User Manual 4.6 Optimization

De Wiki
Aller à : navigation, rechercher
User Manual 4.6 Optimization

Introduction

Scope

This section describes PATRIUS optimization features.

It will focus on the *JOptimizer* functionalities, which provides solvers for general convex optimization problems.

Javadoc

The optimization classes are available in the package `fr.cnes.sirius.patrius.math.optim`.

Library	Javadoc
Patrius	Package fr.cnes.sirius.patrius.math.optim

Links

None as of now.

Useful Documents

None as of now.

Package Overview

The optimization functionalities for joptimizer are organized in the following packages:

- `fr.cnes.sirius.patrius.math.optim.joptimizer.algebra` compounds the classes with the algebra functionalities.
- `fr.cnes.sirius.patrius.math.optim.joptimizer.functions` compounds the classes with the optimization functions.
- `fr.cnes.sirius.patrius.math.optim.joptimizer.optimizers` compounds the classes with the optimizers.
- `fr.cnes.sirius.patrius.math.optim.joptimizer.solvers` compounds the classes with the solvers.
- `fr.cnes.sirius.patrius.math.optim.joptimizer.util` compounds the utility classes.

Features Description

Features descritpion for the joptimizer package.

Optimizers

The `JOptimizer` class implements the convex optimizer (see "S.Boyd and L.Vandenberghe, Convex Optimization").

The algorithm selection is implemented as a Chain of Responsibility pattern, and this class is the client of the chain.

The different methods implemented to solve the convex optimization problem are:

- Interior point methods
  - `PrimalDualMethod` : primal-dual interior-point method.
  - `LPPrimalDualMethod` : primal-dual interior-point method for linear problems.
  - `BarrierMethod`
- Quality constrained minimization
  - `NewtonLEConstrainedFSP` : linear equality constrained newton optimizer, with a feasible starting point.
  - `NewtonLEConstrainedISP` : linear equality constrained newton optimizer, with an infeasible starting point.
- Unconstrained minimization
  - `NewtonUnconstrained` : unconstrained newton optimizer.

Optimization problem

The `OptimizationRequest` class has all the setting field's necessaires to define an optimization problem.

The `LPOptimizationRequest` is an extension of this class for linear optimization problems.

The general form of a linear problem is (1):

```
min(c) s.t.
A.x = b
lb <= x <= ub
```

The `OptimizationResponse` is the class with the getters and setters to set and get the response after the optimization. The `LPOptimizationResponse` is the extended class applied to linear problems.

Standard converter

The `LPStandardConverter` converts a general linear problem stated in the form (2):

```
min(c) s.t.
G.x < h
A.x = b
lb <= x <= ub
```

to the (strictly)standard form:

```
min(c) s.t.
A.x = b
x >= 0
```

or to the (quasi)standard form (1).

Presolver

The `LPPresolver` implements a presolver for a linear problem in the form (1).

It applies a set of techniques to the linear programming problem before a linear programming solver solves it. This set of techniques aims at reducing the size of the LP problem by eliminating redundant constraints and variables and identifying possible infeasibility and unboundedness of the problem.

Solvers

The `AbstractKKTSolver` implements a solver for the KKT system:

```
H.v + [A]T.w = -g
A.v = -h
```

where H is a square and symmetric matrix.

The following classes are an extension of `AbstractKKTSolver`:

- `AugmentedKKTSolver` (for singular H)
- `BasicKKTSolver`
- `UpperDiagonalHKKTSolver` (for upper diagonal H)

Functions

Different functions are implemented, all of them twice differentiable.

- Linear functions

The `LinearMultivariateRealFunction` represents a function in the form of:

```
f(x) = q.x + r
```

- Quadratic functions

The `QuadraticMultivariateRealFunction` represents a function in the form of:

```
f(x) := 1/2 x.P.x + q.x + r
```

where x, q ∈ R^n, P is a symmetric nXn matrix and r ∈ R.

With the extended `PSDQuadraticMultivariateRealFunction` and `PDQuadraticMultivariateRealFunction` classes for P symmetric and positive semi-definite, and P symmetric and positive definite, respectively.

- Barrier functions

The `LogarithmicBarrier` is the default barrier function for the barrier method algorithm.

If $f_i(x)$ are the inequalities of the problem, then the function:

```
Φ(x) = − ∑_i (log(−fi(x)))
```

Algebra

Factorization

The `CholeskyFactorization` implements the Cholesky L . L[T] factorization and inverse for symmetric and positive matrix:

```
Q = L.L[T]
```

with L lower-triangular.

Rescaler

The `Matrix1NornRescaler` calculates the matrix rescaling factors, so that the 1-norm of each row and each column of the scaled matrix asymptotically converges to one.

Getting Started

Example 1

Example of a linear problem optimized by the primal-dual interior-point method.

