BBOWDA Java API

Kevin Kofler <kofler@dagopt.com>

Package
com.dagopt.bbowda

com.dagopt.bbowda
Class Bbowda

```
java.lang.Object
   |
   +-com.dagopt.bbowda.Bbowda
```

public class **Bbowda**
extends java.lang.Object

Constructor Summary

public	Bbowda()

Method Summary

static void	srand(long seed) Set the pseudorandom number generator (PRNG) seed of the C library. BBOWDA uses the C library to generate pseudorandom numbers.

Methods inherited from class `java.lang.Object`

`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructors

Bbowda

`public` **`Bbowda`**`()`

Methods

srand

`public static void` **`srand`**`(long seed)`

Set the pseudorandom number generator (PRNG) seed of the C library.

BBOWDA uses the C library to generate pseudorandom numbers. Therefore, if reproducible results are needed, it is necessary to set this seed.

Parameters:

`seed` - The random seed to set.

com.dagopt.bbowda
Class BbowdaJNI

```
java.lang.Object
   |
   +-com.dagopt.bbowda.BbowdaJNI
```

public class **BbowdaJNI**
extends java.lang.Object

Constructor Summary

public	BbowdaJNI()

Method Summary

static native void	delete_OptimizationProblem(long jarg1)
static native void	delete_SolverParameters(long jarg1)
static native long	new_OptimizationProblem(int jarg1, int jarg2, int jarg3, int jarg4, double[] jarg5, double[] jarg6, double[] jarg7, double[] jarg8, double[] jarg9, double[] jarg10)
static native long	new_SolverParameters(long jarg1, double jarg2, boolean jarg3, double jarg4)
static native double[]	OptimizationProblem_c_get(long jarg1, OptimizationProblem jarg1_)
static native void	OptimizationProblem_change_ownership(OptimizationProblem obj, long cptr, boolean take_or_release)
static native int	OptimizationProblem_dimx_get(long jarg1, OptimizationProblem jarg1_)
static native int	OptimizationProblem_dimy_get(long jarg1, OptimizationProblem jarg1_)
static native int	OptimizationProblem_dimyEq_get(long jarg1, OptimizationProblem jarg1_)
static native void	OptimizationProblem_director_connect(OptimizationProblem obj, long cptr, boolean mem_own, boolean weak_global)
static native void	OptimizationProblem_evaluateF(long jarg1, OptimizationProblem jarg1_, double[] jarg2, double[] jarg3)
static native double[]	OptimizationProblem_flow_get(long jarg1, OptimizationProblem jarg1_)
static native double[]	OptimizationProblem_fup_get(long jarg1, OptimizationProblem jarg1_)

static native double[]	OptimizationProblem_initptsvec_get(long jarg1, OptimizationProblem jarg1_)
static native int	OptimizationProblem_numinitpts_get(long jarg1, OptimizationProblem jarg1_)
static native void	OptimizationProblem_solve(long jarg1, OptimizationProblem jarg1_, long jarg2, SolverParameters jarg2_)
static native double[]	OptimizationProblem_xlow_get(long jarg1, OptimizationProblem jarg1_)
static native double[]	OptimizationProblem_xup_get(long jarg1, OptimizationProblem jarg1_)
static native double	SolverParameters_estimateConstraintTol_get(long jarg1, SolverParameters jarg1_)
static native void	SolverParameters_estimateConstraintTol_set(long jarg1, SolverParameters jarg1_, double jarg2)
static native boolean	SolverParameters_globalSearchIgnoresEqConstraints_get(long jarg1, SolverParameters jarg1_)
static native void	SolverParameters_globalSearchIgnoresEqConstraints_set(long jarg1, SolverParameters jarg1_, boolean jarg2)
static native long	SolverParameters_maxpts_get(long jarg1, SolverParameters jarg1_)
static native void	SolverParameters_maxpts_set(long jarg1, SolverParameters jarg1_, long jarg2)
static native double	SolverParameters_optimumTol_get(long jarg1, SolverParameters jarg1_)
static native void	SolverParameters_optimumTol_set(long jarg1, SolverParameters jarg1_, double jarg2)
static native void	srand(long jarg1)
static void	SwigDirector_OptimizationProblem_evaluateF(OptimizationProblem jself, double[] x, double[] F)

