Ipopt

3.12

Generated by Doxygen 1.8.9.1

Thu Oct 8 2015 22:48:38

Contents

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

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

- basic_fstream< char >
- basic_fstream< wchar_t >
- basic_stringstream< char >
- basic_stringstream< wchar_t >
- std::basic_fstream
  - std::fstream
  - std::wfstream
- std::basic_stringstream
  - std::stringstream
  - std::wstringstream

- std::basic_istringstream
  - std::istringstream
  - std::wistringstream
- std::istream
- std::wistream

- std::basic_ostream
  - basic_iostream< char >
  - basic_iostream< wchar_t >
  - basic_ofstream< char >
  - basic_ofstream< wchar_t >
  - basic_ostringstream< char >
  - basic_ostringstream< wchar_t >
  - std::basic_iostream
  - std::basic_ofstream
    - std::ofstream
    - std::wofstream
  - std::basic_ostringstream
    - std::ostringstream
    - std::wostringstream
  - std::ostream
  - std::wostream
- std::ios
- std::wios

Ipopt::IpoptException . . . 153
std::list< T >::iterator
std::forward_list< T >::iterator
std::map< K, T >::iterator
std::unordered_map< K, T >::iterator
std::multimap< K, T >::iterator
std::basic_string< Char >::iterator
std::unordered_multimap< K, T >::iterator
std::set< K >::iterator
std::string::iterator
std::unordered_set< K >::iterator
std::wstring::iterator
std::multiset< K >::iterator
std::unordered_multiset< K >::iterator
std::vector< T >::iterator
std::deque< T >::iterator
std::list< T >
std::list< Ipopt::DependentResult< Ipopt::SmartPtr< const Ipopt::Matrix > > * >
std::list< Ipopt::DependentResult< Ipopt::SmartPtr< const Ipopt::SymMatrix > > * >
std::list< Ipopt::DependentResult< Ipopt::SmartPtr< const Ipopt::Vector > > * >
std::list< Ipopt::DependentResult< Ipopt::SmartPtr< Ipopt::Vector > > * >

std::unordered_set< K >
std::valarray< T >
std::vector< T >
std::vector< const Ipopt::Subject * >
std::vector< Index >
std::vector< Ipopt::Observer * >
std::vector< Ipopt::PiecewisePenEntry >
std::vector< Ipopt::RegisteredOption::string_entry >
std::vector< Ipopt::SmartPtr< const Ipopt::Matrix > >
std::vector< Ipopt::SmartPtr< const Ipopt::MatrixSpace > >
std::vector< Ipopt::SmartPtr< const Ipopt::SymMatrix > >
std::vector< Ipopt::SmartPtr< const Ipopt::SymMatrixSpace > >
std::vector< Ipopt::SmartPtr< const Ipopt::Vector > >
std::vector< Ipopt::SmartPtr< const Ipopt::VectorSpace > >
std::vector< Ipopt::SmartPtr< Ipopt::Journal > >
std::vector< Ipopt::SmartPtr< Ipopt::Vector > >
std::vector< Ipopt::TaggedObject::Tag >
std::vector< Number >
std::vector< std::string >
std::vector< std::vector< bool > >
std::vector< std::vector< Ipopt::SmartPtr< const Ipopt::Matrix > > >
std::vector< std::vector< Ipopt::SmartPtr< const Ipopt::MatrixSpace > > >
std::vector< std::vector< Ipopt::SmartPtr< Ipopt::Matrix > > >
std::vector< Suffix_Source >
std::vector< Suffix_Type >
std::weak_ptr< T >
K
T

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

Class Documentation

3.1 Ipopt::AdaptiveMuUpdate Class Reference

Non-monotone mu update.

```
#include <IpAdaptiveMuUpdate.hpp>
```

Inheritance diagram for Ipopt::AdaptiveMuUpdate:

3.2 Ipopt::AlgorithmBuilder Class Reference

Builder to create a complete IpoptAlg object.

```
#include <IpAlgBuilder.hpp>
```

Inheritance diagram for Ipopt::AlgorithmBuilder:

Public Member Functions

Constructors/Destructors

- AlgorithmBuilder (SmartPtr< AugSystemSolver > custom_solver=NULL)
  Constructor.
- virtual ∼AlgorithmBuilder ()
  Destructor.

Methods to build parts of the algorithm

- virtual void **BuildIpoptObjects** (const Journalist &jnlst, const OptionsList &options, const std::string &prefix, const SmartPtr< NLP > &nlp, SmartPtr< IpoptNLP > &ip_nlp, SmartPtr< IpoptData > &ip_data, SmartPtr< IpoptCalculatedQuantities > &ip_cq)
- virtual SmartPtr< IpoptAlgorithm > **BuildBasicAlgorithm** (const Journalist &jnlst, const OptionsList &options, const std::string &prefix)

Static Public Member Functions

- static void RegisterOptions (SmartPtr< RegisteredOptions > roptions)
  Methods for IpoptTypeInfo.

3.2.1 Detailed Description

Builder to create a complete IpoptAlg object.

This object contains all subelements (such as line search objects etc). How the resulting IpoptAlg object is built can be influenced by the options.

The optional argument custom_solver allows the expert user to provide a specialized linear solver (e.g., of the type GenAugSystemSolver), possibly for selfmade matrix objects.

TODO: Currently, this is a basic implementation with everything in one method that can be overloaded. This will need to be expanded to allow customization of different parts without recoding everything.

Definition at line 32 of file IpAlgBuilder.hpp.

3.2.2 Member Function Documentation

3.2.2.1 static void Ipopt::AlgorithmBuilder::RegisterOptions (SmartPtr< RegisteredOptions > *roptions*) [static]

Methods for IpoptTypeInfo.

register the options used by the algorithm builder

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

- IpAlgBuilder.hpp

3.3 Ipopt::AlgorithmStrategyObject Class Reference

This is the base class for all algorithm strategy objects.

`#include <IpAlgStrategy.hpp>`

Inheritance diagram for Ipopt::AlgorithmStrategyObject:

Public Member Functions

- bool Initialize (const Journalist &jnlst, IpoptNLP &ip_nlp, IpoptData &ip_data, IpoptCalculatedQuantities &ip_cq, const OptionsList &options, const std::string &prefix)

    This method is called every time the algorithm starts again - it is used to reset any internal state.
- bool ReducedInitialize (const Journalist &jnlst, const OptionsList &options, const std::string &prefix)

    Reduced version of the Initialize method, which does not require special Ipopt information.

Constructors/Destructors

- AlgorithmStrategyObject ()

    Default Constructor.
- virtual ~AlgorithmStrategyObject ()

    Default Destructor.

Protected Member Functions

- virtual bool InitializeImpl (const OptionsList &options, const std::string &prefix)=0

Implementation of the initialization method that has to be overloaded by for each derived class.

Accessor methods for the problem defining objects.

Those should be used by the derived classes.

- const Journalist & **Jnlst** () const
- IpoptNLP & **IpNLP** () const
- IpoptData & **IpData** () const
- IpoptCalculatedQuantities & **IpCq** () const
- bool **HaveIpData** () const

3.3.1 Detailed Description

This is the base class for all algorithm strategy objects.

The AlgorithmStrategyObject base class implements a common interface for all algorithm strategy objects. A strategy object is a component of the algorithm for which different alternatives or implementations exists. It allows to compose the algorithm before execution for a particular configuration, without the need to call alternatives based on enums. For example, the LineSearch object is a strategy object, since different line search options might be used for different runs.

This interface is used for things that are done to all strategy objects, like initialization and setting options.

Definition at line 35 of file IpAlgStrategy.hpp.

3.3.2 Member Function Documentation

3.3.2.1 bool Ipopt::AlgorithmStrategyObject::Initialize (const **Journalist** & *jnlst*, **IpoptNLP** & *ip_nlp*, **IpoptData** & *ip_data*, **IpoptCalculatedQuantities** & *ip_cq*, const **OptionsList** & *options*, const std::string & *prefix*) [inline]

This method is called every time the algorithm starts again - it is used to reset any internal state.

The pointers to the Journalist, as well as to the IpoptNLP, IpoptData, and IpoptCalculatedQuantities objects should be stored in the instanciation of this base class. This method is also used to get all required user options from the OptionsList. Here, if prefix is given, each tag (identifying the options) is first looked for with the prefix in front, and if not found, without the prefix. Note: you should not cue off of the iteration count to indicate the "start" of an algorithm!

Do not overload this method, since it does some general initialization that is common for all strategy objects. Overload the protected InitializeImpl method instead.

Definition at line 66 of file IpAlgStrategy.hpp.

3.3.2.2 bool Ipopt::AlgorithmStrategyObject::ReducedInitialize (const **Journalist** & *jnlst*, const **OptionsList** & *options*, const std::string & *prefix*) [inline]

Reduced version of the Initialize method, which does not require special Ipopt information.

This is useful for algorithm objects that could be used outside Ipopt, such as linear solvers.

Definition at line 92 of file IpAlgStrategy.hpp.

3.3.2.3 virtual bool Ipopt::AlgorithmStrategyObject::InitializeImpl (const **OptionsList** & *options*, const std::string & *prefix*) [protected], [pure virtual]

Implementation of the initialization method that has to be overloaded by for each derived class.

