Université catholique de Louvain
Florida State University

Riemannian Manifold Optimization Library

User Manual

Authors:
Wen Huang
Collaborators:
Kyle A. Gallivan
P.-A. Absil

Affiliations:
Université catholique de Louvain

Florida State University
Université catholique de Louvain

May 17, 2015

Contents

1 Introduction

Riemannian manifold optimization library (ROPTLIB) is used to optimize a smooth 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 alone in C++ environment or in Matlab with Mex set up. More reliable and smaller computational time is obtained when compared to packages written in Matlab, such as Manopt [BM] from which this package is partly inspired. Users only need to provide the cost function, the gradient function, and the action of the Hessian (if a Newton method is used) in Matlab or C++. The package optimizes the function with some parameters specified by users, e.g., the domain manifold, algorithm, stopping criterion.

Table 1: Riemannian algorithms in the package

Riemannian trust-region Newton (RTRNewton)	[ABG07]
Riemannian trust-region symmetric rank-one update (RTRSR1)	[HAG14]
Limited-memory RTRSR1 (LRTRSR1)	[HAG14]
Riemannian trust-region steepest descent (RTRSD)	[AMS08]
Riemannian line-search Newton (RNewton)	[AMS08]
Riemannian Broyden family (RBroydenFamily)	[HGA14]
Riemannian BFGS (RWRBFGS and RBFGS)	[RW12, HGA14]
Limited-memory RBFGS (LRBFGS)	[HGA14]
Riemannian conjugate gradients (RCG)	[NW06, AMS08, SI13]
Riemannian steepest descent (RSD)	[AMS08]

2 Installation and First Example

The package has been tested on Windows 7, Ubuntu 14.04 and MAC OS X 10.8.5 when the code is compiled in Matlab. It also has been tested on Windows 7 when the code is compiled in C++ environment alone.

2.1 Compile in matlab

Users need to set up the mex environment properly. We refer to the following url for details: `www.mathworks.com/support/compilers/R2014b/index.html` Users are not required to install blas and lapack in this case since those libraries are included in Matlab.

It is shown next that how to run the code in matlab. First of all, go to the root directory, i.e., `/ROPTLIB/`. Then run "GenerateMyMex.m". A file called "MyMex.m" will be generated or updated. The file "MyMex.m" is used to compile the package. User can use command "MyMex" followed by a name of any test files in `/ROPTLIB/test/` to compile the test file. For example, run "MyMex TestStieBrockett" to test Brockett cost function on the Stiefel manifold [AMS08, Section 4.8]. A binary file "TestStieBrockett.***" will be generated in `/ROPTLIB/test/BinaryFiles/` , where the suffix *** depends on systems. Finally, use command "TestStieBrockett" to run the binary file. The commands and results can be found in Listing 1. The interpretations of output notation can be found in Appendix B.

Listing 1: Test code

```
1  >> GenerateMyMex
```

```
2  Generate MyMex.m file...
3  >> MyMex TestStieBrockett
4  Building with 'g++-4.7'.
5  MEX completed successfully.
6  >> n = 12; p = 4; B = randn(n, n); B = B + B'; D = (p:-1:1)';
7  >> Xinitial = orth(randn(n, p));
8  >> SolverParams.method = 'RBFGS'; SolverParams.IsCheckParams = 1; SolverParams.DEBUG = 1;
9  >> HasHHR = 0;
10 >> [Xopt, f, gf, gfgf0, iter, nf, ng, nR, nV, nVp, nH, ComTime] = TestStieBrockett(B, D,
       Xinitial, HasHHR, SolverParams);
11 (n, p):12,4
12 GENERAL PARAMETERS:
13 Stop_Criterion:       GRAD_F_0[YES],    Tolerance     :          1e-06[YES]
14 Max_Iteration :            500[YES],    Min_Iteration :              0[YES]
15 OutputGap     :              1[YES],    DEBUG         :    FINALRESULT[YES]
16 LINE SEARCH TYPE METHODS PARAMETERS:
17 LineSearch_LS :          ARMIJO[YES],    LS_alpha      :         0.0001[YES]
18 LS_ratio1     :             0.1[YES],    LS_ratio2     :            0.9[YES]
19 Initstepsize  :               1[YES]
20 Minstepsize   :    2.22045e-16[YES],    Maxstepsize   :           1000[YES]
21 RBFGS METHOD PARAMETERS:
22 nu            :          0.0001[YES],    mu            :              1[YES]
23 isconvex      :               0[YES]
24 ========================RBFGS========================
25 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 Compile in C++ enviroment alone

Users need install blas and lapack first. We refer to the next links for details: `http://www.fi.muni.cz/~xsvobod2/misc/lapack/` and `http://www.netlib.org/` We only state method of compiling this code in Windows 7 using IDE Visual Studio Express 2013. Click "PROJECT→properties" and add directories and libraries of this package, blas, and lapack to "Configuration properties→VC++ Directories→General→Include directories" and "Configuration properties→Linker→Input→Additional Dependencies" respectively.

It is shown next that how to run a test file. First, open `/ROPTLIB/test/Others/def.h`. Uncomment one of commands from 11th to 20th line to specify a test problem. Finally, press F5 or ctrl + F5 to compile and run the test problem.

3 For Matlab Users

This package contains three parts, including problem definition, manifold and solver. In order to use this package, users need define a problem by providing routines of a cost function, its gradient and its action of Hessian (if Newton method is used) and specify a manifold of domain and a solver. In this section, we state how to use this package.

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

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

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 summation of three Brockett cost functions

$$\min_{(X_1,X_2,X_3)\in\mathrm{St}(p,n)\times\mathrm{St}(p,n)\times\mathrm{St}(q,m)} \mathrm{trace}(X_1^TB_1X_1D_1) + \mathrm{trace}(X_2^TB_2X_2D_2) + \mathrm{trace}(X_3^TB_3X_3D_3) \tag{3.2}$$

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

First, use 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.

The script DriverOPT can be called by:

