Vol

1.4

Generated by Doxygen 1.7.1

Mon Nov 25 2013 17:37:35

Contents

1 Class Index

1.1 Class Hierarchy

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

std::basic_fstream< char >
std::basic_fstream< wchar_t >
std::basic_ifstream< char >
std::basic_ifstream< wchar_t >
std::basic_ios< char >
std::basic_ios< wchar_t >
std::basic_iostream< char >
std::basic_iostream< wchar_t >
std::basic_istream< char >
std::basic_istream< wchar_t >
std::basic_istringstream< char >
std::basic_istringstream< wchar_t >
std::basic_ofstream< char >
std::basic_ofstream< wchar_t >
std::basic_ostream< char >
std::basic_ostream< wchar_t >
std::basic_ostringstream< char >
std::basic_ostringstream< wchar_t >
std::basic_string< char >
std::basic_string< wchar_t >

2 Class Index

2.1 Class List

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

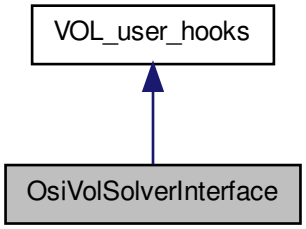
3 Class Documentation

3.1 OsiVolSolverInterface Class Reference

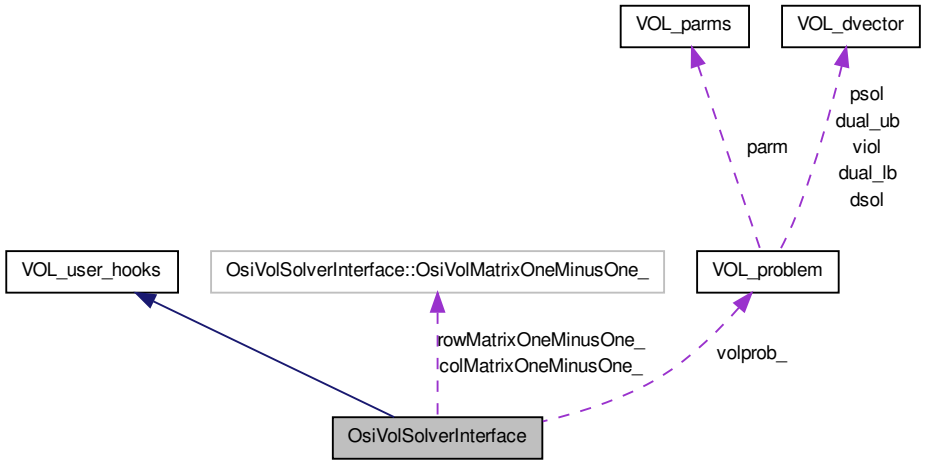
Vol(ume) Solver Interface.

`#include <OsiVolSolverInterface.hpp>`

Inheritance diagram for OsiVolSolverInterface:



Collaboration diagram for OsiVolSolverInterface:



Classes

- class **OsiVolMatrixOneMinusOne_**

Public Member Functions

- virtual void setObjSense (double s)

  Set objective function sense (1 for min (default), -1 for max,).

- virtual void setColSolution (const double *colsol)

  Set the primal solution column values.

- virtual void setRowPrice (const double *rowprice)

  Set dual solution vector.

Solve methods

- virtual void initialSolve ()

  Solve initial LP relaxation.

- virtual void resolve ()

  Resolve an LP relaxation after problem modification.

- virtual void branchAndBound ()

  Invoke solver's built-in enumeration algorithm.

Parameter set/get methods

The set methods return true if the parameter was set to the given value, false otherwise.

There can be various reasons for failure: the given parameter is not applicable for the solver (e.g., refactorization frequency for the volume algorithm), the parameter is not yet implemented for the solver or simply the value of the parameter is out of the range the solver accepts. If a parameter setting call returns false check the details of your solver.

The get methods return true if the given parameter is applicable for the solver and is implemented. In this case the value of the parameter is returned in the second argument. Otherwise they return false.

- bool **setIntParam** (OsiIntParam key, int value)
- bool **setDblParam** (OsiDblParam key, double value)
- bool **setStrParam** (OsiStrParam key, const std::string &value)
- bool **getIntParam** (OsiIntParam key, int &value) const
- bool **getDblParam** (OsiDblParam key, double &value) const
- bool **getStrParam** (OsiStrParam key, std::string &value) const

Methods returning info on how the solution process terminated

- virtual bool isAbandoned () const

  Are there a numerical difficulties?

- virtual bool isProvenOptimal () const

  Is optimality proven?

- virtual bool isProvenPrimalInfeasible () const

  Is primal infeasiblity proven?

- virtual bool isProvenDualInfeasible () const

  Is dual infeasiblity proven?

- virtual bool isPrimalObjectiveLimitReached () const

  Is the given primal objective limit reached?

- virtual bool isDualObjectiveLimitReached () const

  Is the given dual objective limit reached?

- virtual bool isIterationLimitReached () const
  Iteration limit reached?

WarmStart related methods

- virtual CoinWarmStart * getEmptyWarmStart () const
  Get an empty warm start object.
- virtual CoinWarmStart * getWarmStart () const
  Get warmstarting information.
- virtual bool setWarmStart (const CoinWarmStart *warmstart)
  Set warmstarting information.

Hotstart related methods (primarily used in strong branching).

