Rice University
Université catholique de Louvain
Florida State University

ROPTLIB: Riemannian Manifold Optimization Library

User Manual

Author:
Wen Huang
Collaborators:
Kyle A. Gallivan
P.-A. Absil
Paul Hand

Affiliations:
Rice University

Florida State University
Université catholique de Louvain
Rice University

February 24, 2017

Contents

1 Introduction

The Riemannian manifold optimization library ROPTLIB is used to optimize a cost function defined on a Riemannian manifold. State of the art algorithms, shown in Table 1, are included. The package is written in C++ using the standard linear algebra libraries BLAS and LAPACK. It can be used in a C++ environment, a Matlab environment and a Julia environment. Users only need to provide the cost function, the gradient function, and the action of the Hessian (if a Newton method is used) in C++, Matlab or Julia. The package optimizes a given cost function using some parameters specified by users, e.g., the domain manifold, algorithm, stopping criterion.

Table 1: Riemannian algorithms in ROPTLIB

Name	literature	required objects	smoothness of the cost function
Riemannian trust-region Newton (RTRNewton)	[ABG07]	Grad/Hess	Smooth
Riemannian trust-region symmetric rank-one update (RTRSR1)	[HAG15]	Grad	Smooth
Limited-memory RTRSR1 (LRTRSR1)	[HAG15]	Grad	Smooth
Riemannian trust-region steepest descent (RTRSD)	[AMS08]	Grad	Smooth
Riemannian line-search Newton (RNewton)	[AMS08]	Grad/Hess	Smooth
Riemannian Broyden family (RBroydenFamily)	[HGA15]	Grad	Smooth
Riemannian BFGS (RWRBFGS and RBFGS)	[RW12] [HGA15]	Grad	Smooth/L-continuous
Subgradient Riemannian (L)BFGS ((L)RBFGSLPSub)	[AHHY16]	Grad	L-continuous
Limited-memory RBFGS (LRBFGS)	[HGA15]	Grad	Smooth
Riemannian conjugate gradients (RCG)	[NW06] [AMS08] [SI13]	Grad	Smooth
Riemannian steepest descent (RSD)	[AMS08]	Grad	Smooth
Riemannian gradient sampling (RGS)	[Hua13] [SH16]	Grad	L-continuous

2 Installation and First Example

The package has been tested on Windows 7, Ubuntu 16.04 and MAC OS X 10.8.5 when the code is compiled in Matlab. It has been tested on Ubuntu 16.04[1] when the code is compiled in Julia. It also has been tested on Windows 7, Ubuntu 16.04, and MAC OS X 10.10 when the code is compiled in C++ environment alone.

2.1 Compiling in Matlab

The command "mex -setup" in Matlab sets up the MEX environment properly. Users are not required to install BLAS and LAPACK in this case since those libraries are included in Matlab.

To compile ROPTLIB and run a test example, first go to the root directory, i.e., `/ROPTLIB/`. Run "GenerateMyMex.m". A file called "MyMex.m" will be created or updated. It is used to compile the package. The command "MyMex" followed by the name of any test files in `/ROPTLIB/test/` compiles the test file. For example, run "MyMex TestStieBrockett" to test the Brockett cost function on the Stiefel manifold [AMS08, Section 4.8]. A binary file "TestStieBrockett.***" will

[1] ROPTLIB on an Ubuntu system needs dependencies. When installing ROPTLIB in Ubuntu, if "gl.h" is missing, using the command "sudo apt-get install mesa-common-dev"; if "-lgl" is not defined, then use the command "sudo apt-get install build-essential libgl1-mesa-dev".

be generated in `/ROPTLIB/test/BinaryFiles/` , where the suffix *** depends on the systems. Finally, use the command "TestStieBrockett" to run the binary file. The commands and results can be found in Listing 1. The explanations of the notation can be found in Appendix B.

Listing 1: Test code

```
>> GenerateMyMex
Generate MyMex.m file...
>> MyMex TestStieBrockett
Building with 'g++-4.7'.
MEX completed successfully.
>> n = 12; p = 4; B = randn(n, n); B = B + B'; D = (p:-1:1)';
>> Xinitial = orth(randn(n, p));
>> SolverParams.method = 'RBFGS'; SolverParams.IsCheckParams = 1; SolverParams.DEBUG = 1;
>> HasHHR = 0; ParamSet = 1;
>> [Xopt, f, gf, gfgf0, iter, nf, ng, nR, nV, nVp, nH, ComTime] = TestStieBrockett(B, D,
    Xinitial, HasHHR, ParamSet, SolverParams);
(n, p):12,4
GENERAL PARAMETERS:
Stop_Criterion:       GRAD_F_0[YES],    Tolerance     :          1e-06[YES]
Max_Iteration :            500[YES],    Min_Iteration :              0[YES]
OutputGap     :              1[YES],    DEBUG         :    FINALRESULT[YES]
LINE SEARCH TYPE METHODS PARAMETERS:
LineSearch_LS :         ARMIJO[YES],    LS_alpha      :         0.0001[YES]
LS_ratio1     :            0.1[YES],    LS_ratio2     :            0.9[YES]
Initstepsize  :              1[YES]
Minstepsize   :    2.22045e-16[YES],    Maxstepsize   :           1000[YES]
RBFGS METHOD PARAMETERS:
nu            :         0.0001[YES],    mu            :              1[YES]
isconvex      :              0[YES]
=========================RBFGS=========================
Iter:51,f:-5.809e+01,|gf|:1.912e-05,|gf|/|gf0|:4.108e-07,time:0.00e+00,nf:57,ng:52,nR:56,nH
    :51,nV(nVp):51(51),
```

2.2 Compiling in Julia

We first generate a shared library by g++. Julia uses this shared library to call ROPTLIB through a C++ interface Cxx `http://julialang.org/`. The details are as follows:

1. In order to compile ROPTLIB in Julia, the "Cxx" library at `https://github.com/Keno/Cxx.jl` is required. Julia and "Cxx" are installing by following the instruction on `https://github.com/Keno/Cxx.jl`. The steps when this user manual is generated are given below for completeness (it takes a few hours):

   - Download Ubuntu 16.04.1 LTS from `https://www.ubuntu.com/download/desktop`.
   - Install the Ubuntu
   - Install required packages by the command

     ```
     sudo apt-get install mesa-common-dev build-essential libgl1-mesa-dev
     sudo apt-get install cmake libedit-dev libncurses5-dev git
     ```

   - Go to the directory for Julia. Then download julia source codes by the command

     ```
     git clone git://github.com/JuliaLang/julia.git
     cd julia
     git checkout release-0.5
     ```

   - Install required packages for compling julia by

```
sudo apt-get install gfortran m4 libssl-dev
```

- Run "make" to compile julia. (You may need to run "sudo apt-get update or sudo apt-get upgrade when some packages are installed.)
- Run julia and in the command line of julia, run the command

```
Pkg.clone("https://github.com/Keno/Cxx.jl.git")
Pkg.build("Cxx")
```

to install "Cxx".

2. Download the latest version of ROPTLIB and go the directory of ROPTLIB.
3. Open `ROPTLIB/Makefile` and make sure "ROOTPATH" is set to be the correct directory of ROPTLIB and `JULIA_DIR` is the directory of Julia.
4. Install BLAS and LAPACK:

```
sudo apt-get install build-essential
sudo apt-get install liblapack*
sudo apt-get install libblas*
```

5. Run "make JuliaROPTLIB TP=DriverJuliaProb" to obtain a shared library of ROPTLIB for Julia.
6. Open the downloaded Julia. Go to the directory of ROPTLIB in Julia using command

```
julia> cd("directory_of_ROPTLIB")
```

7. Open `ROPTLIB/Julia/BeginROPTLIB.jl` and make sure that the path of ROPTLIB is correct and the path of head files of Julia is correct.
8. Run `ROPTLIB/Julia/BeginROPTLIB.jl` by the command "include("Julia/BeginROPTLIB.jl")" to import ROPTLIB into Julia.
9. Run JTestSimpleExample.jl by the command "include("Julia/JTestSimpleExample.jl")" to run an example.

2.3 Compiling in a Stand-alone C++ Enviroment

Users must first install BLAS and LAPACK. For details on a Windows installation, see the links: `http://www.fi.muni.cz/~xsvobod2/misc/lapack/` and `http://www.netlib.org/`. The steps of compiling this code in Windows 7 using IDE Visual Studio Express 2013 are: i) Click "PROJECT→properties"; ii) add directory of ROPTLIB and the directories of header files of BLAS and LAPACK to "Configuration properties→VC++ Directories→General→Include directories"; iii) add the libraries of BLAS, and LAPACK to "Configuration properties→Linker→Input→Additional Dependencies". To compile and run a test file, first, open `/ROPTLIB/test/Others/def.h`. Uncomment one of commands from line 17 to line 38 to specify a test problem. Finally, press F5 or ctrl + F5 to compile and run the test problem.

In Ubuntu, the steps are:

1. Install BLAS and LAPACK:

```
sudo apt-get install build-essential
sudo apt-get install liblapack*
sudo apt-get install libblas*
```

2. Download the latest version of ROPTLIB and go the the directory of ROPTLIB. Open `ROPTLIB/Makefile` and make sure "ROOTPATH" is set to be the correct directory of ROPTLIB.

3. Run Makefile to generate a binary file for a test problem, i.e., using the command:

   ```
   make ROPTLIB TP=name_of_the_test_file
   ```

   where "`name_of_the_test_file`" can be any test problem in `ROPTLIB/test/*`. For example, if one wants to generate a binary for "TestStieBrockett.cpp", then use the command:

   ```
   make ROPTLIB TP=TestStieBrockett
   ```

   Note the suffix ".cpp" is not used.

4. Run "./TestStieBrockett" to see the test results.

In MAC, the steps to set up in Xcode 7 are:

1. Download BLAS and LAPACK from `http://www.netlib.org/blas/` and `http://www.netlib.org/lapack/`;

2. Unzip both packages. In terminal, run "make" in both folders to generate *.a libraries;

3. Rename BLAS and LAPACK libraries to "libblas.a" and "liblapack.a" respectively;

4. Download the latest version of ROPTLIB and go the the directory of ROPTLIB.

5. Open Xcode, create a project, add ROPTLIB to this project by simply drag the subfolders to the project in Xcode. Note that the last option, "Add to targets", in the pop-up window must be checked.

6. In Xcode, link BLAS and LAPACK libraries by adding them to
   `Build Phases/Link Binary With Libraries`

7. In Xcode, add the path of ROPTLIB and paths of the header files of BLAS and LAPACK to
   `Building Settings/Search Paths/Header Search Paths`
   and
   `Building Settings/Search Paths/User Header Search Paths`

8. To compile and run a test file, first, open `/ROPTLIB/test/Others/def.h`. Uncomment one of commands from line 17 to line 38 to specify a test problem. Then compile and run.

3 For Matlab Users

3.1 Test Problems and Matlab Interface

ROPTLIB contains three parts, including problem definition, manifold and solver. In order to use this package, a user must define a problem by providing functions of a cost function, its gradient

and the action of the Hessian (if Newton's method is used) and specify a domain manifold and a solver.

Two problems are used as examples. The first is the Brockett cost function on the Stiefel manifold $\mathrm{St}(p,n) = \{X \in \mathbb{R}^{n\times p} | X^T X = I_p\}$ [AMS08, Section 4.8]

$$\min_{X\in\mathrm{St}(p,n)} \mathrm{trace}(X^T B X D) \tag{3.1}$$

where $B \in \mathbb{R}^{n\times n}$, $B = B^T$, $D = \mathrm{diag}(\mu_1, \mu_2, \ldots, \mu_p)$ and $\mu_1 \geq \mu_2 \geq \ldots \geq \mu_p$. The second is a summation of three Brockett cost functions

$$\min_{(X_1,X_2,X_3)\in\mathrm{St}(p,n)\times\mathrm{St}(p,n)\times\mathrm{St}(q,m)} \mathrm{trace}(X_1^T B_1 X_1 D_1) + \mathrm{trace}(X_2^T B_2 X_2 D_2) + \mathrm{trace}(X_3^T B_3 X_3 D_3) \tag{3.2}$$

where $B_1, B_2 \in \mathbb{R}^{n\times n}$, $B_3 \in \mathbb{R}^{m\times m}$, $B_1 = B_1^T$, $B_2 = B_2^T$, $B_3 = B_3^T$, $D_1 = \mathrm{diag}(\mu_1, \mu_2, \ldots, \mu_p)$, $\mu_1 \geq \mu_2 \geq \ldots \geq \mu_p$, $D_2 = \mathrm{diag}(\nu_1, \nu_2, \ldots, \nu_p)$, $\nu_1 \geq \nu_2 \geq \ldots \geq \nu_p$, $D_3 = \mathrm{diag}(\sigma_1, \sigma_2, \ldots, \sigma_q)$, and $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_q$. Problem (3.2) is used to illustrate an implementation for a problem on a product manifold.

First, use the command "MyMex DriverMexProb" to generate a binary file. This binary can be called from Matlab by inputting function handles and parameter structures. We have wrapped this function by a script `/ROPTLIB/Matlab/DriverOPT.m`. DriverOPT.m is used to check correctness of the input parameters and reshape the data from C++ solvers.

Listing 2: Matlab interface

```
1 [FinalIterate, fv, gfv, gfgf0, iter, nf, ng, nR, nV, nVp, nH, ComTime, funs, grads, times,
      dists] = DriverOPT(fhandle, gfhandle, Hesshandle, SolverParams,
2                        ManiParams, HasHHR, initialIterate, solution)
```

The script DriverOPT can be called by Listing 2, where "initialIterate" and "finalIterate" are structures that contain initial and final iterates respectively; "fv" is the final cost function value; "gfv" is the norm of the final gradient; "gfgf0" is the norm of the final gradient over the norm of the initial gradient; "iter", "nf", "ng", "nR", "nV/nVp", "nH" donote the number of iterations, the number of function evaluations, the number of gradient evaluations, the number of retraction evaluations, the number of vector transports (expensive/cheap)[2], and the number of evaluations of the action of the Hessian respectively; "ComTime" denotes the total computational time; "funs", "grads", and "times" are arrays that store the function values, norms of gradients and the accumulated computational time at each iteration. If the minimizer that the sequence converges to is known (Given as the last argument in this function), then the array "dist" stores distances between every iterate to the minimizer. "fhandle", "gfhandle", and "Hesshandle" are function handles of cost function, its Euclidean gradient and the action of its Euclidean Hessian; "SolverParams" and "ManiParams" are structures that specify parameters of the solver and manifold respectively; and "HasHHR" indicates whether the locking condition [HGA15, (2.8)] is satisfied using the testing approach in [HGA15, Section 4.1].

[2]Two numbers of vector transports are reported. The first time an action of a vector transport $\mathcal{T}_\eta$ is computed will usually have higher complexity than subsequent times. Specifically, if $\mathcal{T}_\eta\xi_1$ has been computed, then evaluating $\mathcal{T}_\eta\xi_2$ usually can use some results from computations of $\mathcal{T}_\eta\xi_1$.

3.2 A Simple Example

An example for Brockett cost function (3.1) is given in Listing 3 and the code can be found in `/ROPTLIB/Matlab/ForMatlab/testSimpleExample.m`.[3] First, the cost function, the Euclidean gradient and action of the Euclidean Hessian are given from line 32 to line 43. Their function handles are assigned from line 5 to line 7. Iterates and tangent vectors are stored as structures with the field "main", as shown in line 18 and line 34. In order to store temporary data to save computations, users can put the temporary data on an iterate with a different field. For example, the Brockett cost function is trace(X^TBXD) and the Euclidean gradient is $2BXD$. It is required to evaluate BXD in the cost function evaluation. Therefore, one can use the result from the function evaluation to reduce computation in the gradient evaluation. This can be seen from the definitions of f(x, B, D) and gf(x, B, D) in line 33 and line 38. All fields for each solver and manifold are defined in Appendices B and C.

Note that besides using the default line search algorithms and the default stopping criteria, users are allowed to define their own stopping criterion and line search algorithm. Lines 10 to 12 specify the stopping criterion and line search algorithm using the functions defined from line 24 to line 30. The input variables x, eta, $t0$, $s0$ and $output$ defined in

```
output = LinesearchInput(x, eta, t0, s0)
```

represent the current iterate, the search direction, the suggested initial stepsize and the initial slope respectively. If the parameter of line search solvers, *IsPureLSInput* (see Appendix B), is set to be false, then the step size found by "LinesearchInput" will be used as the initial step size for a backtracking algorithm. Otherwise, the step size will be the accepted step size. The variables x, gf, f, ngf and $ngf0$ defined in

```
output = IsStopped(x, gf, f, ngf, ngf0)
```

represent the current iterate, the current gradient, the function value at x, the norm of the gradient at x and the norm of the gradient at the initial iterate respectively.

Listing 3: Test Brockett

```
1  function [FinalX, fv, gfv, gfgf0, iter, nf, ng, nR, nV, nVp, nH, ComTime, funs, grads, times
       ] = testBrockett()
2    n = 5; p = 2;                    % size of the Stiefel manifold
3    B = randn(n, n); B = B + B';     % data matrix
4    D = sparse(diag(p : -1 : 1));    % data matrix
5    fhandle = @(x)f(x, B, D);        % cost function handle
6    gfhandle = @(x)gf(x, B, D);      % gradient
7    Hesshandle = @(x, eta)Hess(x, eta, B, D); % Hessian
8
9    SolverParams.method = 'RSD';     % Use RSD solver
10   SolverParams.IsStopped = @IsStopped; % Don't use one of the default stopping criteria. Use
         the one specified by the IsStopped function handle.
11   SolverParams.LineSearch_LS = 5; % Don't use one of the default line search algorithm. Use
         the one specified by the LinesearchInput function handle.
12   SolverParams.LinesearchInput = @LinesearchInput;
13
14   ManiParams.name = 'Stiefel';     % Domain is the Stiefel manifold
15   ManiParams.n = n;                % assign size to manifold parameter
16   ManiParams.p = p;                % assign size to manifold parameter
17
18   initialX.main = orth(randn(n, p));  % initial iterate
```

[3]The code in the file may not be exactly the same as that in the Listings. The code in the file tests more parameters and runs more/different algorithms. Therefore, the differences are minor and should not cause confusion.

```
19
20    % call the driver
21    [FinalX, fv, gfv, gfgf0, iter, nf, ng, nR, nV, nVp, nH, ComTime, funs, grads, times] =
          DriverOPT(fhandle, gfhandle, Hesshandle, SolverParams, ManiParams, initialX);
22  end
23
24  function output = LinesearchInput(x, eta, t0, s0)
25      output = 1;
26  end
27
28  function output = IsStopped(x, gf, f, ngf, ngf0)
29      output = ngf / ngf0 < 1e-5;
30  end
31
32  function [output, x] = f(x, B, D)
33    x.BUD = B * x.main * D;
34    output = x.main(:)' * x.BUD(:);
35  end
36
37  function [output, x] = gf(x, B, D)
38    output.main = 2 * x.BUD;
39  end
40
41  function [output, x] = Hess(x, eta, B, D)
42    output.main = 2 * B * eta.main * D;
43  end
```

3.3 An Example for a Product of Manifolds

An example for a summation of three Brockett cost functions is given in Listing 4, and the associated code can be found in `/ROPTLIB/Matlab/ForMatlab/testProductExample.m`.[4] An array of structures is used to specify a product of manifolds. Suppose the manifold $\mathcal{M}$ is $\mathcal{M}_1^{t_1} \times \mathcal{M}_2^{t_2} \times \ldots \times \mathcal{M}_s^{t_s} := \mathcal{M}_1 \times \ldots \times \mathcal{M}_1 \times \mathcal{M}_2 \times \ldots \mathcal{M}_2 \times \ldots \times \mathcal{M}_s \times \ldots \times \mathcal{M}_s$, where the number of $\mathcal{M}_i$ is t_i. Then the structure specifying parameters of manifolds is an array with length s and the field "numofmani" in i-th element of the array is assigned to be t_i. One example can be found in the function "testSumBrockett()" of Listing 4 from line 20 to line 27.