```
finalIterate = DriverOPT(fhandle, gfhandle, Hesshandle, SolverParams,
                         ManiParams, HasHHR, initialIterate)
```

where "initialIterate" and "finalIterate" are structures that contain initial and final iterates respectively; "fhandle", "gfhandle", and "Hesshandle" are function handles of cost function, its Euclidean gradient and its action of Euclidean Hessian; "SolverParams" and "ManiParams" are structures that specify parameters of solver and manifold respectively; and "HasHHR" indicates whether the locking condition [HGA14, (2.8)] is satisfied by using the approach in [HGA14, Section 4.1].

3.1 A Simple Example

An example for Brockett cost function (3.1) is given in Listing 2 and the code can be found in `/ROPTLIB/Matlab/ForMatlab/testSimpleExample.m`. First, the cost function, Euclidean gradient and action of Euclidean Hessian are given from line 22 to line 33. The function handles of those routines are assigned from line 5 to line 7. Iterates and tangent vectors are stored as structures with field "main", as shown in line 15 and line 32. In case of storing a temporary data to save computations, users can put the temporary data on an iterate with a different field. For example, Brockett cost function is $\text{trace}(X^TBXD)$ and the Euclidean gradient is $2BXD$. It is required to evaluate BXD in the cost function evaluation. Therefore, one can use the result from function evaluation and yields cheap computation in the gradient evaluation. This can be seen from definitions of f(x, B, D) and gf(x, B, D) in line 23 and line 28. All fields for each solver and manifold can be found in Appendices B and C.

Listing 2: Test Brockett

```
1  function output = 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
11   ManiParams.name = 'Stiefel';     % Domain is the Stiefel manifold
12   ManiParams.n = n;                % assign size to manifold parameter
13   ManiParams.p = p;                % assign size to manifold parameter
14
15   initialX.main = orth(randn(n, p));  % initial iterate
16
17   % call the driver
18   output = DriverOPT(fhandle, gfhandle, Hesshandle,
19                      SolverParams, ManiParams, initialX);
20 end
```

```
21
22 function [output, x] = f(x, B, D)
23   x.BUD = B * x.main * D;
24   output = x.main(:)' * x.BUD(:);
25 end
26
27 function [output, x] = gf(x, B, D)
28   output.main = 2 * x.BUD;
29 end
30
31 function [output, x] = Hess(x, eta, B, D)
32   output.main = 2 * B * eta.main * D;
33 end
```

3.2 An Example for product of manifolds

An exmaple for summation of three Brockett cost function is given in Listing 3 and the code can be found in `/ROPTLIB/Matlab/ForMatlab/testProductExample.m`. 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 3 from line 20 to line 27.

All components of an iterate of products of manifolds is 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. Then 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 29-31, 40-42, 49, 56-58 and 62 in Listing 3.

Listing 3: Test summation of Brockett