The user can create a hotstart (a snapshot) of the optimization process then reoptimize over and over again always starting from there.

***NOTE**: between hotstarted optimizations only bound changes are allowed.*

- virtual void markHotStart ()
  Create a hotstart point of the optimization process.
- virtual void solveFromHotStart ()
  Optimize starting from the hotstart.
- virtual void unmarkHotStart ()
  Delete the snapshot.

Methods related to querying the input data

- virtual int getNumCols () const
  Get number of columns.
- virtual int getNumRows () const
  Get number of rows.
- virtual int getNumElements () const
  Get number of nonzero elements.
- virtual const double * getColLower () const
  Get pointer to array[getNumCols()] of column lower bounds.

- virtual const double * getColUpper () const
  Get pointer to array[getNumCols()] of column upper bounds.
- virtual const char * getRowSense () const
  Get pointer to array[getNumRows()] of row constraint senses.
- virtual const double * getRightHandSide () const
  Get pointer to array[getNumRows()] of rows right-hand sides.
- virtual const double * getRowRange () const
  Get pointer to array[getNumRows()] of row ranges.
- virtual const double * getRowLower () const
  Get pointer to array[getNumRows()] of row lower bounds.
- virtual const double * getRowUpper () const
  Get pointer to array[getNumRows()] of row upper bounds.
- virtual const double * getObjCoefficients () const
  Get pointer to array[getNumCols()] of objective function coefficients.
- virtual double getObjSense () const
  Get objective function sense (1 for min (default), -1 for max).
- virtual bool isContinuous (int colNumber) const
  Return true if column is continuous.
- virtual const CoinPackedMatrix * getMatrixByRow () const
  Get pointer to row-wise copy of matrix.
- virtual const CoinPackedMatrix * getMatrixByCol () const
  Get pointer to column-wise copy of matrix.
- virtual double getInfinity () const
  Get solver's value for infinity.

Methods related to querying the solution

- virtual const double * getColSolution () const
  Get pointer to array[getNumCols()] of primal solution vector.
- virtual const double * getRowPrice () const
  Get pointer to array[getNumRows()] of dual prices.

- virtual const double * getReducedCost () const
  Get a pointer to array[getNumCols()] of reduced costs.

- virtual const double * getRowActivity () const
  Get pointer to array[getNumRows()] of row activity levels (constraint matrix times the solution vector.

- virtual double getObjValue () const
  Get objective function value.

- virtual int getIterationCount () const
  Get how many iterations it took to solve the problem (whatever "iteration" mean to the solver.

- virtual std::vector< double * > getDualRays (int maxNumRays, bool fullRay=false) const
  Get as many dual rays as the solver can provide.

- virtual std::vector< double * > getPrimalRays (int maxNumRays) const
  Get as many primal rays as the solver can provide.

Changing bounds on variables and constraints

- virtual void setObjCoeff (int elementIndex, double elementValue)
  Set an objective function coefficient.

- virtual void setColLower (int elementIndex, double elementValue)
  Set a single column lower bound
  Use -DBL_MAX for -infinity.

- virtual void setColUpper (int elementIndex, double elementValue)
  Set a single column upper bound
  Use DBL_MAX for infinity.

- virtual void setColBounds (int elementIndex, double lower, double upper)
  Set a single column lower and upper bound.

- virtual void setColSetBounds (const int *indexFirst, const int *indexLast, const double *boundList)
  Set the bounds on a number of columns simultaneously
  The default implementation just invokes setColLower() and setColUpper() over and over again.

- virtual void setRowLower (int elementIndex, double elementValue)
  Set a single row lower bound
  Use -DBL_MAX for -infinity.

- virtual void setRowUpper (int elementIndex, double elementValue)
  Set a single row upper bound
  Use DBL_MAX for infinity.

- virtual void setRowBounds (int elementIndex, double lower, double upper)
  Set a single row lower and upper bound.

- virtual void setRowType (int index, char sense, double rightHandSide, double range)
  Set the type of a single row

- virtual void setRowSetBounds (const int *indexFirst, const int *indexLast, const double *boundList)
  Set the bounds on a number of rows simultaneously
  The default implementation just invokes setRowLower() and setRowUpper() over and over again.

- virtual void setRowSetTypes (const int *indexFirst, const int *indexLast, const char *senseList, const double *rhsList, const double *rangeList)
  Set the type of a number of rows simultaneously
  The default implementation just invokes setRowType() over and over again.

Integrality related changing methods

- virtual void setContinuous (int index)
  Set the index-th variable to be a continuous variable.

- virtual void setInteger (int index)
  Set the index-th variable to be an integer variable.

- virtual void setContinuous (const int *indices, int len)
  Set the variables listed in indices (which is of length len) to be continuous variables.

- virtual void setInteger (const int *indices, int len)
  Set the variables listed in indices (which is of length len) to be integer variables.

Methods to expand a problem.

Note that if a column is added then by default it will correspond to a continuous variable.