All components of an iterate of products of manifolds are stored in a consecutive memory. Suppose the length of the i-th component of iterate in product of manifold $\mathcal{M}_1 \times \mathcal{M}_2 \times \ldots \times \mathcal{M}_w$ is ℓ_i. The i-th component of the iterate is stored in the space from $\sum_{j=1}^{i-1} \ell_j + 1$ to $\sum_{j=1}^{i} \ell_j$ in the field "main" of the iterate structure. The same method is used to store tangent vectors. An example is given in lines 29 to 31, 40 to 42, 49, 56 to 58 and 62 in Listing 4.

Listing 4: Test Summation of Brockett

```
1   function [FinalX, fv, gfv, gfgf0, iter, nf, ng, nR, nV, nVp, nH, ComTime, funs, grads, times
        ] = testSumBrockett()
2     n = 5;
3     p = 2;
4     m = 6;
5     q = 3;
6     B1 = randn(n, n); B1 = B1 + B1';
7     D1 = sparse(diag(p : -1 : 1));
8     B2 = randn(n, n); B2 = B2 + B2';
9     D2 = sparse(diag(p : -1 : 1));
10    B3 = randn(m, m); B3 = B3 + B3';
```

[4] The code in the file may not be exactly the same as that in the Listings. The code in the file tests more parameters and runs more/different algorithms. Therefore, the differences are minor and should not cause confusion.

```
11    D3 = sparse(diag(q : -1 : 1));
12
13    fhandle = @(x)f(x, B1, D1, B2, D2, B3, D3);
14    gfhandle = @(x)gf(x, B1, D1, B2, D2, B3, D3);
15    Hesshandle = @(x, eta)Hess(x, eta, B1, D1, B2, D2, B3, D3);
16
17    SolverParams.method = 'RSD';
18
19    % Set up domain of manifold, St(p, n)^2 \times St(q, m)
20    ManiParams(1).name = 'Stiefel';
21    ManiParams(1).numofmani = 2;      % the number of St(p, n) is two
22    ManiParams(1).n = n;
23    ManiParams(1).p = p;
24    ManiParams(2).name = 'Stiefel';
25    ManiParams(2).numofmani = 1;   % the number of St(q, m) is one
26    ManiParams(2).n = m;
27    ManiParams(2).p = q;
28
29    % generate initial iterate
30    X1 = orth(randn(n, p)); X2 = orth(randn(n, p)); X3 = orth(randn(m, q));
31    initialX.main = [X1(:); X2(:); X3(:)];
32
33    [FinalX, fv, gfv, gfgf0, iter, nf, ng, nR, nV, nVp, nH, ComTime, funs, grads, times] =
          DriverOPT(fhandle, gfhandle, Hesshandle, SolverParams, ManiParams, initialX);
34  end
35
36  function [output, x] = f(x, B1, D1, B2, D2, B3, D3)
37    n = size(B1, 1); p = size(D1, 1);
38    m = size(B3, 1); q = size(D3, 1);
39
40    X1 = reshape(x.main(1 : n * p), n, p);
41    X2 = reshape(x.main(n * p + 1 : 2 * n * p), n, p);
42    X3 = reshape(x.main(2 * n * p + 1 : 2 * n * p + m * q), m, q);
43
44    x.BUD1 = B1 * X1 * D1; x.BUD2 = B2 * X2 * D2; x.BUD3 = B3 * X3 * D3;
45    output = X1(:)' * x.BUD1(:) + X2(:)' * x.BUD2(:) + X3(:)' * x.BUD3(:);
46  end
47
48  function [output, x] = gf(x, B1, D1, B2, D2, B3, D3)
49    output.main = [x.BUD1(:); x.BUD2(:); x.BUD3(:)];
50    output.main = 2 * output.main;
51  end
52
53  function [output, x] = Hess(x, eta, B1, D1, B2, D2, B3, D3)
54    n = size(B1, 1); p = size(D1, 1);
55    m = size(B3, 1); q = size(D3, 1);
56    eta1 = reshape(eta.main(1 : n * p), n, p);
57    eta2 = reshape(eta.main(n * p + 1 : 2 * n * p), n, p);
58    eta3 = reshape(eta.main(2 * n * p + 1 : 2 * n * p + m * q), m, q);
59    xi1 = 2 * B1 * eta1 * D1;
60    xi2 = 2 * B2 * eta2 * D2;
61    xi3 = 2 * B3 * eta3 * D3;
62    output.main = [xi1(:); xi2(:); xi3(:)];
63  end
```

3.4 Checking the Correctness of the Gradient and the Action of the Hessian

ROPTLIB provides a function to test the correctness of the gradient and the action of the Hessian. Let $\hat{f}_x(\eta_x)$ be $f(R_x(\eta_x))$. If $f \in C^2$, then using Taylor's theorem yields

$$\begin{aligned}\hat{f}_x(\eta_x) =& \hat{f}_x(0_x) + \langle \operatorname{grad} \hat{f}_x(0_x), \eta_x\rangle + \frac{1}{2}\langle \operatorname{Hess} \hat{f}_x(0_x)[\eta_x], \eta_x\rangle + o(\|\eta_x\|^2)\\ =& f(x) + \langle \operatorname{grad} f(x), \eta_x\rangle + \frac{1}{2}\langle \operatorname{Hess} \hat{f}_x(0_x)[\eta_x], \eta_x\rangle + o(\|\eta_x\|^2).\end{aligned}$$

If the retraction R is a second-order retraction or x is a stationary point of f, then $\operatorname{Hess} \hat{f}_x(0_x) = \operatorname{Hess} f(x)$ by [AMS08, Propositions 5.5.5 and 5.5.6]. It follows that

$$f(y) = f(x) + \langle \operatorname{grad} f(x), \eta_x\rangle + \frac{1}{2}\langle \operatorname{Hess} f(R_x(\eta_x))[\eta_x], \eta_x\rangle + o(\|\eta_x\|^2),$$

where $y = R_x(\eta_x)$. The function in this package computes

$$(f(y) - f(x))/\langle \operatorname{grad} f(x), \eta_x\rangle \tag{3.3}$$

and

$$(f(y) - f(x) - \langle \operatorname{grad} f(x), \eta_x\rangle)/(0.5\langle \operatorname{Hess} f(R_x(\eta_x))[\eta_x], \eta_x\rangle) \tag{3.4}$$

for $\eta_x = \alpha\xi$ such that $\|\xi\| = 1$, α decreases from 100 to $100 * 2^{-35}$. Suppose there exists an interval of α such that numerical errors do not have significant effect and the values of α are sufficiently small so that the higher order term is negligible. If (3.3) is approximately 1 in the interval, then $\operatorname{grad} f$ is probably correct. Likewise, if (3.4) is approximately 1 in the interval, the retraction R is a second-order retraction or x is a stationary point of f, then the $\operatorname{Hess} f$ is probably correct.

To run the function, users must set the field "IsCheckGradHess" to 1 in the solver's parameters. For example, adding the command "SolverParams.IsCheckGradHess = 1" in line 13 of the function "testBrockett()" in Listing 3 sets "IsCheckGradHess" to 1. Two sets of values (3.3) and (3.4) are output. One is at the initial iterate and the other at the final iterate obtained by the solver. The values of (3.3) at the initial iterate indicates if the Riemannian gradient and Euclidean gradient are correct and the values of (3.4) at the final iterate indicates if the actions of the Riemannian Hessian and Euclidean Hessian are correct.

4 For Julia Users

4.1 A Simple Example

In Julia, a shared library of ROPTLIB needs to be generated first (see Section 2 for details). ROPTLIB then can be added to Julia by running `ROPTLIB/Julia/BeginROPTLIB.jl`. The interface in Julia is given by

```
1 [FinalIterate, fv, gfv, gfgf0, iter, nf, ng, nR, nV, nVp, nH, ComTime, funs, grads, times,
    dists] = DriverJuliaOPT(Handles, SolverParams, ManiParams, HasHHR, initialIterate,
    solution)
```

where the notation is the same as those in Listing 2 except that "Handles" is a composite type containing all the function names (see an example in lines 8 to 10 in Listing 5).

Listing 5 shows an example to optimize the Brockett cost function in (3.1). The code is available in `/ROPTLIB/Julia/JTestSimpleExample.jl`. The manifold is specified from lines 1 to 12. Note that the name and size of a manifold are defined to be an array. This is done to make the code compatible for a product of manifolds. See `/ROPTLIB/Julia/JTestProductExample.jl` or Section 4.2 for an example on a product of manifolds and related information. The names of the cost function, gradient, action of Hessian, stopping criterion, and line search algorithm are given from lines 15 to 17. The functions are defined later from lines 44 to 85. The solver-related parameters are defined from lines 22 to 30. Unlike *ManiParams* and *FunHandles*, an object **Sparams** of *SolverParams* has been defined in BeginROPTLIB.jl. Therefore, users do not need to create an object of type *SolverParams* but need only modify **Sparams**. The default values of **Sparams** can be found in Appendix B.

The Julia interface also supports sharing data across functions. As shown in line 32 of Listing 9, the temporary data is stored in the object **outTmp**. In the gradient evaluation, the temporary data is given in the object **inTmp** and can be used to avoid redundant computations.

Note that size information about all data is not explicitly stored with the data in ROPTLIB and, therefore, it is required to reshape the data, as shown in lines 30, 37, and 43. This has little impact on the efficiency of ROPTLIB since the data in memory do not change when reshaped.

Listing 5: Test Brockett

```
1  # set domain manifold to be the Stiefel manifold.
2  # Note that every parameter in ManiParams is an array. The idea to use an array
3  # is to make the framework compatible with produce of manifolds. See details in
4  # JTestProductExample.jl
5  mani1 = "Stiefel"; ManArr = [pointer(mani1)]
6  UseDefaultArr = [-1] # -1 means that the default value in C++ is used.
7  numofmani = [1]
8  # The size is R^{5 \times 3}
9  ns = [5];
10 ps = [3];
11 paramsets = [2];
12 Mparams = ManiParams(1, length(ManArr), pointer(ManArr), pointer(numofmani), pointer(
       paramsets), pointer(UseDefaultArr), pointer(ns), pointer(ps))
13
14 # set function handles
15 fname = "func"
16 gfname = "gfunc"
17 hfname = "hfunc"
18 isstopped = "stopfunc" # or empty string "" if use a default one
19 LinesearchInput = "LSfunc" # or empty string "" if use a default one
20 Handles = FunHandles(pointer(fname), pointer(gfname), pointer(hfname), pointer(isstopped),
       pointer(LinesearchInput))
21
22 # set solvers by modifying the default one.
23 method = "LRBFGS"
24 Sparams.name = pointer(method)
25 Sparams.OutputGap = 1
26 Sparams.LineSearch_LS = 5
27 Sparams.Max_Iteration = 50
28 Sparams.IsPureLSInput = 0
29 Sparams.Stop_Criterion = 0
30 Sparams.IsCheckGradHess = 1
31
32 # use locking condition or not
33 HasHHR = 0
34
35 # Initial iterate and problem
36 srand(1)
```

```
37 n = ns[1]
38 p = ps[1]
39 B = randn(n, n)
40 B = B + B'
41 D = sparse(diagm(linspace(p, 1, p)))
42 initialX = qr(randn(ns[1], ps[1]))[1]
43
44 # Define function handles
45 # The function names are assigned to the "FunHandles" struct.
46 # See lines 17-21
47 function func(x, inTmp)
48         x = reshape(x, n, p) # All the input argument is a vector. One has to reshape it to
              have a proper size
49         outTmp = B * x * D
50         fx = vecdot(x, outTmp)
51         return (fx, outTmp) # The temparary data "outTmp" will replace the "inTmp"
52 end
53
54 function gfunc(x, inTmp) # The inTmp is the temparary data computed in "func".
55         inTmp = reshape(inTmp, n, p) # All the input argument is a vector. One has to
              reshape it to have a proper size
56         gf = 2.0::Float64 * inTmp
57         return (gf, []) # If one does not want to change the temparary data, then let the
              outTmp be an empty array.
58 end
59
60 function hfunc(x, inTmp, eta)
61         eta = reshape(eta, n, p) # All the input argument is a vector. One has to reshape it
               to have a proper size
62         result = 2.0::Float64 * B * eta * D
63         return (result, []) # If one does not want to change the temparary data, then let
              the outTmp be an empty array.
64 end
65
66 # Users can define their own stopping criterion by passing the name
67 # of the function to the "isstopped" field in the object of structure FunHandles
68 function stopfunc(x, gf, fx, ngfx, ngfx0)
69 # x: the current iterate
70 # gf: the gradient at x
71 # gx: the function value at x
72 # ngfx: the norm of gradient at x
73 # ngfx0: the norm of gradient at the initial iterate
74         return (ngfx / ngfx0 < 1e-6)
75 end
76
77 # Users can define their own line search method by passing the name
78 # of the function to the "LinesearchInput" field in the object of structure FunHandles
79 function LSfunc(x, eta, t0, s0)
80 # x: the current iterate
81 # eta: the search direction
82 # t0: the initial step size
83 # s0: the slope of the line search scalar function at zero
84         return 1.0::Float64
85 end
86
87 # Call the solver and get results. See the user manual for details about the outputs.
88 (FinalIterate, fv, gfv, gfgf0, iter, nf, ng, nR, nV, nVp, nH, ComTime, funs, grads, times) =
       DriverJuliaOPT(Handles, Sparams, Mparams, HasHHR, initialX)
```

4.2 An Example for a Product of Manifolds

An example for minimzing a summation of Brockett cost functions 3.2 is given in Listing 7. The setting is the same as that in Section 4.1. All the data are stored in consecutive memory and

the shape information is not explicitly stored with the data. Therefore, as in e.g., lines 58 to 60 and lines 72 to 74, each component is obtained by extracting and reshaping the input variables using knowledge of the manner in which the initial data was specified and the relevant manifold parameters.

As shown in Table 32 of Appendix C, the size information of a manifold is specified by at most three letters, m, n, and p. The example shown in Listing 7 only involves p and n. Here we show what if m is also involved. Suppose the product manifold is $\mathrm{St}(p,n) \times \mathcal{OB}(n,m) \times (\mathcal{O}(n))^3$. The code to generate such a product manifold is given in Listing 6. If a manifold does not need a value of a letter, such as the Stiefel manifold does not need a value of m, then its corresponding value can be set to be any value and we use 0 in the code.

Listing 6: Generate the product of manifolds

```
# set domain manifold to be the product of Stiefel manifolds: St(p, n)^2 \times St(q, m).
mani1 = "Stiefel"
mani2 = "Oblique"
mani3 = "OrthGroup"
ManArr = [pointer(mani1), pointer(mani2), pointer(mani3)]
UseDefaultArr = [-1, -1, -1] # -1 means that the default values in C++ are used.
numofmani = [1, 1, 3] # Orthogonal group has power 3, therefore, the corresponding number is set to be 3.
St_p = 3, St_n = 5, OB_n = 4, OB_m = 6, Or_n = 10
ms = [0, OB_m, 0]; # first one is for Stiefel, Second one is for Oblique and the last one is for Orthogonal group
ns = [St_n, OB_n, Or_n];
ps = [St_p, 0, 0];
paramsets = [1, 1, 1];
Mparams = ManiParams(1, length(ManArr), pointer(ManArr), pointer(numofmani), pointer(paramsets), pointer(ms), pointer(ns), pointer(ps))
```