Implemented in Ipopt::Ma97SolverInterface, Ipopt::Ma86SolverInterface, Ipopt::Ma77SolverInterface, Ipopt::↩
SparseSymLinearSolverInterface, Ipopt::PDSystemSolver, Ipopt::IpoptAlgorithm, Ipopt::SymLinearSolver, Ipopt::↩
BacktrackingLineSearch, Ipopt::IterativeSolverTerminationTester, Ipopt::AugSystemSolver, Ipopt::TSymLinearSolver, Ipopt::PDFullSpaceSolver, Ipopt::RestorationPhase, Ipopt::ConvergenceCheck, Ipopt::DefaultIterateInitializer, Ipopt↩
::RestoFilterConvergenceCheck, Ipopt::RestoPenaltyConvergenceCheck, Ipopt::RestoConvergenceCheck, Ipopt::↩
AugRestoSystemSolver, Ipopt::LowRankSSAugSystemSolver, Ipopt::MinC_1NrmRestorationPhase, Ipopt::Inexact↩
PDSolver, Ipopt::StdAugSystemSolver, Ipopt::Ma57TSolverInterface, Ipopt::IterativePardisoSolverInterface, Ipopt::↩
LimMemQuasiNewtonUpdater, Ipopt::RestoIterateInitializer, Ipopt::MumpsSolverInterface, Ipopt::InexactSearchDir↩
Calculator, Ipopt::FilterLSAcceptor, Ipopt::PenaltyLSAcceptor, Ipopt::RestoIterationOutput, Ipopt::RestoRestoration↩
Phase, Ipopt::CGPenaltyLSAcceptor, Ipopt::CGPerturbationHandler, Ipopt::CGSearchDirCalculator, Ipopt::InexactLS↩
Acceptor, Ipopt::InexactTSymScalingMethod, Ipopt::AdaptiveMuUpdate, Ipopt::HessianUpdater, Ipopt::IterateInitializer, Ipopt::IterationOutput, Ipopt::LeastSquareMultipliers, Ipopt::LowRankAugSystemSolver, Ipopt::PDPerturbationHandler, Ipopt::PardisoSolverInterface, Ipopt::WsmpSolverInterface, Ipopt::BacktrackingLSAcceptor, Ipopt::IterativeWsmp↩
SolverInterface, Ipopt::SlackBasedTSymScalingMethod, Ipopt::TSymScalingMethod, Ipopt::InexactNormalStep↩
Calculator, Ipopt::EqMultiplierCalculator, Ipopt::ExactHessianUpdater, Ipopt::GenAugSystemSolver, Ipopt::Monotone↩
MuUpdate, Ipopt::MuOracle, Ipopt::MuUpdate, Ipopt::SearchDirectionCalculator, Ipopt::InexactDoglegNormalStep, Ipopt::InexactNewtonNormalStep, Ipopt::InexactNormalTerminationTester, Ipopt::InexactPDTerminationTester, Ipopt::↩
PDSearchDirCalculator, Ipopt::QualityFunctionMuOracle, Ipopt::WarmStartIterateInitializer, Ipopt::Mc19TSymScaling↩
Method, Ipopt::OptimalityErrorConvergenceCheck, Ipopt::ProbingMuOracle, Ipopt::GenKKTSolverInterface, Ipopt::↩
TDependencyDetector, Ipopt::LoqoMuOracle, Ipopt::OrigIterationOutput, Ipopt::Ma27TSolverInterface, Ipopt::TSym↩
DependencyDetector, and Ipopt::Ma28TDependencyDetector.

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

- IpAlgStrategy.hpp

3.4 Ipopt::AmplOptionsList::AmplOption Class Reference

Ampl Option class, contains name, type and description for an AMPL option.

`#include <AmplTNLP.hpp>`

Inheritance diagram for Ipopt::AmplOptionsList::AmplOption:

3.4.1 Detailed Description

Ampl Option class, contains name, type and description for an AMPL option.

Definition at line 115 of file AmplTNLP.hpp.

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

- AmplTNLP.hpp

3.5 Ipopt::AmplOptionsList Class Reference

Class for storing a number of AMPL options that should be registered to the AMPL Solver library interface.

`#include <AmplTNLP.hpp>`

Inheritance diagram for Ipopt::AmplOptionsList:

Classes

- class AmplOption

    Ampl Option class, contains name, type and description for an AMPL option.
- class PrivatInfo

Public Member Functions

- AmplOptionsList ()

    Default Constructor.
- ~AmplOptionsList ()

    Destructor.
- void AddAmplOption (const std::string ampl_option_name, const std::string ipopt_option_name, AmplOptions←List::AmplOptionType type, const std::string description)

    Adding a new AMPL Option.
- Index NumberOfAmplOptions ()

    Number of AMPL Options.
- void * Keywords (const SmartPtr< OptionsList > &options, SmartPtr< const Journalist > jnlst, void **nerror)

    ASL keywords list for the stored options.

3.5.1 Detailed Description

Class for storing a number of AMPL options that should be registered to the AMPL Solver library interface.

Definition at line 102 of file AmplTNLP.hpp.

3.5.2 Member Function Documentation

3.5.2.1 void* Ipopt::AmplOptionsList::Keywords (const **SmartPtr**< **OptionsList** > & *options*, **SmartPtr**< const **Journalist** > *jnlst*, void ** *nerror*)

ASL keywords list for the stored options.

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

- AmplTNLP.hpp

3.6 Ipopt::AmplSuffixHandler Class Reference

Inheritance diagram for Ipopt::AmplSuffixHandler:

Friends

- class AmplTNLP

    Method called by AmplTNLP to retrieve the suffixes from asl.

3.6.1 Detailed Description

Definition at line 27 of file AmplTNLP.hpp.

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

- AmplTNLP.hpp

3.7 Ipopt::AmplTNLP Class Reference

Ampl Interface.

`#include <AmplTNLP.hpp>`

Inheritance diagram for Ipopt::AmplTNLP:

Public Member Functions

- DECLARE_STD_EXCEPTION (NONPOSITIVE_SCALING_FACTOR)

  Exceptions.
- void set_active_objective (Index obj_no)

  A method for setting the index of the objective function to be considered.
- SmartPtr< AmplSuffixHandler > get_suffix_handler ()

  Method for returning the suffix handler.

Constructors/Destructors

- AmplTNLP (const SmartPtr< const Journalist > &jnlst, const SmartPtr< OptionsList > options, char ∗∗&argv, SmartPtr< AmplSuffixHandler > suffix_handler=NULL, bool allow_discrete=false, SmartPtr< AmplOptionsList > ampl_options_list=NULL, const char ∗ampl_option_string=NULL, const char ∗ampl_invokation_string=N←ULL, const char ∗ampl_banner_string=NULL, std::string ∗nl_file_content=NULL)

  Constructor.
- virtual ∼AmplTNLP ()

  Default destructor.

methods to gather information about the NLP. These

methods are overloaded from TNLP.

See TNLP for their more detailed documentation.

- virtual bool get_nlp_info (Index &n, Index &m, Index &nnz_jac_g, Index &nnz_h_lag, IndexStyleEnum &index←_style)

  returns dimensions of the nlp.
- virtual bool get_var_con_metadata (Index n, StringMetaDataMapType &var_string_md, IntegerMetaData←MapType &var_integer_md, NumericMetaDataMapType &var_numeric_md, Index m, StringMetaDataMapType &con_string_md, IntegerMetaDataMapType &con_integer_md, NumericMetaDataMapType &con_numeric_←md)

  returns names and other meta data for the variables and constraints Overloaded from TNLP
- virtual bool get_bounds_info (Index n, Number ∗x_l, Number ∗x_u, Index m, Number ∗g_l, Number ∗g_u)

  returns bounds of the nlp.
- virtual bool get_constraints_linearity (Index m, LinearityType ∗const_types)

  Returns the constraint linearity.

- virtual bool get_starting_point (Index n, bool init_x, Number ∗x, bool init_z, Number ∗z_L, Number ∗z_U, Index m, bool init_lambda, Number ∗lambda)

  provides a starting point for the nlp variables.
- virtual bool eval_f (Index n, const Number ∗x, bool new_x, Number &obj_value)

  evaluates the objective value for the nlp.
- virtual bool eval_grad_f (Index n, const Number ∗x, bool new_x, Number ∗grad_f)

  evaluates the gradient of the objective for the nlp.
- virtual bool eval_g (Index n, const Number ∗x, bool new_x, Index m, Number ∗g)

  evaluates the constraint residuals for the nlp.
- virtual bool eval_jac_g (Index n, const Number ∗x, bool new_x, Index m, Index nele_jac, Index ∗iRow, Index ∗jCol, Number ∗values)

  specifies the jacobian structure (if values is NULL) and evaluates the jacobian values (if values is not NULL) for the nlp.
- virtual bool eval_h (Index n, const Number ∗x, bool new_x, Number obj_factor, Index m, const Number ∗lambda, bool new_lambda, Index nele_hess, Index ∗iRow, Index ∗jCol, Number ∗values)

  specifies the structure of the hessian of the lagrangian (if values is NULL) and evaluates the values (if values is not NULL).
- virtual bool get_scaling_parameters (Number &obj_scaling, bool &use_x_scaling, Index n, Number ∗x_scaling, bool &use_g_scaling, Index m, Number ∗g_scaling)

  retrieve the scaling parameters for the variables, objective function, and constraints.

Solution Methods

- virtual void finalize_solution (SolverReturn status, Index n, const Number ∗x, const Number ∗z_L, const Number ∗z_U, Index m, const Number ∗g, const Number ∗lambda, Number obj_value, const IpoptData ∗ip_data, IpoptCalculatedQuantities ∗ip_cq)

  This method is called when the algorithm is complete so the TNLP can store/write the solution.

Method for quasi-Newton approximation information.

- virtual Index **get_number_of_nonlinear_variables** ()
- virtual bool **get_list_of_nonlinear_variables** (Index num_nonlin_vars, Index ∗pos_nonlin_vars)

Ampl specific methods

- ASL_pfgh ∗ AmplSolverObject ()

  Return the ampl solver object (ASL∗)
- void write_solution_file (const std::string &message) const

  Write the solution file.
- void get_discrete_info (Index &nlvb_, Index &nlvbi_, Index &nlvc_, Index &nlvci_, Index &nlvo_, Index &nlvoi_, Index &nbv_, Index &niv_) const

  ampl orders the variables like (continuous, binary, integer).

Methods to set meta data for the variables

and constraints.

These values will be passed on to the TNLP in get_var_con_meta_data

- void **set_string_metadata_for_var** (std::string tag, std::vector< std::string > meta_data)
- void **set_integer_metadata_for_var** (std::string tag, std::vector< Index > meta_data)
- void **set_numeric_metadata_for_var** (std::string tag, std::vector< Number > meta_data)
- void **set_string_metadata_for_con** (std::string tag, std::vector< std::string > meta_data)
- void **set_integer_metadata_for_con** (std::string tag, std::vector< Index > meta_data)
- void **set_numeric_metadata_for_con** (std::string tag, std::vector< Number > meta_data)

Additional Inherited Members

3.7.1 Detailed Description

Ampl Interface.