Methods inherited from class `java.lang.Object`

`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructors

BbowdaJNI

`public` **`BbowdaJNI`**`()`

Methods

new_SolverParameters

```
public final static native long new_SolverParameters(long jarg1,
        double jarg2,
        boolean jarg3,
        double jarg4)
```

SolverParameters_maxpts_set

```
public final static native void SolverParameters_maxpts_set(long jarg1,
        SolverParameters jarg1_,
        long jarg2)
```

SolverParameters_maxpts_get

```
public final static native long SolverParameters_maxpts_get(long jarg1,
        SolverParameters jarg1_)
```

SolverParameters_optimumTol_set

```
public final static native void SolverParameters_optimumTol_set(long jarg1,
        SolverParameters jarg1_,
        double jarg2)
```

SolverParameters_optimumTol_get

```
public final static native double SolverParameters_optimumTol_get(long jarg1,
        SolverParameters jarg1_)
```

SolverParameters_globalSearchIgnoresEqConstraints_set

```
public final static native void
SolverParameters_globalSearchIgnoresEqConstraints_set(long jarg1,
        SolverParameters jarg1_,
        boolean jarg2)
```

SolverParameters_globalSearchIgnoresEqConstraints_get

```
public final static native boolean
SolverParameters_globalSearchIgnoresEqConstraints_get(long jarg1,
        SolverParameters jarg1_)
```

(continued from last page)

SolverParameters_estimateConstraintTol_set

```
public final static native void SolverParameters_estimateConstraintTol_set(long jarg1,
        SolverParameters jarg1_,
        double jarg2)
```

SolverParameters_estimateConstraintTol_get

```
public final static native double SolverParameters_estimateConstraintTol_get(long
jarg1,
        SolverParameters jarg1_)
```

delete_SolverParameters

```
public final static native void delete_SolverParameters(long jarg1)
```

new_OptimizationProblem

```
public final static native long new_OptimizationProblem(int jarg1,
        int jarg2,
        int jarg3,
        int jarg4,
        double[] jarg5,
        double[] jarg6,
        double[] jarg7,
        double[] jarg8,
        double[] jarg9,
        double[] jarg10)
```

delete_OptimizationProblem

```
public final static native void delete_OptimizationProblem(long jarg1)
```

OptimizationProblem_solve

```
public final static native void OptimizationProblem_solve(long jarg1,
        OptimizationProblem jarg1_,
        long jarg2,
        SolverParameters jarg2_)
```

OptimizationProblem_dimx_get

```
public final static native int OptimizationProblem_dimx_get(long jarg1,
        OptimizationProblem jarg1_)
```

(continued from last page)

OptimizationProblem_dimy_get

```
public final static native int OptimizationProblem_dimy_get(long jarg1,
        OptimizationProblem jarg1_)
```

OptimizationProblem_dimyEq_get

```
public final static native int OptimizationProblem_dimyEq_get(long jarg1,
        OptimizationProblem jarg1_)
```

OptimizationProblem_numinitpts_get

```
public final static native int OptimizationProblem_numinitpts_get(long jarg1,
        OptimizationProblem jarg1_)
```

OptimizationProblem_c_get

```
public final static native double[] OptimizationProblem_c_get(long jarg1,
        OptimizationProblem jarg1_)
```

OptimizationProblem_xlow_get

```
public final static native double[] OptimizationProblem_xlow_get(long jarg1,
        OptimizationProblem jarg1_)
```

OptimizationProblem_xup_get

```
public final static native double[] OptimizationProblem_xup_get(long jarg1,
        OptimizationProblem jarg1_)
```

OptimizationProblem_flow_get

```
public final static native double[] OptimizationProblem_flow_get(long jarg1,
        OptimizationProblem jarg1_)
```

OptimizationProblem_fup_get

```
public final static native double[] OptimizationProblem_fup_get(long jarg1,
        OptimizationProblem jarg1_)
```

(continued from last page)

OptimizationProblem_initptsvec_get

```
public final static native double[] OptimizationProblem_initptsvec_get(long jarg1,
        OptimizationProblem jarg1_)
```

OptimizationProblem_evaluateF

```
public final static native void OptimizationProblem_evaluateF(long jarg1,
        OptimizationProblem jarg1_,
        double[] jarg2,
        double[] jarg3)
```

OptimizationProblem_director_connect

```
public final static native void
OptimizationProblem_director_connect(OptimizationProblem obj,
        long cptr,
        boolean mem_own,
        boolean weak_global)
```