Listing 7: Test summation of Brockett

```
# set domain manifold to be the product of Stiefel manifolds: St(p, n)^2 \times St(q, m).
mani1 = "Stiefel"
mani2 = "Stiefel"
ManArr = [pointer(mani1), pointer(mani2)]
UseDefaultArr = [-1, -1] # -1 means that the default values in C++ are used.
numofmani = [2, 1] # St(p, n) has power 2, therefore, the corresponding number is set to be 2.
# p = 3, n = 5, q = 2, m = 6
ns = [5, 6];
ps = [3, 2];
paramsets = [1, 1];
Mparams = ManiParams(1, length(ManArr), pointer(ManArr), pointer(numofmani), pointer(paramsets), pointer(UseDefaultArr), pointer(ns), pointer(ps))

# set function handles
fname = "func_P"
gfname = "gfunc_P"
hfname = "hfunc_P"
isstopped = "stopfunc_P"
LinesearchInput = "LSfunc_P"
Handles = FunHandles(pointer(fname), pointer(gfname), pointer(hfname), pointer(isstopped), pointer(LinesearchInput))

# set solvers by modifying the default one.
method = "LRBFGS"
Sparams.name = pointer(method)
Sparams.OutputGap = 1
Sparams.LineSearch_LS = 5
Sparams.Max_Iteration = 50
```

```
27  Sparams.IsPureLSInput = 0
28
29  # use locking condition or not
30  HasHHR = 0
31
32  # Initial iterate and problem
33  srand(1)
34  n = ns[1]
35  p = ps[1]
36  m = ns[2]
37  q = ps[2]
38  B1 = randn(n, n)
39  B1 = B1 + B1'
40  D1 = sparse(diagm(linspace(p, 1, p)))
41  B2 = randn(n, n)
42  B2 = B2 + B2'
43  D2 = sparse(diagm(linspace(p, 1, p)))
44  B3 = randn(m, m)
45  B3 = B3 + B3'
46  D3 = sparse(diagm(linspace(q, 1, q)))
47
48  initialX1 = qr(randn(ns[1], ps[1]))[1]
49  initialX2 = qr(randn(ns[1], ps[1]))[1]
50  initialX3 = qr(randn(ns[2], ps[2]))[1]
51  initialX = [reshape(initialX1, n * p, 1); reshape(initialX2, n * p, 1); reshape(initialX3, m
        * q, 1)]
52
53
54  # Define function handles
55  # The function names are assigned to the "FunHandles" struct.
56  # See lines 17-21
57  function func_P(x, inTmp) # All the input argument is a vector.
58          x1 = reshape(view(x, 1 : n * p), n, p)
59          x2 = reshape(view(x, n * p + 1 : 2 * n * p), n, p)
60          x3 = reshape(view(x, 2 * n * p + 1 : 2 * n * p + m * q), m, q)
61          outTmp = [reshape(B1 * x1 * D1, n * p, 1); reshape(B2 * x2 * D2, n * p, 1); reshape(
                B3 * x3 * D3, m * q, 1)]
62          fx = vecdot(x, outTmp)
63          return (fx, outTmp) # The temparary data "outTmp" will replace the "inTmp"
64  end
65
66  function gfunc_P(x, inTmp)
67          gf = 2.0::Float64 * inTmp
68          return (gf, []) # If one does not want to change the temparary data, then let the
                outTmp be an empty array.
69  end
70
71  function hfunc_P(x, inTmp, eta)
72          eta1 = reshape(view(eta, 1:n*p), n, p) # All the input argument is a vector. One has
                 to reshape it to have a proper size
73          eta2 = reshape(view(eta, n*p+1:2*n*p), n, p)
74          eta3 = reshape(view(eta, 2*n*p+1:2*n*p+m*q), m, q)
75
76          result = [reshape(2.0::Float64 * B1 * eta1 * D1, n * p, 1); reshape(2.0::Float64 *
                B2 * eta2 * D2, n * p, 1); reshape(2.0::Float64 * B3 * eta3 * D3, m * q, 1)]
77          return (result, []) # If one does not want to change the temparary data, then let
                the outTmp be an empty array.
78  end
79
80  function stopfunc_P(x, gf, fx, ngfx, ngfx0)
81          return (ngfx / ngfx0 < 1e-6)
82  end
83
84  function LSfunc_P(x, eta, t0, s0)
85          return 1.0::Float64
```

```
86  end
87
88  (FinalIterate, fv, gfv, gfgf0, iter, nf, ng, nR, nV, nVp, nH, ComTime, funs, grads, times) =
        DriverJuliaOPT(Handles, Sparams, Mparams, HasHHR, initialX)
```

5 For C++ Users

The classes in the package and their relationships are given in Figures 1 to 4. All the classes that store data inherit an abstract class, *SmartSpace*. The copy-on-write strategy is used in *SmartSpace*. In the abstract class *Manifold*, all functions only related to manifolds are declared, e.g., retraction, vector transport. Some of these functions are also given default definitions, e.g., the default metric is the Frobenius inner product. The abstract class *Problem* contains all prototypes of the cost function, the Riemannian gradient, the Euclidean gradient, the action of the Riemannian Hessian and the action of the Euclidean Hessian. It not only automatically chooses functions that have been overridden (polymorphism), but also includes a function to check the correctness of the gradient and the action of the Hessian, see Section 3.4. The domain of a problem must also be specified using one of the manifold classes. Note that class *mexProblem* is a bridge between C++ and Matlab. It uses function handles of Matlab and produces C++ functions. Each solver accepts an object of *Problem* and an object of *Variable* (an initial iterate), and outputs a final iterate based on the given parameters.

Users must write a problem class by inheriting the abstract class `/ROPTLIB/Problems/Problem.h` and override either functions of cost function, Riemannian gradient and action of Riemannian Hessian

```
virtual double f(Variable *x) const;
virtual void RieGrad(Variable *x, Vector *gf) const;
virtual void RieHessianEta(Variable *x, Vector *etax, Vector *xix) const;
```

or functions of cost function, Euclidean gradient and action of Euclidean Hessian.

```
virtual double f(Variable *x) const;
virtual void EucGrad(Variable *x, Vector *gf) const;
virtual void EucHessianEta(Variable *x, Vector *etax, Vector *exix) const;
```

Throughout this section, a class or a routine is written in *this font* and an object is written in **this font**.

5.1 A Simple Example

An example for the Brockett cost function (3.1) is given in Listings 8, 9 and 10. Listings 8 and 9 give details of two files, StieBrockett.h and StieBrockett.cpp, which inherit the class *Problem* and define the Brockett problem. The Euclidean gradient and the action of the Euclidean Hessian are overridden. Listing 10 gives a test file for the Brockett cost function minimization problem. Those codes can be found in `/ROPTLIB/Problems/StieBrockett/*` and `/ROPTLIB/test/TestSimpleExample.cpp`.[5] If all the test files are included in the user's C++ project, then the user must comment out all the "#define TEST*" except "#define TESTSTIEBROCKETT" in file `/ROPTLIB/Others/def.h` to specify that only TestStieBrockett.cpp is compiled. Otherwise, a user need only include the test file in the user's C++ project.

[5] The code in the file may not be exactly the same as that in the Listings. The code in the file tests more parameters and runs more/different algorithms. Therefore, the differences are minor and should not cause confusion.

For any class derived from *SmartSpace*, any one of the following three functions can be used to obtain a double pointer to the data:

```
virtual const double *ObtainReadData(void) const;
virtual double *ObtainWriteEntireData(void);
virtual double *ObtainWritePartialData(void);
```

ObtainReadData returns a constant pointer and users are not allowed to modify the data. This is the fastest way to access the data but users have the most limited authority. The memory functions *ObtainWriteEntireData* and *ObtainWritePartialData* are allowed to access the data and modify them. *ObtainWriteEntireData* may not preserve the old data in memory and this function is used when users want to completely overwrite the data. *ObtainWritePartialData* guarantees that the memory retains the old data. This is the most inefficient approach but it preserves the old data information and is used if users only partially modify the data.

C++ code provides a way to share information in the computation of the cost function, the gradients and the actions of the Hessians. A class *SharedSpace* is used to store temporary data. One can attach an arbitrary length double array or a derived class of *Element* on it. After constructing a *ShareSpace* object, users can attach it to an object of a class derived from *Element*. One example can be found in Listing 9. A *SharedSpace* object **Temp** is constructed in line 21. The pointer to the data of **Temp** is obtained in line 22. The codes of lines 28 to 34 assign values to the data that **Temp** points to. The codes in line 40 attach the object **Temp** on **x** with name "BxD". Note that the pointer **Temp** is assigned to be null after it is attached to **x**. Therefore, one mush attach the *SharedSpace* object to the element after computing the *SharedSpace* object. In line 51 of Listing 9, a pointer to a *SharedSpace* object is obtained from **x** by using name "BxD". The data in the *SharedSpace* object is obtained in line 52. More examples can be found in files under directory `/ROPTLIB/Problems/`.

In order to avoid memory leaking, a user must delete all the objects that are constructed by the command "new" or "ConstructEmpty()" except objects that are attached to an element. An example is in lines 37 to 44 of Listing 9. The object **Temp** is constructed by the command "new" on line 21 and the object **BxD** is constructed by the function *ConstructEmpty* on line 20. Since **BxD** is attached to **Temp**, manipulating **Temp** takes care of the memory of **BxD** automatically. If **Temp** is attached to **x**, then it cannot be deleted. Otherwise, **Temp** must be deleted.

Users are allowed to define a line search algorithm and a stopping criterion. Lines 15 to 25 in Listing 10 show an example of a definition of a line search algorithm and stopping criterion. The input variables are the same as those in Matlab, see Section 3.2. Their function pointers are assigned to the solvers on lines 69 and 71 in Listing 10. The false value of the parameter *IsPureLSInput* in line 70 indicates that the step size returned by the user-specified function is used as an initial step size in a back tracking algorithm to satisfy the Armijo condition. If the value is true, then the step size is used as the accepted step size. Note that in this case, users must guarantee that the step size is sufficient for convergence.

C++ codes, of course, support checking correctness of the gradients and the actions of the Hessians. An example is given in line 77 to 79 of Listing 10.

Listing 8: File "StieBrockett.h" for test Brockett in C++

```
1  // File: StieBrockett.h
2
3  #ifndef STIEBROCKETT_H
4  #define STIEBROCKETT_H
5
```

```
6  #include "Stiefel.h"
7  #include "StieVariable.h"
8  #include "StieVector.h"
9  #include "Problem.h"
10 #include "SharedSpace.h"
11 #include "def.h"
12
13 // min_X X^T B X D, where B is a symmetric positive definite matrix,
14 // D is a diagonal matrix and X \in St(p, n).
15 class StieBrockett : public Problem{
16 public:
17         StieBrockett(double *inB, double *inD, integer inn, integer inp);
18         virtual ~StieBrockett();
19         virtual double f(Variable *x) const;
20         virtual void EucGrad(Variable *x, Vector *egf) const;
21         virtual void EucHessianEta(Variable *x, Vector *etax,
22                                    Vector *exix) const;
23
24         double *B;
25         double *D;
26         integer n;
27         integer p;
28 };
29 #endif // end of STIEBROCKETT_H
```

Listing 9: File "StieBrockett.cpp" for test Brockett in C++

```
1  // File: StieBrockett.cpp
2
3  #include "StieBrockett.h"
4
5  StieBrockett::StieBrockett(double *inB, double *inD, integer inn, integer inp)
6  {
7          B = inB;
8          D = inD;
9          n = inn;
10         p = inp;
11 };
12
13 StieBrockett::~StieBrockett(void)
14 {
15 };
16
17 double StieBrockett::f(Variable *x) const
18 {
19         const double *xxM = x->ObtainReadData();
20         Vector *BxD = x->ConstructEmpty();
21         SharedSpace *Temp = new SharedSpace(BxD);
22         double *temp = BxD->ObtainWriteEntireData();
23         double result = 0;
24
25         char *transn = const_cast<char *> ("n");
26         double one = 1, zero = 0;
27         integer inc = 1, N = n, P = p;
28         dgemm_(transn, transn, &N, &P, &N, &one, B, &N, const_cast<double *> (xxM), &N, &
               zero, temp, &N);
29
30         for (integer i = 0; i < p; i++)
31         {
32                 dscal_(&N, &D[i], temp + i * n, &inc);
33         }
34         integer length = N * P;
35         result = ddot_(&length, temp, &inc,
36             const_cast<double *> (xxM), &inc);
37         if (UseGrad)
```

```
38          {
39                  x->AddToTempData("BxD", Temp);
40          }
41          else
42          {
43                  delete Temp;
44          }
45          return result;
46  };
47
48  void StieBrockett::EucGrad(Variable *x, Vector *egf) const
49  {
50          const SharedSpace *Temp = x->ObtainReadTempData("BxD");
51          Vector *BxD = Temp->GetSharedElement();
52          Domain->ScaleTimesVector(x, 2.0, BxD, egf);
53  };
54
55  void StieBrockett::EucHessianEta(Variable *x, Vector *etax,
56                                   Vector *exix) const
57  {
58          const double *etaxTV = etax->ObtainReadData();
59          double *exixTV = exix->ObtainWriteEntireData();
60
61          char *transn = const_cast<char *> ("n");
62          integer N = n, P = p, inc = 1, Length = N * P;
63          double one = 1, zero = 0, negone = -1, two = 2;
64          dgemm_(transn, transn, &N, &P, &N, &one, B, &N,
65             const_cast<double *> (etaxTV), &N, &zero, exixTV, &N);
66          for (integer i = 0; i < p; i++)
67          {
68                  dscal_(&N, &D[i], exixTV + i * n, &inc);
69          }
70          Domain->ScaleTimesVector(x, 2.0, exix, exix);
71  };
```

Listing 10: File "TestSimpleExample.cpp" for test Brockett in C++

```
1   // File: TestSimpleExample.cpp
2
3   #ifndef TESTSIMPLEEXAMPLE_CPP
4   #define TESTSIMPLEEXAMPLE_CPP
5
6   #include "StieBrockett.h"
7   #include "StieVector.h"
8   #include "StieVariable.h"
9   #include "Stiefel.h"
10  #include "RTRNewton.h"
11  #include "def.h"
12
13  #ifdef TESTSIMPLEEXAMPLE
14
15  /*User-specified linesearch algorithm*/
16  double LinesearchInput(integer iter, Variable *x1, Vector *eta1, double initialstepsize,
        double initialslope, const Problem *prob, const Solvers *solver)
17  {
18          return 1;
19  }
20
21  /*User-specified stopping criterion*/
22  bool MyStop(Variable *x, Vector *gf, double f, double ngf, double ngf0, const Problem *prob,
         const Solvers *solver)
23  {
24          return (ngf / ngf0 < 1e-6);
25  };
26
```

```
27  int main(void)
28  {
29          // choose a random seed
30          unsigned tt = (unsigned)time(NULL);
31          init_genrand(tt);
32
33          // size of the Stiefel manifold
34          integer n = 12, p = 8;
35
36          // Generate the matrices in the Brockett problem.
37          double *B = new double[n * n + p];
38          double *D = B + n * n;
39          for (integer i = 0; i < n; i++)
40          {
41                  for (integer j = i; j < n; j++)
42                  {
43                          B[i + j * n] = genrand_gaussian();
44                          B[j + i * n] = B[i + j * n];
45                  }
46          }
47          for (integer i = 0; i < p; i++)
48                  D[i] = static_cast<double> (i + 1);
49
50          // Obtain an initial iterate
51          StieVariable StieX(n, p);
52          StieX.RandInManifold();
53
54          // Define the Stiefel manifold
55          Stiefel Domain(n, p);
56
57          // Define the Brockett problem
58          StieBrockett Prob(B, D, n, p);
59
60          // Set the domain of the problem to be the Stiefel manifold
61          Prob.SetDomain(&Domain);
62
63          // output the parameters of the manifold of domain
64          Domain.CheckParams();
65
66          //test RBFGS
67          RBFGS *RBFGSsolver = new RBFGS(&Prob, &StieX);
68          RBFGSsolver->LineSearch_LS = INPUTFUN;
69          RBFGSsolver->LinesearchInput = &LinesearchInput;
70          RBFGSsolver->IsPureLSInput = false;
71          RBFGSsolver->StopPtr = &MyStop;
72          RBFGSsolver->Debug = ITERRESULT;
73          RBFGSsolver->CheckParams();
74          RBFGSsolver->Run();
75
76          // Check gradient and Hessian
77          Prob.CheckGradHessian(&StieX);
78          const Variable *xopt = RBFGSsolver->GetXopt();
79          Prob.CheckGradHessian(xopt);
80
81          delete RBFGSsolver;
82          delete[] B;
83
84          return 0;
85  }
86  #endif
87  #endif
```

5.2 An Example for a Product of Manifolds

This section gives the C++ code for the problem (3.2) defined on a product of manifolds (see Section 3.3). The codes in Listing 11, 12 and 13 can be found in `/ROPTLIB/Problems/StieSumBrockett/*` and `/ROPTLIB/test/TestProductExample.cpp`.[6]

The codes defining a product of manifolds and a point on the manifold is given from line 85 to line 99 of Listing 13. The space for all components required by a point on a product of manifolds is stored in consecutive memory locations. For example, as shown in line 25 of Listing 12, a pointer to a segment of memory with length of $2np + mq$ doubles is obtained. The first np doubles are the first component of the iterate. The next np doubles are the second component and the last mq doubles are the last component of the iterate. Double pointers **xX1**, **xX2** and **xX3** are used to point the first addresses of the three components. Note that the order of components must be consistent with the order in initial iterate the user constructed in line 93 of Listing 13.

It is allowed to cast **x** to be a pointer of *ProductElement*, as shown for example in line 29 of Listing 12. Each component of **x** can be obtained by using the member function *GetElement(integer)*, e.g. line 30 of Listing 12. If users want to overwrite data that is pointed to by a pointer obtained by *GetElement(integer)*, then it is required to first use *NewMemoryOnWrite(void)* or *CopyOnWrite(void)* for all the *ProductElement* objects. *NewMemoryOnWrite(void)* creates new memory if necessary. *CopyOnWrite(void)* not only creates new memory if necessary, it also copies the data from old memory to the new memory. For example, in the routine *EucGrad* of Listing 12, **egf** is the output gradient and it is not important what data is in **egf**. It is important to make sure **egf** has sufficient memory to store the results. Therefore, before overwriting it, we use *NewMemoryOnWrite(void)* routine in line 107 to ensure it contains enough space. The same routine is used similarly in line 121.