The problem is:

```
min(-100x + y) s.t.
x - y = 0
0 <= x <= 1
0 <= y <= 1
```

First, the definition of the variables:

```
final double[] c = new double[] { -100, 1 }
final double[][] a = new double[][] { { 1, -1 } }
final double[] b = new double[] { 0 }
final double[] lb = new double[] { 0, 0 }
final double[] ub = new double[] { 1, 1 }
```

Definition of the optimization problem by setting the variables:

```
final LPOptimizationRequest or = new LPOptimizationRequest()
or.setC(c)
or.setA(a)
or.setB(b)
or.setLb(lb)
or.setUb(ub)
```

Additional parameters (tolerance, check the solution accuracy, etc) can also be setted:

```
or.setCheckKKTSolutionAccuracy(true)
or.setToleranceFeas(1.E-7)
or.setTolerance(1.E-7)
or.setDumpProblem(true)
or.setRescalingDisabled(true)
```

Definition of the optimizer and setting the optimization problem:

```
LPPrimalDualMethod opt = new LPPrimalDualMethod()
opt.setLPOptimizationRequest(or)
```

Optimization and check that it has not failed:

```
final int returnCode = opt.optimize()
if (returnCode == OptimizationResponse.FAILED) {
    fail()
}
```

Recuperate the response and the solution:

```
final LPOptimizationResponse response = opt.getLPOptimizationResponse()
final double[] sol = response.getSolution()
```

Validation:

```
final RealVector cVector = new ArrayRealVector(c)
final RealVector solVector = new ArrayRealVector(sol)
final double value = cVector.dotProduct(solVector)
assertEquals(2, sol.length)
assertEquals(1, sol[0], or.getTolerance())
assertEquals(1, sol[1], or.getTolerance())
assertEquals(-99, value, or.getTolerance())
```

Example 2

Example of the optimization of a linear objective function with quadratic constraints.

The problem is:

```
min(-e.x) s.t.
1/2 x.P.x < v
x + y + z = 1
x > 0
y > 0
z > 0
```

Definition of the linear objective function:

```
final double[] e = { -0.018, -0.025, -0.01 }
final LinearMultivariateRealFunction objectiveFunction = new
LinearMultivariateRealFunction(e, 0)
```

Definition of the quadratic and linear constraints:

```
final double[][] p = { { 1.68, 0.34, 0.38 }, { 0.34, 3.09, -1.59 }, { 0.38,
-1.59, 1.54 } }
final double v = 0.3
final PDQuadraticMultivariateRealFunction qc0 = new
PDQuadraticMultivariateRealFunction(p, null,-v)
final LinearMultivariateRealFunction lc0 = new
LinearMultivariateRealFunction(new double[] { -1, 0, 0 }, 0)
final LinearMultivariateRealFunction lc1 = new
LinearMultivariateRealFunction(new double[] { 0, -1, 0 }, 0)
final LinearMultivariateRealFunction lc2 = new
LinearMultivariateRealFunction(new double[] { 0, 0, -1 }, 0)
final ConvexMultivariateRealFunction[] constraints = new
ConvexMultivariateRealFunction[] { qc0, lc0, lc1, lc2 }
```

Definition of the equality constraint:

```
final double[][] a = {{ 1, 1, 1 }}
final double[] b = { 1 }
```

Definition of the optimization problem and setting the parameters:

```
final OptimizationRequest or = new OptimizationRequest()
or.setF0(objectiveFunction)
or.setFi(constraints)
or.setA(a)
or.setB(b)
or.setToleranceFeas(1.e-6)  // additional parameter
```

Definition of the optimizer and setting the optimization problem:

```
final JOptimizer opt = new JOptimizer()
opt.setLPOptimizationRequest(or)
```

Optimization and check that it has not failed:

```
final int returnCode = opt.optimize()
if (returnCode == OptimizationResponse.FAILED) {
    fail()
}
```

Recuperate the response and the solution:

```
final LPOptimizationResponse response = opt.getLPOptimizationResponse()
final double[] sol = response.getSolution()
```

Validation:

```
assertEquals(1., sol[0] + sol[1] + sol[2], 1.e-6)
assertTrue(sol[0] > 0)
assertTrue(sol[1] > 0)
assertTrue(sol[2] > 0)
final RealVector xVector = MatrixUtils.createRealVector(sol)
final RealMatrix pMatrix = MatrixUtils.createRealMatrix(p)
final double xPx = xVector.dotProduct(pMatrix.operate(xVector))
assertTrue(0.5 * xPx < v)
```

Contents

Interfaces

The interfaces related to the joptimizer are listed here :

Interface	Summary	Javadoc
BarrierFunction	Interface for the barrier function used by a given barrier optimization method.	...
ConvexMultivariateRealFunction	Interface for convex multivariate real functions.	...
MatrixRescaler	An interface to classes that implement an algorithm to rescale matrices.	...
StrictlyConvexMultivariateRealFunction	Interface for striclty convex multivariate real functions.	...
TwiceDifferentiableMultivariateRealFunction	Interface for twice-differentiable multivariate functions.	...