- virtual void **addCol** (const CoinPackedVectorBase &vec, const double collb, const double colub, const double obj)

- virtual void **addCols** (const int numcols, const CoinPackedVectorBase ∗const ∗cols, const double ∗collb, const double ∗colub, const double ∗obj)
- virtual void **deleteCols** (const int num, const int ∗colIndices)
- virtual void **addRow** (const CoinPackedVectorBase &vec, const double rowlb, const double rowub)
- virtual void **addRow** (const CoinPackedVectorBase &vec, const char rowsen, const double rowrhs, const double rowrng)
- virtual void **addRows** (const int numrows, const CoinPackedVectorBase ∗const ∗rows, const double ∗rowlb, const double ∗rowub)
- virtual void **addRows** (const int numrows, const CoinPackedVectorBase ∗const ∗rows, const char ∗rowsen, const double ∗rowrhs, const double ∗rowrng)
- virtual void **deleteRows** (const int num, const int ∗rowIndices)

Methods to input a problem

- virtual void loadProblem (const CoinPackedMatrix &matrix, const double ∗collb, const double ∗colub, const double ∗obj, const double ∗rowlb, const double ∗rowub)

  Load in an problem by copying the arguments (the constraints on the rows are given by lower and upper bounds).

- virtual void assignProblem (CoinPackedMatrix ∗&matrix, double ∗&collb, double ∗&colub, double ∗&obj, double ∗&rowlb, double ∗&rowub)

  Load in an problem by assuming ownership of the arguments (the constraints on the rows are given by lower and upper bounds).

- virtual void loadProblem (const CoinPackedMatrix &matrix, const double ∗collb, const double ∗colub, const double ∗obj, const char ∗rowsen, const double ∗rowrhs, const double ∗rowrng)

  Load in an problem by copying the arguments (the constraints on the rows are given by sense/rhs/range triplets).

- virtual void assignProblem (CoinPackedMatrix ∗&matrix, double ∗&collb, double ∗&colub, double ∗&obj, char ∗&rowsen, double ∗&rowrhs, double ∗&rowrng)

  Load in an problem by assuming ownership of the arguments (the constraints on the rows are given by sense/rhs/range triplets).

- virtual void loadProblem (const int numcols, const int numrows, const int ∗start, const int ∗index, const double ∗value, const double ∗collb, const double ∗colub, const double ∗obj, const double ∗rowlb, const double ∗rowub)

  Just like the other loadProblem() methods except that the matrix is given in a standard column major ordered format (without gaps).

- virtual void loadProblem (const int numcols, const int numrows, const int ∗start, const int ∗index, const double ∗value, const double ∗collb, const double

∗colub, const double ∗obj, const char ∗rowsen, const double ∗rowrhs, const double ∗rowrng)

Just like the other loadProblem() methods except that the matrix is given in a standard column major ordered format (without gaps).

- virtual int readMps (const char ∗filename, const char ∗extension="mps")

  Read an mps file from the given filename.

- virtual void writeMps (const char ∗filename, const char ∗extension="mps", double objSense=0.0) const

  Write the problem into an mps file of the given filename.

OSL specific public interfaces

- VOL_problem ∗ volprob ()

  Get pointer to Vol model.

Constructors and destructors

- OsiVolSolverInterface ()

  Default Constructor.

- virtual OsiSolverInterface ∗ clone (bool copyData=true) const

  Clone.

- OsiVolSolverInterface (const OsiVolSolverInterface &)

  Copy constructor.

- OsiVolSolverInterface & operator= (const OsiVolSolverInterface &rhs)

  Assignment operator.

- virtual ∼OsiVolSolverInterface ()

  Destructor.

Protected Member Functions

Helper methods for problem input

- void **initFromRlbRub** (const int rownum, const double ∗rowlb, const double ∗rowub)
- void **initFromRhsSenseRange** (const int rownum, const char ∗rowsen, const double ∗rowrhs, const double ∗rowrng)

- void **initFromClbCubObj** (const int colnum, const double ∗collb, const double ∗colub, const double ∗obj)

Protected methods

- virtual void applyRowCut (const OsiRowCut &rc)
  Apply a row cut (append to constraint matrix).
- virtual void applyColCut (const OsiColCut &cc)
  Apply a column cut (adjust one or more bounds).

Friends

- void OsiVolSolverInterfaceUnitTest (const std::string &mpsDir, const std::string &netlibDir)
  A function that tests the methods in the OsiVolSolverInterface class.

3.1.1 Detailed Description

Vol(ume) Solver Interface. Instantiation of OsiVolSolverInterface for the Volume Algorithm

Definition at line 27 of file OsiVolSolverInterface.hpp.

3.1.2 Member Function Documentation

3.1.2.1 virtual CoinWarmStart∗ OsiVolSolverInterface::getEmptyWarmStart ( ) const [virtual]

Get an empty warm start object.

This routine returns an empty warm start object. Its purpose is to provide a way to give a client a warm start object of the appropriate type, which can resized and modified as desired.

3.1.2.2 virtual bool OsiVolSolverInterface::setWarmStart (const CoinWarmStart ∗ *warmstart*) [virtual]

Set warmstarting information.

Return true/false depending on whether the warmstart information was accepted or not.

3.1.2.3 virtual const char* OsiVolSolverInterface::getRowSense ( ) const [inline, virtual]

Get pointer to array[getNumRows()] of row constraint senses.

- 'L' <= constraint
- 'E' = constraint
- 'G' >= constraint
- 'R' ranged constraint
- 'N' free constraint

Definition at line 196 of file OsiVolSolverInterface.hpp.

3.1.2.4 virtual const double* OsiVolSolverInterface::getRightHandSide ( ) const [inline, virtual]

Get pointer to array[getNumRows()] of rows right-hand sides.