```
1  function output = 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';
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    output = DriverOPT(fhandle, gfhandle, Hesshandle,
34                       SolverParams, ManiParams, initialX);
35  end
36
37  function [output, x] = f(x, B1, D1, B2, D2, B3, D3)
38    n = size(B1, 1); p = size(D1, 1);
39    m = size(B3, 1); q = size(D3, 1);
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
```

3.3 Check Gradient and action of Hessian

This package provides a routine to test whether gradient and action of Hessian are correct or not. 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 take effect and the values of α are sufficient small satisfying the higher order term is negligible. If (3.3) is approximately one in the interval, then grad f is probably correct. Likewise, if (3.4) is approximately one in the interval, the retraction R is a second-order retraction or x is a stationary point of f, then the Hess f is probably correct.

To run the function, users need to set the field "IsCheckGradHess" to be one in solver's parameters. Take Listing 2 to be an example. Use command "SolverParams.IsCheckGradHess = 1" in function "testBrockett()". Two sets of values (3.3) and (3.4) will be output. One is the results with x be the initial iterate and the other one is with x be the final iterate obtained by the solver.

4 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 abstract class *Manifold*, all routines only related to manifolds are declared, e.g., retraction, vector transport. Some of these routines are also defined as default, e.g., the default metric is the Frobenius inner product. Abstract class *Problem* contains all prototypes of cost function, Riemannian gradient, Euclidean gradient, action of Riemannian Hessian and action of Euclidean Hessian. It not only automatically chooses routines that have been overloaded (polymorphism), but also includes a routine to check the correctness of gradient and Hessian, see mathematical derivations in Section 3.3. The domain of a problem is also required to be specified using one of manifold classes. Note that class *mexProblem* is a bridge between C++ and Matlab. It uses function handles of Matlab and produces C++ routines. Each solver accepts an object of *Problem* and an object of *Variable*, which is an initial iterate, and outputs a final iterate based on the given parameters.

Users need write a problem class by inheriting the abstract class `/ROPTLIB/Problems/Problem.h` and overload the routines of cost function, gradient and also action of Hessian if Newton method is used. More specifically, either routines of cost function, Riemannian gradient and action of Riemannian Hessian

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

or routines of cost function, Euclidean gradient and action of Euclidean Hessian

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

are needed to be overloaded.

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

4.1 A Simple Example

An example for Brockett cost function (3.1) is given in Listings 4, 5 and 6. Listings 4 and 5 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 Euclidean Hessian are overloaded. Listing 6 gives a test file for the Brockett problem. Those codes can be found in

/ROPTLIB/Problems/StieBrockett.* and /ROPTLIB/test/TestSimpleExample.cpp. To run the codes, users need ensure comment out lines from 11 to 17 and line 20 and uncomment line 19 in file /ROPTLIB/Others/def.h.

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

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

The pointer obtained from *ObtainReadData* is constant and the data it points to are not allowed to be modified. The pointer obtained from *ObtainWriteEntireData* is not constant. Therefore, the data it points to can be modified. However, the old data may or may not exist. The pointer obtained from *ObtainWritePartialData* is not constant and the old data it points to still exists. Their efficiencies have relationships that $ObtainReadData \geq ObtainWriteEntireData \geq ObtainWritePartialData$, where $A \geq B$ means the efficiency of A is at least as good as B.

C++ code provides a way to share information in computation of function, gradient and action of Hessian. 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 onto an object of a class derived from *Element*. One example can be found in Listing 5. 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 28-34 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 attached to **x**. Therefore, one mush attach *SharedSpace* object to element after finish computing the *SharedSpace* object. In Line 51 of Listing 5, a pointer to the *SharedSpace* object is obtained from **x** by using name "BxD". The data in the *SharedSpace* object is obtained in Line 52. More example can be found in files under directory /ROPTLIB/Problems/.

In order to avoid memory leaking, user need delete all the objects that is constructed by command "new" except *SharedSpace* objects that are attached to an element. One example is in Lines 38 to 45 of Listing 5. The object **Temp** is constructed by command "new" on Line 21. If **Temp** is attached to **x**, then it can not be deleted. Otherwise, **Temp** must be deleted.

C++ codes, of course, support checking correctness of gradient and action of Hessian. One example is given in Line 62 to 64 of Listing 6.

Listing 4: 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 5: 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 6: 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  int main(void)
16  {
17          // choose a random seed
18          unsigned tt = (unsigned)time(NULL);
19          init_genrand(tt);
20
21          // size of the Stiefel manifold
22          integer n = 12, p = 8;
23
24          // Generate the matrices in the Brockett problem.
25          double *B = new double[n * n + p];
26          double *D = B + n * n;
27          for (integer i = 0; i < n; i++)
28          {
29                  for (integer j = i; j < n; j++)
30                  {
31                          B[i + j * n] = genrand_gaussian();
32                          B[j + i * n] = B[i + j * n];
33                  }
34          }
35          for (integer i = 0; i < p; i++)
36                  D[i] = static_cast<double> (i + 1);
37
38          // Obtain an initial iterate
39          StieVariable StieX(n, p);
40          StieX.RandInManifold();
41
```

```
42          // Define the Stiefel manifold
43          Stiefel Domain(n, p);
44
45          // Define the Brockett problem
46          StieBrockett Prob(B, D, n, p);
47
48          // Set the domain of the problem to be the Stiefel manifold
49          Prob.SetDomain(&Domain);
50
51          // output the parameters of the manifold of domain
52          Domain.CheckParams();
53
54          // test RTRNewton
55          std::cout << "*********Check RTRNewton**********" << std::endl;
56          RTRNewton RTRNewtonsolver(&Prob, &StieX);
57          RTRNewtonsolver.DEBUG = FINALRESULT;
58          RTRNewtonsolver.CheckParams();
59          RTRNewtonsolver.Run();
60
61          // Check gradient and Hessian
62          Prob.CheckGradHessian(&StieX);
63          const Variable *xopt = RTRNewtonsolver.GetXopt();
64          Prob.CheckGradHessian(xopt);
65
66          delete[] B;
67
68          return 0;
69  }
70  #endif
71  #endif
```

4.2 An Example for Product of Manifolds

This section gives the C++ code for the problem (3.2) defined on product of manifolds (see Section 3.2).

The codes of defining a product of manifolds and a point on the manifold is given from Line 85 to 99 of Listing 9. The space of all components required by a point on a product of manifolds are stored in consecutive memories. For example, as shown in Line 25 of Listing 8, a pointer to a 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 three components. Note that the order of components must be consistent with the order of initial iterate user constructed in Line 93 of Listing 9.

It is allowed to cast **x** to be a pointer to *ProductElement* for problems on a product of manifolds, as shown for example in Line 29 of Listing 8. Each component of **x** can be obtained by using member function *GetElement(integer)*, e.g. Line 30 of Listing 8. If users want to overwrite data which is pointed by a pointer obtained by *GetElement(integer)*, then it is required to use *NewMemoryOnWrite(void)* or *CopyOnWrite(void)* for the *ProductElement* object first. *NewMemoryOnWrite(void)* creates new memories if necessary. *CopyOnWrite(void)* not only creates new memories if necessary, it also copies the data from old memories to the new memories. For example, in the routine *EucGrad* of Listing 8, **egf** is the output gradient and it is not important what data is in **egf**. What important is to make sure **egf** has efficient memories to store results. Therefore, before overwriting it, we use *NewMemoryOnWrite(void)* routine in Line 107 to ensure it contains enough space. Similarly, the same routine is used in Line 121.