Listing 11: File "StieSumBrockett.h" for test summation of Brockett in C++

```
1  // File: StieSumBrockett.h
2
3  #ifndef STIESUMBROCKETT_H
4  #define STIESUMBROCKETT_H
5
6  #include "Stiefel.h"
7  #include "StieVariable.h"
8  #include "StieVector.h"
9  #include <ProductElement.h>
10 #include <ProductManifold.h>
11 #include "Problem.h"
12 #include "SharedSpace.h"
13 #include "def.h"
14
15 // min_X X^T B X D, where B is a symmetric positive definite matrix, D is a diagonal matrix
16 // and X \in St(p, n).
17 class StieSumBrockett : public Problem{
18 public:
19         StieSumBrockett(double *inB1, double *inD1, double *inB2, double *inD2, double *inB3
            , double *inD3, integer inn, integer inp, integer inm, integer inq);
20         virtual ~StieSumBrockett();
21         virtual double f(Variable *x) const;
22
23         virtual void EucGrad(Variable *x, Vector *egf) const;
24         virtual void EucHessianEta(Variable *x, Vector *etax, Vector *exix) const;
```

[6]The code in the file may not be exactly the same as that in the Listings. The code in the file tests more parameters and runs more/different algorithms. Therefore, the differences are minor and should not cause confusion.

```
25
26          double *B1;
27          double *D1;
28          double *B2;
29          double *D2;
30          double *B3;
31          double *D3;
32          integer n;
33          integer p;
34          integer m;
35          integer q;
36  };
37  #endif // end of STIESUMBROCKETT_H
```

Listing 12: File "StieSumBrockett.cpp" for test summation of Brockett in C++

```
1   // File: StieSumBrockett.cpp
2
3   #include "StieSumBrockett.h"
4
5   StieSumBrockett::StieSumBrockett(double *inB1, double *inD1, double *inB2, double *inD2,
        double *inB3, double *inD3, integer inn, integer inp, integer inm, integer inq)
6   {
7           B1 = inB1;
8           D1 = inD1;
9           B2 = inB2;
10          D2 = inD2;
11          B3 = inB3;
12          D3 = inD3;
13          n = inn;
14          p = inp;
15          m = inm;
16          q = inq;
17  };
18
19  StieSumBrockett::~StieSumBrockett(void)
20  {
21  };
22
23  double StieSumBrockett::f(Variable *x) const
24  {
25          const double *xX1 = x->ObtainReadData();
26          const double *xX2 = xX1 + n * p;
27          const double *xX3 = xX2 + n * p;
28
29          ProductElement *prodx = dynamic_cast<ProductElement *> (x);
30          Vector *BxD1 = prodx->GetElement(0)->ConstructEmpty();
31          SharedSpace *Temp1 = new SharedSpace(BxD1);
32          double *temp1 = BxD1->ObtainWriteEntireData();
33          double result = 0;
34
35          char *transn = const_cast<char *> ("n");
36          double one = 1, zero = 0;
37          integer inc = 1, N = n, P = p;
38          dgemm_(transn, transn, &N, &P, &N, &one, B1, &N, const_cast<double *> (xX1), &N, &
                zero, temp1, &N);
39          for (integer i = 0; i < p; i++)
40          {
41                  dscal_(&N, &D1[i], temp1 + i * n, &inc);
42          }
43          integer length = N * P;
44          result += ddot_(&length, temp1, &inc, const_cast<double *> (xX1), &inc);
45          if (UseGrad)
46          {
47                  x->AddToTempData("BxD1", Temp1);
```

```
48          }
49          else
50          {
51                  delete Temp1;
52          }
53
54          Vector *BxD2 = prodx->GetElement(1)->ConstructEmpty();
55          SharedSpace *Temp2 = new SharedSpace(BxD2);
56          double *temp2 = BxD2->ObtainWriteEntireData();
57
58          dgemm_(transn, transn, &N, &P, &N, &one, B2, &N, const_cast<double *> (xX2), &N, &
                zero, temp2, &N);
59          for (integer i = 0; i < p; i++)
60          {
61                  dscal_(&N, &D2[i], temp2 + i * n, &inc);
62          }
63          result += ddot_(&length, temp2, &inc, const_cast<double *> (xX2), &inc);
64          if (UseGrad)
65          {
66                  x->AddToTempData("BxD2", Temp2);
67          }
68          else
69          {
70                  delete Temp2;
71          }
72
73          Vector *BxD3 = prodx->GetElement(2)->ConstructEmpty();
74          SharedSpace *Temp3 = new SharedSpace(BxD3);
75          double *temp3 = BxD3->ObtainWriteEntireData();
76          integer M = m, Q = q;
77          length = M * Q;
78          dgemm_(transn, transn, &M, &Q, &M, &one, B3, &M, const_cast<double *> (xX3), &M, &
                zero, temp3, &M);
79          for (integer i = 0; i < q; i++)
80          {
81                  dscal_(&M, &D3[i], temp3 + i * m, &inc);
82          }
83          result += ddot_(&length, temp3, &inc, const_cast<double *> (xX3), &inc);
84          if (UseGrad)
85          {
86                  x->AddToTempData("BxD3", Temp3);
87          }
88          else
89          {
90                  delete Temp3;
91          }
92
93          return result;
94  };
95
96  void StieSumBrockett::EucGrad(Variable *x, Vector *egf) const
97  {
98          const SharedSpace *Temp1 = x->ObtainReadTempData("BxD1");
99          const SharedSpace *Temp2 = x->ObtainReadTempData("BxD2");
100         const SharedSpace *Temp3 = x->ObtainReadTempData("BxD3");
101         Vector *BxD1 = Temp1->GetSharedElement();
102         Vector *BxD2 = Temp2->GetSharedElement();
103         Vector *BxD3 = Temp3->GetSharedElement();
104
105         ProductElement *prodegf = dynamic_cast<ProductElement *> (egf);
106         ProductElement *prodx = dynamic_cast<ProductElement *> (x);
107         prodegf->NewMemoryOnWrite();
108
109         ProductManifold *ProdDomain = dynamic_cast<ProductManifold *> (Domain);
110
```

```
111         ProdDomain->GetManifold(0)->ScaleTimesVector(prodx->GetElement(0), 2.0, BxD1,
                prodegf->GetElement(0));
112         ProdDomain->GetManifold(0)->ScaleTimesVector(prodx->GetElement(1), 2.0, BxD2,
                prodegf->GetElement(1));
113         ProdDomain->GetManifold(1)->ScaleTimesVector(prodx->GetElement(2), 2.0, BxD3,
                prodegf->GetElement(2));
114 };
115
116 void StieSumBrockett::EucHessianEta(Variable *x, Vector *etax, Vector *exix) const
117 {
118         ProductElement *prodx = dynamic_cast<ProductElement *> (x);
119         ProductElement *prodetax = dynamic_cast<ProductElement *> (etax);
120         ProductElement *prodexix = dynamic_cast<ProductElement *> (exix);
121         prodexix->NewMemoryOnWrite();
122         ProductManifold *ProdDomain = dynamic_cast<ProductManifold *> (Domain);
123
124         const double *etax1TV = prodetax->GetElement(0)->ObtainReadData();
125         double *exix1TV = prodexix->GetElement(0)->ObtainWriteEntireData();
126         char *transn = const_cast<char *> ("n");
127         integer N = n, P = p, inc = 1, Length = N * P;
128         double one = 1, zero = 0, negone = -1, two = 2;
129         dgemm_(transn, transn, &N, &P, &N, &one, B1, &N, const_cast<double *> (etax1TV), &N,
                &zero, exix1TV, &N);
130         for (integer i = 0; i < p; i++)
131         {
132                 dscal_(&N, &D1[i], exix1TV + i * n, &inc);
133         }
134         ProdDomain->GetManifold(0)->ScaleTimesVector(prodx->GetElement(0), 2.0, prodexix->
                GetElement(0), prodexix->GetElement(0));
135
136         const double *etax2TV = prodetax->GetElement(1)->ObtainReadData();
137         double *exix2TV = prodexix->GetElement(1)->ObtainWriteEntireData();
138         dgemm_(transn, transn, &N, &P, &N, &one, B2, &N, const_cast<double *> (etax2TV), &N,
                &zero, exix2TV, &N);
139         for (integer i = 0; i < p; i++)
140         {
141                 dscal_(&N, &D2[i], exix2TV + i * n, &inc);
142         }
143         ProdDomain->GetManifold(0)->ScaleTimesVector(prodx->GetElement(1), 2.0, prodexix->
                GetElement(1), prodexix->GetElement(1));
144
145         const double *etax3TV = prodetax->GetElement(2)->ObtainReadData();
146         double *exix3TV = prodexix->GetElement(2)->ObtainWriteEntireData();
147         integer M = m, Q = q;
148         Length = N * P;
149         dgemm_(transn, transn, &M, &Q, &M, &one, B3, &M, const_cast<double *> (etax3TV), &M,
                &zero, exix3TV, &M);
150         for (integer i = 0; i < q; i++)
151         {
152                 dscal_(&M, &D3[i], exix3TV + i * m, &inc);
153         }
154         ProdDomain->GetManifold(1)->ScaleTimesVector(prodx->GetElement(2), 2.0, prodexix->
                GetElement(2), prodexix->GetElement(2));
155 };
```

Listing 13: File "TestProductExample.cpp" for test summation of Brockett in C++

```
1  // File: TestProductExample.cpp
2
3  #ifndef TESTPRODUCTEXAMPLE_CPP
4  #define TESTPRODUCTEXAMPLE_CPP
5
6  #include "ForDebug.h"
7  #include <iostream>
8  #include "randgen.h"
```

```
 9  #include "Manifold.h"
10  #include "Problem.h"
11  #include "SolversLS.h"
12  #include <ctime>
13
14  #include "StieSumBrockett.h"
15  #include "StieVector.h"
16  #include "StieVariable.h"
17  #include "Stiefel.h"
18  #include <ProductElement.h>
19  #include <ProductManifold.h>
20
21  #include "RSD.h"
22  #include "RNewton.h"
23  #include "RCG.h"
24  #include "RBroydenFamily.h"
25  #include "RWRBFGS.h"
26  #include "RBFGS.h"
27  #include "LRBFGS.h"
28
29  #include "SolversTR.h"
30  #include "RTRSD.h"
31  #include "RTRNewton.h"
32  #include "RTRSR1.h"
33  #include "LRTRSR1.h"
34
35  #include "def.h"
36
37  #ifdef TESTPRODUCTEXAMPLE
38
39  int main(void)
40  {
41          // choose a random seed
42          unsigned tt = (unsigned)time(NULL);
43          init_genrand(tt);
44
45          // size of the Stiefel manifold
46          integer n = 12, p = 8, m = 6, q = 2;
47
48          // Generate the matrices in the Brockett problem.
49          double *B1 = new double[n * n * 2 + p * 2 + m * m + q];
50          double *B2 = B1 + n * n;
51          double *B3 = B2 + n * n;
52          double *D1 = B3 + m * m;
53          double *D2 = D1 + p;
54          double *D3 = D2 + p;
55
56          for (integer i = 0; i < n; i++)
57          {
58                  for (integer j = i; j < n; j++)
59                  {
60                          B1[i + j * n] = genrand_gaussian();
61                          B1[j + i * n] = B1[i + j * n];
62
63                          B2[i + j * n] = genrand_gaussian();
64                          B2[j + i * n] = B2[i + j * n];
65                  }
66          }
67          for (integer i = 0; i < m; i++)
68          {
69                  for (integer j = i; j < m; j++)
70                  {
71                          B3[i + j * m] = genrand_gaussian();
72                          B3[j + i * m] = B3[i + j * m];
73                  }
```

```
74          }
75          for (integer i = 0; i < p; i++)
76          {
77                  D1[i] = static_cast<double> (i + 1);
78                  D2[i] = D1[i];
79          }
80          for (integer i = 0; i < q; i++)
81          {
82                  D3[i] = static_cast<double> (i + 1);
83          }
84
85          // number of manifolds in product of manifold
86          integer numofmanis = 2; // two kinds of manifolds
87          integer numofmani1 = 2; // the number of first one is two
88          integer numofmani2 = 1; // the number of second one is one
89
90          // Obtain an initial iterate
91          StieVariable StieX1(n, p);
92          StieVariable StieX2(m, q);
93          ProductElement ProdX(numofmanis, &StieX1, numofmani1, &StieX2, numofmani2);
94          ProdX.RandInManifold();
95
96          // Define the Stiefel manifold
97          Stiefel mani1(n, p);
98          Stiefel mani2(m, q);
99          ProductManifold Domain(numofmanis, &mani1, numofmani1, &mani2, numofmani2);
100
101         // Define the Brockett problem
102         StieSumBrockett Prob(B1, D1, B2, D2, B3, D3, n, p, m, q);
103
104         // Set the domain of the problem to be the Stiefel manifold
105         Prob.SetDomain(&Domain);
106
107         // output the parameters of the manifold of domain
108         Domain.CheckParams();
109
110         // test RTRNewton
111         std::cout << "*********************************Check RTRNewton
                ***********************************" << std::endl;
112         RTRNewton RTRNewtonsolver(&Prob, &ProdX);
113         RTRNewtonsolver.DEBUG = FINALRESULT;
114         RTRNewtonsolver.CheckParams();
115         RTRNewtonsolver.Run();
116
117         // Check gradient and Hessian
118         Prob.CheckGradHessian(&ProdX);
119         const Variable *xopt = RTRNewtonsolver.GetXopt();
120         Prob.CheckGradHessian(xopt);
121
122         delete[] B1;
123
124         return 0;
125 }
126 #endif
127 #endif
```

A Relationships among Classes in the Package

A.1 Manifold-related Classes

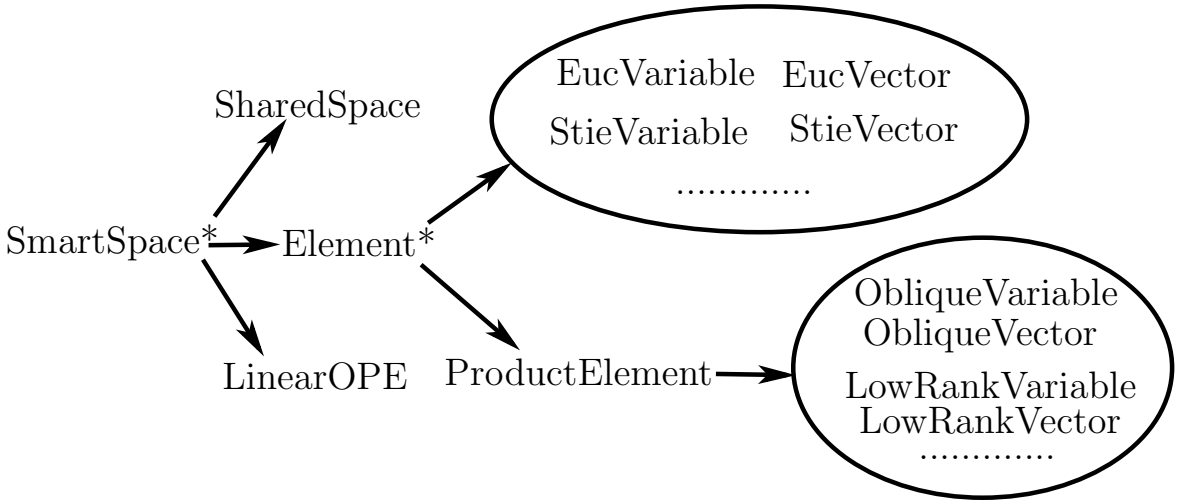

Figure 1: The class hierarchy of space-related classes in ROPTLIB. Note that Variable *and* Vector *are defined to be* Element.

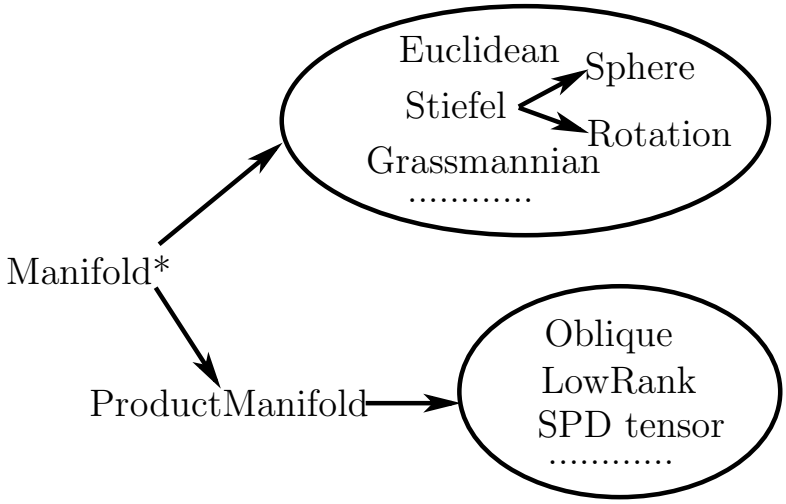

Figure 2: The class hierarchy of manifold-related classes in ROPTLIB. We refer to the documentation in the code for detailed explanations of the functions.

A.2 Problem-related Classes

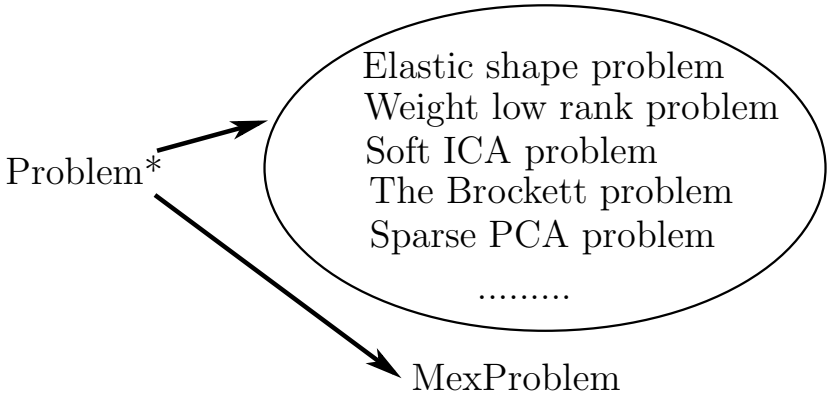

Figure 3: The class hierarchy of problem-related classes in ROPTLIB. We refer to the documentation in the code for detailed explanations of the functions.

A.3 Solver-related Classes

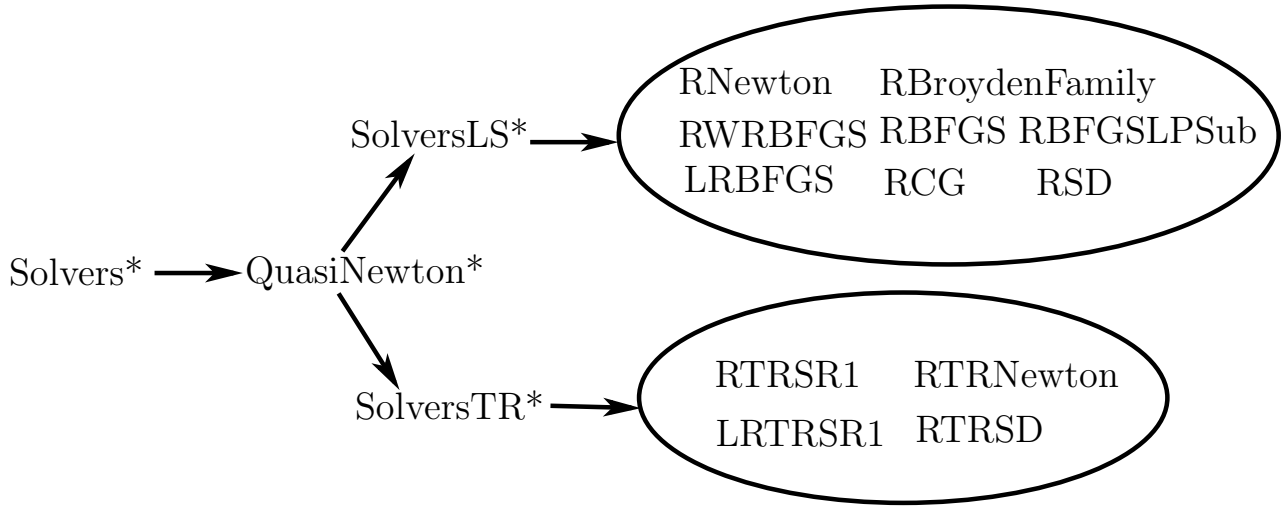

Figure 4: The class hierarchy of solver-related classes in ROPTLIB. We refer to the documentation in the code for detailed explanations of the functions.