- if rowsense()[i] == 'L' then rhs()[i] == rowupper()[i]
- if rowsense()[i] == 'G' then rhs()[i] == rowlower()[i]
- if rowsense()[i] == 'R' then rhs()[i] == rowupper()[i]
- if rowsense()[i] == 'N' then rhs()[i] == 0.0

Definition at line 206 of file OsiVolSolverInterface.hpp.

3.1.2.5 virtual const double* OsiVolSolverInterface::getRowRange ( ) const [inline, virtual]

Get pointer to array[getNumRows()] of row ranges.

- if rowsense()[i] == 'R' then rowrange()[i] == rowupper()[i] - rowlower()[i]

- if rowsense()[i] != 'R' then rowrange()[i] is undefined

Definition at line 216 of file OsiVolSolverInterface.hpp.

3.1.2.6 virtual int OsiVolSolverInterface::getIterationCount ( ) const [inline, virtual]

Get how many iterations it took to solve the problem (whatever "iteration" mean to the solver.

Definition at line 287 of file OsiVolSolverInterface.hpp.

3.1.2.7 virtual std::vector<double*> OsiVolSolverInterface::getDualRays (int *maxNumRays*, bool *fullRay* = `false`) const [virtual]

Get as many dual rays as the solver can provide.

(In case of proven primal infeasibility there should be at least one.)

The first getNumRows() ray components will always be associated with the row duals (as returned by getRowPrice()). If `fullRay` is true, the final getNumCols() entries will correspond to the ray components associated with the nonbasic variables. If the full ray is requested and the method cannot provide it, it will throw an exception.

NOTE for implementers of solver interfaces:

The double pointers in the vector should point to arrays of length getNumRows() and they should be allocated via new[].

NOTE for users of solver interfaces:

It is the user's responsibility to free the double pointers in the vector using delete[].

3.1.2.8 virtual std::vector<double*> OsiVolSolverInterface::getPrimalRays (int *maxNumRays*) const [virtual]

Get as many primal rays as the solver can provide.

(In case of proven dual infeasibility there should be at least one.)

NOTE for implementers of solver interfaces:

The double pointers in the vector should point to arrays of length getNumCols() and they should be allocated via new[].

NOTE for users of solver interfaces:

It is the user's responsibility to free the double pointers in the vector using delete[].

3.1.2.9 virtual void OsiVolSolverInterface::setColLower (int *elementIndex*, double *elementValue*) [inline, virtual]

Set a single column lower bound

Use -DBL_MAX for -infinity.

Definition at line 345 of file OsiVolSolverInterface.hpp.

3.1.2.10 virtual void OsiVolSolverInterface::setColUpper (int *elementIndex*, double *elementValue*) [inline, virtual]

Set a single column upper bound

Use DBL_MAX for infinity.

Definition at line 352 of file OsiVolSolverInterface.hpp.

3.1.2.11 virtual void OsiVolSolverInterface::setColSetBounds (const int * *indexFirst*, const int * *indexLast*, const double * *boundList*) [virtual]

Set the bounds on a number of columns simultaneously

The default implementation just invokes setColLower() and setColUpper() over and over again.

Parameters

indexFirst,indexLast pointers to the beginning and after the end of the array of the indices of the variables whose *either* bound changes

boundList the new lower/upper bound pairs for the variables

3.1.2.12 virtual void OsiVolSolverInterface::setRowLower (int *elementIndex*, double *elementValue*) [inline, virtual]

Set a single row lower bound

Use -DBL_MAX for -infinity.

Definition at line 377 of file OsiVolSolverInterface.hpp.

3.1.2.13 virtual void OsiVolSolverInterface::setRowUpper (int *elementIndex,* double *elementValue*) [inline, virtual]

Set a single row upper bound

Use DBL_MAX for infinity.

Definition at line 386 of file OsiVolSolverInterface.hpp.

3.1.2.14 virtual void OsiVolSolverInterface::setRowSetBounds (const int * *indexFirst,* const int * *indexLast,* const double * *boundList*) [virtual]

Set the bounds on a number of rows simultaneously

The default implementation just invokes setRowLower() and setRowUpper() over and over again.

Parameters

indexFirst,indexLast pointers to the beginning and after the end of the array of the indices of the constraints whose *either* bound changes

boundList the new lower/upper bound pairs for the constraints

3.1.2.15 virtual void OsiVolSolverInterface::setRowSetTypes (const int * *indexFirst,* const int * *indexLast,* const char * *senseList,* const double * *rhsList,* const double * *rangeList*) [virtual]

Set the type of a number of rows simultaneously

The default implementation just invokes setRowType() over and over again.

Parameters

indexFirst,indexLast pointers to the beginning and after the end of the array of the indices of the constraints whose *any* characteristics changes

senseList the new senses

rhsList the new right hand sides

rangeList the new ranges

3.1.2.16 virtual void OsiVolSolverInterface::setColSolution (const double * *colsol*) [virtual]

Set the primal solution column values.

colsol[numcols()] is an array of values of the problem column variables. These values are copied to memory owned by the solver object or the solver. They will be returned as the result of colsol() until changed by another call to setColsol() or by a call to any solver routine. Whether the solver makes use of the solution in any way is solver-dependent.

3.1.2.17 virtual void OsiVolSolverInterface::setRowPrice (const double * *rowprice*) [virtual]

Set dual solution vector.

rowprice[numrows()] is an array of values of the problem row dual variables. These values are copied to memory owned by the solver object or the solver. They will be returned as the result of rowprice() until changed by another call to setRowprice() or by a call to any solver routine. Whether the solver makes use of the solution in any way is solver-dependent.