Listing 7: 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;
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 8: 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 9: 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

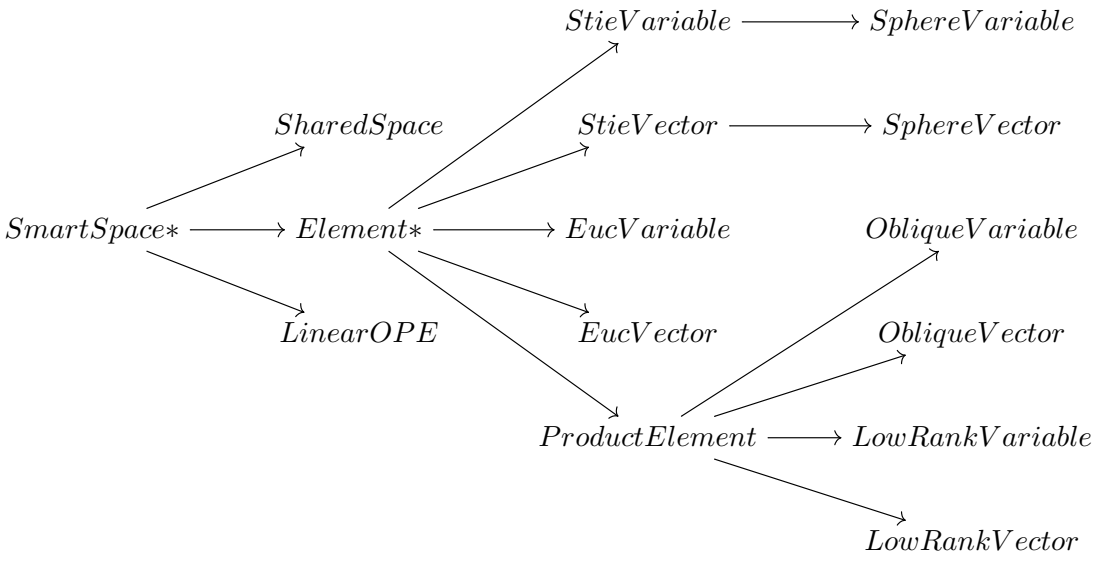

Figure 1: Relationships among classes of variables and tangent vectors in the package. Arrows are from base class to derived class, i.e., base class → derived class. All the inheritances are public. A class with star ∗ is abstract. Variable and Vector are defined to be Element.

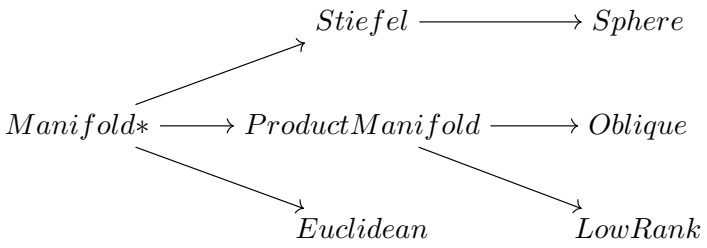

Figure 2: Relationships among classes of Manifolds in the package. Arrows are from base class to derived class, i.e., base class → derived class. All the inheritances are public. A class with star ∗ is abstract.

A.2 Problem-related Classes

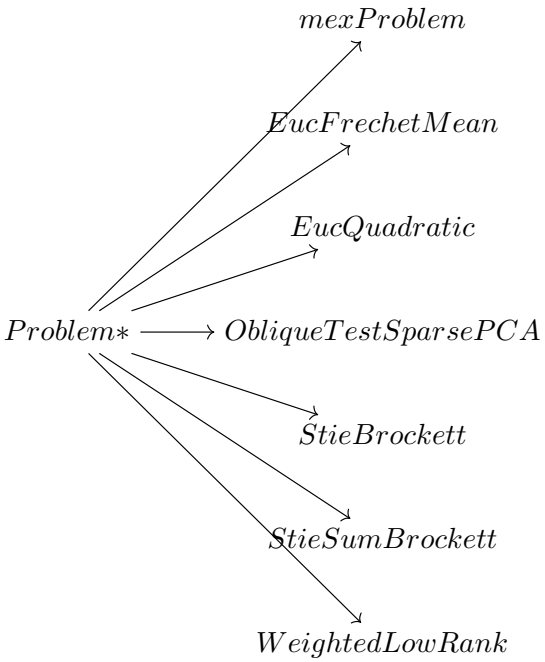

Figure 3: Relationships among classes of problems in the package. Arrows are from base class to derived class, i.e., base class → derived class. All the inheritances are public. A class with star ∗ is abstract.

