bbowda

Generated by Doxygen 1.9.1

Chapter 1

Main Page

1.1 Introduction

BBOWDA is an algorithm and a reference implementation to solve optimization problems where:

- both the objective function and the constraints may be black box functions,
- we do not have any gradient or Hessian information for those black box functions,
- the functions are assumed to be expensive to compute, thus the number of function evaluations shall be kept as small as possible,

using methods from data analysis:

- covariance models,
- Gaussian mixture models (GMMs) and the Expectation-Maximization (EM) iteration, and
- ratio-reject (outlier rejection algorithm by Tax and Duin).

The algorithm is a so-called incomplete global optimizer, i.e. it attempts to find a global solution for the optimization problem, but is unable to guarantee globality. In fact, it cannot even guarantee always finding a local optimum, due to the lack of gradients and any sort of global information. Despite this lack of guarantees, the algorithm performs well in practice.

BBOWDA is a surrogate model method, hence it depends on third-party solvers for Non-Linear (optimization) Programs (NLPs) and Linear (optimization) Programs (LPs).

For NLPs, BBOWDA currently supports any of:

- NLopt SLSQP (MIT and 3-clause BSD licenses),
- Ipopt from COIN-OR (Eclipse Public License version 2.0),
- DONLP2 (donlp2_intv_dyn) by Prof. Peter Spellucci (free for research only),
- DONLP3 (reentrant C++ version of DONLP2, see above, from COCONUT).

Note that DONLP2 is NOT reentrant. Use another solver if you require BBOWDA to be reentrant. All other currently supported solvers (including DONLP3) are reentrant.

For LPs, BBOWDA currently requires lp_solve (GNU LGPL version 2.1 or later).

The implementation is licensed under the GNU General Public License, version 3 or later, with special exceptions allowing to link with the third-party optimizers used. See licenses.txt for details.

1.2 API Documentation

This documentation documents the public C and C++ API of BBOWDA.

1.2.1 C API (Procedural API)

See bbowda.h. The main entry point is bbowda.

1.2.2 C++ API (Object-oriented API)

See bbowdapp/bbowdapp.hh. The main entry point is Bbowda::OptimizationProblem::solve.

Chapter 2

Namespace Index

2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 4

Class Index

4.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

Chapter 6

Namespace Documentation

6.1 Bbowda Namespace Reference

Classes

- class SolverParameters

  Solver parameters for the BBOWDA algorithm.
- class OptimizationProblem

  Definition of a black box optimization problem.

Chapter 7

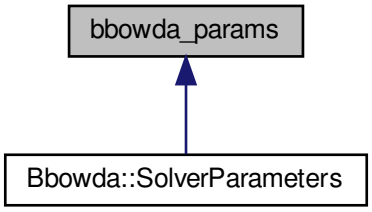
Class Documentation

7.1 bbowda_params Struct Reference

Solver parameters for the BBOWDA algorithm.

`#include <bbowda.h>`

Inheritance diagram for bbowda_params:



Public Attributes

- size_t maxpts

  Maximum number of points to evaluate.
- double optimum_tol

  Tolerance for optimum feasibility (infinity norm).
- int global_search_ignores_eq_constraints

  Whether to ignore implicit equality constraints for global search.
- double estimate_constraint_tol

  Tolerance for the constraints estimating the implicit equality constraints during global search.

7.1.1 Detailed Description

Solver parameters for the BBOWDA algorithm.

Parameters allowing to tune how the BBOWDA algorithm operates.

7.1.2 Member Data Documentation

7.1.2.1 estimate_constraint_tol

```
double bbowda_params::estimate_constraint_tol
```

Tolerance for the constraints estimating the implicit equality constraints during global search.

If global_search_ignores_eq_constraints is false (zero), the global search attempts to estimate global enclosures for the implicit equality constraints. But those enclosures are estimated approximations and not guaranteed to hold exactly. This tolerance specifies by how much the bounds for the estimated enclosures should be relaxed to account for that.

This parameter has no effect if the problem does not include any implicit equality constraints or if global_search_ignores_eq_constraints is true (non-zero).

7.1.2.2 global_search_ignores_eq_constraints

```
int bbowda_params::global_search_ignores_eq_constraints
```

Whether to ignore implicit equality constraints for global search.

A true (non-zero) value means that implicit equality constraints will be ignored by the global search. Hence, it will suggest evaluation points to fill any gaps in the search space even if they are nowhere near feasible for the implicit equality constraints.

The false (zero) value means that implicit equality constraints will be honored by the global search. The global search will compute estimates that attempt to globally enclose the implicit equality constraints with the help of an external LP solver, helping to suggest only points that are expected to be approximately feasible.

This parameter has no effect if the problem does not include any implicit equality constraints.

7.1.2.3 maxpts

```
size_t bbowda_params::maxpts
```

Maximum number of points to evaluate.