3.1.2.18 virtual void OsiVolSolverInterface::loadProblem (const CoinPackedMatrix & *matrix,* const double * *collb,* const double * *colub,* const double * *obj,* const double * *rowlb,* const double * *rowub*) [virtual]

Load in an problem by copying the arguments (the constraints on the rows are given by lower and upper bounds).

If a pointer is 0 then the following values are the default:

- `colub`: all columns have upper bound infinity
- `collb`: all columns have lower bound 0

- `rowub`: all rows have upper bound infinity
- `rowlb`: all rows have lower bound -infinity
- `obj`: all variables have 0 objective coefficient

3.1.2.19 virtual void OsiVolSolverInterface::assignProblem (CoinPackedMatrix *& *matrix*, double *& *collb*, double *& *colub*, double *& *obj*, double *& *rowlb*, double *& *rowub*) [virtual]

Load in an problem by assuming ownership of the arguments (the constraints on the rows are given by lower and upper bounds).

For default values see the previous method.

WARNING: The arguments passed to this method will be freed using the C++ `delete` and `delete[]` functions.

3.1.2.20 virtual void OsiVolSolverInterface::loadProblem (const CoinPackedMatrix & *matrix*, const double * *collb*, const double * *colub*, const double * *obj*, const char * *rowsen*, const double * *rowrhs*, const double * *rowrng*) [virtual]

Load in an problem by copying the arguments (the constraints on the rows are given by sense/rhs/range triplets).

If a pointer is 0 then the following values are the default:

- `colub`: all columns have upper bound infinity
- `collb`: all columns have lower bound 0
- `obj`: all variables have 0 objective coefficient
- `rowsen`: all rows are >=
- `rowrhs`: all right hand sides are 0
- `rowrng`: 0 for the ranged rows

3.1.2.21 virtual void OsiVolSolverInterface::assignProblem (CoinPackedMatrix *& *matrix*, double *& *collb*, double *& *colub*, double *& *obj*, char *& *rowsen*, double *& *rowrhs*, double *& *rowrng*) [virtual]

Load in an problem by assuming ownership of the arguments (the constraints on the rows are given by sense/rhs/range triplets).

For default values see the previous method.

WARNING: The arguments passed to this method will be freed using the C++ `delete` and `delete[]` functions.

3.1.2.22 virtual void OsiVolSolverInterface::loadProblem (const int *numcols*, const int *numrows*, const int * *start*, const int * *index*, const double * *value*, const double * *collb*, const double * *colub*, const double * *obj*, const double * *rowlb*, const double * *rowub*) [virtual]

Just like the other loadProblem() methods except that the matrix is given in a standard column major ordered format (without gaps).

3.1.2.23 virtual void OsiVolSolverInterface::loadProblem (const int *numcols*, const int *numrows*, const int * *start*, const int * *index*, const double * *value*, const double * *collb*, const double * *colub*, const double * *obj*, const char * *rowsen*, const double * *rowrhs*, const double * *rowrng*) [virtual]

Just like the other loadProblem() methods except that the matrix is given in a standard column major ordered format (without gaps).

3.1.2.24 virtual void OsiVolSolverInterface::writeMps (const char * *filename*, const char * *extension* = *"mps"*, double *objSense* = *0.0*) const [virtual]

Write the problem into an mps file of the given filename.

If objSense is non zero then -1.0 forces the code to write a maximization objective and +1.0 to write a minimization one. If 0.0 then solver can do what it wants

3.1.2.25 virtual void OsiVolSolverInterface::applyRowCut (const OsiRowCut & *rc*) [protected, virtual]

Apply a row cut (append to constraint matrix).

3.1.2.26 virtual void OsiVolSolverInterface::applyColCut (const OsiColCut & *cc*) [protected, virtual]

Apply a column cut (adjust one or more bounds).

3.1.3 Friends And Related Function Documentation

3.1.3.1 void OsiVolSolverInterfaceUnitTest (const std::string & *mpsDir*, const std::string & *netlibDir*) [friend]

A function that tests the methods in the OsiVolSolverInterface class.

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

- OsiVolSolverInterface.hpp

3.2 VOL_alpha_factor Class Reference

3.2.1 Detailed Description

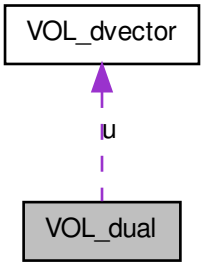
Definition at line 489 of file VolVolume.hpp.

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

- VolVolume.hpp

3.3 VOL_dual Class Reference

Collaboration diagram for VOL_dual:



3.3.1 Detailed Description

Definition at line 354 of file VolVolume.hpp.

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

- VolVolume.hpp

3.4 VOL_dvector Class Reference

vector of doubles.

`#include <VolVolume.hpp>`

Public Member Functions

- VOL_dvector (const int s)

  Construct a vector of size s.

- VOL_dvector ()

  Default constructor creates a vector of size 0.

- VOL_dvector (const VOL_dvector &x)

Copy constructor makes a replica of x.

- ~VOL_dvector ()

  The destructor deletes the data array.

- int size () const

  Return the size of the vector.

- double & operator[ ] (const int i)

  Return a reference to the `i`-th entry.