A.3 Solver-related Classes

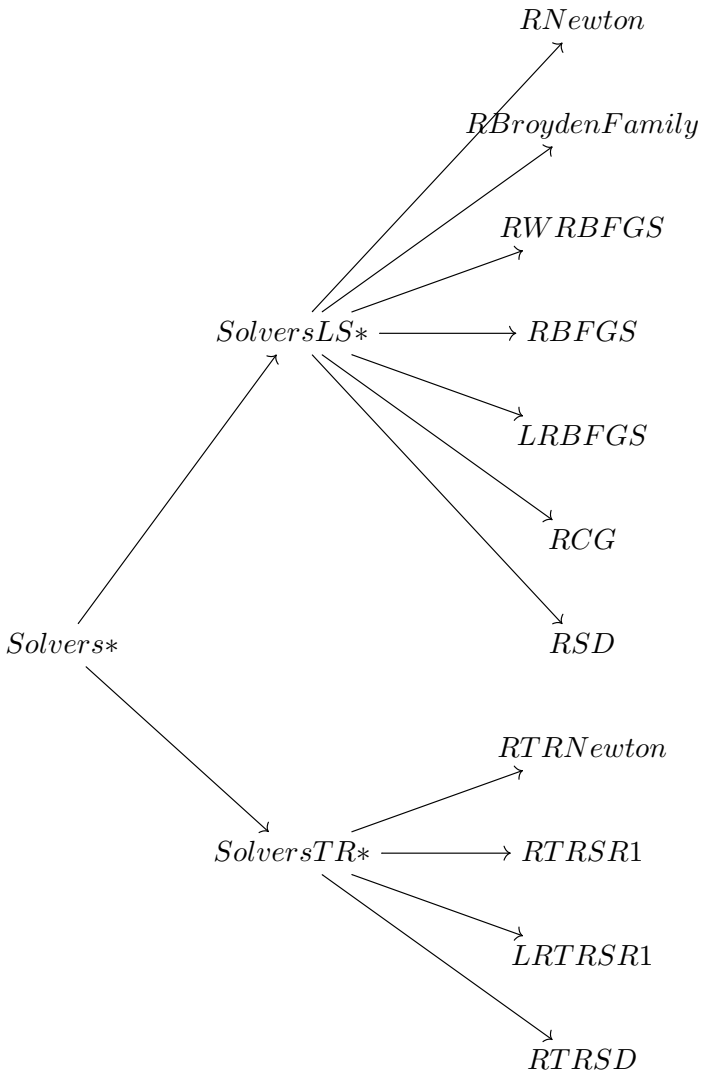

Figure 4: Relationships among classes of solvers in the package. Arrows are from base class to derived class, i.e., base class → derived class. All the inheritances are public. A class with star ∗ is abstract.

B Input Parameters and Output Notation of Solvers

B.1 RTRNewton

Table 2: Input Parameters of RTRNewton

name of fields	interpretation	default value	applicable values
		C++/Matlab	C++/Matlab : interpretation
IsCheckParams	output parameters of Solvers	Matlab only : 0	0 or 1
IsCheckGradHess	Check the correctness of gradient and Hessian	Matlab only : 0	0 or 1
Stop_Criterion	Stopping criterion[1]	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
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	1000	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

[1]Users can write their own function to check stopping criterion by overloading the C++ function "bool Solver-

Table 3: Output notation of RTRNewton

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 [2]
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

s::IsStopped(void)" in /ROPTLIB/Solvers/Solvers.cpp.

[2]The first time to compute an action of a vector transport $\mathcal{T}_\eta$ usually have higher complexities than other 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.

B.2 RTRSR1

Table 4: Input Parameters of RTRSR1

name of fields	interpretation	default value	applicable values
		C++/Matlab	C++/Matlab : interpretation
IsCheckParams	output parameters of Solvers	Matlab only : 0	0 or 1
IsCheckGradHess	Check the correctness of gradient and Hessian	Matlab only : 0	0 or 1
Stop_Criterion	Stopping criterion[3]	GRAD_F_0 / 2	FUN_REL / 0 : $\lvert(f(x_{i-1}) - f(x_i))/f(x_i)\rvert$ GRAD_F / 1 : $\lVert \operatorname{grad} f(x_i)\rVert$ GRAD_F_0 / 2 : $\lVert \operatorname{grad} f(x_i)\rVert / \lVert \operatorname{grad} f(x_0)\rVert$
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
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	1000	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
isconvex	whether the cost function is convex	false / 0	false / 0 or true / 1

[3]Users can write their own function to check stopping criterion by overloading the C++ function "bool Solvers::IsStopped(void)" in /ROPTLIB/Solvers/Solvers.cpp.

Table 5: Output notation of RTRSR1

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 [4]
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

[4]The first time to compute an action of a vector transport $\mathcal{T}_\eta$ usually have higher complexities than other 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.

B.3 LRTRSR1

Table 6: Input Parameters of LRTRSR1

name of fields	interpretation	default value	applicable values
		C++/Matlab	C++/Matlab : interpretation
IsCheckParams	output parameters of Solvers	Matlab only : 0	0 or 1
IsCheckGradHess	Check the correctness of gradient and Hessian	Matlab only : 0	0 or 1
Stop_Criterion	Stopping criterion[5]	GRAD_F_0 / 2	FUN_REL / 0 : $\lvert(f(x_{i-1}) - f(x_i))/f(x_i)\rvert$ GRAD_F / 1 : $\lVert \operatorname{grad} f(x_i)\rVert$ GRAD_F_0 / 2 : $\lVert \operatorname{grad} f(x_i)\rVert / \lVert \operatorname{grad} f(x_0)\rVert$
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
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	1000	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
isconvex	whether the cost function is convex	false / 0	false / 0 or true / 1
LengthSY	the same as ℓ in [HGA14, Algorithm 2]	4	greater than or equal to 0

Table 7: Output notation of LRTRSR1

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 [6]
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

[5] Users can write their own function to check stopping criterion by overloading the C++ function "bool Solvers::IsStopped(void)" in /ROPTLIB/Solvers/Solvers.cpp.