Ampl Interface, implemented as a TNLP.

Definition at line 271 of file AmplTNLP.hpp.

3.7.2 Constructor & Destructor Documentation

3.7.2.1 Ipopt::AmplTNLP::AmplTNLP (const SmartPtr< const Journalist > & *jnlst*, const SmartPtr< OptionsList > *options*, char **& *argv*, SmartPtr< AmplSuffixHandler > *suffix_handler* = NULL, bool *allow_discrete* = false, SmartPtr< AmplOptionsList > *ampl_options_list* = NULL, const char * *ampl_option_string* = NULL, const char * *ampl_invokation_string* = NULL, const char * *ampl_banner_string* = NULL, std::string * *nl_file_content* = NULL)

Constructor.

3.7.3 Member Function Documentation

3.7.3.1 virtual bool Ipopt::AmplTNLP::get_nlp_info (Index & *n*, Index & *m*, Index & *nnz_jac_g*, Index & *nnz_h_lag*, IndexStyleEnum & *index_style*) [virtual]

returns dimensions of the nlp.

Overloaded from TNLP

Implements Ipopt::TNLP.

3.7.3.2 virtual bool Ipopt::AmplTNLP::get_bounds_info (Index *n*, Number * *x_l*, Number * *x_u*, Index *m*, Number * *g_l*, Number * *g_u*) [virtual]

returns bounds of the nlp.

Overloaded from TNLP

Implements Ipopt::TNLP.

3.7.3.3 virtual bool Ipopt::AmplTNLP::get_constraints_linearity (Index *m*, LinearityType * *const_types*) [virtual]

Returns the constraint linearity.

array will be alocated with length n. (default implementation just return false and does not fill the array).

Reimplemented from Ipopt::TNLP.

3.7.3.4 virtual bool Ipopt::AmplTNLP::get_starting_point (Index *n*, bool *init_x*, Number * *x*, bool *init_z*, Number * *z_L*, Number * *z_U*, Index *m*, bool *init_lambda*, Number * *lambda*) [virtual]

provides a starting point for the nlp variables.

Overloaded from TNLP

Implements Ipopt::TNLP.

3.7.3.5 virtual bool Ipopt::AmplTNLP::eval_f (Index *n*, const Number ∗ *x*, bool *new_x*, Number & *obj_value*) [virtual]

evaluates the objective value for the nlp.

Overloaded from TNLP

Implements Ipopt::TNLP.

3.7.3.6 virtual bool Ipopt::AmplTNLP::eval_grad_f (Index *n*, const Number ∗ *x*, bool *new_x*, Number ∗ *grad_f*) [virtual]

evaluates the gradient of the objective for the nlp.

Overloaded from TNLP

Implements Ipopt::TNLP.

3.7.3.7 virtual bool Ipopt::AmplTNLP::eval_g (Index *n*, const Number ∗ *x*, bool *new_x*, Index *m*, Number ∗ *g*) [virtual]

evaluates the constraint residuals for the nlp.

Overloaded from TNLP

Implements Ipopt::TNLP.

3.7.3.8 virtual bool Ipopt::AmplTNLP::eval_jac_g (Index *n*, const Number ∗ *x*, bool *new_x*, Index *m*, Index *nele_jac*, Index ∗ *iRow*, Index ∗ *jCol*, Number ∗ *values*) [virtual]

specifies the jacobian structure (if values is NULL) and evaluates the jacobian values (if values is not NULL) for the nlp.

Overloaded from TNLP

Implements Ipopt::TNLP.

3.7.3.9 virtual bool Ipopt::AmplTNLP::eval_h (Index *n*, const Number ∗ *x*, bool *new_x*, Number *obj_factor*, Index *m*, const Number ∗ *lambda*, bool *new_lambda*, Index *nele_hess*, Index ∗ *iRow*, Index ∗ *jCol*, Number ∗ *values*) [virtual]

specifies the structure of the hessian of the lagrangian (if values is NULL) and evaluates the values (if values is not NULL).

Overloaded from TNLP

Reimplemented from Ipopt::TNLP.

3.7.3.10 virtual bool Ipopt::AmplTNLP::get_scaling_parameters (Number & *obj_scaling*, bool & *use_x_scaling*, Index *n*, Number ∗ *x_scaling*, bool & *use_g_scaling*, Index *m*, Number ∗ *g_scaling*) [virtual]

retrieve the scaling parameters for the variables, objective function, and constraints.

Reimplemented from Ipopt::TNLP.

3.7.3.11 void Ipopt::AmplTNLP::write_solution_file (const std::string & *message*) const

Write the solution file.

This is a wrapper for AMPL's write_sol. TODO Maybe this should be at a different place, or collect the numbers itself?

3.7.3.12 void Ipopt::AmplTNLP::get_discrete_info (Index & *nlvb_*, Index & *nlvbi_*, Index & *nlvc_*, Index & *nlvci_*, Index & *nlvo_*, Index & *nlvoi_*, Index & *nbv_*, Index & *niv_*) const

ampl orders the variables like (continuous, binary, integer).

This method gives the number of binary and integer variables. For details, see Tables 3 and 4 in "Hooking Your Solver to AMPL"

3.7.3.13 void Ipopt::AmplTNLP::set_active_objective (Index *obj_no*)

A method for setting the index of the objective function to be considered.

This method must be called after the constructor, and before anything else is called. It can only be called once, and if there is more than one objective function in the AMPL model, it MUST be called.

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

- AmplTNLP.hpp

3.8 Ipopt::AugRestoSystemSolver Class Reference

Class that converts the an augmented system with compound restoration pieces into a smaller "pivoted" system to be solved with an existing AugSystemSolver.

```
#include <IpAugRestoSystemSolver.hpp>
```

Inheritance diagram for Ipopt::AugRestoSystemSolver:

Public Member Functions

- bool InitializeImpl (const OptionsList &options, const std::string &prefix)

    overloaded from AlgorithmStrategyObject
- virtual ESymSolverStatus Solve (const SymMatrix ∗W, double W_factor, const Vector ∗D_x, double delta_x, const Vector ∗D_s, double delta_s, const Matrix ∗J_c, const Vector ∗D_c, double delta_c, const Matrix ∗J_d, const Vector ∗D_d, double delta_d, const Vector &rhs_x, const Vector &rhs_s, const Vector &rhs_c, const Vector &rhs_d, Vector &sol_x, Vector &sol_s, Vector &sol_c, Vector &sol_d, bool check_NegEVals, Index numberOfNegEVals)

    Translate the augmented system (in the full space of the restoration variables) into the smaller space of the original variables.
- virtual Index NumberOfNegEVals () const

    Returns the number of negative eigenvalues from the original augmented system call.
- virtual bool ProvidesInertia () const

    Query whether inertia is computed by linear solver.
- virtual bool IncreaseQuality ()

    Request to increase quality of solution for next solve.

Constructors/Destructors

- AugRestoSystemSolver (AugSystemSolver &orig_aug_solver, bool skip_orig_aug_solver_init=true)

    Constructor.
- virtual ∼AugRestoSystemSolver ()

    Default destructor.

Additional Inherited Members

3.8.1 Detailed Description

Class that converts the an augmented system with compound restoration pieces into a smaller "pivoted" system to be solved with an existing AugSystemSolver.

This is really a decorator that changes the behavior of the AugSystemSolver to account for the known structure of the restoration phase.

Definition at line 23 of file IpAugRestoSystemSolver.hpp.

3.8.2 Constructor & Destructor Documentation

3.8.2.1 Ipopt::AugRestoSystemSolver::AugRestoSystemSolver (**AugSystemSolver** & *orig_aug_solver,* bool *skip_orig_aug_solver_init* = `true`)

Constructor.

Here, orig_aug_solver is the object for solving the original augmented system. The flag skip_orig_aug_solver_init indicates, if the initialization call (to Initialize) should be skipped; this flag will usually be true, so that the symbolic factorization of the main algorithm will be used.

3.8.3 Member Function Documentation

3.8.3.1 virtual bool Ipopt::AugRestoSystemSolver::ProvidesInertia ( ) const `[inline]`, `[virtual]`

Query whether inertia is computed by linear solver.

Returns true, if linear solver provides inertia.

Implements Ipopt::AugSystemSolver.

Definition at line 84 of file IpAugRestoSystemSolver.hpp.

3.8.3.2 virtual bool Ipopt::AugRestoSystemSolver::IncreaseQuality ( ) `[inline]`, `[virtual]`

Request to increase quality of solution for next solve.

Ask underlying linear solver to increase quality of solution for the next solve (e.g. increase pivot tolerance). Returns false, if this is not possible (e.g. maximal pivot tolerance already used.)

Implements Ipopt::AugSystemSolver.

Definition at line 95 of file IpAugRestoSystemSolver.hpp.

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

- IpAugRestoSystemSolver.hpp

3.9 Ipopt::AugSystemSolver Class Reference

Base class for Solver for the augmented system.

`#include <IpAugSystemSolver.hpp>`

Inheritance diagram for Ipopt::AugSystemSolver:

Public Member Functions

- virtual bool InitializeImpl (const OptionsList &options, const std::string &prefix)=0

  overloaded from AlgorithmStrategyObject
- virtual ESymSolverStatus Solve (const SymMatrix ∗W, double W_factor, const Vector ∗D_x, double delta_x, const Vector ∗D_s, double delta_s, const Matrix ∗J_c, const Vector ∗D_c, double delta_c, const Matrix ∗J_d, const Vector ∗D_d, double delta_d, const Vector &rhs_x, const Vector &rhs_s, const Vector &rhs_c, const Vector &rhs_d, Vector &sol_x, Vector &sol_s, Vector &sol_c, Vector &sol_d, bool check_NegEVals, Index numberOfNegEVals)

  Set up the augmented system and solve it for a given right hand side.
- virtual ESymSolverStatus MultiSolve (const SymMatrix ∗W, double W_factor, const Vector ∗D_x, double delta_x, const Vector ∗D_s, double delta_s, const Matrix ∗J_c, const Vector ∗D_c, double delta_c, const Matrix ∗J_d, const Vector ∗D_d, double delta_d, std::vector< SmartPtr< const Vector > > &rhs_xV, std::vector< SmartPtr< const Vector > > &rhs_sV, std::vector< SmartPtr< const Vector > > &rhs_cV, std::vector< SmartPtr< const Vector > > &rhs_dV, std::vector< SmartPtr< Vector > > &sol_xV, std::vector< SmartPtr< Vector > > &sol_sV, std::vector< SmartPtr< Vector > > &sol_cV, std::vector< SmartPtr< Vector > > &sol_dV, bool check_NegEVals, Index numberOfNegEVals)

  Like Solve, but for multiple right hand sides.
- virtual Index NumberOfNegEVals () const =0

  Number of negative eigenvalues detected during last solve.
- virtual bool ProvidesInertia () const =0

  Query whether inertia is computed by linear solver.
- virtual bool IncreaseQuality ()=0

  Request to increase quality of solution for next solve.