- double operator[ ] (const int i) const

  Return the `i`-th entry.

- void clear ()

  Delete the content of the vector and replace it with a vector of length 0.

- void cc (const double gamma, const VOL_dvector &w)

  Convex combination.

- void allocate (const int s)

  delete the current vector and allocate space for a vector of size `s`.

- void swap (VOL_dvector &w)

  swaps the vector with `w`.

- VOL_dvector & operator= (const VOL_dvector &w)

  Copy `w` into the vector.

- VOL_dvector & operator= (const double w)

  Replace every entry in the vector with `w`.

Public Attributes

- double * v

  The array holding the vector.

- int sz

  The size of the vector.

3.4.1 Detailed Description

vector of doubles. It is used for most vector operations.

Note: If `VOL_DEBUG` is `defined` to be 1 then each time an entry is accessed in the vector the index of the entry is tested for nonnegativity and for being less than the size of the vector. It's good to turn this on while debugging, but in final runs it should be turned off (beause of the performance hit).

Definition at line 148 of file VolVolume.hpp.

3.4.2 Constructor & Destructor Documentation

3.4.2.1 VOL_dvector::VOL_dvector (const int *s*) [inline]

Construct a vector of size s.

The content of the vector is undefined.

Definition at line 157 of file VolVolume.hpp.

3.4.2.2 VOL_dvector::VOL_dvector ( ) [inline]

Default constructor creates a vector of size 0.

Definition at line 162 of file VolVolume.hpp.

3.4.2.3 VOL_dvector::VOL_dvector (const VOL_dvector & *x*) [inline]

Copy constructor makes a replica of x.

Definition at line 164 of file VolVolume.hpp.

3.4.2.4 VOL_dvector::~VOL_dvector ( ) [inline]

The destructor deletes the data array.

Definition at line 172 of file VolVolume.hpp.

3.4.3 Member Function Documentation

3.4.3.1 int VOL_dvector::size ( ) const [inline]

Return the size of the vector.

Definition at line 175 of file VolVolume.hpp.

3.4.3.2 double& VOL_dvector::operator[ ] (const int *i*) [inline]

Return a reference to the `i`-th entry.

Definition at line 178 of file VolVolume.hpp.

3.4.3.3 double VOL_dvector::operator[ ] (const int *i*) const [inline]

Return the `i`-th entry.

Definition at line 184 of file VolVolume.hpp.

3.4.3.4 void VOL_dvector::clear ( ) [inline]

Delete the content of the vector and replace it with a vector of length 0.

Definition at line 191 of file VolVolume.hpp.

3.4.3.5 void VOL_dvector::cc (const double *gamma,* const VOL_dvector & *w*) [inline]

Convex combination.

Replace the current vector `v` with `v = (1-gamma) v + gamma w`.

Definition at line 198 of file VolVolume.hpp.

3.4.3.6 void VOL_dvector::allocate (const int *s*) [inline]

delete the current vector and allocate space for a vector of size `s`.

Definition at line 214 of file VolVolume.hpp.

3.4.3.7 void VOL_dvector::swap (VOL_dvector & *w*) [inline]

swaps the vector with `w`.

Definition at line 221 of file VolVolume.hpp.

3.4.3.8 VOL_dvector& VOL_dvector::operator= (const VOL_dvector & *w*)

Copy `w` into the vector.

3.4.3.9 VOL_dvector& VOL_dvector::operator= (const double *w*)

Replace every entry in the vector with `w`.

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

- VolVolume.hpp

3.5 VOL_indc Class Reference

3.5.1 Detailed Description

Definition at line 538 of file VolVolume.hpp.

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

- VolVolume.hpp

3.6 VOL_ivector Class Reference

vector of ints.

`#include <VolVolume.hpp>`

Public Member Functions

- VOL_ivector (const int s)

  Construct a vector of size s.

- VOL_ivector ()

  Default constructor creates a vector of size 0.

- VOL_ivector (const VOL_ivector &x)

  Copy constructor makes a replica of x.

- ~VOL_ivector ()

  The destructor deletes the data array.

- int size () const

  Return the size of the vector.

- int & operator[ ] (const int i)

  Return a reference to the `i`-th entry.

- int operator[ ] (const int i) const

  Return the `i`-th entry.

- void clear ()

  Delete the content of the vector and replace it with a vector of length 0.

- void allocate (const int s)

  delete the current vector and allocate space for a vector of size `s`.

- void swap (VOL_ivector &w)

  swaps the vector with `w`.

- VOL_ivector & operator= (const VOL_ivector &v)

  Copy `w` into the vector.

- VOL_ivector & operator= (const int w)

  Replace every entry in the vector with `w`.

Public Attributes

- int * v

  The array holding the vector.

- int sz

  The size of the vector.

3.6.1 Detailed Description

vector of ints. It's used to store indices, it has similar functions as VOL_dvector.

Note: If `VOL_DEBUG` is `defined` to be 1 then each time an entry is accessed in the vector the index of the entry is tested for nonnegativity and for being less than the size of the vector. It's good to turn this on while debugging, but in final runs it should be turned off (beause of the performance hit).

Definition at line 242 of file VolVolume.hpp.

3.6.2 Constructor & Destructor Documentation

3.6.2.1 VOL_ivector::VOL_ivector (const int *s*) [inline]

Construct a vector of size s.

The content of the vector is undefined.