[6] The first time to compute an action of a vector transport $\mathcal{T}_\eta$ usually have higher complexities than other 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.

B.4 RTRSD

Table 8: Input Parameters of RTRSD

name of fields	interpretation	default value	applicable values
		C++/Matlab	C++/Matlab : interpretation
IsCheckParams	output parameters of Solvers	Matlab only : 0	0 or 1
IsCheckGradHess	Check the correctness of gradient and Hessian	Matlab only : 0	0 or 1
Stop_Criterion	Stopping criterion[7]	GRAD_F_0 / 2	FUN_REL / 0 : $\lvert(f(x_{i-1}) - f(x_i))/f(x_i)\rvert$ GRAD_F / 1 : $\lVert \operatorname{grad} f(x_i)\rVert$ GRAD_F_0 / 2 : $\lVert \operatorname{grad} f(x_i)\rVert/\lVert \operatorname{grad} f(x_0)\rVert$
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
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	1000	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

[7]Users can write their own function to check stopping criterion by overloading the C++ function "bool Solvers::IsStopped(void)" in /ROPTLIB/Solvers/Solvers.cpp.

Table 9: Output notation of RTRSD

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 [8]
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

[8]The first time to compute an action of a vector transport $\mathcal{T}_\eta$ usually have higher complexities than other 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.

B.5 RNewton

Table 10: Input Parameters of RNewton

name of fields	interpretation	default value	applicable values
		C++/Matlab	C++/Matlab : interpretation
IsCheckParams	output parameters of Solvers	Matlab only : 0	0 or 1
IsCheckGradHess	Check the correctness of gradient and Hessian	Matlab only : 0	0 or 1
Stop_Criterion	Stopping criterion[9]	GRAD_F_0 / 2	FUN_REL / 0 : $\lvert(f(x_{i-1}) - f(x_i))/f(x_i)\rvert$ GRAD_F / 1 : $\lVert \operatorname{grad} f(x_i)\rVert$ GRAD_F_0 / 2 : $\lVert \operatorname{grad} f(x_i)\rVert / \lVert \operatorname{grad} f(x_0)\rVert$
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
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
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
LS_ratio	coefficient in the Armijo condition	0.25	between 0 and 1, i.e., $\in (0, 1)$
Initstepsize	initial step size in first iteration	1	greater than 0
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

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 [10]
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

[9] Users can write their own function to check stopping criterion by overloading the C++ function "bool Solvers::IsStopped(void)" in /ROPTLIB/Solvers/Solvers.cpp.

[10] The first time to compute an action of a vector transport $\mathcal{T}_\eta$ usually have higher complexities than other 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.

B.6 RBroydenFamily

Table 12: Input Parameters of RBroydenFamily

name of fields	interpretation	default value	applicable values
		C++/Matlab	C++/Matlab : interpretation
IsCheckParams	output parameters of Solvers	Matlab only : 0	0 or 1
IsCheckGradHess	Check the correctness of gradient and Hessian	Matlab only : 0	0 or 1
Stop_Criterion	Stopping criterion[11]	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
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
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
LS_ratio	coefficient in the Armijo condition	0.25	between 0 and 1, i.e., $\in (0, 1)$
Initstepsize	initial step size in first iteration	1	greater than 0
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

[11] Users can write their own function to check stopping criterion by overloading the C++ function "bool Solvers::IsStopped(void)" in /ROPTLIB/Solvers/Solvers.cpp.

Table 13: Output notation of RBroydenFamily

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 [12]
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 [HGA14, Step 6 of Algorithm 1]
Phic	the coefficient ϕ_i in the update [HGA14, (2.3)]
inpss	$\langle s_i, s_i\rangle$
inpsy	$\langle s_i, y_i\rangle$
IsUpdateHessian	Whether update inverse Hessian approximation or not

[12]The first time to compute an action of a vector transport $\mathcal{T}_\eta$ usually have higher complexities than other 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.

B.7 RWRBFGS

Table 14: Input Parameters of RWRBFGS

name of fields	interpretation	default value	applicable values
		C++/Matlab	C++/Matlab : interpretation
IsCheckParams	output parameters of Solvers	Matlab only : 0	0 or 1
IsCheckGradHess	Check the correctness of gradient and Hessian	Matlab only : 0	0 or 1
Stop_Criterion	Stopping criterion[13]	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
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
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
LS_ratio	coefficient in the Armijo condition	0.25	between 0 and 1, i.e., $\in (0, 1)$
Initstepsize	initial step size in first iteration	1	greater than 0
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

[13] Users can write their own function to check stopping criterion by overloading the C++ function "bool Solvers::IsStopped(void)" in /ROPTLIB/Solvers/Solvers.cpp.

Table 15: Output notation of RWRBFGS

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 [14]
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

[14]The first time to compute an action of a vector transport $\mathcal{T}_\eta$ usually have higher complexities than other 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.