B Input Parameters and Output Notation of Solvers

B.1 RTRNewton

Table 2: Input Parameters of RTRNewton

Name of field	Interpretation	Default value	Applicable values
		C++/(Matlab,Julia)	C++/(Matlab,Julia) : interpretation

IsCheckParams	output parameters of Solvers	Matlab and Julia only : 0	0 or 1
IsCheckGradHess	Check the correctness of gradient and Hessian	Matlab and Julia only : 0	0 or 1
Stop_Criterion	Stopping criterion	GRAD_F_0 / 2	FUN_REL / 0 : $(f(x_{i-1}) - f(x_i))/f(x_i)$ GRAD_F / 1 : $\|\operatorname{grad} f(x_i)\|$ GRAD_F_0 / 2 : $\|\operatorname{grad} f(x_i)\|/\|\operatorname{grad} f(x_0)\|$
Tolerance	Algorithm stops if "Stop_Criterion" $<$ tolerance	10^{-6}	greater than 0
Min_Iteration	minimum number of iterations	0	greater than or equal to 0 and smaller than or equal to Max_Iteration
Max_Iteration	maximum number of iterations	500	greater than or equal to Min_Iteration
OutputGap	Output every "OutputGap" iterations	1	greater than or equal to 1
DEBUG	output information	ITERRESULT / 2	NOOUTPUT / 0 : no output FINALRESULT / 1 : Only final result ITERRESULT / 2 : Output every "OutputGap" iterations DETAILED / 3: Output Detailed information
TimeBound	maximum computational time	$60 * 60 * 24 * 365$	greater than 0
Acceptence_Rho	Accept candidate if Rho $>$ Acceptence_Rho	0.1	between 0 and 0.25, i.e., $\in (0, 0.25)$
Shrinked_tau	coefficient in reducing radius	0.25	between 0 and 1, i.e., $\in (0, 1)$
Magnified_tau	coefficient in increasing radius	2	greater than 1
minimum_Delta	minimum allowed radius	machine eps	greater than 0 and smaller than or equal to maximum_Delta
maximum_Delta	maximum allowed radius	10000	greater than or equal to minimum_Delta
useRand	whether use Rand in truncate conjugate gradient	false / 0	false / 0 or true / 1
Min_Inner_Iter	minimum number of iterations in truncate conjugate gradient	0	greater than or equal to ZERO and smaller than or equal to Max_Inner_Iter
Max_Inner_Iter	maximum number of iterations in truncate conjugate gradient	1000	greater than or equal to Min_Inner_Iter
theta	in [AMS08, (7.10)]	1	greater than or equal to 0
kappa	in [AMS08, (7.10)]	0.1	between 0 and 1, i.e., $\in (0, 1)$
initial_Delta	initial radius	1	greater than 0

Table 3: Output notation of RTRNewton. Note that the first time an action of a vector transport $\mathcal{T}_\eta$ is computed will usually have higher complexity than subsequent times. Specifically, if $\mathcal{T}_\eta\xi_1$ has been computed, then evaluating $\mathcal{T}_\eta\xi_2$ usually can use some results from computations of $\mathcal{T}_\eta\xi_1$. nV denotes the number of evaluations of vector transport first time. nVp denotes the number of other times.

Notation	Interpretation
i	the number of iterations
f	function value
df/f	$(f(x_{i-1}) - f(x_i))/f(x_i)$

gf	$\|\operatorname{grad} f(x_i)\|$
time	computational time (second)
nf	the number of function evaluations
ng	the number of gradient evaluations
nR	the number of retraction evaluations
nV/nVp	the number of actions of vector transport
nH	the number of actions of Hessian
rho	[AMS08, (7.7)]
radius	the radius of trust region
tCGstatus	status of truncate conjugate gradient
innerIter	the number of iterations in truncate conjugate gradient

B.2 RTRSR1

Table 4: Input Parameters of RTRSR1

Name of field	Interpretation	Default value	Applicable values
		C++/(Matlab,Julia)	C++/(Matlab,Julia) : interpretation
IsCheckParams	output parameters of Solvers	Matlab and Julia only : 0	0 or 1
IsCheckGradHess	Check the correctness of gradient and Hessian	Matlab and Julia only : 0	0 or 1
Stop_Criterion	Stopping criterion	GRAD_F_0 / 2	FUN_REL / 0 : $(f(x_{i-1}) - f(x_i))/f(x_i)$ GRAD_F / 1 : $\|\operatorname{grad} f(x_i)\|$ GRAD_F_0 / 2 : $\|\operatorname{grad} f(x_i)\|/\|\operatorname{grad} f(x_0)\|$
Tolerance	Algorithm stops if "Stop_Criterion" < tolerance	10^{-6}	greater than 0
Min_Iteration	minimum number of iterations	0	greater than or equal to 0 and smaller than or equal to Max_Iteration
Max_Iteration	maximum number of iterations	500	greater than or equal to Min_Iteration
OutputGap	Output every "OutputGap" iterations	1	greater than or equal to 1
DEBUG	output information	ITERRESULT / 2	NOOUTPUT / 0 : no output FINALRESULT / 1 : Only final result ITERRESULT / 2 : Output every "OutputGap" iterations DETAILED / 3: Output Detailed information
TimeBound	maximum computational time	$60 * 60 * 24 * 365$	greater than 0
Acceptence_Rho	Accept candidate if Rho > Acceptence_Rho	0.1	between 0 and 0.25, i.e., $\in (0, 0.25)$
Shrinked_tau	coefficient in reducing radius	0.25	between 0 and 1, i.e., $\in (0, 1)$
Magnified_tau	coefficient in increasing radius	2	greater than 1
minimum_Delta	minimum allowed radius	machine eps	greater than 0 and smaller than or equal to maximum_Delta
maximum_Delta	maximum allowed radius	10000	greater than or equal to minimum_Delta
useRand	whether use Rand in truncate conjugate gradient	false / 0	false / 0 or true / 1

Min_Inner_Iter	minimum number of iterations in truncate conjugate gradient	0	greater than or equal to ZERO and smaller than or equal to Max_Inner_Iter
Max_Inner_Iter	maximum number of iterations in truncate conjugate gradient	1000	greater than or equal to Min_Inner_Iter
theta	in [AMS08, (7.10)]	0.1	greater than or equal to 0
kappa	in [AMS08, (7.10)]	0.1	between 0 and 1, i.e., $\in (0,1)$
initial_Delta	initial radius	1	greater than 0
isconvex	whether the cost function is convex	false / 0	false / 0 or true / 1

Table 5: Output notation of RTRSR1. Note that the first time an action of a vector transport $\mathcal{T}_\eta$ is computed will usually have higher complexity than subsequent times. Specifically, if $\mathcal{T}_\eta\xi_1$ has been computed, then evaluating $\mathcal{T}_\eta\xi_2$ usually can use some results from computations of $\mathcal{T}_\eta\xi_1$. nV denotes the number of evaluations of vector transport first time. nVp denotes the number of other times.

Notation	Interpretation
i	the number of iterations
f	function value
df/f	$(f(x_{i-1}) - f(x_i))/f(x_i)$
gf	$\|\operatorname{grad} f(x_i)\|$
time	computational time (second)
nf	the number of function evaluations
ng	the number of gradient evaluations
nR	the number of retraction evaluations
nV/nVp	the number of actions of vector transport
nH	the number of actions of Hessian
rho	[AMS08, (7.7)]
radius	the radius of trust region
tCGstatus	status of truncate conjugate gradient
innerIter	the number of iterations in truncate conjugate gradient
inpss	$\langle s_i, s_i \rangle$
IsUpdateHessian	Whether update Hessian approximation or not

B.3 LRTRSR1

Table 6: Input Parameters of LRTRSR1

Name of field	Interpretation	Default value	Applicable values
		C++/(Matlab,Julia)	C++/(Matlab,Julia) : interpretation
IsCheckParams	output parameters of Solvers	Matlab and Julia only : 0	0 or 1
IsCheckGradHess	Check the correctness of gradient and Hessian	Matlab and Julia only : 0	0 or 1
Stop_Criterion	Stopping criterion	GRAD_F_0 / 2	FUN_REL / 0 : $(f(x_{i-1}) - f(x_i))/f(x_i)$ GRAD_F / 1 : $\|\operatorname{grad} f(x_i)\|$ GRAD_F_0 / 2 : $\|\operatorname{grad} f(x_i)\|/\|\operatorname{grad} f(x_0)\|$
Tolerance	Algorithm stops if "Stop_Criterion" < tolerance	10^{-6}	greater than 0
Min_Iteration	minimum number of iterations	0	greater than or equal to 0 and smaller than or equal to Max_Iteration
Max_Iteration	maximum number of iterations	500	greater than or equal to Min_Iteration

OutputGap	Output every "OutputGap" iterations	1	greater than or equal to 1
DEBUG	output information	ITERRESULT / 2	NOOUTPUT / 0 : no output FINALRESULT / 1 : Only final result ITERRESULT / 2 : Output every "OutputGap" iterations DETAILED / 3: Output Detailed information
TimeBound	maximum computational time	$60 * 60 * 24 * 365$	greater than 0
Acceptence_Rho	Accept candidate if Rho > Acceptence_Rho	0.1	between 0 and 0.25, i.e., $\in (0, 0.25)$
Shrinked_tau	coefficient in reducing radius	0.25	between 0 and 1, i.e., $\in (0, 1)$
Magnified_tau	coefficient in increasing radius	2	greater than 1
minimum_Delta	minimum allowed radius	machine eps	greater than 0 and smaller than or equal to maximum_Delta
maximum_Delta	maximum allowed radius	10000	greater than or equal to minimum_Delta
useRand	whether use Rand in truncate conjugate gradient	false / 0	false / 0 or true / 1
Min_Inner_Iter	minimum number of iterations in truncate conjugate gradient	0	greater than or equal to ZERO and smaller than or equal to Max_Inner_Iter
Max_Inner_Iter	maximum number of iterations in truncate conjugate gradient	1000	greater than or equal to Min_Inner_Iter
theta	in [AMS08, (7.10)]	0.1	greater than or equal to 0
kappa	in [AMS08, (7.10)]	0.1	between 0 and 1, i.e., $\in (0, 1)$
initial_Delta	initial radius	1	greater than 0
isconvex	whether the cost function is convex	false / 0	false / 0 or true / 1
LengthSY	the same as ℓ in [HGA15, Algorithm 2]	4	greater than or equal to 0

Table 7: Output notation of LRTRSR1. Note that the first time an action of a vector transport $\mathcal{T}_\eta$ is computed will usually have higher complexity than subsequent times. Specifically, if $\mathcal{T}_\eta\xi_1$ has been computed, then evaluating $\mathcal{T}_\eta\xi_2$ usually can use some results from computations of $\mathcal{T}_\eta\xi_1$. nV denotes the number of evaluations of vector transport first time. nVp denotes the number of other times.

Notation	Interpretation
i	the number of iterations
f	function value
df/f	$(f(x_{i-1}) - f(x_i))/f(x_i)$
gf	$\|\operatorname{grad} f(x_i)\|$
time	computational time (second)
nf	the number of function evaluations
ng	the number of gradient evaluations
nR	the number of retraction evaluations
nV/nVp	the number of actions of vector transport
nH	the number of actions of Hessian
rho	[AMS08, (7.7)]
radius	the radius of trust region
tCGstatus	status of truncate conjugate gradient

innerIter	the number of iterations in truncate conjugate gradient
gamma	$\langle y_i, y_i \rangle / \langle s_i, y_i \rangle$
inpss	$\langle s_i, s_i \rangle$
inpsy	$\langle s_i, y_i \rangle$
inpyy	$\langle y_i, y_i \rangle$
IsUpdateHessian	Whether update Hessian approximation or not

B.4 RTRSD

Table 8: Input Parameters of RTRSD

Name of field	Interpretation	Default value	Applicable values
		C++/(Matlab,Julia)	C++/(Matlab,Julia) : interpretation
IsCheckParams	output parameters of Solvers	Matlab and Julia only : 0	0 or 1
IsCheckGradHess	Check the correctness of gradient and Hessian	Matlab and Julia only : 0	0 or 1
Stop_Criterion	Stopping criterion	GRAD_F_0 / 2	FUN_REL / 0 : $(f(x_{i-1}) - f(x_i))/f(x_i)$ GRAD_F / 1 : $\|\operatorname{grad} f(x_i)\|$ GRAD_F_0 / 2 : $\|\operatorname{grad} f(x_i)\|/\|\operatorname{grad} f(x_0)\|$
Tolerance	Algorithm stops if "Stop_Criterion" < tolerance	10^{-6}	greater than 0
Min_Iteration	minimum number of iterations	0	greater than or equal to 0 and smaller than or equal to Max_Iteration
Max_Iteration	maximum number of iterations	500	greater than or equal to Min_Iteration
OutputGap	Output every "OutputGap" iterations	1	greater than or equal to 1
DEBUG	output information	ITERRESULT / 2	NOOUTPUT / 0 : no output FINALRESULT / 1 : Only final result ITERRESULT / 2 : Output every "OutputGap" iterations DETAILED / 3: Output Detailed information
TimeBound	maximum computational time	$60 * 60 * 24 * 365$	greater than 0
Acceptence_Rho	Accept candidate if Rho > Acceptence_Rho	0.1	between 0 and 0.25, i.e., $\in (0, 0.25)$
Shrinked_tau	coefficient in reducing radius	0.25	between 0 and 1, i.e., $\in (0, 1)$
Magnified_tau	coefficient in increasing radius	2	greater than 1
minimum_Delta	minimum allowed radius	machine eps	greater than 0 and smaller than or equal to maximum_Delta
maximum_Delta	maximum allowed radius	10000	greater than or equal to minimum_Delta
useRand	whether use Rand in truncate conjugate gradient	false / 0	false / 0 or true / 1
Min_Inner_Iter	minimum number of iterations in truncate conjugate gradient	0	greater than or equal to ZERO and smaller than or equal to Max_Inner_Iter
Max_Inner_Iter	maximum number of iterations in truncate conjugate gradient	1000	greater than or equal to Min_Inner_Iter

theta	in [AMS08, (7.10)]	0.1	greater than or equal to 0
kappa	in [AMS08, (7.10)]	0.9	between 0 and 1, i.e., $\in (0,1)$
initial_Delta	initial radius	1	greater than 0

Table 9: Output notation of RTRSD. Note that the first time an action of a vector transport $\mathcal{T}_\eta$ is computed will usually have higher complexity than subsequent times. Specifically, if $\mathcal{T}_\eta\xi_1$ has been computed, then evaluating $\mathcal{T}_\eta\xi_2$ usually can use some results from computations of $\mathcal{T}_\eta\xi_1$. nV denotes the number of evaluations of vector transport first time. nVp denotes the number of other times.

Notation	Interpretation
i	the number of iterations
f	function value
df/f	$(f(x_{i-1}) - f(x_i))/f(x_i)$
gf	$\|\operatorname{grad} f(x_i)\|$
time	computational time (second)
nf	the number of function evaluations
ng	the number of gradient evaluations
nR	the number of retraction evaluations
nV/nVp	the number of actions of vector transport
nH	the number of actions of Hessian
rho	[AMS08, (7.7)]
radius	the radius of trust region
tCGstatus	status of truncate conjugate gradient
innerIter	the number of iterations in truncate conjugate gradient

B.5 RNewton

Table 10: Input Parameters of RNewton

Name of field	Interpretation	Default value	Applicable values
		C++/(Matlab,Julia)	C++/(Matlab,Julia) : interpretation
IsCheckParams	output parameters of Solvers	Matlab and Julia only : 0	0 or 1
IsCheckGradHess	Check the correctness of gradient and Hessian	Matlab and Julia only : 0	0 or 1
Stop_Criterion	Stopping criterion	GRAD_F_0 / 2	FUN_REL / 0 : $(f(x_{i-1}) - f(x_i))/f(x_i)$ GRAD_F / 1 : $\|\operatorname{grad} f(x_i)\|$ GRAD_F_0 / 2 : $\|\operatorname{grad} f(x_i)\|/\|\operatorname{grad} f(x_0)\|$
Tolerance	Algorithm stops if "Stop_Criterion" < tolerance	10^{-6}	greater than 0
Min_Iteration	minimum number of iterations	0	greater than or equal to 0 and smaller than or equal to Max_Iteration
Max_Iteration	maximum number of iterations	500	greater than or equal to Min_Iteration
OutputGap	Output every "OutputGap" iterations	1	greater than or equal to 1
DEBUG	output information	ITERRESULT / 2	NOOUTPUT / 0 : no output FINALRESULT / 1 : Only final result ITERRESULT / 2 : Output every "OutputGap" iterations DETAILED / 3: Output Detailed information
TimeBound	maximum computational time	$60 * 60 * 24 * 365$	greater than 0
LineSearch_LS	Algorithm in linesearch	ARMIJO / 0	ARMIJO / 0 : Back tracking WOLFE / 1 : [DS83, Algorithm A6.3.1mod]

			STRONGWOLFE / 2 : [NW06, Algorithm 3.5] EXACT / 3 : scaled BFGS WOLFELP / 4 : [AHHY16] INPUTFUN / 5 : Given by users
IsPureLSInput	Whether backtracking is used for step size given by users' algorithm	false / 0	false / 0 or true / 1
LS_alpha	coefficient in the Wolfe first condition	0.0001	between 0 and 0.5, i.e. $\in (0, 0.5)$
LS_beta	coefficient in the Wolfe second condition	0.999	between 0 and 1, i.e., $\in (0, 1)$
Minstepsize	minimum allowed step size	machine eps	greater than 0 and smaller than or equal to Maxstepsize
Maxstepsize	maximum allowed step size	1000	greater than or equal to Minstepsize
Accuracy	fixed the stepsize if $\|gf_k\|/\|gf_0\| <$ accuracy	0	between 0 and 1, i.e., $\in [0, 1]$
Finalstepsize	Use this step size if $\|gf_k\|/\|gf_0\| <$ accuracy	1	all real number (negative number means the stepsize by method in "Initstepsize" is used)
LS_ratio1	coefficient in the Armijo condition	0.1	between 0 and 1, i.e., $\in (0, 1)$
LS_ratio2	coefficient in the Armijo condition	0.9	between 0 and 1, i.e., $\in (0, 1)$
Initstepsize	initial step size in first iteration	1	greater than 0
Num_pre_funs	the number of computed functions values stored for nonmonotonic linesearch	0	greater than or equal to 0
InitSteptype	Initial step size	QUADINTMOD / 3	ONESTEP / 0 : use one BBSTEP / 1 : g(s, s) / g(s, y) QUADINT / 2 : [NW06, (3.60)] QUADINTMOD / 3 : [NW06, page 60]
useRand	whether use Rand in truncate conjugate gradient	false / 0	false / 0 or true / 1
Min_Inner_Iter	minimum number of iterations in truncate conjugate gradient	0	greater than or equal to ZERO and smaller than or equal to Max_Inner_Iter
Max_Inner_Iter	maximum number of iterations in truncate conjugate gradient	1000	greater than or equal to Min_Inner_Iter
theta	in [AMS08, (7.10)]	1	greater than or equal to 0
kappa	in [AMS08, (7.10)]	0.1	between 0 and 1, i.e., $\in (0, 1)$

Table 11: Output notation of RNewton. Note that the first time an action of a vector transport $\mathcal{T}_\eta$ is computed will usually have higher complexity than subsequent times. Specifically, if $\mathcal{T}_\eta\xi_1$ has been computed, then evaluating $\mathcal{T}_\eta\xi_2$ usually can use some results from computations of $\mathcal{T}_\eta\xi_1$. nV denotes the number of evaluations of vector transport first time. nVp denotes the number of other times.