Definition at line 250 of file VolVolume.hpp.

3.6.2.2 VOL_ivector::VOL_ivector ( ) [inline]

Default constructor creates a vector of size 0.

Definition at line 255 of file VolVolume.hpp.

3.6.2.3 VOL_ivector::VOL_ivector (const VOL_ivector & *x*) [inline]

Copy constructor makes a replica of x.

Definition at line 257 of file VolVolume.hpp.

3.6.2.4 VOL_ivector::~VOL_ivector ( ) [inline]

The destructor deletes the data array.

Definition at line 265 of file VolVolume.hpp.

3.6.3 Member Function Documentation

3.6.3.1 int VOL_ivector::size ( ) const [inline]

Return the size of the vector.

Definition at line 270 of file VolVolume.hpp.

3.6.3.2 int& VOL_ivector::operator[ ] (const int *i*) [inline]

Return a reference to the `i`-th entry.

Definition at line 272 of file VolVolume.hpp.

3.6.3.3 int VOL_ivector::operator[ ] (const int *i*) const [inline]

Return the `i`-th entry.

Definition at line 278 of file VolVolume.hpp.

3.6.3.4 void VOL_ivector::clear ( ) [inline]

Delete the content of the vector and replace it with a vector of length 0.

Definition at line 285 of file VolVolume.hpp.

3.6.3.5 void VOL_ivector::allocate (const int *s*) [inline]

delete the current vector and allocate space for a vector of size `s`.

Definition at line 293 of file VolVolume.hpp.

3.6.3.6 void VOL_ivector::swap (VOL_ivector & *w*) [inline]

swaps the vector with w.

Definition at line 300 of file VolVolume.hpp.

3.6.3.7 VOL_ivector& VOL_ivector::operator= (const VOL_ivector & *v*)

Copy w into the vector.

3.6.3.8 VOL_ivector& VOL_ivector::operator= (const int *w*)

Replace every entry in the vector with w.

3.6.4 Member Data Documentation

3.6.4.1 int* VOL_ivector::v

The array holding the vector.

Definition at line 245 of file VolVolume.hpp.

3.6.4.2 int VOL_ivector::sz

The size of the vector.

Definition at line 247 of file VolVolume.hpp.

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

- VolVolume.hpp

3.7 VOL_parms Struct Reference

This class contains the parameters controlling the Volume Algorithm.

`#include <VolVolume.hpp>`

Public Attributes

- double lambdainit

  initial value of lambda

- double alphainit

  initial value of alpha

- double alphamin

  minimum value for alpha

- double alphafactor

  when little progress is being done, we multiply alpha by alphafactor

- double ubinit

  initial upper bound of the value of an integer solution

- double primal_abs_precision

  accept if max abs viol is less than this

- double gap_abs_precision

  accept if abs gap is less than this

- double gap_rel_precision

  accept if rel gap is less than this

- double granularity

  terminate if best_ub - lcost < granularity

- double minimum_rel_ascent

  terminate if the relative increase in lcost through `ascent_check_invl` steps is less than this

- int ascent_first_check

  when to check for sufficient relative ascent the first time

- int ascent_check_invl

through how many iterations does the relative ascent have to reach a minimum

- int maxsgriters
  maximum number of iterations
- int printflag
  controls the level of printing.
- int printinvl
  controls how often do we print
- int heurinvl
  controls how often we run the primal heuristic
- int greentestinvl
  how many consecutive green iterations are allowed before changing lambda
- int yellowtestinvl
  how many consecutive yellow iterations are allowed before changing lambda
- int redtestinvl
  how many consecutive red iterations are allowed before changing lambda
- int alphaint
  number of iterations before we check if alpha should be decreased
- char * temp_dualfile
  name of file for saving dual solution

3.7.1 Detailed Description

This class contains the parameters controlling the Volume Algorithm.

Definition at line 71 of file VolVolume.hpp.

3.7.2 Member Data Documentation

3.7.2.1 int VOL_parms::printflag

controls the level of printing.

The flag should the the 'OR'-d value of the following options:

- 0 - print nothing
- 1 - print iteration information
- 2 - add lambda information
- 4 - add number of Red, Yellow, Green iterations

Default: 3

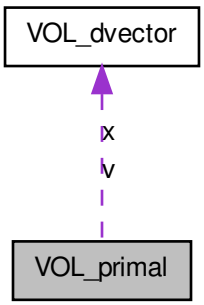
Definition at line 115 of file VolVolume.hpp.

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

- VolVolume.hpp

3.8 VOL_primal Class Reference

Collaboration diagram for VOL_primal:



3.8.1 Detailed Description

Definition at line 313 of file VolVolume.hpp.

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

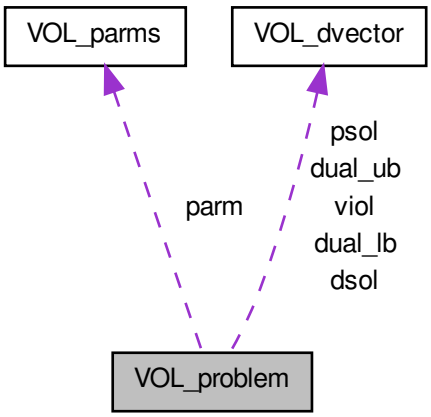
- VolVolume.hpp

3.9 VOL_problem Class Reference

This class holds every data for the Volume Algorithm and its `solve` method must be invoked to solve the problem.