B.8 RBFGS

Table 16: Input Parameters of RBFGS

name of fields	interpretation	default value	applicable values
		C++/Matlab	C++/Matlab : interpretation
IsCheckParams	output parameters of Solvers	Matlab only : 0	0 or 1
IsCheckGradHess	Check the correctness of gradient and Hessian	Matlab only : 0	0 or 1
Stop_Criterion	Stopping criterion[15]	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
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
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
LS_ratio	coefficient in the Armijo condition	0.25	between 0 and 1, i.e., $\in (0, 1)$
Initstepsize	initial step size in first iteration	1	greater than 0
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

[15]Users can write their own function to check stopping criterion by overloading the C++ function "bool Solvers::IsStopped(void)" in /ROPTLIB/Solvers/Solvers.cpp.

Table 17: Output notation of RBFGS

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 [16]
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 [HGA14, 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

[16] The first time to compute an action of a vector transport $\mathcal{T}_\eta$ usually have higher complexities than other 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.

B.9 LRBFGS

Table 18: Input Parameters of LRBFGS

name of fields	interpretation	default value	applicable values
		C++/Matlab	C++/Matlab : interpretation
IsCheckParams	output parameters of Solvers	Matlab only : 0	0 or 1
IsCheckGradHess	Check the correctness of gradient and Hessian	Matlab only : 0	0 or 1
Stop_Criterion	Stopping criterion[17]	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
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
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
LS_ratio	coefficient in the Armijo condition	0.25	between 0 and 1, i.e., $\in (0, 1)$
Initstepsize	initial step size in first iteration	1	greater than 0
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 [HGA14, Algorithm 2]	4	greater than or equal to 0

[17]Users can write their own function to check stopping criterion by overloading the C++ function "bool Solvers::IsStopped(void)" in /ROPTLIB/Solvers/Solvers.cpp.

Table 19: Output notation of LRBFGS

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 [18]
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 [HGA14, 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

[18] The first time to compute an action of a vector transport $\mathcal{T}_\eta$ usually have higher complexities than other 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.

B.10 RCG

Table 20: Input Parameters of RCG

name of fields	interpretation	default value	applicable values
		C++/Matlab	C++/Matlab : interpretation
IsCheckParams	output parameters of Solvers	Matlab only : 0	0 or 1
IsCheckGradHess	Check the correctness of gradient and Hessian	Matlab only : 0	0 or 1
Stop_Criterion	Stopping criterion[19]	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
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
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
LS_ratio	coefficient in the Armijo condition	0.25	between 0 and 1, i.e., $\in (0, 1)$
Initstepsize	initial step size in first iteration	1	greater than 0
RCGmethod	method in choosing β in [AMS08, (8.26)]	POLAK_RIBIERE_MOD / 1	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

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 [20]
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 $\mathcal{T}_{\alpha_i \eta_i}(\eta_i)$

[19] Users can write their own function to check stopping criterion by overloading the C++ function "bool Solvers::IsStopped(void)" in /ROPTLIB/Solvers/Solvers.cpp.

[20] The first time to compute an action of a vector transport $\mathcal{T}_\eta$ usually have higher complexities than other 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.

B.11 RSD

Table 22: Input Parameters of RSD

name of fields	interpretation	default value	applicable values
		C++/Matlab	C++/Matlab : interpretation
IsCheckParams	output parameters of Solvers	Matlab only : 0	0 or 1
IsCheckGradHess	Check the correctness of gradient and Hessian	Matlab only : 0	0 or 1
Stop_Criterion	Stopping criterion[21]	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
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
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
LS_ratio	coefficient in the Armijo condition	0.25	between 0 and 1, i.e., $\in (0, 1)$
Initstepsize	initial step size in first iteration	1	greater than 0

[21] Users can write their own function to check stopping criterion by overloading the C++ function "bool Solvers::IsStopped(void)" in /ROPTLIB/Solvers/Solvers.cpp.

Table 23: Output notation of RSD

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 [22]
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

[22]The first time to compute an action of a vector transport $\mathcal{T}_\eta$ usually have higher complexities than other 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.

C Manifold Parameters

This package provides 5 commonly encountered manifolds, i.e., the Euclidean space, the sphere, the compact Stiefel manifold, the low rank manifold and the oblique manifold. In the future, we will add more manifolds with more geometric objects.

Table 24: Parameters for Matlab. An example can be found in Lines 11 to 13 of Listing 2.

Manifolds	name of fields	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 26
Unit sphere $\mathbb{S}^n=\{x\in\mathbb{R}^n\|x^Tx=1\}$	name	'Sphere'
	n	positive integer
	ParamSet	see Table 27
$\mathbb{L}^2$ 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 28
Orthogonal group $\mathbb{O}(n)=\{X\in\mathbb{R}^{n\times n}\|X^TX=1\}$	name	'OrthGroup'
	n	positive integer
	ParamSet	see Table 29
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 30
Low-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

Table 25: Euclidean space

member function	parameters	values
initialized in C++ constructor	Metric	Euclidean
	Retraction	exponential $R_x(\eta)=x+\eta$
	Vector transport	parallel translation $\mathcal{T}_{S_\eta}\xi=\xi$
	Use intrinsic approach	no (There is no difference.)
	Compute β_i in [HGA14, Step 6 of Algorithm 1]	no
	Use house holder reflection	no
	Satisfy the locking condition	yes