OptimizationProblem_change_ownership

```
public final static native void
OptimizationProblem_change_ownership(OptimizationProblem obj,
        long cptr,
        boolean take_or_release)
```

srand

```
public final static native void srand(long jarg1)
```

SwigDirector_OptimizationProblem_evaluateF

```
public static void SwigDirector_OptimizationProblem_evaluateF(OptimizationProblem
jself,
        double[] x,
        double[] F)
```

com.dagopt.bbowda
Class OptimizationProblem

```
java.lang.Object
   |
   +-com.dagopt.bbowda.OptimizationProblem
```

public class **OptimizationProblem**
extends java.lang.Object

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

Field Summary

protected transient	swigCMemOwn

Constructor Summary

protected	OptimizationProblem(long cPtr, boolean cMemoryOwn)
public	OptimizationProblem(int dimx, int dimy, int dimy_eq, int numinitpts, double[] c, double[] xlow, double[] xup, double[] Flow, double[] Fup, double[] initpts) Main constructor.
public	OptimizationProblem(int dimx, int dimy, int dimy_eq, int numinitpts, double[] c, double[] xlow, double[] xup, double[] Flow, double[] Fup, double[][] initpts) Convenience constructor. This constructor overload is syntactic sugar allowing to pass a 2-dimensional array as *initpts*.

Method Summary

void	delete()
void	evaluateF(double[] x, double[] F) 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).
void	finalize()
double[]	getC() Coefficients in the (linear) objective function. The coefficient vector *c*.

static long	getCPtr(OptimizationProblem obj)
int	getDimx() Number of input variables. The vector dimension of *x*.
int	getDimy() Number of explicit equality constraints. The vector dimension of *y* = F1(*x*).
int	getDimyEq() Number of implicit equality constraints. Vector dimension of F2(*x*).
double[]	getFlow() Lower bounds for explicit equality constraints. Lower bounds for *y* = F1(*x*).
double[]	getFup() Upper bounds for explicit equality constraints. Upper bounds for *y* = F1(*x*).
double[][]	getInitpts() User-provided starting points, as a 2-dimensional jagged array. A jagged matrix of dimension numinitpts * dimx (in row-major order, i.e., first all components of the first starting point, then the second one, etc.).
double[]	getInitptsvec() User-provided starting points, as a contiguous array. 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.).
int	getNuminitpts() Number of user-specified starting points. Can be 0.
double[]	getXlow() Lower bounds for input variables. Lower bounds for *x*.
double[]	getXup() Upper bounds for input variables. Upper bounds for *x*.
void	solve(SolverParameters params) Main entry point of the BBOWDA object-oriented API. Runs the BBOWDA algorithm on this problem with the given parameters.
void	swigDirectorDisconnect()
void	swigReleaseOwnership()
void	swigTakeOwnership()

Methods inherited from class `java.lang.Object`

`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Fields

(continued from last page)

swigCMemOwn

```
protected transient boolean swigCMemOwn
```

Constructors

OptimizationProblem

```
protected OptimizationProblem(long cPtr,
                              boolean cMemoryOwn)
```

OptimizationProblem

```
public OptimizationProblem(int dimx,
                           int dimy,
                           int dimy_eq,
                           int numinitpts,
                           double[] c,
                           double[] xlow,
                           double[] xup,
                           double[] Flow,
                           double[] Fup,
                           double[] initpts)
```

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` - User-provided starting points. See initpts.

(continued from last page)

OptimizationProblem

```
public OptimizationProblem(int dimx,
                           int dimy,
                           int dimy_eq,
                           int numinitpts,
                           double[] c,
                           double[] xlow,
                           double[] xup,
                           double[] Flow,
                           double[] Fup,
                           double[][] initpts)
```

Convenience constructor.

This constructor overload is syntactic sugar allowing to pass a 2-dimensional array as *initpts*.

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` - User-provided starting points. See initpts.

Methods

getCPtr

```
protected static long getCPtr(OptimizationProblem obj)
```

finalize

```
protected void finalize()
```

(continued from last page)

delete

```
public void delete()
```

swigDirectorDisconnect

```
protected void swigDirectorDisconnect()
```

swigReleaseOwnership

```
public void swigReleaseOwnership()
```

swigTakeOwnership

```
public void swigTakeOwnership()
```

solve

```
public void solve(SolverParameters params)
```

Main entry point of the BBOWDA object-oriented API.