Constructors/Destructors

- AugSystemSolver ()

  Default constructor.
- virtual ∼AugSystemSolver ()

  Default destructor.

Additional Inherited Members

3.9.1 Detailed Description

Base class for Solver for the augmented system.

This is the base class for linear solvers that solve the augmented system, which is defined as

$$\left[\begin{array}{cccc} W+D_x+\delta_x I & 0 & J_c^T & J_d^T \\ 0 & D_s+\delta_s I & 0 & -I \\ J_c & 0 & D_c-\delta_c I & 0 \\ J_d & -I & 0 & D_d-\delta_d I \end{array}\right]\left(\begin{array}{c} sol_x \\ sol_s \\ sol_c \\ sol_d \end{array}\right)=\left(\begin{array}{c} rhs_x \\ rhs_s \\ rhs_c \\ rhs_d \end{array}\right)$$

Since this system might be solved repeatedly for different right hand sides, it is desirable to step the factorization of a direct linear solver if possible.

Definition at line 37 of file IpAugSystemSolver.hpp.

3.9.2 Constructor & Destructor Documentation

3.9.2.1 Ipopt::AugSystemSolver::AugSystemSolver ( ) [inline]

Default constructor.

Definition at line 43 of file IpAugSystemSolver.hpp.

3.9.3 Member Function Documentation

3.9.3.1 virtual ESymSolverStatus Ipopt::AugSystemSolver::Solve (const SymMatrix ∗ *W*, double *W_factor*, const Vector ∗ *D_x*, double *delta_x*, const Vector ∗ *D_s*, double *delta_s*, const Matrix ∗ *J_c*, const Vector ∗ *D_c*, double *delta_c*, const Matrix ∗ *J_d*, const Vector ∗ *D_d*, double *delta_d*, const Vector & *rhs_x*, const Vector & *rhs_s*, const Vector & *rhs_c*, const Vector & *rhs_d*, Vector & *sol_x*, Vector & *sol_s*, Vector & *sol_c*, Vector & *sol_d*, bool *check_NegEVals*, Index *numberOfNegEVals*) [inline], [virtual]

Set up the augmented system and solve it for a given right hand side.

If desired (i.e. if check_NegEVals is true), then the solution is only computed if the number of negative eigenvalues matches numberOfNegEVals.

The return value is the return value of the linear solver object.

Reimplemented in Ipopt::AugRestoSystemSolver, Ipopt::LowRankSSAugSystemSolver, and Ipopt::LowRankAug←SystemSolver.

Definition at line 61 of file IpAugSystemSolver.hpp.

3.9.3.2 virtual ESymSolverStatus Ipopt::AugSystemSolver::MultiSolve (const SymMatrix ∗ *W*, double *W_factor*, const Vector ∗ *D_x*, double *delta_x*, const Vector ∗ *D_s*, double *delta_s*, const Matrix ∗ *J_c*, const Vector ∗ *D_c*, double *delta_c*, const Matrix ∗ *J_d*, const Vector ∗ *D_d*, double *delta_d*, std::vector< SmartPtr< const Vector > > & *rhs_xV*, std::vector< SmartPtr< const Vector > > & *rhs_sV*, std::vector< SmartPtr< const Vector > > & *rhs_cV*, std::vector< SmartPtr< const Vector > > & *rhs_dV*, std::vector< SmartPtr< Vector > > & *sol_xV*, std::vector< SmartPtr< Vector > > & *sol_sV*, std::vector< SmartPtr< Vector > > & *sol_cV*, std::vector< SmartPtr< Vector > > & *sol_dV*, bool *check_NegEVals*, Index *numberOfNegEVals*) [inline], [virtual]

Like Solve, but for multiple right hand sides.

The inheriting class has to be overload at least one of Solve and MultiSolve.

Reimplemented in Ipopt::StdAugSystemSolver, and Ipopt::GenAugSystemSolver.

Definition at line 110 of file IpAugSystemSolver.hpp.

3.9.3.3 virtual Index Ipopt::AugSystemSolver::NumberOfNegEVals ( ) const [pure virtual]

Number of negative eigenvalues detected during last solve.

Returns the number of negative eigenvalues of the most recent factorized matrix. This must not be called if the linear solver does not compute this quantities (see ProvidesInertia).

Implemented in Ipopt::LowRankSSAugSystemSolver, Ipopt::StdAugSystemSolver, Ipopt::AugRestoSystemSolver, Ipopt::LowRankAugSystemSolver, and Ipopt::GenAugSystemSolver.

3.9.3.4 virtual bool Ipopt::AugSystemSolver::ProvidesInertia ( ) const [pure virtual]

Query whether inertia is computed by linear solver.

Returns true, if linear solver provides inertia.

Implemented in Ipopt::AugRestoSystemSolver, Ipopt::LowRankSSAugSystemSolver, Ipopt::StdAugSystemSolver, Ipopt::LowRankAugSystemSolver, and Ipopt::GenAugSystemSolver.

3.9.3.5 virtual bool Ipopt::AugSystemSolver::IncreaseQuality ( ) [pure virtual]

Request to increase quality of solution for next solve.

Ask underlying linear solver to increase quality of solution for the next solve (e.g. increase pivot tolerance). Returns false, if this is not possible (e.g. maximal pivot tolerance already used.)

Implemented in Ipopt::AugRestoSystemSolver, Ipopt::LowRankSSAugSystemSolver, Ipopt::StdAugSystemSolver, Ipopt::LowRankAugSystemSolver, and Ipopt::GenAugSystemSolver.

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

- IpAugSystemSolver.hpp

3.10 Ipopt::BacktrackingLineSearch Class Reference

General implementation of a backtracking line search.

#include <IpBacktrackingLineSearch.hpp>

Inheritance diagram for Ipopt::BacktrackingLineSearch:

Public Member Functions

- virtual bool InitializeImpl (const OptionsList &options, const std::string &prefix)

  InitializeImpl - overloaded from AlgorithmStrategyObject.
- virtual void FindAcceptableTrialPoint ()

  Perform the line search.
- virtual void Reset ()

  Reset the line search.
- virtual void SetRigorousLineSearch (bool rigorous)

  Set flag indicating whether a very rigorous line search should be performed.
- virtual bool CheckSkippedLineSearch ()

  Check if the line search procedure didn't accept a new iterate during the last call of FindAcceptableTrialPoint().
- virtual bool ActivateFallbackMechanism ()

  Activate fallback mechanism.

Constructors/Destructors

- BacktrackingLineSearch (const SmartPtr< BacktrackingLSAcceptor > &acceptor, const SmartPtr< RestorationPhase > &resto_phase, const SmartPtr< ConvergenceCheck > &conv_check)

  Constructor.
- virtual ∼BacktrackingLineSearch ()

  Default destructor.

Static Public Member Functions

- static void RegisterOptions (SmartPtr< RegisteredOptions > roptions)

  Methods for OptionsList.

Additional Inherited Members

3.10.1 Detailed Description

General implementation of a backtracking line search.

This class can be used to perform the filter line search procedure or other procedures. The BacktrackingLSAcceptor is used to determine whether trial points are acceptable (e.g., based on a filter or other methods).

This backtracking line search knows of a restoration phase (which is called when the trial step size becomes too small or no search direction could be computed). It also has the notion of a "soft restoration phase," which uses the regular steps but decides on the acceptability based on other measures than the regular ones (e.g., reduction of the PD error instead of acceptability to a filter mechanism).

Definition at line 36 of file IpBacktrackingLineSearch.hpp.

3.10.2 Constructor & Destructor Documentation

3.10.2.1 Ipopt::BacktrackingLineSearch::BacktrackingLineSearch (const SmartPtr< BacktrackingLSAcceptor > & *acceptor*, const SmartPtr< RestorationPhase > & *resto_phase*, const SmartPtr< ConvergenceCheck > & *conv_check*)

Constructor.

The acceptor implements the acceptance test for the line search. The ConvergenceCheck object is used to determine whether the current iterate is acceptable (for example, the restoration phase is not started if the acceptability level has been reached). If conv_check is NULL, we assume that the current iterate is not acceptable (in the sense of the acceptable_tol option).

3.10.3 Member Function Documentation

3.10.3.1 virtual void Ipopt::BacktrackingLineSearch::FindAcceptableTrialPoint ( ) [virtual]

Perform the line search.

It is assumed that the search direction is computed in the data object.

Implements Ipopt::LineSearch.

3.10.3.2 virtual void Ipopt::BacktrackingLineSearch::Reset ( ) [virtual]

Reset the line search.

This function should be called if all previous information should be discarded when the line search is performed the next time. For example, this method should be called if the barrier parameter is changed.

Implements Ipopt::LineSearch.

3.10.3.3 virtual void Ipopt::BacktrackingLineSearch::SetRigorousLineSearch (bool *rigorous*) [inline], [virtual]

Set flag indicating whether a very rigorous line search should be performed.

If this flag is set to true, the line search algorithm might decide to abort the line search and not to accept a new iterate. If the line search decided not to accept a new iterate, the return value of CheckSkippedLineSearch() is true at the next call. For example, in the non-monotone barrier parameter update procedure, the filter algorithm should not switch to the restoration phase in the free mode; instead, the algorithm should swtich to the fixed mode.