Table 26: 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 [HAG14, (2.3.1)]
		Use intrinsic approach [Hua13, §9.5]	yes
		Compute β_i in [HGA14, Step 6 of Algorithm 1]	no
		Use house holder reflection	no
		Satisfy the locking condition	no
2	ChooseStieParamsSet2()	Metric	Euclidean
		Retraction	constructed retraction [HGA14, (7.3)]
		Vector transport	by parallelization [HAG14, (2.3.1)]
		Use intrinsic approach [Hua13, §9.5]	yes
		Compute β_i in [HGA14, Step 6 of Algorithm 1]	no
		Use house holder reflection	no
		Satisfy the locking condition	yes

Table 27: 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 [HAG14, (2.3.1)]
		Use intrinsic approach [Hua13, §9.5]	yes
		Compute β_i in [HGA14, Step 6 of Algorithm 1]	no
		Use house holder reflection	no
		Satisfy the locking condition	no
2	ChooseSphereParamsSet1()	Metric	Euclidean
		Retraction	exponential mapping [AMS08, (5.25)]
		Vector transport	parallel translation [HAG14, (2.3.1)]
		Use intrinsic approach [Hua13, §9.5]	no
		Compute β_i in [HGA14, Step 6 of Algorithm 1]	no
		Use house holder reflection	no
		Satisfy the locking condition	yes
3	ChooseSphereParamsSet2()	Metric	Euclidean
		Retraction	qf retraction [AMS08, (4.8)]
		Vector transport	parallel translation [HAG14, (2.3.1)]
		Use intrinsic approach [Hua13, §9.5]	no
		Compute β_i in [HGA14, Step 6 of Algorithm 1]	no
		Use house holder reflection	no
		Satisfy the locking condition	no
4	ChooseSphereParamsSet3()	Metric	Euclidean
		Retraction	qf retraction [AMS08, (4.8)]
		Vector transport	parallel translation [HAG14, (2.3.1)]
		Use intrinsic approach [Hua13, §9.5]	no
		Compute β_i in [HGA14, Step 6 of Algorithm 1]	yes
		Use house holder reflection	no
		Satisfy the locking condition	yes

Table 28: $\mathbb{L}^2$ unit sphere

member function	parameters	values
initialized in C++ constructor	Metric	trapezoidal rule
	Retraction	exponential mapping
	Vector transport	parallel translation
	Use intrinsic approach	no
	Compute β_i in [HGA14, Step 6 of Algorithm 1]	no
	Use house holder reflection	no
	Satisfy the locking condition	yes

Table 29: 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 [HAG14, (2.3.1)]
		Use intrinsic approach [Hua13, §9.5]	yes
		Compute β_i in [HGA14, Step 6 of Algorithm 1]	no
		Use house holder reflection	no
		Satisfy the locking condition	no
2	ChooseStieParamsSet2()	Metric	Euclidean
		Retraction	constructed retraction [HGA14, (7.3)] (equivalent to exponential mapping)
		Vector transport	by parallelization [HAG14, (2.3.1)] (equivalent to parallel translation)
		Use intrinsic approach [Hua13, §9.5]	yes
		Compute β_i in [HGA14, Step 6 of Algorithm 1]	no
		Use house holder reflection	no
		Satisfy the locking condition	yes

Table 30: 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 [HAG14, (2.3.1)]
		Use intrinsic approach [Hua13, §9.5]	yes
		Compute β_i in [HGA14, 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 [HAG14, (2.3.1)]
		Use intrinsic approach [Hua13, §9.5]	no
		Compute β_i in [HGA14, 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 [HAG14, (2.3.1)]
		Use intrinsic approach [Hua13, §9.5]	no
		Compute β_i in [HGA14, 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 [HAG14, (2.3.1)]
		Use intrinsic approach [Hua13, §9.5]	no
		Compute β_i in [HGA14, Step 6 of Algorithm 1]	yes
		Use house holder reflection	no
		Satisfy the locking condition	yes

Table 31: low rank manifold using representation: the compact Stiefel manifold $\mathrm{St}(r,n)$ $\times$ *the Euclidean space* $\mathbb{R}^{r\times r}$ $\times$ *the compact Stiefel manifold* $\mathrm{St}(r,m)$

member function	components	parameters	values
initialized in constructor	Euclidean	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 [HGA14, Step 6 of Algorithm 1]	no
		Use house holder reflection	no
		Satisfy the locking condition	yes
	Stiefel	Metric	Euclidean
		Retraction	qf retraction [AMS08, (4.8)]
		Vector transport	by parallelization [HAG14, (2.3.1)]
		Use intrinsic approach [Hua13, §9.5]	yes
		Compute β_i in [HGA14, 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.

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

[BM] N. Boumal and B. Mishra. The Manopt toolbox.

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

[HAG14] W. Huang, P.-A. Absil, and K. A. Gallivan. A Riemannian symmetric rank-one trust-region method. *Mathematical Programming*, February 2014. doi:10.1007/s10107-014-0765-1.

[HGA14] W. Huang, K. A. Gallivan, and P.-A. Absil. A Broyden class of quasi-Newton methods for Riemannian optimization. *Submitted for publication*, 2014.

[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.

[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.

[SI13] H. Sato and T. Iwai. A new, globally convergent riemannian conjugate gradient method. *Optimization*, page 22, February 2013. 1302.0125.