Notation	Interpretation
i	the number of iterations
f	function value
df/f	$(f(x_{i-1}) - f(x_i))/f(x_i)$

gf	$\|\operatorname{grad} f(x_i)\|$
time	computational time (second)
nf	the number of function evaluations
ng	the number of gradient evaluations
nR	the number of retraction evaluations
nV/nVp	the number of actions of vector transport
LSstatus	status of line search result
initslope	initial slope in line search
newslope	the slope of final point in line search
initstepsize	initial step size in line search
stepsize	the final stepsize
nH	the number of actions of Hessian
tCGstatus	status of truncate conjugate gradient
innerIter	the number of iterations in truncate conjugate gradient

B.6 RBroydenFamily

Table 12: Input Parameters of RBroydenFamily

Name of field	Interpretation	Default value	Applicable values
		C++/(Matlab,Julia)	C++/(Matlab,Julia) : interpretation
IsCheckParams	output parameters of Solvers	Matlab and Julia only : 0	0 or 1
IsCheckGradHess	Check the correctness of gradient and Hessian	Matlab and Julia only : 0	0 or 1
Stop_Criterion	Stopping criterion	GRAD_F_0 / 2	FUN_REL / 0 : $(f(x_{i-1}) - f(x_i))/f(x_i)$ GRAD_F / 1 : $\|\operatorname{grad} f(x_i)\|$ GRAD_F_0 / 2 : $\|\operatorname{grad} f(x_i)\|/\|\operatorname{grad} f(x_0)\|$
Tolerance	Algorithm stops if "Stop_Criterion" $<$ tolerance	10^{-6}	greater than 0
Min_Iteration	minimum number of iterations	0	greater than or equal to 0 and smaller than or equal to Max_Iteration
Max_Iteration	maximum number of iterations	500	greater than or equal to Min_Iteration
OutputGap	Output every "OutputGap" iterations	1	greater than or equal to 1
DEBUG	output information	ITERRESULT / 2	NOOUTPUT / 0 : no output FINALRESULT / 1 : Only final result ITERRESULT / 2 : Output every "OutputGap" iterations DETAILED / 3: Output Detailed information
TimeBound	maximum computational time	$60 * 60 * 24 * 365$	greater than 0
LineSearch_LS	Algorithm in linesearch	ARMIJO / 0	ARMIJO / 0 : Back tracking WOLFE / 1 : [DS83, Algorithm A6.3.1mod] STRONGWOLFE / 2 : [NW06, Algorithm 3.5] EXACT / 3 : scaled BFGS WOLFELP / 4 : [AHHY16] INPUTFUN / 5 : Given by users
IsPureLSInput	Whether backtracking is used for step size given by users' algorithm	false / 0	false / 0 or true / 1
LS_alpha	coefficient in the Wolfe first condition	0.0001	between 0 and 0.5, i.e. $\in (0, 0.5)$

LS_beta	coefficient in the Wolfe second condition	0.999	between 0 and 1, i.e., $\in (0, 1)$
Minstepsize	minimum allowed step size	machine eps	greater than 0 and smaller than or equal to Maxstepsize
Maxstepsize	maximum allowed step size	1000	greater than or equal to Minstepsize
Accuracy	fixed the stepsize if $\|gf_k\|/\|gf_0\| <$ accuracy	0	between 0 and 1, i.e., $\in [0, 1]$
Finalstepsize	Use this step size if $\|gf_k\|/\|gf_0\| <$ accuracy	1	all real number (negative number means the stepsize by method in "Initstepsize" is used)
LS_ratio1	coefficient in the Armijo condition	0.1	between 0 and 1, i.e., $\in (0, 1)$
LS_ratio2	coefficient in the Armijo condition	0.9	between 0 and 1, i.e., $\in (0, 1)$
Initstepsize	initial step size in first iteration	1	greater than 0
Num_pre_funs	the number of computed functions values stored for nonmonotonic linesearch	0	greater than or equal to 0
InitSteptype	Initial step size	QUADINTMOD / 3	ONESTEP / 0 : use one BBSTEP / 1 : g(s, s) / g(s, y) QUADINT / 2 : [NW06, (3.60)] QUADINTMOD / 3 : [NW06, page 60]
isconvex	whether the cost function is convex	false / 0	false / 0 or true / 1
nu	the same as ϵ in [LF01, (3.2)]	10^{-4}	greater than or equal to 0 and smaller than 1
mu	the same as α in [LF01, (3.2)]	1	greater than or equal to 0

Table 13: Output notation of RBroydenFamily. Note that the first time an action of a vector transport $\mathcal{T}_\eta$ is computed will usually have higher complexity than subsequent times. Specifically, if $\mathcal{T}_\eta\xi_1$ has been computed, then evaluating $\mathcal{T}_\eta\xi_2$ usually can use some results from computations of $\mathcal{T}_\eta\xi_1$. nV denotes the number of evaluations of vector transport first time. nVp denotes the number of other times.

Notation	Interpretation
i	the number of iterations
f	function value
df/f	$(f(x_{i-1}) - f(x_i))/f(x_i)$
gf	$\|\operatorname{grad} f(x_i)\|$
time	computational time (second)
nf	the number of function evaluations
ng	the number of gradient evaluations
nR	the number of retraction evaluations
nV/nVp	the number of actions of vector transport
LSstatus	status of line search result
initslope	initial slope in line search
newslope	the slope of final point in line search
initstepsize	initial step size in line search
stepsize	the final stepsize
betay	$\alpha_i\eta_i/\mathcal{T}_{R_{\alpha_i\eta_i}}(\alpha_i\eta_i)$ see [HGA15, Step 6 of Algorithm 1]
Phic	the coefficient ϕ_i in the update [HGA15, (2.3)]
inpss	$\langle s_i, s_i\rangle$
inpsy	$\langle s_i, y_i\rangle$
IsUpdateHessian	Whether update inverse Hessian approximation or not

B.7 RWRBFGS

Table 14: Input Parameters of RWRBFGS

Name of field	Interpretation	Default value	Applicable values
		C++/(Matlab,Julia)	C++/(Matlab,Julia) : interpretation
IsCheckParams	output parameters of Solvers	Matlab and Julia only : 0	0 or 1
IsCheckGradHess	Check the correctness of gradient and Hessian	Matlab and Julia only : 0	0 or 1
Stop_Criterion	Stopping criterion	GRAD_F_0 / 2	FUN_REL / 0 : $(f(x_{i-1}) - f(x_i))/f(x_i)$ GRAD_F / 1 : $\|\operatorname{grad} f(x_i)\|$ GRAD_F_0 / 2 : $\|\operatorname{grad} f(x_i)\|/\|\operatorname{grad} f(x_0)\|$
Tolerance	Algorithm stops if "Stop_Criterion" < tolerance	10^{-6}	greater than 0
Min_Iteration	minimum number of iterations	0	greater than or equal to 0 and smaller than or equal to Max_Iteration
Max_Iteration	maximum number of iterations	500	greater than or equal to Min_Iteration
OutputGap	Output every "OutputGap" iterations	1	greater than or equal to 1
DEBUG	output information	ITERRESULT / 2	NOOUTPUT / 0 : no output FINALRESULT / 1 : Only final result ITERRESULT / 2 : Output every "OutputGap" iterations DETAILED / 3: Output Detailed information
TimeBound	maximum computational time	$60 * 60 * 24 * 365$	greater than 0
LineSearch_LS	Algorithm in linesearch	ARMIJO / 0	ARMIJO / 0 : Back tracking WOLFE / 1 : [DS83, Algorithm A6.3.1mod] STRONGWOLFE / 2 : [NW06, Algorithm 3.5] EXACT / 3 : scaled BFGS WOLFELP / 4 : [AHHY16] INPUTFUN / 5 : Given by users
IsPureLSInput	Whether backtracking is used for step size given by users' algorithm	false / 0	false / 0 or true / 1
LS_alpha	coefficient in the Wolfe first condition	0.0001	between 0 and 0.5, i.e. $\in (0, 0.5)$
LS_beta	coefficient in the Wolfe second condition	0.999	between 0 and 1, i.e., $\in (0, 1)$
Minstepsize	minimum allowed step size	machine eps	greater than 0 and smaller than or equal to Maxstepsize
Maxstepsize	maximum allowed step size	1000	greater than or equal to Minstepsize
Accuracy	fixed the stepsize if $\|gf_k\|/\|gf_0\| <$ accuracy	0	between 0 and 1, i.e., $\in [0, 1]$
Finalstepsize	Use this step size if $\|gf_k\|/\|gf_0\| <$ accuracy	1	all real number (negative number means the stepsize by method in "Initstepsize" is used)
LS_ratio1	coefficient in the Armijo condition	0.1	between 0 and 1, i.e., $\in (0, 1)$
LS_ratio2	coefficient in the Armijo condition	0.9	between 0 and 1, i.e., $\in (0, 1)$

Initstepsize	initial step size in first iteration	1	greater than 0
Num_pre_funs	the number of computed functions values stored for nonmonotonic linesearch	0	greater than or equal to 0
InitSteptype	Initial step size	QUADINTMOD / 3	ONESTEP / 0 : use one BBSTEP / 1 : g(s, s) / g(s, y) QUADINT / 2 : [NW06, (3.60)] QUADINTMOD / 3 : [NW06, page 60]
isconvex	whether the cost function is convex	false / 0	false / 0 or true / 1
nu	the same as ϵ in [LF01, (3.2)]	10^{-4}	greater than or equal to 0 and smaller than 1
mu	the same as α in [LF01, (3.2)]	1	greater than or equal to 0

Table 15: Output notation of RWRBFGS. Note that the first time an action of a vector transport $\mathcal{T}_\eta$ is computed will usually have higher complexity than subsequent times. Specifically, if $\mathcal{T}_\eta\xi_1$ has been computed, then evaluating $\mathcal{T}_\eta\xi_2$ usually can use some results from computations of $\mathcal{T}_\eta\xi_1$. nV denotes the number of evaluations of vector transport first time. nVp denotes the number of other times.

Notation	Interpretation
i	the number of iterations
f	function value
df/f	$(f(x_{i-1}) - f(x_i))/f(x_i)$
gf	$\|\operatorname{grad} f(x_i)\|$
time	computational time (second)
nf	the number of function evaluations
ng	the number of gradient evaluations
nR	the number of retraction evaluations
nV/nVp	the number of actions of vector transport
LSstatus	status of line search result
initslope	initial slope in line search
newslope	the slope of final point in line search
initstepsize	initial step size in line search
stepsize	the final stepsize
inpss	$\langle s_i, s_i\rangle$
inpsy	$\langle s_i, y_i\rangle$
IsUpdateHessian	Whether update inverse Hessian approximation or not

B.8 RBFGS

Table 16: Input Parameters of RBFGS

Name of field	Interpretation	Default value	Applicable values
		C++/(Matlab,Julia)	C++/(Matlab,Julia) : interpretation
IsCheckParams	output parameters of Solvers	Matlab and Julia only : 0	0 or 1
IsCheckGradHess	Check the correctness of gradient and Hessian	Matlab and Julia only : 0	0 or 1
Stop_Criterion	Stopping criterion	GRAD_F_0 / 2	FUN_REL / 0 : $(f(x_{i-1}) - f(x_i))/f(x_i)$ GRAD_F / 1 : $\|\operatorname{grad} f(x_i)\|$ GRAD_F_0 / 2 : $\|\operatorname{grad} f(x_i)\|/\|\operatorname{grad} f(x_0)\|$ PSSUBGRAD / 3 : See [LO13, Section 6.3]

Tolerance	Algorithm stops if "Stop_Criterion" < tolerance	10^{-6}	greater than 0
Min_Iteration	minimum number of iterations	0	greater than or equal to 0 and smaller than or equal to Max_Iteration
Max_Iteration	maximum number of iterations	500	greater than or equal to Min_Iteration
OutputGap	Output every "OutputGap" iterations	1	greater than or equal to 1
DEBUG	output information	ITERRESULT / 2	NOOUTPUT / 0 : no output FINALRESULT / 1 : Only final result ITERRESULT / 2 : Output every "OutputGap" iterations DETAILED / 3: Output Detailed information
TimeBound	maximum computational time	$60 * 60 * 24 * 365$	greater than 0
LineSearch_LS	Algorithm in linesearch	ARMIJO / 0	ARMIJO / 0 : Back tracking WOLFE / 1 : [DS83, Algorithm A6.3.1mod] STRONGWOLFE / 2 : [NW06, Algorithm 3.5] EXACT / 3 : scaled BFGS WOLFELP / 4 : [AHHY16] INPUTFUN / 5 : Given by users
IsPureLSInput	Whether backtracking is used for step size given by users' algorithm	false / 0	false / 0 or true / 1
LS_alpha	coefficient in the Wolfe first condition	0.0001	between 0 and 0.5, i.e. $\in (0, 0.5)$
LS_beta	coefficient in the Wolfe second condition	0.999	between 0 and 1, i.e., $\in (0, 1)$
Minstepsize	minimum allowed step size	machine eps	greater than 0 and smaller than or equal to Maxstepsize
Maxstepsize	maximum allowed step size	1000	greater than or equal to Minstepsize
Accuracy	fixed the stepsize if $\|gf_k\|/\|gf_0\| <$ accuracy	0	between 0 and 1, i.e., $\in [0, 1]$
Finalstepsize	Use this step size if $\|gf_k\|/\|gf_0\| <$ accuracy	1	all real number (negative number means the stepsize by method in "Initstepsize" is used)
LS_ratio1	coefficient in the Armijo condition	0.1	between 0 and 1, i.e., $\in (0, 1)$
LS_ratio2	coefficient in the Armijo condition	0.9	between 0 and 1, i.e., $\in (0, 1)$
Initstepsize	initial step size in first iteration	1	greater than 0
Num_pre_funs	the number of computed functions values stored for nonmonotonic linesearch	0	greater than or equal to 0
InitSteptype	Initial step size	QUADINTMOD / 3	ONESTEP / 0 : use one BBSTEP / 1 : g(s, s) / g(s, y) QUADINT / 2 : [NW06, (3.60)] QUADINTMOD / 3 : [NW06, page 60]
isconvex	whether the cost function is convex	false / 0	false / 0 or true / 1
nu	the same as ϵ in [LF01, (3.2)]	10^{-4}	greater than or equal to 0 and smaller than 1

mu	the same as α in [LF01, (3.2)]	1	greater than or equal to 0
Diffx	the same as τ_x in [LO13, Section 6.3]	10^{-6}	greater than 0

Table 17: Output notation of RBFGS. Note that the first time an action of a vector transport $\mathcal{T}_\eta$ is computed will usually have higher complexity than subsequent times. Specifically, if $\mathcal{T}_\eta\xi_1$ has been computed, then evaluating $\mathcal{T}_\eta\xi_2$ usually can use some results from computations of $\mathcal{T}_\eta\xi_1$. nV denotes the number of evaluations of vector transport first time. nVp denotes the number of other times.

Notation	Interpretation
i	the number of iterations
f	function value
df/f	$(f(x_{i-1}) - f(x_i))/f(x_i)$
gf	$\|\operatorname{grad} f(x_i)\|$
time	computational time (second)
nf	the number of function evaluations
ng	the number of gradient evaluations
nR	the number of retraction evaluations
nV/nVp	the number of actions of vector transport
LSstatus	status of line search result
initslope	initial slope in line search
newslope	the slope of final point in line search
initstepsize	initial step size in line search
stepsize	the final stepsize
betay	$\alpha_i\eta_i/\mathcal{T}_{R_{\alpha_i\eta_i}}(\alpha_i\eta_i)$ see [HGA15, Step 6 of Algorithm 1]
inpss	$\langle s_i, s_i\rangle$
inpsy	$\langle s_i, y_i\rangle$
IsUpdateHessian	Whether update inverse Hessian approximation or not

B.9 LRBFGS

Table 18: Input Parameters of LRBFGS

Name of field	Interpretation	Default value	Applicable values
		C++/(Matlab,Julia)	C++/(Matlab,Julia) : interpretation
IsCheckParams	output parameters of Solvers	Matlab and Julia only : 0	0 or 1
IsCheckGradHess	Check the correctness of gradient and Hessian	Matlab and Julia only : 0	0 or 1
Stop_Criterion	Stopping criterion	GRAD_F_0 / 2	FUN_REL / 0 : $(f(x_{i-1}) - f(x_i))/f(x_i)$ GRAD_F / 1 : $\|\operatorname{grad} f(x_i)\|$ GRAD_F_0 / 2 : $\|\operatorname{grad} f(x_i)\|/\|\operatorname{grad} f(x_0)\|$
Tolerance	Algorithm stops if "Stop_Criterion" < tolerance	10^{-6}	greater than 0
Min_Iteration	minimum number of iterations	0	greater than or equal to 0 and smaller than or equal to Max_Iteration
Max_Iteration	maximum number of iterations	500	greater than or equal to Min_Iteration
OutputGap	Output every "OutputGap" iterations	1	greater than or equal to 1
DEBUG	output information	ITERRESULT / 2	NOOUTPUT / 0 : no output FINALRESULT / 1 : Only final result ITERRESULT / 2 : Output every "OutputGap" iterations

			DETAILED / 3: Output Detailed information
TimeBound	maximum computational time	$60 * 60 * 24 * 365$	greater than 0
LineSearch_LS	Algorithm in linesearch	ARMIJO / 0	ARMIJO / 0 : Back tracking WOLFE / 1 : [DS83, Algorithm A6.3.1mod] STRONGWOLFE / 2 : [NW06, Algorithm 3.5] EXACT / 3 : scaled BFGS WOLFELP / 4 : [AHHY16] INPUTFUN / 5 : Given by users
IsPureLSInput	Whether backtracking is used for step size given by users' algorithm	false / 0	false / 0 or true / 1
LS_alpha	coefficient in the Wolfe first condition	0.0001	between 0 and 0.5, i.e. $\in (0, 0.5)$
LS_beta	coefficient in the Wolfe second condition	0.999	between 0 and 1, i.e., $\in (0, 1)$
Minstepsize	minimum allowed step size	machine eps	greater than 0 and smaller than or equal to Maxstepsize
Maxstepsize	maximum allowed step size	1000	greater than or equal to Minstepsize
Accuracy	fixed the stepsize if $\|gf_k\|/\|gf_0\| <$ accuracy	0	between 0 and 1, i.e., $\in [0, 1]$
Finalstepsize	Use this step size if $\|gf_k\|/\|gf_0\| <$ accuracy	1	all real number (negative number means the stepsize by method in "Initstepsize" is used)
LS_ratio1	coefficient in the Armijo condition	0.1	between 0 and 1, i.e., $\in (0, 1)$
LS_ratio2	coefficient in the Armijo condition	0.9	between 0 and 1, i.e., $\in (0, 1)$
Initstepsize	initial step size in first iteration	1	greater than 0
Num_pre_funs	the number of computed functions values stored for nonmonotonic linesearch	0	greater than or equal to 0
InitSteptype	Initial step size	QUADINTMOD / 3	ONESTEP / 0 : use one BBSTEP / 1 : g(s, s) / g(s, y) QUADINT / 2 : [NW06, (3.60)] QUADINTMOD / 3 : [NW06, page 60]
isconvex	whether the cost function is convex	false / 0	false / 0 or true / 1
nu	the same as ϵ in [LF01, (3.2)]	10^{-4}	greater than or equal to 0 and smaller than 1
mu	the same as α in [LF01, (3.2)]	1	greater than or equal to 0
LengthSY	the same as ℓ in [HGA15, Algorithm 2]	4	greater than or equal to 0

Table 19: Output notation of LRBFGS. Note that the first time an action of a vector transport $\mathcal{T}_\eta$ is computed will usually have higher complexity than subsequent times. Specifically, if $\mathcal{T}_\eta\xi_1$ has been computed, then evaluating $\mathcal{T}_\eta\xi_2$ usually can use some results from computations of $\mathcal{T}_\eta\xi_1$. nV denotes the number of evaluations of vector transport first time. nVp denotes the number of other times.