Classes

The classes related to the joptimizer are listed here :

Class	Summary	Javadoc
AlgebraUtils	Algebraic utility operations	...
CholeskyFactorization	Implements the Cholesky L.L[T] factorization and inverse for symmetric and positive matrix.	...
Matrix1NornRescaler	Calculates the matrix rescaling factors so that the 1-norm of each row and each column of the scaled matrix asymptotically converges to one.	...
FunctionsUtils	Utility class for optimization function building.	...
LinearMultivariateRealFunction	Represents a function f(x) = q.x + r.	...
LogarithmicBarrier	Default barrier function for the barrier method algorithm.	...
PDQuadraticMultivariateRealFunction	Extends the class QuadraticMultivariateRealFunction with P symmetric and positive definite.	...
PSDQuadraticMultivariateRealFunction	Extends the class QuadraticMultivariateRealFunction with P symmetric and positive semi-definite.	...
QuadraticMultivariateRealFunction	Represents a quadratic multivariate function in the form of f(x):= 1/2 x.P.x + q.x + r.	...
AbstractLPOptimizationRequestHandler	Abstract class for Linear Problem Optimization Request Handler.	...
BarrierMethod	Implements the Barrier Method.	...
BasicPhaseIBM	Implements the Basic Phase I Method as a Barried Method.	...
BasicPhaseILPPDM	Implements the Basic Phase I Method form LP problems as a Primal-Dual Method.	...
BasicPhaseIPDM	Implements the Basic Phase I Method as a Primal-Dual Method.	...
JOptimizer	Implements the convex optimizer.	...
LPPresolver	Presolver for a linear problem.	...
LPPrimalDualMethod	Implements the Primal-dual interior-point method for linear problems.	...
LPStandardConverter	Converts a general LP problem into a strictly standard or quasi standard form.	...
NewtonLEConstrainedFSP	Linear equality constrained newton optimizer, with feasible starting point.	...
NewtonLEConstrainedISP	Linear equality constrained newton optimizer, with infeasible starting point.	...
NewtonUnconstrained	Unconstrained newton optimizer.	...
OptimizationRequest	Implements an optimization problem.	...
OptimizationRequestHandler	Generic class for optimization process.	...
OptimizationResponse	Optimization process output: stores the solution as well as an exit code.	...
PrimalDualMethod	Implements a primal-dual interior-point method.	...
AbstractKKTSolver	Abstract class for solving KKT systems.	...

AugmentedKKTSolver	Extension of AbstractKKTSolver for singular matrix.	...
BasicKKTSolver	Extension of AbstractKKTSolver for the basic solver.	...
UpperDiagonalHKKTSolver	Extends the class AbstractKKTSolver for upper diagonal matrix.	...
ArrayUtils	Class offering operations on arrays, primitive arrays (like int[]) and primitive wrapper arrays (like Integer[]).	...
MutableInt	A mutable (int) wrapper.	...
Utils	Utility class.	...

PATRIUS

- Welcome

Evolutions

- Main differences between V4.15 and V4.14
- Main differences between V4.14 and V4.13
- Main differences between V4.13 and V4.12
- Main differences between V4.12 and V4.11
- Main differences between V4.11 and V4.10
- Main differences between V4.10 and V4.9
- Main differences between V4.9 and V4.8
- Main differences between V4.8 and V4.7
- Main differences between V4.7 and V4.6.1
- Main differences between V4.6.1 and V4.5.1
- Main differences between V4.5.1 and V4.4
- Main differences between V4.4 and V4.3
- Main differences between V4.3 and V4.2
- Main differences between V4.2 and V4.1.1
- Main differences between V4.1.1 and V4.1
- Main differences between V4.1 and V4.0
- Main differences between V4.0 and V3.4.1

User Manual

- User Manual 4.15
- User Manual 4.14
- User Manual 4.13
- User Manual 4.12
- User Manual 4.11
- User Manual 4.10
- User Manual 4.9
- User Manual 4.8
- User Manual 4.7
- User Manual 4.6.1
- User Manual 4.5.1
- User Manual 4.4
- User Manual 4.3
- User Manual 4.2
- User Manual 4.1
- User Manual 4.0
- User Manual 3.4.1
- User Manual 3.3

Tutorials

- Tutorials 4.15

- Tutorials 4.14
- Tutorials 4.13.5
- Tutorials 4.12.1
- Tutorials 4.8.1
- Tutorials 4.5.1
- Tutorials 4.4
- Tutorials 4.1
- Tutorials 4.0

Links

- CNES freeware server

Navigation

- Accueil
- Modifications récentes
- Page au hasard
- Aide

Outils

- Pages liées
- Suivi des pages liées
- Pages spéciales
- Adresse de cette version
- Information sur la page
- Citer cette page