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

Parameters:

`params` - The solver parameters for the BBOWDA algorithm.

getDimx

```
public int getDimx()
```

Number of input variables.

The vector dimension of *x*.

getDimy

```
public int getDimy()
```

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.

getDimyEq

```
public int getDimyEq()
```

(continued from last page)

Number of implicit equality constraints.

Vector dimension of F2(*x*).

getNuminitpts

```
public int getNuminitpts()
```

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.

getC

```
public double[] getC()
```

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.

getXlow

```
public double[] getXlow()
```

Lower bounds for input variables.

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

getXup

```
public double[] getXup()
```

Upper bounds for input variables.

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

getFlow

```
public double[] getFlow()
```

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.

getFup

```
public double[] getFup()
```

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.

getInitptsvec

```
public double[] getInitptsvec()
```

User-provided starting points, as a contiguous array.

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).

evaluateF

```
protected void evaluateF(double[] x,
         double[] F)
```

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.

getInitpts

```
public double[][] getInitpts()
```

User-provided starting points, as a 2-dimensional jagged array.

A jagged 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 array is empty.

com.dagopt.bbowda
Class SolverParameters

```
java.lang.Object
   |
   +-com.dagopt.bbowda.SolverParameters
```

public class **SolverParameters**
extends java.lang.Object

Solver parameters for the BBOWDA algorithm.

Parameters allowing to tune how the BBOWDA algorithm operates.

Field Summary

protected transient	swigCMemOwn

Constructor Summary

protected	SolverParameters(long cPtr, boolean cMemoryOwn)
public	SolverParameters(long maxpts, double optimum_tol, boolean global_search_ignores_eq_constraints, double estimate_constraint_tol) Constructor.

Method Summary

void	delete()
void	finalize()
static long	getCPtr(SolverParameters obj)
double	getEstimateConstraintTol() Tolerance for the constraints estimating the implicit equality constraints during global search. If global_search_ignores_eq_constraints is false, the global search attempts to estimate global enclosures for the implicit equality constraints.
boolean	getGlobalSearchIgnoresEqConstraints() Whether to ignore implicit equality constraints for global search. True means that implicit equality constraints will be ignored by the global search.
long	getMaxpts() 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.

double	getOptimumTol() 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.
void	setEstimateConstraintTol(double value) Tolerance for the constraints estimating the implicit equality constraints during global search. If global_search_ignores_eq_constraints is false, the global search attempts to estimate global enclosures for the implicit equality constraints.
void	setGlobalSearchIgnoresEqConstraints(boolean value) Whether to ignore implicit equality constraints for global search. True means that implicit equality constraints will be ignored by the global search.
void	setMaxpts(long value) 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.
void	setOptimumTol(double value) 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.

Methods inherited from class `java.lang.Object`

`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Fields

swigCMemOwn

```
protected transient boolean swigCMemOwn
```

Constructors

SolverParameters

```
protected SolverParameters(long cPtr,
                           boolean cMemoryOwn)
```

SolverParameters

```
public SolverParameters(long maxpts,
                        double optimum_tol,
                        boolean global_search_ignores_eq_constraints,
                        double estimate_constraint_tol)
```

(continued from last page)

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.

Methods

getCPtr

```
protected static long getCPtr(SolverParameters obj)
```

finalize

```
protected void finalize()
```

delete

```
public void delete()
```

setMaxpts

```
public void setMaxpts(long value)
```

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.

getMaxpts

```
public long getMaxpts()
```

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.

(continued from last page)

setOptimumTol

`public void **setOptimumTol**(double value)`

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*.

---

getOptimumTol

`public double **getOptimumTol**()`

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*.

---

setGlobalSearchIgnoresEqConstraints

`public void **setGlobalSearchIgnoresEqConstraints**(boolean value)`

Whether to ignore implicit equality constraints for global search.

True 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.

False 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.

---

getGlobalSearchIgnoresEqConstraints

`public boolean **getGlobalSearchIgnoresEqConstraints**()`

(continued from last page)

Whether to ignore implicit equality constraints for global search.

True 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.

False 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.

setEstimateConstraintTol

```
public void setEstimateConstraintTol(double value)
```

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

If global_search_ignores_eq_constraints is false, 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.

getEstimateConstraintTol

```
public double getEstimateConstraintTol()
```

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

If global_search_ignores_eq_constraints is false, 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.

Index