Implements Ipopt::LineSearch.

Definition at line 85 of file IpBacktrackingLineSearch.hpp.

3.10.3.4 virtual bool Ipopt::BacktrackingLineSearch::ActivateFallbackMechanism ( ) [virtual]

Activate fallback mechanism.

Return false, if that is not possible.

Implements Ipopt::LineSearch.

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

- IpBacktrackingLineSearch.hpp

3.11 Ipopt::BacktrackingLSAcceptor Class Reference

Base class for backtracking line search acceptors.

#include <IpBacktrackingLSAcceptor.hpp>

Inheritance diagram for Ipopt::BacktrackingLSAcceptor:

Public Member Functions

- virtual bool InitializeImpl (const OptionsList &options, const std::string &prefix)=0

  InitializeImpl - overloaded from AlgorithmStrategyObject.
- virtual void Reset ()=0

  Reset the acceptor.
- virtual void InitThisLineSearch (bool in_watchdog)=0

  Initialization for the next line search.
- virtual void PrepareRestoPhaseStart ()=0

  Method that is called before the restoration phase is called.
- virtual Number CalculateAlphaMin ()=0

  Method returning the lower bound on the trial step sizes.
- virtual bool CheckAcceptabilityOfTrialPoint (Number alpha_primal)=0

  Method for checking if current trial point is acceptable.
- virtual bool TrySecondOrderCorrection (Number alpha_primal_test, Number &alpha_primal, SmartPtr< IteratesVector > &actual_delta)=0

  Try a second order correction for the constraints.
- virtual bool TryCorrector (Number alpha_primal_test, Number &alpha_primal, SmartPtr< IteratesVector > &actual_delta)=0

  Try higher order corrector (for fast local convergence).

- virtual char UpdateForNextIteration (Number alpha_primal_test)=0

  Method for ending the current line search.
- virtual void StartWatchDog ()=0

  Method for setting internal data if the watchdog procedure is started.
- virtual void StopWatchDog ()=0

  Method for setting internal data if the watchdog procedure is stopped.
- virtual bool RestoredIterate ()

  Method for telling the BacktrackingLineSearch object that a previous iterate has been restored.
- virtual bool NeverRestorationPhase ()

  Method called by BacktrackingLineSearch object to determine whether the restoration phase should never be called.
- virtual bool DoFallback ()

  Method for doing a fallback approach in case no search direction could be computed.
- virtual Number ComputeAlphaForY (Number alpha_primal, Number alpha_dual, SmartPtr< IteratesVector > &delta)

  Method for computing the step for the constraint multipliers in the line search acceptor method.
- virtual bool HasComputeAlphaForY () const

  Method returning true of ComputeAlphaForY is implemented for this acceptor.

Constructors/Destructors

- BacktrackingLSAcceptor ()

  Constructor.
- virtual ∼BacktrackingLSAcceptor ()

  Default destructor.

Static Public Member Functions

- static void RegisterOptions (SmartPtr< RegisteredOptions > roptions)

  Methods for OptionsList.

Additional Inherited Members

3.11.1 Detailed Description

Base class for backtracking line search acceptors.

Definition at line 21 of file IpBacktrackingLSAcceptor.hpp.

3.11.2 Constructor & Destructor Documentation

3.11.2.1 Ipopt::BacktrackingLSAcceptor::BacktrackingLSAcceptor ( ) [inline]

Constructor.

Definition at line 27 of file IpBacktrackingLSAcceptor.hpp.

3.11.3 Member Function Documentation

3.11.3.1 virtual void Ipopt::BacktrackingLSAcceptor::Reset ( ) [pure virtual]

Reset the acceptor.

This function should be called if all previous information should be discarded when the line search is performed the next time. For example, this method should be called if the barrier parameter is changed.

Implemented in Ipopt::FilterLSAcceptor, Ipopt::PenaltyLSAcceptor, Ipopt::CGPenaltyLSAcceptor, and Ipopt::InexactL↩ SAcceptor.

3.11.3.2 virtual void Ipopt::BacktrackingLSAcceptor::InitThisLineSearch (bool *in_watchdog*) [pure virtual]

Initialization for the next line search.

The flag in_watchdog indicates if we are currently in an active watchdog procedure.

Implemented in Ipopt::FilterLSAcceptor, Ipopt::PenaltyLSAcceptor, Ipopt::CGPenaltyLSAcceptor, and Ipopt::InexactL↩ SAcceptor.

3.11.3.3 virtual void Ipopt::BacktrackingLSAcceptor::PrepareRestoPhaseStart ( ) [pure virtual]

Method that is called before the restoration phase is called.

Here, we can set up things that are required in the termination test for the restoration phase, such as augmenting a filter.

Implemented in Ipopt::FilterLSAcceptor, Ipopt::PenaltyLSAcceptor, Ipopt::CGPenaltyLSAcceptor, and Ipopt::InexactL↩ SAcceptor.

3.11.3.4 virtual Number Ipopt::BacktrackingLSAcceptor::CalculateAlphaMin ( ) [pure virtual]

Method returning the lower bound on the trial step sizes.

If the backtracking procedure encounters a trial step size below this value after the first trial set, it swtiches to the (soft) restoration phase.

Implemented in Ipopt::CGPenaltyLSAcceptor, Ipopt::FilterLSAcceptor, Ipopt::PenaltyLSAcceptor, and Ipopt::InexactL↩ SAcceptor.

3.11.3.5 virtual bool Ipopt::BacktrackingLSAcceptor::CheckAcceptabilityOfTrialPoint (Number *alpha_primal*) [pure virtual]

Method for checking if current trial point is acceptable.

It is assumed that the delta information in ip_data is the search direction used in criteria. The primal trial point has to be set before the call. alpha_primal is the step size which is to be used for the test; if it is zero, then this method is called during the soft restoration phase.

Implemented in Ipopt::CGPenaltyLSAcceptor, Ipopt::FilterLSAcceptor, Ipopt::PenaltyLSAcceptor, and Ipopt::InexactL↩ SAcceptor.

3.11.3.6 virtual bool Ipopt::BacktrackingLSAcceptor::TrySecondOrderCorrection (Number *alpha_primal_test,* Number & *alpha_primal,* SmartPtr< IteratesVector > & *actual_delta*) [pure virtual]

Try a second order correction for the constraints.

If the first trial step (with incoming alpha_primal) has been reject, this tries second order corrections, e.g., for the constraints. Here, alpha_primal_test is the step size that has to be used in the filter acceptance tests. On output actual_delta_ has been set to the step including the second order correction if it has been accepted, otherwise it is unchanged. If the SOC step has been accepted, alpha_primal has the fraction-to-the-boundary value for the SOC step on output. The return value is true, if a SOC step has been accepted.

Implemented in Ipopt::CGPenaltyLSAcceptor, Ipopt::FilterLSAcceptor, Ipopt::PenaltyLSAcceptor, and Ipopt::InexactLSAcceptor.

3.11.3.7 virtual bool Ipopt::BacktrackingLSAcceptor::TryCorrector (Number *alpha_primal_test,* Number & *alpha_primal,* SmartPtr< IteratesVector > & *actual_delta*) [pure virtual]

Try higher order corrector (for fast local convergence).

In contrast to a second order correction step, which tries to make an unacceptable point acceptable by improving constraint violation, this corrector step is tried even if the regular primal-dual step is acceptable.

Implemented in Ipopt::CGPenaltyLSAcceptor, Ipopt::FilterLSAcceptor, Ipopt::PenaltyLSAcceptor, and Ipopt::InexactLSAcceptor.

3.11.3.8 virtual char Ipopt::BacktrackingLSAcceptor::UpdateForNextIteration (Number *alpha_primal_test*) [pure virtual]

Method for ending the current line search.

When it is called, the internal data should be updates, e.g., the filter might be augmented. alpha_primal_test is the value of alpha that has been used for in the acceptence test ealier. Return value is a character for the info_alpha_primal_char field in IpData.

Implemented in Ipopt::CGPenaltyLSAcceptor, Ipopt::FilterLSAcceptor, Ipopt::PenaltyLSAcceptor, and Ipopt::InexactLSAcceptor.

3.11.3.9 virtual void Ipopt::BacktrackingLSAcceptor::StartWatchDog ( ) [pure virtual]

Method for setting internal data if the watchdog procedure is started.

Implemented in Ipopt::CGPenaltyLSAcceptor, Ipopt::FilterLSAcceptor, Ipopt::PenaltyLSAcceptor, and Ipopt::InexactLSAcceptor.

3.11.3.10 virtual void Ipopt::BacktrackingLSAcceptor::StopWatchDog ( ) [pure virtual]

Method for setting internal data if the watchdog procedure is stopped.

Implemented in Ipopt::CGPenaltyLSAcceptor, Ipopt::FilterLSAcceptor, Ipopt::PenaltyLSAcceptor, and Ipopt::InexactLSAcceptor.

3.11.3.11 virtual bool Ipopt::BacktrackingLSAcceptor::RestoredIterate ( ) [inline], [virtual]

Method for telling the BacktrackingLineSearch object that a previous iterate has been restored.

Reimplemented in Ipopt::CGPenaltyLSAcceptor.

Definition at line 115 of file IpBacktrackingLSAcceptor.hpp.

3.11.3.12 virtual bool Ipopt::BacktrackingLSAcceptor::NeverRestorationPhase ( ) [inline], [virtual]

Method called by BacktrackingLineSearch object to determine whether the restoration phase should never be called.

Reimplemented in Ipopt::CGPenaltyLSAcceptor.

Definition at line 122 of file IpBacktrackingLSAcceptor.hpp.

3.11.3.13 virtual bool Ipopt::BacktrackingLSAcceptor::DoFallback ( ) [inline], [virtual]

Method for doing a fallback approach in case no search direction could be computed.

If no such fall back option is available, return false. If possible, the new point is assumed to be in the trial fields of IpData now.

Reimplemented in Ipopt::CGPenaltyLSAcceptor.

Definition at line 131 of file IpBacktrackingLSAcceptor.hpp.