Notation	Interpretation
i	the number of iterations

f	function value
df/f	$(f(x_{i-1}) - f(x_i))/f(x_i)$
gf	$\|\operatorname{grad} f(x_i)\|$
time	computational time (second)
nf	the number of function evaluations
ng	the number of gradient evaluations
nR	the number of retraction evaluations
nV/nVp	the number of actions of vector transport [7]
LSstatus	status of line search result
initslope	initial slope in line search
newslope	the slope of final point in line search
initstepsize	initial step size in line search
stepsize	the final stepsize
betay	$\alpha_i\eta_i/\mathcal{T}_{R_{\alpha_i\eta_i}}(\alpha_i\eta_i)$ see [HGA15, Step 6 of Algorithm 1]
rho	$1/\langle s_i, y_i\rangle$
gamma	$\langle s_i, y_i\rangle/\langle y_i, y_i\rangle$
inpss	$\langle s_i, s_i\rangle$
inpsy	$\langle s_i, y_i\rangle$
IsUpdateHessian	Whether update inverse Hessian approximation or not

B.10 RCG

Table 20: Input Parameters of RCG

Name of field	Interpretation	Default value	Applicable values
		C++/(Matlab,Julia)	C++/(Matlab,Julia) : interpretation
IsCheckParams	output parameters of Solvers	Matlab and Julia only : 0	0 or 1
IsCheckGradHess	Check the correctness of gradient and Hessian	Matlab and Julia only : 0	0 or 1
Stop_Criterion	Stopping criterion	GRAD_F_0 / 2	FUN_REL / 0 : $(f(x_{i-1}) - f(x_i))/f(x_i)$ GRAD_F / 1 : $\|\operatorname{grad} f(x_i)\|$ GRAD_F_0 / 2 : $\|\operatorname{grad} f(x_i)\|/\|\operatorname{grad} f(x_0)\|$
Tolerance	Algorithm stops if "Stop_Criterion" $<$ tolerance	10^{-6}	greater than 0
Min_Iteration	minimum number of iterations	0	greater than or equal to 0 and smaller than or equal to Max_Iteration
Max_Iteration	maximum number of iterations	500	greater than or equal to Min_Iteration
OutputGap	Output every "OutputGap" iterations	1	greater than or equal to 1
DEBUG	output information	ITERRESULT / 2	NOOUTPUT / 0 : no output FINALRESULT / 1 : Only final result ITERRESULT / 2 : Output every "OutputGap" iterations DETAILED / 3: Output Detailed information
TimeBound	maximum computational time	$60 * 60 * 24 * 365$	greater than 0
LineSearch_LS	Algorithm in linesearch	ARMIJO / 0	ARMIJO / 0 : Back tracking WOLFE / 1 : [DS83, Algorithm A6.3.1mod] STRONGWOLFE / 2 : [NW06, Algorithm 3.5] EXACT / 3 : scaled BFGS WOLFELP / 4 : [AHHY16] INPUTFUN / 5 : Given by users

IsPureLSInput	Whether backtracking is used for step size given by users' algorithm	false / 0	false / 0 or true / 1
LS_alpha	coefficient in the Wolfe first condition	0.0001	between 0 and 0.5, i.e. $\in (0, 0.5)$
LS_beta	coefficient in the Wolfe second condition	0.999	between 0 and 1, i.e., $\in (0, 1)$
Minstepsize	minimum allowed step size	machine eps	greater than 0 and smaller than or equal to Maxstepsize
Maxstepsize	maximum allowed step size	1000	greater than or equal to Minstepsize
Accuracy	fixed the stepsize if $\|gf_k\|/\|gf_0\| <$ accuracy	0	between 0 and 1, i.e., $\in [0, 1]$
Finalstepsize	Use this step size if $\|gf_k\|/\|gf_0\| <$ accuracy	1	all real number (negative number means the stepsize by method in "Initstepsize" is used)
LS_ratio1	coefficient in the Armijo condition	0.1	between 0 and 1, i.e., $\in (0, 1)$
LS_ratio2	coefficient in the Armijo condition	0.9	between 0 and 1, i.e., $\in (0, 1)$
Initstepsize	initial step size in first iteration	1	greater than 0
Num_pre_funs	the number of computed functions values stored for nonmonotonic linesearch	0	greater than or equal to 0
InitSteptype	Initial step size	BBSTEP / 1	ONESTEP / 0 : use one BBSTEP / 1 : g(s, s) / g(s, y) QUADINT / 2 : [NW06, (3.60)] QUADINTMOD / 3 : [NW06, page 60]
RCGmethod	method in choosing β in [AMS08, (8.26)]	HESTENES_STIEFEL / 2	FLETCHER_REEVES / 0 : [AMS08, (8.28)] POLAK_RIBIERE_MOD / 1 : Riemannian generalization of [NW06, (5.45)] HESTENES_STIEFEL / 2 : Riemannian generalization of [NW06, (5.46)] FR_PR / 3 : Riemannian generalization of [NW06, (5.48)] DAI_YUAN / 4 : Riemannian generalization of [NW06, (5.49)] HAGER_ZHANG / 5 : Riemannian generalization of [NW06, (5.50)]
ManDim	search direction is reset every "ManDim" iterations	machine maximum integer	greater than or equal to 0

Table 21: Output notation of RCG. Note that the first time an action of a vector transport $\mathcal{T}_\eta$ is computed will usually have higher complexity than subsequent times. Specifically, if $\mathcal{T}_\eta\xi_1$ has been computed, then evaluating $\mathcal{T}_\eta\xi_2$ usually can use some results from computations of $\mathcal{T}_\eta\xi_1$. nV denotes the number of evaluations of vector transport first time. nVp denotes the number of other times.

Notation	Interpretation
i	the number of iterations
f	function value
df/f	$(f(x_{i-1}) - f(x_i))/f(x_i)$

gf	$\|\operatorname{grad} f(x_i)\|$
time	computational time (second)
nf	the number of function evaluations
ng	the number of gradient evaluations
nR	the number of retraction evaluations
nV/nVp	the number of actions of vector transport
LSstatus	status of line search result
initslope	initial slope in line search
newslope	the slope of final point in line search
initstepsize	initial step size in line search
stepsize	the final stepsize
sigma	the coefficient between $\operatorname{grad} f(x_i)$ and $\mathcal{T}_{\alpha_i\eta_i}(\eta_i)$

B.11 RSD

Table 22: Input Parameters of RSD

Name of field	Interpretation	Default value	Applicable values
		C++/(Matlab,Julia)	C++/(Matlab,Julia) : interpretation
IsCheckParams	output parameters of Solvers	Matlab and Julia only : 0	0 or 1
IsCheckGradHess	Check the correctness of gradient and Hessian	Matlab and Julia only : 0	0 or 1
Stop_Criterion	Stopping criterion	GRAD_F_0 / 2	FUN_REL / 0 : $(f(x_{i-1}) - f(x_i))/f(x_i)$ GRAD_F / 1 : $\|\operatorname{grad} f(x_i)\|$ GRAD_F_0 / 2 : $\|\operatorname{grad} f(x_i)\|/\|\operatorname{grad} f(x_0)\|$
Tolerance	Algorithm stops if "Stop_Criterion" $<$ tolerance	10^{-6}	greater than 0
Min_Iteration	minimum number of iterations	0	greater than or equal to 0 and smaller than or equal to Max_Iteration
Max_Iteration	maximum number of iterations	500	greater than or equal to Min_Iteration
OutputGap	Output every "OutputGap" iterations	1	greater than or equal to 1
DEBUG	output information	ITERRESULT / 2	NOOUTPUT / 0 : no output FINALRESULT / 1 : Only final result ITERRESULT / 2 : Output every "OutputGap" iterations DETAILED / 3: Output Detailed information
TimeBound	maximum computational time	$60 * 60 * 24 * 365$	greater than 0
LineSearch_LS	Algorithm in linesearch	ARMIJO / 0	ARMIJO / 0 : Back tracking WOLFE / 1 : [DS83, Algorithm A6.3.1mod] STRONGWOLFE / 2 : [NW06, Algorithm 3.5] EXACT / 3 : scaled BFGS WOLFELP / 4 : [AHHY16] INPUTFUN / 5 : Given by users
IsPureLSInput	Whether backtracking is used for step size given by users' algorithm	false / 0	false / 0 or true / 1
LS_alpha	coefficient in the Wolfe first condition	0.0001	between 0 and 0.5, i.e. $\in (0, 0.5)$
LS_beta	coefficient in the Wolfe second condition	0.999	between 0 and 1, i.e., $\in (0, 1)$

Minstepsize	minimum allowed step size	machine eps	greater than 0 and smaller than or equal to Maxstepsize
Maxstepsize	maximum allowed step size	1000	greater than or equal to Minstepsize
Accuracy	fixed the stepsize if $\|gf_k\|/\|gf_0\| <$ accuracy	0	between 0 and 1, i.e., $\in [0, 1]$
Finalstepsize	Use this step size if $\|gf_k\|/\|gf_0\| <$ accuracy	1	all real number (negative number means the stepsize by method in "Initstepsize" is used)
LS_ratio1	coefficient in the Armijo condition	0.1	between 0 and 1, i.e., $\in (0, 1)$
LS_ratio2	coefficient in the Armijo condition	0.9	between 0 and 1, i.e., $\in (0, 1)$
Initstepsize	initial step size in first iteration	1	greater than 0
Num_pre_funs	the number of computed functions values stored for nonmonotonic linesearch	0	greater than or equal to 0
InitSteptype	Initial step size	BBSTEP / 1	ONESTEP / 0 : use one BBSTEP / 1 : g(s, s) / g(s, y) QUADINT / 2 : [NW06, (3.60)] QUADINTMOD / 3 : [NW06, page 60]

Table 23: Output notation of RSD. Note that the first time an action of a vector transport $\mathcal{T}_\eta$ is computed will usually have higher complexity than subsequent times. Specifically, if $\mathcal{T}_\eta\xi_1$ has been computed, then evaluating $\mathcal{T}_\eta\xi_2$ usually can use some results from computations of $\mathcal{T}_\eta\xi_1$. nV denotes the number of evaluations of vector transport first time. nVp denotes the number of other times.

Notation	Interpretation
i	the number of iterations
f	function value
df/f	$(f(x_{i-1}) - f(x_i))/f(x_i)$
gf	$\|\operatorname{grad} f(x_i)\|$
time	computational time (second)
nf	the number of function evaluations
ng	the number of gradient evaluations
nR	the number of retraction evaluations
nV/nVp	the number of actions of vector transport
LSstatus	status of line search result
initslope	initial slope in line search
newslope	the slope of final point in line search
initstepsize	initial step size in line search
stepsize	the final stepsize

B.12 RBFGSLPSub

Table 24: Input Parameters of RBFGSLPSub

Name of field	Interpretation	Default value	Applicable values
		C++/(Matlab,Julia)	C++/(Matlab,Julia) : interpretation
IsCheckParams	output parameters of Solvers	Matlab and Julia only : 0	0 or 1
IsCheckGradHess	Check the correctness of gradient and Hessian	Matlab and Julia only : 0	0 or 1

Tolerance	Algorithm stops if $\|gf\|_P <$ tolerance and Eps equals Min_Eps	10^{-6}	greater than 0
Min_Iteration	minimum number of iterations	0	greater than or equal to 0 and smaller than or equal to Max_Iteration
Max_Iteration	maximum number of iterations	500	greater than or equal to Min_Iteration
OutputGap	Output every "OutputGap" iterations	1	greater than or equal to 1
DEBUG	output information	ITERRESULT / 2	NOOUTPUT / 0 : no output FINALRESULT / 1 : Only final result ITERRESULT / 2 : Output every "OutputGap" iterations DETAILED / 3: Output Detailed information
TimeBound	maximum computational time	$60 * 60 * 24 * 365$	greater than 0
IsPureLSInput	Whether backtracking is used for step size given by users' algorithm	false / 0	false / 0 or true / 1
LineSearch_LS	Algorithm in linesearch	WOLFELP / 4	WOLFELP / 4 : [AHHY16]
LS_alpha	coefficient in the Wolfe first condition	0.0001	between 0 and 0.5, i.e. $\in (0, 0.5)$
LS_beta	coefficient in the Wolfe second condition	0.999	between 0 and 1, i.e., $\in (0, 1)$
Minstepsize	minimum allowed step size	machine eps	greater than 0 and smaller than or equal to Maxstepsize
Maxstepsize	maximum allowed step size	1000	greater than or equal to Minstepsize
Initstepsize	initial step size in first iteration	1	greater than 0
InitSteptype	Initial step size	ONESTEP / 0	ONESTEP / 0 : use one BBSTEP / 1 : g(s, s) / g(s, y) QUADINT / 2 : [NW06, (3.60)] QUADINTMOD / 3 : [NW06, page 60]
isconvex	whether the cost function is convex	false / 0	false / 0 or true / 1
lambdaLower	λ in [AHHY16]	10^{-2}	greater than 0 and smaller than lambdaUpper
lambdaUpper	Λ in [AHHY16]	10^2	greater than lambdaLower
Eps	ϵ in [AHHY16]	1	in $(0, 1)$
Theta_eps	θ_δ in [AHHY16]	0.01	in $(0, 1)$
Min_Eps	lower bound of ϵ	10^{-6}	in $(0, 1)$
Del	δ in [AHHY16]	1	in $(0, 1)$
Theta_del	θ_δ in [AHHY16]	0.01	in $(0, 1)$

Table 25: Output notation of RBFGSLPSub. Note that the first time an action of a vector transport $\mathcal{T}_\eta$ is computed will usually have higher complexity than subsequent times. Specifically, if $\mathcal{T}_\eta\xi_1$ has been computed, then evaluating $\mathcal{T}_\eta\xi_2$ usually can use some results from computations of $\mathcal{T}_\eta\xi_1$. nV denotes the number of evaluations of vector transport first time. nVp denotes the number of other times.

Notation	Interpretation
i	the number of iterations
f	function value
df/f	$(f(x_{i-1}) - f(x_i))/f(x_i)$

gf	$\|\operatorname{grad} f(x_i)\|$
time	computational time (second)
nf	the number of function evaluations
ng	the number of gradient evaluations
nR	the number of retraction evaluations
nV/nVp	the number of actions of vector transport
LSstatus	status of line search result
initslope	initial slope in line search
newslope	the slope of final point in line search
initstepsize	initial step size in line search
stepsize	the final stepsize
betay	$\alpha_i\eta_i/\mathcal{T}_{R_{\alpha_i\eta_i}}(\alpha_i\eta_i)$ see [HGA15, Step 6 of Algorithm 1]
inpss	$\langle s_i, s_i\rangle$
inpsy	$\langle s_i, y_i\rangle$
IsUpdateHessian	Whether update inverse Hessian approximation or not
nsubprob	The number of solving quadratic programming problem

B.13 LRBFGSLPSub

Table 26: Input Parameters of RBFGSLPSub

Name of field	Interpretation	Default value	Applicable values
		C++/(Matlab,Julia)	C++/(Matlab,Julia) : interpretation
IsCheckParams	output parameters of Solvers	Matlab and Julia only : 0	0 or 1
IsCheckGradHess	Check the correctness of gradient and Hessian	Matlab and Julia only : 0	0 or 1
Tolerance	Algorithm stops if $\|gf\|_P <$ tolerance and Eps equals Min_Eps	10^{-6}	greater than 0
Min_Iteration	minimum number of iterations	0	greater than or equal to 0 and smaller than or equal to Max_Iteration
Max_Iteration	maximum number of iterations	500	greater than or equal to Min_Iteration
OutputGap	Output every "OutputGap" iterations	1	greater than or equal to 1
DEBUG	output information	ITERRESULT / 2	NOOUTPUT / 0 : no output FINALRESULT / 1 : Only final result ITERRESULT / 2 : Output every "OutputGap" iterations DETAILED / 3: Output Detailed information
TimeBound	maximum computational time	$60 * 60 * 24 * 365$	greater than 0
IsPureLSInput	Whether backtracking is used for step size given by users' algorithm	false / 0	false / 0 or true / 1
LineSearch_LS	Algorithm in linesearch	WOLFELP / 4	WOLFELP / 4 : [AHHY16]
LS_alpha	coefficient in the Wolfe first condition	0.0001	between 0 and 0.5, i.e. $\in (0, 0.5)$
LS_beta	coefficient in the Wolfe second condition	0.999	between 0 and 1, i.e., $\in (0, 1)$
Minstepsize	minimum allowed step size	machine eps	greater than 0 and smaller than or equal to Maxstepsize

Maxstepsize	maximum allowed step size	1000	greater than or equal to Minstepsize
Initstepsize	initial step size in first iteration	1	greater than 0
InitSteptype	Initial step size	ONESTEP / 0	ONESTEP / 0 : use one BBSTEP / 1 : g(s, s) / g(s, y) QUADINT / 2 : [NW06, (3.60)] QUADINTMOD / 3 : [NW06, page 60]
isconvex	whether the cost function is convex	false / 0	false / 0 or true / 1
lambdaLower	λ in [AHHY16]	10^{-2}	greater than 0 and smaller than lambdaUpper
lambdaUpper	Λ in [AHHY16]	10^{2}	greater than lambdaLower
Eps	ϵ in [AHHY16]	1	in $(0, 1)$
Theta_eps	θ_δ in [AHHY16]	0.01	in $(0, 1)$
Min_Eps	lower bound of ϵ	10^{-6}	in $(0, 1)$
Del	δ in [AHHY16]	1	in $(0, 1)$
Theta_del	θ_δ in [AHHY16]	0.01	in $(0, 1)$
LengthSY	The same as ℓ in [HGA15, Algorithm 2]	2	greater than or equal to 0

Table 27: Output notation of RBFGSLPSub. Note that the first time an action of a vector transport $\mathcal{T}_\eta$ is computed will usually have higher complexity than subsequent times. Specifically, if $\mathcal{T}_\eta\xi_1$ has been computed, then evaluating $\mathcal{T}_\eta\xi_2$ usually can use some results from computations of $\mathcal{T}_\eta\xi_1$. nV denotes the number of evaluations of vector transport first time. nVp denotes the number of other times.