Currently, the algorithm will always run until exactly this many points are evaluated, because no other stopping criteria are implemented yet. In the presence of implicit equality constraints, it may then evaluate one more point for the final extrapolation attempt.

7.1.2.4 optimum_tol

```
double bbowda_params::optimum_tol
```

Tolerance for optimum feasibility (infinity norm).

A point will be considered feasible, and thus a valid candidate for the optimum, if the bound constraints for *x* are satisfied exactly, and if none of the other constraints is violated by more than *optimum_tol*, i.e., if the componentwise inequalities *Flow* - *optimum_tol* <= F1(*x*) <= *Fup* + *optimum_tol* and - *optimum_tol* < F2(*x*) < *optimum_tol* hold. In vector terms, this means that the infinity norm of the constraint violation must be less than *optimum_tol*.

The documentation for this struct was generated from the following file:

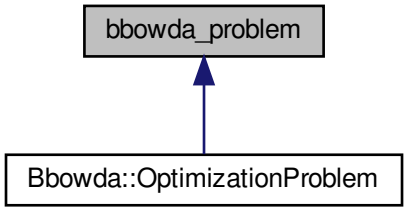
- bbowda.h

7.2 bbowda_problem Struct Reference

Constants that, together with the black box function, define a black box optimization problem.

```
#include <bbowda.h>
```

Inheritance diagram for bbowda_problem:



Public Attributes

- int dimx
  Number of input variables.
- int dimy
  Number of explicit equality constraints.
- int dimy_eq
  Number of implicit equality constraints.
- int numinitpts
  Number of user-specified starting points.
- const double * c
  Coefficients in the (linear) objective function.
- const double * xlow
  Lower bounds for input variables.

- const double * xup

  Upper bounds for input variables.
- const double * Flow

  Lower bounds for explicit equality constraints.
- const double * Fup

  Upper bounds for explicit equality constraints.
- const double * initpts_p

  User-provided starting points.

7.2.1 Detailed Description

Constants that, together with the black box function, define a black box optimization problem.

The problem is assumed to be of the form:

```
min cT (x ; y)
s.t. y = F1(x) [explicit equality constraints]
     F2(x) = 0 [implicit equality constraints]
     xlow <= x <= xup
     Flow <= y <= Fup
```

7.2.2 Member Data Documentation

7.2.2.1 c

```
const double* bbowda_problem::c
```

Coefficients in the (linear) objective function.

The coefficient vector *c*. Must have dimension dimx + dimy. First list all coefficients of *x* in order, then all coefficients of *y* = F1(*x*) in order.

7.2.2.2 dimx

```
int bbowda_problem::dimx
```

Number of input variables.

The vector dimension of *x*.

7.2.2.3 dimy

```
int bbowda_problem::dimy
```

Number of explicit equality constraints.

The vector dimension of *y* = F1(*x*). Also known as the number of interval inequality constraints if the coefficients of *y* in c are 0.

7.2.2.4 dimy_eq

```
int bbowda_problem::dimy_eq
```

Number of implicit equality constraints.

Vector dimension of F2(x).

7.2.2.5 Flow

```
const double* bbowda_problem::Flow
```

Lower bounds for explicit equality constraints.

Lower bounds for y = F1(x). Must have dimension dimy.

Note

No bounds need to be specified for the implicit equality constraints F2(x) because those bounds are by definition always 0.

7.2.2.6 Fup

```
const double* bbowda_problem::Fup
```

Upper bounds for explicit equality constraints.

Upper bounds for y = F1(x). Must have dimension dimy.

Note

No bounds need to be specified for the implicit equality constraints F2(x) because those bounds are by definition always 0.

7.2.2.7 initpts_p

```
const double* bbowda_problem::initpts_p
```

User-provided starting points.

Must be a contiguous matrix of dimension numinitpts $*$ dimx (in row-major order, i.e., first all components of the first starting point, then the second one, etc.). If numinitpts is 0, this is just an empty vector (and can be NULL). Otherwise, it can be a pointer of the form `&initpts[0][0]`, where *initpts* is defined as `double initpts[numinitpts][dimx]`. (In fact, that will work in practice even if numinitpts is 0 because C allows pointers to the end of an array.)

7.2.2.8 numinitpts

```
int bbowda_problem::numinitpts
```

Number of user-specified starting points.

Can be 0. If no or not enough starting points are provided by the user, BBOWDA will automatically sample some starting points within the bounds.

7.2.2.9 xlow

```
const double* bbowda_problem::xlow
```

Lower bounds for input variables.

Lower bounds for *x*. Must have dimension dimx.

7.2.2.10 xup

```
const double* bbowda_problem::xup
```

Upper bounds for input variables.

Upper bounds for *x*. Must have dimension dimx.