3.11.3.14 virtual Number Ipopt::BacktrackingLSAcceptor::ComputeAlphaForY (Number *alpha_primal*, Number *alpha_dual*, SmartPtr< IteratesVector > & *delta*) [inline], [virtual]

Method for computing the step for the constraint multipliers in the line search acceptor method.

This is activated with choosing the option alpha_for_y=acceptor

Reimplemented in Ipopt::InexactLSAcceptor.

Definition at line 139 of file IpBacktrackingLSAcceptor.hpp.

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

- IpBacktrackingLSAcceptor.hpp

3.12 Ipopt::CachedResults< T > Class Template Reference

Cache Priority Enum.

`#include <IpCachedResults.hpp>`

Inheritance diagram for Ipopt::CachedResults< T >:

Public Member Functions

- bool InvalidateResult (const std::vector< const TaggedObject ∗ > &dependents, const std::vector< Number > &scalar_dependents)

  Invalidates the result for given dependencies.
- void Clear ()

  Invalidates all cached results.
- void Clear (Int max_cache_size)

  Invalidate all cached results and changes max_cache_size.

Constructors and Destructors.

- CachedResults (Int max_cache_size)

Constructor, where max_cache_size is the maximal number of results that should be cached.

- virtual ~CachedResults ()

  Destructor.

Generic methods for adding and retrieving cached results.

- void AddCachedResult (const T &result, const std::vector< const TaggedObject ∗ > &dependents, const std::vector< Number > &scalar_dependents)

  Generic method for adding a result to the cache, given a std::vector of TaggesObjects and a std::vector of Numbers.
- bool GetCachedResult (T &retResult, const std::vector< const TaggedObject ∗ > &dependents, const std::vector< Number > &scalar_dependents) const

  Generic method for retrieving a cached results, given the dependencies as a std::vector of TaggesObjects and a std::vector of Numbers.
- void AddCachedResult (const T &result, const std::vector< const TaggedObject ∗ > &dependents)

  Method for adding a result, providing only a std::vector of TaggedObjects.
- bool GetCachedResult (T &retResult, const std::vector< const TaggedObject ∗ > &dependents) const

  Method for retrieving a cached result, providing only a std::vector of TaggedObjects.

Pointer-based methods for adding and retrieving cached

results, providing dependencies explicitly.

- void AddCachedResult1Dep (const T &result, const TaggedObject ∗dependent1)

  Method for adding a result to the cache, proving one dependency as a TaggedObject explicitly.
- bool GetCachedResult1Dep (T &retResult, const TaggedObject ∗dependent1)

  Method for retrieving a cached result, proving one dependency as a TaggedObject explicitly.
- void AddCachedResult2Dep (const T &result, const TaggedObject ∗dependent1, const TaggedObject ∗dependent2)

  Method for adding a result to the cache, proving two dependencies as a TaggedObject explicitly.
- bool GetCachedResult2Dep (T &retResult, const TaggedObject ∗dependent1, const TaggedObject ∗dependent2)

  Method for retrieving a cached result, proving two dependencies as a TaggedObject explicitly.
- void AddCachedResult3Dep (const T &result, const TaggedObject ∗dependent1, const TaggedObject ∗dependent2, const TaggedObject ∗dependent3)

  Method for adding a result to the cache, proving three dependencies as a TaggedObject explicitly.
- bool GetCachedResult3Dep (T &retResult, const TaggedObject ∗dependent1, const TaggedObject ∗dependent2, const TaggedObject ∗dependent3)

  Method for retrieving a cached result, proving three dependencies as a TaggedObject explicitly.

Pointer-free version of the Add and Get methods

- bool **GetCachedResult1Dep** (T &retResult, const TaggedObject &dependent1)
- bool **GetCachedResult2Dep** (T &retResult, const TaggedObject &dependent1, const TaggedObject &dependent2)
- bool **GetCachedResult3Dep** (T &retResult, const TaggedObject &dependent1, const TaggedObject &dependent2, const TaggedObject &dependent3)
- void **AddCachedResult1Dep** (const T &result, const TaggedObject &dependent1)
- void **AddCachedResult2Dep** (const T &result, const TaggedObject &dependent1, const TaggedObject &dependent2)
- void **AddCachedResult3Dep** (const T &result, const TaggedObject &dependent1, const TaggedObject &dependent2, const TaggedObject &dependent3)

3.12.1 Detailed Description

template<class T>class Ipopt::CachedResults< T >

Cache Priority Enum.

Templated class for Cached Results. This class stores up to a given number of "results", entities that are stored here together with identifiers, that can be used to later retrieve the information again.

Typically, T is a SmartPtr for some calculated quantity that should be stored (such as a Vector). The identifiers (or dependencies) are a (possibly varying) number of Tags from TaggedObjects, and a number of Numbers. Results are added to the cache using the AddCachedResults methods, and the can be retrieved with the GetCachedResults methods. The second set of methods checks whether a result has been cached for the given identifiers. If a corresponding result is found, a copy of it is returned and the method evaluates to true, otherwise it evaluates to false.

Note that cached results can become "stale", namely when a TaggedObject that is used to identify this CachedResult is changed. When this happens, the cached result can never be asked for again, so that there is no point in storing it any longer. For this purpose, a cached result, which is stored as a DependentResult, inherits off an Observer. This Observer retrieves notification whenever a TaggedObject dependency has changed. Stale results are later removed from the cache.

Definition at line 70 of file IpCachedResults.hpp.

3.12.2 Constructor & Destructor Documentation

3.12.2.1 template<class T > **Ipopt::CachedResults< T >::CachedResults** (Int *max_cache_size*)

Constructor, where max_cache_size is the maximal number of results that should be cached.

If max_cache_size is negative, we allow an infinite amount of cache.

Definition at line 476 of file IpCachedResults.hpp.

3.12.3 Member Function Documentation

3.12.3.1 template<class T > bool **Ipopt::CachedResults< T >::InvalidateResult** (const std::vector< const **TaggedObject** * > & *dependents*, const std::vector< Number > & *scalar_dependents*)

Invalidates the result for given dependencies.

Sets the stale flag for the corresponding cached result to true if it is found. Returns true, if the result was found.

Definition at line 685 of file IpCachedResults.hpp.

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

- IpCachedResults.hpp

3.13 Ipopt::CGPenaltyCq Class Reference

Class for all Chen-Goldfarb penalty method specific calculated quantities.

`#include <IpCGPenaltyCq.hpp>`

Inheritance diagram for Ipopt::CGPenaltyCq:

Public Member Functions

- bool Initialize (const Journalist &jnlst, const OptionsList &options, const std::string &prefix)

  This method must be called to initialize the global algorithmic parameters.

Constructors/Destructors

- CGPenaltyCq (IpoptNLP ∗ip_nlp, IpoptData ∗ip_data, IpoptCalculatedQuantities ∗ip_cg)

  Constructor.
- virtual ∼CGPenaltyCq ()

  Default destructor.

Methods for the Chen-Goldfarb line search

- Number curr_jac_cd_norm (Index nrm_type)

  Compute ||delta_c, delta_d||_infty.
- Number curr_scaled_y_Amax ()

  Compute gradient scaling based y->Amax.
- Number curr_added_y_nrm2 ()

  Compute the 2-norm of y plus delta y.
- Number curr_penalty_function ()

  Method for the penalty function at current point.
- Number trial_penalty_function ()

  Method for the penalty function at trial point.
- Number curr_direct_deriv_penalty_function ()

  Method for the directional derivative of the penalty function at current point with current step in delta.
- Number curr_fast_direct_deriv_penalty_function ()

  Method for the directional derivative of the penalty function at current point with current "fast" step in delta_cgpen.
- Number dT_times_barH_times_d ()

  Quality of d^T Aug(H) d.
- Number curr_cg_pert_fact ()

  Method for the current value for the perturbation factor for the Chen-Goldfarb method.
- Number compute_curr_cg_penalty (const Number)

  Method for choose line search penalty parameter.
- Number compute_curr_cg_penalty_scale ()

  Method for choose penalty parameters for scaling the KKT system.

Static Public Member Functions

- static void RegisterOptions (const SmartPtr< RegisteredOptions > &roptions)

  Methods for IpoptType.

3.13.1 Detailed Description

Class for all Chen-Goldfarb penalty method specific calculated quantities.

Definition at line 22 of file IpCGPenaltyCq.hpp.

3.13.2 Member Function Documentation

3.13.2.1 bool Ipopt::CGPenaltyCq::Initialize (const Journalist & *jnlst*, const OptionsList & *options*, const std::string & *prefix*) [virtual]

This method must be called to initialize the global algorithmic parameters.

The parameters are taken from the OptionsList object.

Implements Ipopt::IpoptAdditionalCq.

3.13.2.2 Number Ipopt::CGPenaltyCq::curr_cg_pert_fact ( )

Method for the current value for the perturbation factor for the Chen-Goldfarb method.

The factor is computed as 2-norm of the constraints devided by the current penbalty parameter

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

- IpCGPenaltyCq.hpp

3.14 Ipopt::CGPenaltyData Class Reference

Class to organize all the additional data required by the Chen-Goldfarb penalty function algorithm.

`#include <IpCGPenaltyData.hpp>`

Inheritance diagram for Ipopt::CGPenaltyData:

Public Member Functions

- bool Initialize (const Journalist &jnlst, const OptionsList &options, const std::string &prefix)
  This method must be called to initialize the global algorithmic parameters.
- bool InitializeDataStructures ()
  Initialize Data Structures.
- SmartPtr< const IteratesVector > delta_cgpen () const
  Delta for the Chen-Goldfarb search direction.
- void set_delta_cgpen (SmartPtr< IteratesVector > &delta_pen)
  Set the delta_cgpen - like the trial point, this method copies the pointer for efficiency (no copy and to keep cache tags the same) so after you call set, you cannot modify the data.
- void set_delta_cgpen (SmartPtr< const IteratesVector > &delta_pen)
  Set the delta_cgpen - like the trial point, this method copies the pointer for efficiency (no copy and to keep cache tags the same) so after you call set, you cannot modify the data.
- SmartPtr< const IteratesVector > delta_cgfast () const
  Delta for the fast Chen-Goldfarb search direction.
- void set_delta_cgfast (SmartPtr< IteratesVector > &delta_fast)
  Set the delta_cgpen - like the trial point, this method copies the pointer for efficiency (no copy and to keep cache tags the same) so after you call set, you cannot modify the data.