Notation	Interpretation
i	the number of iterations
f	function value
df/f	$(f(x_{i-1}) - f(x_i))/f(x_i)$
gf	$\|\operatorname{grad} f(x_i)\|$
time	computational time (second)
nf	the number of function evaluations
ng	the number of gradient evaluations
nR	the number of retraction evaluations
nV/nVp	the number of actions of vector transport
LSstatus	status of line search result
initslope	initial slope in line search
newslope	the slope of final point in line search
initstepsize	initial step size in line search
stepsize	the final stepsize
betay	$\alpha_i\eta_i/\mathcal{T}_{R_{\alpha_i\eta_i}}(\alpha_i\eta_i)$ see [HGA15, Step 6 of Algorithm 1]
inpss	$\langle s_i, s_i\rangle$
inpsy	$\langle s_i, y_i\rangle$
IsUpdateHessian	Whether update inverse Hessian approximation or not
nsubprob	The number of solving quadratic programming problem

B.14 RGS

Table 28: Input Parameters of RBFGSLPSub

Name of field	Interpretation	Default value	Applicable values
		C++/(Matlab,Julia)	C++/(Matlab,Julia) : interpretation
IsCheckParams	output parameters of Solvers	Matlab and Julia only : 0	0 or 1
IsCheckGradHess	Check the correctness of gradient and Hessian	Matlab and Julia only : 0	0 or 1

Tolerance	Algorithm stops if $\|gf\|_P <$ tolerance and Eps equals Min_Eps	10^{-6}	greater than 0
Min_Iteration	minimum number of iterations	0	greater than or equal to 0 and smaller than or equal to Max_Iteration
Max_Iteration	maximum number of iterations	500	greater than or equal to Min_Iteration
OutputGap	Output every "OutputGap" iterations	1	greater than or equal to 1
DEBUG	output information	ITERRESULT / 2	NOOUTPUT / 0 : no output FINALRESULT / 1 : Only final result ITERRESULT / 2 : Output every "OutputGap" iterations DETAILED / 3: Output Detailed information
TimeBound	maximum computational time	$60 * 60 * 24 * 365$	greater than 0
IsPureLSInput	Whether backtracking is used for step size given by users' algorithm	false / 0	false / 0 or true / 1
LineSearch_LS	Algorithm in linesearch	ARMIJO / 0	ARMIJO / 0 : Back tracking
LS_alpha	coefficient in the Wolfe first condition	0.0001	between 0 and 0.5, i.e. $\in (0, 0.5)$
LS_beta	coefficient in the Wolfe second condition	0.999	between 0 and 1, i.e., $\in (0, 1)$
Minstepsize	minimum allowed step size	machine eps	greater than 0 and smaller than or equal to Maxstepsize
Maxstepsize	maximum allowed step size	1000	greater than or equal to Minstepsize
Initstepsize	initial step size in first iteration	1	greater than 0
InitSteptype	Initial step size	ONESTEP / 0	ONESTEP / 0 : use one BBSTEP / 1 : g(s, s) / g(s, y) QUADINT / 2 : [NW06, (3.60)] QUADINTMOD / 3 : [NW06, page 60]
Eps	ϵ in [AHHY16]	1	in $(0, 1)$
Theta_eps	θ_δ in [AHHY16]	0.01	in $(0, 1)$
Min_Eps	lower bound of ϵ	10^{-6}	in $(0, 1)$
Del	δ in [AHHY16]	1	in $(0, 1)$
Theta_del	θ_δ in [AHHY16]	0.01	in $(0, 1)$

Table 29: Output notation of RBFGSLPSub. Note that the first time an action of a vector transport $\mathcal{T}_\eta$ is computed will usually have higher complexity than subsequent times. Specifically, if $\mathcal{T}_\eta\xi_1$ has been computed, then evaluating $\mathcal{T}_\eta\xi_2$ usually can use some results from computations of $\mathcal{T}_\eta\xi_1$. nV denotes the number of evaluations of vector transport first time. nVp denotes the number of other times.

Notation	Interpretation
i	the number of iterations
f	function value
df/f	$(f(x_{i-1}) - f(x_i))/f(x_i)$
gf	$\|\operatorname{grad} f(x_i)\|$
time	computational time (second)
nf	the number of function evaluations
ng	the number of gradient evaluations
nR	the number of retraction evaluations

nV/nVp	the number of actions of vector transport
LSstatus	status of line search result
initslope	initial slope in line search
newslope	the slope of final point in line search
initstepsize	initial step size in line search
stepsize	the final stepsize
nsubprob	The number of solving quadratic programming problem

C Manifold Parameters

This package provides 11 commonly encountered manifolds and the set of non-negative numbers. In the future, we will add more manifolds with more geometric objects.

Table 30: Parameters for Matlab. An example can be found in Lines 11 to 13 of Listing 3.

Manifolds	Name of field	Applicable values
Euclidean space $\mathbb{R}^{n\times m}$	name	'Euclidean'
	n	positive integer
	m	positive integer
Stiefel manifold $\mathrm{St}(p,n)=\{X\in\mathbb{R}^{n\times p}\|X^TX=I_p\}$	name	'Stiefel'
	n	positive integer
	p	positive integer and smaller than or equal to n
	ParamSet	see Table 32
Unit sphere $\mathbb{S}^n=\{x\in\mathbb{R}^n\|x^Tx=1\}$	name	'Sphere'
	n	positive integer
	ParamSet	see Table 33
$\mathbb{L}^2$ Unit sphere $\mathbb{S}^{\mathbb{L}^2}=\{x\in\mathbb{L}^2([0,1],\mathbb{R})\|\int_0^1 x^2(t)dt=1\}$	name	'L2Sphere'
	n	positive integer
	ParamSet	see Table 34
Orthogonal group $\mathbb{O}(n)=\{X\in\mathbb{R}^{n\times n}\|X^TX=1\}$	name	'OrthGroup'
	n	positive integer
	ParamSet	see Table 35
Oblique manifold $\mathcal{OB}(n,m)=\{X\in\mathbb{R}^{n\times p}\|(X^TX)_{ii}=1\}$	name	'Oblique'
	n	positive integer
	m	positive integer
	ParamSet	see Table 36
Fixed-rank manifold $\mathcal{LR}(n,m,p)=\{X\in\mathbb{R}^{n\times m}\|\mathrm{rank}(X)=p\}$	name	'LowRank'
	n	positive integer
	m	positive integer
	p	positive integer
The manifold of symmetric positive definite matrices $\mathbb{S}_n$	name	'SPDManifold'
	n	positive integer
$\mathbb{C}_*^{n\times p}/\mathcal{U}_p=\{[Y]\|Y\in\mathbb{C}_*^{n\times p}\},[Y]=\{YO\|O\in\mathcal{U}_p,i.e.,O^HO=I_p\}$	name	'CpxNStQOrth'
	n	positive integer
	p	positive integer
Euclidean space with nonnegative entries $\mathbb{R}_+^{n\times m}$	name	'EucPositive'
	n	positive integer
	m	positive integer
The tensor of manifolds of SPD matrices: $\mathbb{S}_n^m$	name	'SPDTensor'
	n	positive integer
	m	positive integer
Grassmann manifold: $Gr(p,n)$	name	'Grassmann'
	n	positive integer
	p	positive integer

Table 31: Euclidean space: Parameters of initialized in the C++ constructor

Parameters	Values
Metric	Euclidean
Retraction	exponential $R_x(\eta) = x + \eta$
Vector transport	parallel translation $\mathcal{T}_{\mathrm{S}_\eta}\xi = \xi$
Use intrinsic approach	no (There is no difference.)
Compute β_i in [HGA15, Step 6 of Algorithm 1]	no
Use house holder reflection	no
Satisfy the locking condition	yes

Table 32: The compact Stiefel manifold

Matlab ParamSet value	C++ Member function	Parameters	Values
1	ChooseStieParamsSet1()	Metric	Euclidean
		Retraction	qf retraction [AMS08, (4.8)]
		Vector transport	by parallelization [HAG15, (2.3.1)]
		Use intrinsic approach [Hua13, §9.5]	yes
		Compute β_i in [HGA15, Step 6 of Algorithm 1]	no
		Use house holder reflection	no
		Satisfy the locking condition	no
2	ChooseStieParamsSet2()	Metric	Euclidean
		Retraction	constructed retraction [HGA15, (7.3)]
		Vector transport	by parallelization [HAG15, (2.3.1)]
		Use intrinsic approach [Hua13, §9.5]	yes
		Compute β_i in [HGA15, Step 6 of Algorithm 1]	no
		Use house holder reflection	no
		Satisfy the locking condition	yes
3	ChooseStieParamsSet3()	Metric	Euclidean
		Retraction	qf retraction [HGA15, (7.3)]
		Vector transport	by projection
		Use intrinsic approach [Hua13, §9.5]	no
		Compute β_i in [HGA15, Step 6 of Algorithm 1]	no
		Use house holder reflection	no
		Satisfy the locking condition	no
4	ChooseStieParamsSet4()	Metric	Euclidean
		Retraction	Cayley retraction [Zhu16]
		Vector transport	Cayley vector transport [Zhu16]
		Use intrinsic approach [Hua13, §9.5]	no
		Compute β_i in [HGA15, Step 6 of Algorithm 1]	no
		Use house holder reflection	no
		Satisfy the locking condition	no

Table 33: Unit sphere

Matlab ParamSet value	C++ Member function	Parameters	Values
1	ChooseStieParamsSet1() (default)	Metric	Euclidean
		Retraction	qf retraction [AMS08, (4.8)]
		Vector transport	by parallelization [HAG15, (2.3.1)]
		Use intrinsic approach [Hua13, §9.5]	yes
		Compute β_i in [HGA15, Step 6 of Algorithm 1]	no
		Use house holder reflection	no
		Satisfy the locking condition	no
2	ChooseSphereParamsSet2()	Metric	Euclidean
		Retraction	exponential mapping [AMS08, (5.25)]
		Vector transport	parallel translation [HAG15, (2.3.1)]
		Use intrinsic approach [Hua13, §9.5]	no
		Compute β_i in [HGA15, Step 6 of Algorithm 1]	no
		Use house holder reflection	no
		Satisfy the locking condition	yes
3	ChooseSphereParamsSet3()	Metric	Euclidean
		Retraction	qf retraction [AMS08, (4.8)]
		Vector transport	parallel translation [HAG15, (2.3.1)]
		Use intrinsic approach [Hua13, §9.5]	no
		Compute β_i in [HGA15, Step 6 of Algorithm 1]	no
		Use house holder reflection	no
		Satisfy the locking condition	no
4	ChooseSphereParamsSet4()	Metric	Euclidean
		Retraction	qf retraction [AMS08, (4.8)]
		Vector transport	parallel translation [HAG15, (2.3.1)]
		Use intrinsic approach [Hua13, §9.5]	no
		Compute β_i in [HGA15, Step 6 of Algorithm 1]	yes
		Use house holder reflection	no
		Satisfy the locking condition	yes

Table 34: $\mathbb{L}^2$ unit sphere: Parameters of initialized in the C++ constructor

Parameters	Values
Metric	trapezoidal rule
Retraction	exponential mapping
Vector transport	parallel translation
Use intrinsic approach	no
Compute β_i in [HGA15, Step 6 of Algorithm 1]	no
Use house holder reflection	no
Satisfy the locking condition	yes

Table 35: The orthogonal group $\mathcal{O}_n$

Matlab ParamSet value	C++ Member function	Parameters	Values
1	ChooseStieParamsSet1()	Metric	Euclidean
		Retraction	qf retraction [AMS08, (4.8)]
		Vector transport	by parallelization [HAG15, (2.3.1)]
		Use intrinsic approach [Hua13, §9.5]	yes
		Compute β_i in [HGA15, Step 6 of Algorithm 1]	no
		Use house holder reflection	no
		Satisfy the locking condition	no
2	ChooseStieParamsSet2()	Metric	Euclidean
		Retraction	constructed retraction [HGA15, (7.3)] (equivalent to exponential mapping)
		Vector transport	by parallelization [HAG15, (2.3.1)] (equivalent to parallel translation)
		Use intrinsic approach [Hua13, §9.5]	yes
		Compute β_i in [HGA15, Step 6 of Algorithm 1]	no
		Use house holder reflection	no
		Satisfy the locking condition	yes

Table 36: Product of unit spheres (Oblique manifold)

Matlab ParamSet value	C++ Member function	Parameters	Values
1	ChooseObliqueParamsSet1() (default)	Metric	Euclidean
		Retraction	qf retraction [AMS08, (4.8)]
		Vector transport	by parallelization [HAG15, (2.3.1)]
		Use intrinsic approach [Hua13, §9.5]	yes
		Compute β_i in [HGA15, Step 6 of Algorithm 1]	no
		Use house holder reflection	no
		Satisfy the locking condition	no
2	ChooseObliqueParamsSet2()	Metric	Euclidean
		Retraction	exponential mapping [AMS08, (5.25)]
		Vector transport	parallel translation [HAG15, (2.3.1)]
		Use intrinsic approach [Hua13, §9.5]	no
		Compute β_i in [HGA15, Step 6 of Algorithm 1]	no
		Use house holder reflection	no
		Satisfy the locking condition	yes
3	ChooseObliqueParamsSet3()	Metric	Euclidean
		Retraction	qf retraction [AMS08, (4.8)]
		Vector transport	parallel translation [HAG15, (2.3.1)]
		Use intrinsic approach [Hua13, §9.5]	no
		Compute β_i in [HGA15, Step 6 of Algorithm 1]	no
		Use house holder reflection	no
		Satisfy the locking condition	no
4	ChooseObliqueParamsSet4()	Metric	Euclidean
		Retraction	qf retraction [AMS08, (4.8)]
		Vector transport	parallel translation [HAG15, (2.3.1)]
		Use intrinsic approach [Hua13, §9.5]	no
		Compute β_i in [HGA15, Step 6 of Algorithm 1]	yes
		Use house holder reflection	no
		Satisfy the locking condition	yes

Table 37: Fixed-rank manifold using representation $X = USV^T$, where $U \in \mathrm{St}(r,m)$, $S \in \mathbb{R}^{r\times r}$, and $V \in \mathrm{St}(r,n)$. Parameters of initialized in the C++ constructor. Let $\eta = \dot{U}_1 S V^T + U\dot{S}_1 V^T + US\dot{V}_1^T$ and $\xi = \dot{U}_2 S V^T + U\dot{S}_2 V^T + US\dot{V}_2^T$ be two tangent vectors at x. The metric is $g_X(\eta,\xi) = \mathrm{trace}(\eta^T\xi) = \mathrm{trace}(S^T\dot{U}_1^T\dot{U}_2 S) + \mathrm{trace}(\dot{S}_1^T\dot{S}_2) + \mathrm{trace}(S\dot{V}_1^T\dot{V}_2 S^T)$.

Components	Parameters	Values
Euclidean	Metric	see caption
	Retraction	exponential $R_x(\eta) = x + \eta$
	Vector transport	parallel translation $\mathcal{T}_{\mathrm{S}_\eta}\xi = \xi$
	Use intrinsic approach	no (There is no difference.)
	Compute β_i in [HGA15, Step 6 of Algorithm 1]	no
	Use house holder reflection	no
	Satisfy the locking condition	yes
Stiefel	Metric	see caption
	Retraction	qf retraction [AMS08, (4.8)]
	Vector transport	by parallelization [HAG15, (2.3.1)]
	Use intrinsic approach [Hua13, §9.5]	yes
	Compute β_i in [HGA15, Step 6 of Algorithm 1]	no
	Use house holder reflection	no
	Satisfy the locking condition	no

References

[ABG07] P.-A. Absil, C. G. Baker, and K. A. Gallivan. Trust-region methods on Riemannian manifolds. *Foundations of Computational Mathematics*, 7(3):303–330, 2007.

[AHHY16] P.-A. Absil, H. Hosseini, Wen Huang, and R. Yousefpour. Line search algorithms for locally Lipschitz functions on Riemannian manifolds. Technical report, 2016.

[AMS08] P.-A. Absil, R. Mahony, and R. Sepulchre. *Optimization algorithms on matrix manifolds*. Princeton University Press, Princeton, NJ, 2008.

[DS83] J. E. Dennis and R. B. Schnabel. *Numerical methods for unconstrained optimization and nonlinear equations*. Springer, New Jersey, 1983.

[HAG15] W. Huang, P.-A. Absil, and K. A. Gallivan. A Riemannian symmetric rank-one trust-region method. *Mathematical Programming*, 150(2):179–216, February 2015.

[HGA15] Wen Huang, K. A. Gallivan, and P.-A. Absil. A Broyden Class of Quasi-Newton Methods for Riemannian Optimization. *SIAM Journal on Optimization*, 25(3):1660–1685, 2015.

[Hua13] W. Huang. *Optimization algorithms on Riemannian manifolds with applications*. PhD thesis, Florida State University, Department of Mathematics, 2013.

[LF01] D.-H. Li and M. Fukushima. On the global convergence of the BFGS method for nonconvex unconstrained optimization problems. *SIAM Journal on Optimization*, 11(4):1054–1064, January 2001. doi:10.1137/S1052623499354242.

[LO13] A. S. Lewis and M. L. Overton. Nonsmooth optimization via quasi-Newton methods. *Mathematical Programming*, 141(1-2):135–163, February 2013. doi:10.1007/s10107-012-0514-2.

[NW06] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer, second edition, 2006.

[RW12] W. Ring and B. Wirth. Optimization methods on Riemannian manifolds and their application to shape space. *SIAM Journal on Optimization*, 22(2):596–627, January 2012. doi:10.1137/11082885X.

[SH16] A. Uschmajew S. Hosseini. A Riemannian gradient sampling algorithm for nonsmooth optimization on manifolds. *Institut für Numerische Simulation*, page INS Preprint No. 1607, 2016.

[SI13] H. Sato and T. Iwai. A Riemannian optimization approach to the matrix singular value decomposition. *SIAM Journal on Optimization*, 23(1):188–212, 2013.

[Zhu16] X. Zhu. A riemannian conjugate gradient method for optimization on the stiefel manifold. *Computational Optimization and Applications*, pages 1–38, 2016.