The documentation for this struct was generated from the following file:

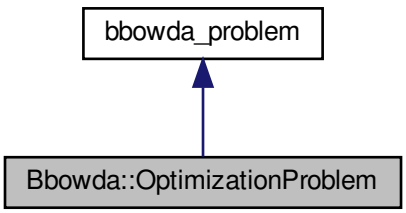
- bbowda.h

7.3 Bbowda::OptimizationProblem Class Reference

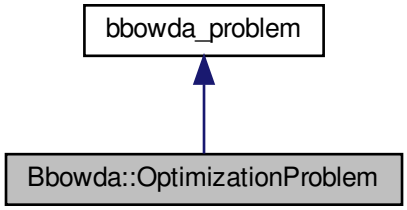
Definition of a black box optimization problem.

```
#include <bbowdapp.hh>
```

Inheritance diagram for Bbowda::OptimizationProblem:



Collaboration diagram for Bbowda::OptimizationProblem:



Public Member Functions

- OptimizationProblem (int dimx, int dimy, int dimy_eq, int numinitpts, const double ∗c, const double ∗xlow, const double ∗xup, const double ∗Flow, const double ∗Fup, const double ∗initpts_p, bool copyVectors=true)

  Main constructor.
- virtual ∼OptimizationProblem ()

  Virtual destructor.
- void solve (const SolverParameters &params)

  Main entry point of the BBOWDA C++ API.

Protected Member Functions

- virtual void evaluateF (const double ∗x, double ∗F)=0

  Pure virtual callback evaluating the black box function.

Additional Inherited Members

7.3.1 Detailed Description

Definition of a black box optimization problem.

This class represents both the constants (inherited from bbowda_problem) and the black box function (in the form of the pure virtual method evaluateF) that together represent the black box optimization problem. Subclass this class to implement your optimization problem.

The problem is assumed to be of the form:

```
min cT (x ; y)
s.t. y = F1(x) [explicit equality constraints]
     F2(x) = 0 [implicit equality constraints]
     xlow <= x <= xup
     Flow <= y <= Fup
```

7.3.2 Constructor & Destructor Documentation

7.3.2.1 OptimizationProblem()

```
Bbowda::OptimizationProblem::OptimizationProblem (
            int dimx,
            int dimy,
            int dimy_eq,
            int numinitpts,
            const double * c,
            const double * xlow,
            const double * xup,
            const double * Flow,
            const double * Fup,
            const double * initpts_p,
            bool copyVectors = true )
```

Main constructor.

Parameters

dimx	Number of input variables. See dimx.
dimy	Number of explicit equality constraints. See dimy.
dimy_eq	Number of implicit equality constraints. See dimy_eq.
numinitpts	Number of user-specified starting points. See numinitpts.
c	Coefficients in the (linear) objective function. See c.
xlow	Lower bounds for input variables. See xlow.
xup	Upper bounds for input variables. See xup.
Flow	Lower bounds for explicit equality constraints. See Flow.
Fup	Upper bounds for explicit equality constraints. See Fup.
initpts_p	User-provided starting points. See initpts_p.
copyVectors	Whether to copy the vectors.

7.3.2.2 ~OptimizationProblem()

```
virtual Bbowda::OptimizationProblem::~OptimizationProblem ( )  [virtual]
```

Virtual destructor.

7.3.3 Member Function Documentation

7.3.3.1 evaluateF()

```
virtual void Bbowda::OptimizationProblem::evaluateF (
            const double * x,
            double * F )  [protected], [pure virtual]
```

Pure virtual callback evaluating the black box function.

Callback evaluating the black box function (or obtaining the evaluation result from an external source) at the point *x* (of dimension dimx) and writing the result to *F* (of dimension dimy + dimy_eq). Must be implemented by the user through subclassing.

Note

There is no *user_data* pointer because any user data can and should be included in the user-provided subclass.

7.3.3.2 solve()

```
void Bbowda::OptimizationProblem::solve (
            const SolverParameters & params )
```

Main entry point of the BBOWDA C++ API.

Runs the BBOWDA algorithm on this problem with the given parameters.

Object-oriented wrapper around the C API entry point bbowda.

Parameters

params	The solver parameters for the BBOWDA algorithm.

The documentation for this class was generated from the following file:

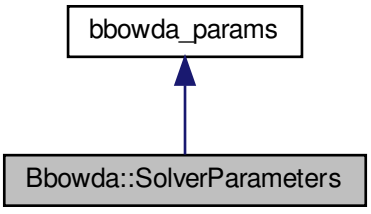
- bbowdapp/bbowdapp.hh

7.4 Bbowda::SolverParameters Class Reference

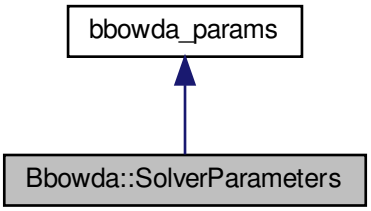
Solver parameters for the BBOWDA algorithm.

```
#include <bbowdapp.hh>
```

Inheritance diagram for Bbowda::SolverParameters:



Collaboration diagram for Bbowda::SolverParameters:



Public Member Functions

- SolverParameters (size_t maxpts, double optimum_tol, bool global_search_ignores_eq_constraints, double estimate_constraint_tol)

  Constructor.