Constructors/Destructors

- CGPenaltyData ()

Constructor.
- ∼CGPenaltyData ()

  Default destructor.

Chen-Goldfarb step2. Those fields can be used to store

directions related to the Chen-Goldfarb algorithm

- bool **HaveCgPenDeltas** () const
- void **SetHaveCgPenDeltas** (bool have_cgpen_deltas)
- bool **HaveCgFastDeltas** () const
- void **SetHaveCgFastDeltas** (bool have_cgfast_deltas)

Public Methods for updating iterates

- void AcceptTrialPoint ()

  Set the current iterate values from the trial values.

3.14.1 Detailed Description

Class to organize all the additional data required by the Chen-Goldfarb penalty function algorithm.

Definition at line 22 of file IpCGPenaltyData.hpp.

3.14.2 Member Function Documentation

3.14.2.1 bool Ipopt::CGPenaltyData::Initialize (const Journalist & *jnlst,* const OptionsList & *options,* const std::string & *prefix*) [virtual]

This method must be called to initialize the global algorithmic parameters.

The parameters are taken from the OptionsList object.

Implements Ipopt::IpoptAdditionalData.

3.14.2.2 void Ipopt::CGPenaltyData::set_delta_cgpen (SmartPtr< const IteratesVector > & *delta_pen*) [inline]

Set the delta_cgpen - like the trial point, this method copies the pointer for efficiency (no copy and to keep cache tags the same) so after you call set, you cannot modify the data.

This is the version that is happy with a pointer to const IteratesVector.

Definition at line 299 of file IpCGPenaltyData.hpp.

3.14.2.3 void Ipopt::CGPenaltyData::AcceptTrialPoint ( ) [virtual]

Set the current iterate values from the trial values.

Implements Ipopt::IpoptAdditionalData.

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

- IpCGPenaltyData.hpp

3.15 Ipopt::CGPenaltyLSAcceptor Class Reference

Line search acceptor, based on the Chen-Goldfarb penalty function approach.

`#include <IpCGPenaltyLSAcceptor.hpp>`

Inheritance diagram for Ipopt::CGPenaltyLSAcceptor:

Public Member Functions

- virtual bool InitializeImpl (const OptionsList &options, const std::string &prefix)

  InitializeImpl - overloaded from AlgorithmStrategyObject.
- virtual void Reset ()

  Reset the acceptor.
- virtual void InitThisLineSearch (bool in_watchdog)

  Initialization for the next line search.
- virtual void PrepareRestoPhaseStart ()

  Method that is called before the restoration phase is called.
- virtual Number CalculateAlphaMin ()

  Method returning the lower bound on the trial step sizes.
- virtual bool CheckAcceptabilityOfTrialPoint (Number alpha_primal)

  Method for checking if current trial point is acceptable.
- virtual bool TrySecondOrderCorrection (Number alpha_primal_test, Number &alpha_primal, SmartPtr< IteratesVector > &actual_delta)

  Try a second order correction for the constraints.
- virtual bool TryCorrector (Number alpha_primal_test, Number &alpha_primal, SmartPtr< IteratesVector > &actual_delta)

  Try higher order corrector (for fast local convergence).
- virtual char UpdateForNextIteration (Number alpha_primal_test)

  Method for ending the current line search.
- virtual void StartWatchDog ()

  Method for setting internal data if the watchdog procedure is started.
- virtual void StopWatchDog ()

  Method for setting internal data if the watchdog procedure is stopped.
- virtual bool RestoredIterate ()

  Method for telling the BacktrackingLineSearch object that a previous iterate has been restored.
- virtual bool NeverRestorationPhase ()

  Method for telling the BacktrackingLineSearch object that the restoration is not needed.
- virtual bool DoFallback ()

  Method for doing a fallback approach in case no search direction could be computed.

Constructors/Destructors

- CGPenaltyLSAcceptor (const SmartPtr< PDSystemSolver > &pd_solver)

  Constructor.
- virtual ~CGPenaltyLSAcceptor ()

  Default destructor.

Static Public Member Functions

- static void RegisterOptions (SmartPtr< RegisteredOptions > roptions)

  Methods for OptionsList.

Additional Inherited Members

3.15.1 Detailed Description

Line search acceptor, based on the Chen-Goldfarb penalty function approach.

Definition at line 23 of file IpCGPenaltyLSAcceptor.hpp.

3.15.2 Constructor & Destructor Documentation

3.15.2.1 Ipopt::CGPenaltyLSAcceptor::CGPenaltyLSAcceptor (const SmartPtr< PDSystemSolver > & *pd_solver*)

Constructor.

The PDSystemSolver object only needs to be provided (i.e. not NULL) if second order correction or corrector steps are to be used.

3.15.3 Member Function Documentation

3.15.3.1 virtual void Ipopt::CGPenaltyLSAcceptor::Reset ( ) [virtual]

Reset the acceptor.

This function should be called if all previous information should be discarded when the line search is performed the next time. For example, this method should be called if the barrier parameter is changed.

Implements Ipopt::BacktrackingLSAcceptor.

3.15.3.2 virtual void Ipopt::CGPenaltyLSAcceptor::InitThisLineSearch (bool *in_watchdog*) [virtual]

Initialization for the next line search.

The flag in_watchdog indicates if we are currently in an active watchdog procedure.

Implements Ipopt::BacktrackingLSAcceptor.

3.15.3.3 virtual void Ipopt::CGPenaltyLSAcceptor::PrepareRestoPhaseStart ( ) [virtual]

Method that is called before the restoration phase is called.

Here, we can set up things that are required in the termination test for the restoration phase.

Implements Ipopt::BacktrackingLSAcceptor.

3.15.3.4 virtual Number Ipopt::CGPenaltyLSAcceptor::CalculateAlphaMin ( ) [virtual]

Method returning the lower bound on the trial step sizes.

If the backtracking procedure encounters a trial step size below this value after the first trial set, it swtiches to the (soft) restoration phase.

Implements Ipopt::BacktrackingLSAcceptor.

3.15.3.5 virtual bool Ipopt::CGPenaltyLSAcceptor::CheckAcceptabilityOfTrialPoint (Number *alpha_primal*) [virtual]

Method for checking if current trial point is acceptable.

It is assumed that the delta information in ip_data is the search direction used in criteria. The primal trial point has to be set before the call.

Implements Ipopt::BacktrackingLSAcceptor.

3.15.3.6 virtual bool Ipopt::CGPenaltyLSAcceptor::TrySecondOrderCorrection (Number *alpha_primal_test*, Number & *alpha_primal*, SmartPtr< IteratesVector > & *actual_delta*) [virtual]

Try a second order correction for the constraints.

If the first trial step (with incoming alpha_primal) has been reject, this tries up to max_soc_ second order corrections for the constraints. Here, alpha_primal_test is the step size that has to be used in the merit function acceptance tests. On output actual_delta_ has been set to the step including the second order correction if it has been accepted, otherwise it is unchanged. If the SOC step has been accepted, alpha_primal has the fraction-to-the-boundary value for the SOC step on output. The return value is true, if a SOC step has been accepted.

Implements Ipopt::BacktrackingLSAcceptor.

3.15.3.7 virtual bool Ipopt::CGPenaltyLSAcceptor::TryCorrector (Number *alpha_primal_test*, Number & *alpha_primal*, SmartPtr< IteratesVector > & *actual_delta*) [virtual]

Try higher order corrector (for fast local convergence).

In contrast to a second order correction step, which tries to make an unacceptable point acceptable by improving constraint violation, this corrector step is tried even if the regular primal-dual step is acceptable.

Implements Ipopt::BacktrackingLSAcceptor.

3.15.3.8 virtual char Ipopt::CGPenaltyLSAcceptor::UpdateForNextIteration (Number *alpha_primal_test*) [virtual]

Method for ending the current line search.

When it is called, the internal data should be updates, e.g., the penalty parameter might be updated. alpha_primal_test is the value of alpha that has been used for in the acceptence test ealier.

Implements Ipopt::BacktrackingLSAcceptor.

3.15.3.9 virtual void Ipopt::CGPenaltyLSAcceptor::StartWatchDog ( ) [virtual]

Method for setting internal data if the watchdog procedure is started.

Implements Ipopt::BacktrackingLSAcceptor.

3.15.3.10 virtual void Ipopt::CGPenaltyLSAcceptor::StopWatchDog ( ) [virtual]

Method for setting internal data if the watchdog procedure is stopped.

Implements Ipopt::BacktrackingLSAcceptor.

3.15.3.11 virtual bool Ipopt::CGPenaltyLSAcceptor::RestoredIterate ( ) [virtual]

Method for telling the BacktrackingLineSearch object that a previous iterate has been restored.

Reimplemented from Ipopt::BacktrackingLSAcceptor.

3.15.3.12 virtual bool Ipopt::CGPenaltyLSAcceptor::DoFallback ( ) [virtual]

Method for doing a fallback approach in case no search direction could be computed.

If no such fall back option is available, return false.

Reimplemented from Ipopt::BacktrackingLSAcceptor.

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

- IpCGPenaltyLSAcceptor.hpp

3.16 Ipopt::CGPerturbationHandler Class Reference

Class for handling the perturbation factors delta_x, delta_s, delta_c, and delta_d in the primal dual system.

`#include <IpCGPerturbationHandler.hpp>`

Inheritance diagram for Ipopt::CGPerturbationHandler:

Public Member Functions

- virtual bool InitializeImpl (const OptionsList &options, const std::string &prefix)

    Implementation of the initialization method that has to be overloaded by for each derived class.
- bool ConsiderNewSystem (Number &delta_x, Number &delta_s, Number &delta_c, Number &delta_d)

    This method must be called for each new matrix, and before any other method for generating perturbation factors.
- bool PerturbForSingularity (Number &delta_x, Number &delta_s, Number &delta_c, Number &delta_d)

    This method returns pertubation factors for the case when the most recent factorization resulted in a singular matrix.
- bool PerturbForWrongInertia (Number &delta_x, Number &delta_s, Number &delta_c, Number &delta_d)

    This method returns pertubation factors for the case when the most recent factorization resulted in a matrix with an incorrect number of negative eigenvalues.
- void CurrentPerturbation (Number &delta_x, Number &delta_s, Number &delta_c, Number &delta_d)

    Just return the perturbation values that have been determined most recently.