`#include <VolVolume.hpp>`

Collaboration diagram for VOL_problem:



Public Member Functions

Constructors and destructor

- VOL_problem ()
  Default constructor.
- VOL_problem (const char *filename)
  Create a a `VOL_problem` object and read in the parameters from `filename`.
- ~VOL_problem ()
  Destruct the object.

Method to solve the problem.

- int solve (VOL_user_hooks &hooks, const bool use_preset_dual=false)
  Solve the problem using the `hooks`.

Methods returning final data

- int iter () const
  returns the iteration number
- double alpha () const
  returns the value of alpha
- double lambda () const
  returns the value of lambda

Public Attributes

- int iter_
  iteration number

External data (containing the result after solve)

- double value
  final lagrangian value (OUTPUT)
- VOL_dvector dsol
  final dual solution (INPUT/OUTPUT)
- VOL_dvector psol
  final primal solution (OUTPUT)
- VOL_dvector viol
  violations (b-Ax) for the relaxed constraints

External data (may be changed by the user before calling solve)

- VOL_parms parm
  The parameters controlling the Volume Algorithm (INPUT).
- int psize
  length of primal solution (INPUT)

- int dsize
  length of dual solution (INPUT)
- VOL_dvector dual_lb
  lower bounds for the duals (if 0 length, then filled with -inf) (INPUT)
- VOL_dvector dual_ub
  upper bounds for the duals (if 0 length, then filled with +inf) (INPUT)

3.9.1 Detailed Description

This class holds every data for the Volume Algorithm and its `solve` method must be invoked to solve the problem. The INPUT fields must be filled out completely before `solve` is invoked. `dsol` have to be filled out if and only if the last argument to `solve` is `true`.

Definition at line 605 of file VolVolume.hpp.

3.9.2 Constructor & Destructor Documentation

3.9.2.1 VOL_problem::VOL_problem ( )

Default constructor.

3.9.2.2 VOL_problem::VOL_problem (const char * *filename*)

Create a a `VOL_problem` object and read in the parameters from `filename`.

3.9.2.3 VOL_problem::~VOL_problem ( )

Destruct the object.

3.9.3 Member Function Documentation

3.9.3.1 int VOL_problem::solve (VOL_user_hooks & *hooks*, const bool *use_preset_dual* = `false`)

Solve the problem using the `hooks`.

Any information needed in the hooks must be stored in the structure `user_data` points to.

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

- VolVolume.hpp

3.10 VOL_swing Class Reference

3.10.1 Detailed Description

Definition at line 390 of file VolVolume.hpp.

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

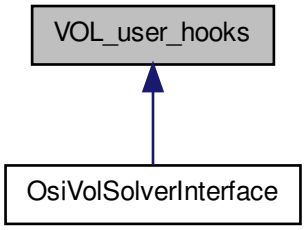
- VolVolume.hpp

3.11 VOL_user_hooks Class Reference

The user hooks should be overridden by the user to provide the problem specific routines for the volume algorithm.

`#include <VolVolume.hpp>`

Inheritance diagram for VOL_user_hooks:



Public Member Functions

- virtual int compute_rc (const VOL_dvector &u, VOL_dvector &rc)=0

compute reduced costs

- virtual int solve_subproblem (const VOL_dvector &dual, const VOL_dvector &rc, double &lcost, VOL_dvector &x, VOL_dvector &v, double &pcost)=0

  Solve the subproblem for the subgradient step.

- virtual int heuristics (const VOL_problem &p, const VOL_dvector &x, double &heur_val)=0

  Starting from the primal vector x, run a heuristic to produce an integer solution.

3.11.1 Detailed Description

The user hooks should be overridden by the user to provide the problem specific routines for the volume algorithm. The user should derive a class ...

for all hooks: return value of -1 means that volume should quit

Definition at line 563 of file VolVolume.hpp.

3.11.2 Member Function Documentation

3.11.2.1 virtual int VOL_user_hooks::compute_rc (const VOL_dvector & *u*, VOL_dvector & *rc*) [pure virtual]

compute reduced costs

Parameters

u (IN) the dual variables

rc (OUT) the reduced cost with respect to the dual values

3.11.2.2 virtual int VOL_user_hooks::solve_subproblem (const VOL_dvector & *dual*, const VOL_dvector & *rc*, double & *lcost*, VOL_dvector & *x*, VOL_dvector & *v*, double & *pcost*) [pure virtual]

Solve the subproblem for the subgradient step.

Parameters

dual (IN) the dual variables

rc (IN) the reduced cost with respect to the dual values

lcost (OUT) the lagrangean cost with respect to the dual values

x (OUT) the primal result of solving the subproblem

v (OUT) b-Ax for the relaxed constraints

pcost (OUT) the primal objective value of `x`

3.11.2.3 virtual int VOL_user_hooks::heuristics (const VOL_problem & *p*, const VOL_dvector & *x*, double & *heur_val*) [pure virtual]

Starting from the primal vector x, run a heuristic to produce an integer solution.

Parameters

x (IN) the primal vector

heur_val (OUT) the value of the integer solution (return `DBL_MAX` here if no feas sol was found

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

- VolVolume.hpp

3.12 VOL_vh Class Reference

3.12.1 Detailed Description

Definition at line 515 of file VolVolume.hpp.

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

- VolVolume.hpp

Index