Additional Inherited Members

7.4.1 Detailed Description

Solver parameters for the BBOWDA algorithm.

Parameters allowing to tune how the BBOWDA algorithm operates.

7.4.2 Constructor & Destructor Documentation

7.4.2.1 SolverParameters()

```
Bbowda::SolverParameters::SolverParameters (
            size_t maxpts,
            double optimum_tol,
            bool global_search_ignores_eq_constraints,
            double estimate_constraint_tol )
```

Constructor.

Parameters

maxpts	Maximum number of points to evaluate. See maxpts.
optimum_tol	Tolerance for optimum feasibility (infinity norm). See optimum_tol.
global_search_ignores_eq_constraints	Whether to ignore implicit equality constraints for global search. See global_search_ignores_eq_constraints.
estimate_constraint_tol	Tolerance for the constraints estimating the implicit equality constraints during global search. See estimate_constraint_tol.

The documentation for this class was generated from the following file:

- bbowdapp/bbowdapp.hh

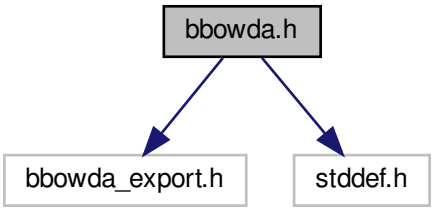
Chapter 8

File Documentation

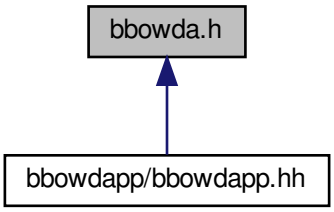
8.1 bbowda.h File Reference

```
#include "bbowda_export.h"
#include <stddef.h>
```

Include dependency graph for bbowda.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct bbowda_problem

  Constants that, together with the black box function, define a black box optimization problem.
- struct bbowda_params

  Solver parameters for the BBOWDA algorithm.

Functions

- BBOWDA_EXPORT void bbowda (const struct bbowda_problem ∗problem, const struct bbowda_params ∗params, void(∗evaluate_F)(const struct bbowda_problem ∗problem, const double ∗x, double ∗F, void ∗user_data), void ∗eval_user_data)

  Main entry point of the BBOWDA C API.

8.1.1 Function Documentation

8.1.1.1 bbowda()

```
BBOWDA_EXPORT void bbowda (
            const struct bbowda_problem * problem,
            const struct bbowda_params * params,
            void(*)(const struct bbowda_problem *problem, const double *x, double *F, void
*user_data) evaluate_F,
            void * eval_user_data )
```

Main entry point of the BBOWDA C API.

Runs the BBOWDA algorithm on the given problem with the given parameters.

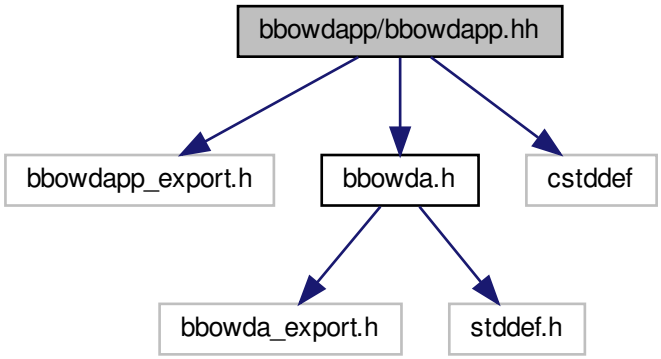
Parameters

problem	The constants that, together with the black box function, define the problem.
params	The solver parameters for the BBOWDA algorithm.
evaluate_F	Function pointer to the callback evaluating the black box function (or obtaining the evaluation result from an external source) at the point *x* (of dimension bbowda_problem::dimx) and writing the result to *F* (of dimension bbowda_problem::dimy + bbowda_problem::dimy_eq).
eval_user_data	Pointer that will be passed unchanged as *user_data* to all invocations of *evaluate_F*.

8.2 bbowdapp/bbowdapp.hh File Reference

```
#include "bbowdapp_export.h"
#include "bbowda.h"
#include <cstddef>
```

Include dependency graph for bbowdapp.hh:



Classes

- class Bbowda::SolverParameters

  Solver parameters for the BBOWDA algorithm.
- class Bbowda::OptimizationProblem

  Definition of a black box optimization problem.

Namespaces

- Bbowda

Index