Constructors/Destructors

- CGPerturbationHandler ()

    Default Constructor.
- virtual ~CGPerturbationHandler ()

    Default destructor.

Static Public Member Functions

- static void RegisterOptions (SmartPtr< RegisteredOptions > roptions)

  Methods for IpoptType.

Additional Inherited Members

3.16.1 Detailed Description

Class for handling the perturbation factors delta_x, delta_s, delta_c, and delta_d in the primal dual system.

This class is used by the PDFullSpaceSolver to handle the cases where the primal-dual system is singular or has the wrong inertia. The perturbation factors are obtained based on simple heuristics, taking into account the size of previous perturbations.

Definition at line 25 of file IpCGPerturbationHandler.hpp.

3.16.2 Member Function Documentation

3.16.2.1 virtual bool Ipopt::CGPerturbationHandler::InitializeImpl (const OptionsList & *options*, const std::string & *prefix*) [virtual]

Implementation of the initialization method that has to be overloaded by for each derived class.

Reimplemented from Ipopt::PDPerturbationHandler.

3.16.2.2 bool Ipopt::CGPerturbationHandler::ConsiderNewSystem (Number & *delta_x*, Number & *delta_s*, Number & *delta_c*, Number & *delta_d*) [virtual]

This method must be called for each new matrix, and before any other method for generating perturbation factors.

Usually, the returned perturbation factors are zero, but if the system is thought to be structurally singular, they might be positive. If the return value is false, no suitable perturbation could be found.

Reimplemented from Ipopt::PDPerturbationHandler.

3.16.2.3 bool Ipopt::CGPerturbationHandler::PerturbForSingularity (Number & *delta_x*, Number & *delta_s*, Number & *delta_c*, Number & *delta_d*) [virtual]

This method returns pertubation factors for the case when the most recent factorization resulted in a singular matrix.

If the return value is false, no suitable perturbation could be found.

Reimplemented from Ipopt::PDPerturbationHandler.

3.16.2.4 bool Ipopt::CGPerturbationHandler::PerturbForWrongInertia (Number & *delta_x*, Number & *delta_s*, Number & *delta_c*, Number & *delta_d*) [virtual]

This method returns pertubation factors for the case when the most recent factorization resulted in a matrix with an incorrect number of negative eigenvalues.

If the return value is false, no suitable perturbation could be found.

Reimplemented from Ipopt::PDPerturbationHandler.

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

- IpCGPerturbationHandler.hpp

3.17 Ipopt::CGSearchDirCalculator Class Reference

Implementation of the search direction calculator that computes the Chen-Goldfarb step for the current barrier and penalty parameter.

`#include <IpCGSearchDirCalc.hpp>`

Inheritance diagram for Ipopt::CGSearchDirCalculator:

Public Member Functions

- virtual bool InitializeImpl (const OptionsList &options, const std::string &prefix)
  overloaded from AlgorithmStrategyObject
- virtual bool ComputeSearchDirection ()
  Method for computing the search direction.

Constructors/Destructors

- CGSearchDirCalculator (const SmartPtr< PDSystemSolver > &pd_solver)
  Constructor.
- virtual ∼CGSearchDirCalculator ()
  Default destructor.

Static Public Member Functions

- static void RegisterOptions (SmartPtr< RegisteredOptions > roptions)
  Methods for IpoptType.

Additional Inherited Members

3.17.1 Detailed Description

Implementation of the search direction calculator that computes the Chen-Goldfarb step for the current barrier and penalty parameter.

Definition at line 25 of file IpCGSearchDirCalc.hpp.

3.17.2 Member Function Documentation

3.17.2.1 virtual bool Ipopt::CGSearchDirCalculator::ComputeSearchDirection ( ) [virtual]

Method for computing the search direction.

If the penalty paraemeter has not yet been initialized, it is initialized now. The computed direction is stored in Ip←Data().delta().

Implements Ipopt::SearchDirectionCalculator.

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

- IpCGSearchDirCalc.hpp

3.18 Ipopt::CompoundMatrix Class Reference

Class for Matrices consisting of other matrices.

`#include <IpCompoundMatrix.hpp>`

Inheritance diagram for Ipopt::CompoundMatrix:

Public Member Functions

- void SetComp (Index irow, Index jcol, const Matrix &matrix)

  Method for setting an individual component at position (irow, icol) in the compound matrix.
- void SetCompNonConst (Index irow, Index jcol, Matrix &matrix)

  Method to set a non-const Matrix entry.
- void CreateBlockFromSpace (Index irow, Index jcol)

  Method to create a new matrix from the space for this block.
- SmartPtr< const Matrix > GetComp (Index irow, Index jcol) const

  Method for retrieving one block from the compound matrix as a const Matrix.
- SmartPtr< Matrix > GetCompNonConst (Index irow, Index jcol)

  Method for retrieving one block from the compound matrix as a non-const Matrix.
- Index NComps_Rows () const

  Number of block rows of this compound matrix.
- Index NComps_Cols () const

  Number of block colmuns of this compound matrix.

Constructors / Destructors

- CompoundMatrix (const CompoundMatrixSpace ∗owner_space)

  Constructor, taking the owner_space.
- virtual ∼CompoundMatrix ()

  Destructor.

Protected Member Functions

Methods overloaded from Matrix

- virtual void MultVectorImpl (Number alpha, const Vector &x, Number beta, Vector &y) const

  Matrix-vector multiply.
- virtual void TransMultVectorImpl (Number alpha, const Vector &x, Number beta, Vector &y) const

  Matrix(transpose) vector multiply.
- virtual void AddMSinvZImpl (Number alpha, const Vector &S, const Vector &Z, Vector &X) const

  X = beta∗X + alpha∗(Matrix S^{-1} Z).
- virtual void SinvBlrmZMTdBrImpl (Number alpha, const Vector &S, const Vector &R, const Vector &Z, const Vector &D, Vector &X) const

X = S^{-1} (r + alpha∗Z∗M^ Td).

- virtual bool HasValidNumbersImpl () const

  Method for determining if all stored numbers are valid (i.e., no Inf or Nan).
- virtual void ComputeRowAMaxImpl (Vector &rows_norms, bool init) const

  Compute the max-norm of the rows in the matrix.
- virtual void ComputeColAMaxImpl (Vector &cols_norms, bool init) const

  Compute the max-norm of the columns in the matrix.
- virtual void PrintImpl (const Journalist &jnlst, EJournalLevel level, EJournalCategory category, const std::string &name, Index indent, const std::string &prefix) const

  Print detailed information about the matrix.

Additional Inherited Members

3.18.1 Detailed Description

Class for Matrices consisting of other matrices.

This matrix is a matrix that consists of zero, one or more Matrices's which are arranged like this: $M_{\rm compound} = \left(\begin{array}{cccc}M_{00} & M_{01} & \ldots & M_{0,{\rm ncomp\_cols}-1} \\ \dots & & & \dots \\ M_{{\rm ncomp\_rows}-1,0} & M_{{\rm ncomp\_rows}-1,1} & \dots & M_{{\rm ncomp\_rows}-1,{\rm ncomp\_cols}-1}\end{array}\right)$. The individual components can be associated to different MatrixSpaces. The individual components can also be const and non-const Matrices. If a component is not set (i.e., it's pointer is NULL), then this components is treated like a zero-matrix of appropriate dimensions.

Definition at line 34 of file IpCompoundMatrix.hpp.

3.18.2 Constructor & Destructor Documentation

3.18.2.1 Ipopt::CompoundMatrix::CompoundMatrix (const **CompoundMatrixSpace** ∗ *owner_space*)

Constructor, taking the owner_space.

The owner_space has to be defined, so that at each block row and column contain at least one non-NULL component. The individual components can be set afterwards with the SeteComp and SetCompNonConst methods.

3.18.3 Member Function Documentation

3.18.3.1 void Ipopt::CompoundMatrix::SetComp (Index *irow*, Index *jcol*, const **Matrix** & *matrix*)

Method for setting an individual component at position (irow, icol) in the compound matrix.

The counting of indices starts at 0.

3.18.3.2 SmartPtr<Matrix> Ipopt::CompoundMatrix::GetCompNonConst (Index *irow*, Index *jcol*) [inline]

Method for retrieving one block from the compound matrix as a non-const Matrix.

Note that calling this method with mark the CompoundMatrix as changed. Therefore, only use this method if you are intending to change the Matrix that you receive.

Definition at line 76 of file IpCompoundMatrix.hpp.

3.18.3.3 Index Ipopt::CompoundMatrix::NComps_Rows ( ) const [inline]

Number of block rows of this compound matrix.

Definition at line 305 of file IpCompoundMatrix.hpp.

3.18.3.4 Index Ipopt::CompoundMatrix::NComps_Cols ( ) const [inline]

Number of block colmuns of this compound matrix.

Definition at line 311 of file IpCompoundMatrix.hpp.

3.18.3.5 virtual void Ipopt::CompoundMatrix::MultVectorImpl (Number *alpha*, const Vector & *x*, Number *beta*, Vector & *y*) const [protected], [virtual]

Matrix-vector multiply.

Computes y = alpha ∗ Matrix ∗ x + beta ∗ y

Implements Ipopt::Matrix.

3.18.3.6 virtual void Ipopt::CompoundMatrix::TransMultVectorImpl (Number *alpha*, const Vector & *x*, Number *beta*, Vector & *y*) const [protected], [virtual]

Matrix(transpose) vector multiply.

Computes y = alpha ∗ Matrix^T ∗ x + beta ∗ y

Implements Ipopt::Matrix.

3.18.3.7 virtual void Ipopt::CompoundMatrix::AddMSinvZImpl (Number *alpha*, const Vector & *S*, const Vector & *Z*, Vector & *X*) const [protected], [virtual]

X = beta∗X + alpha∗(Matrix S^{-1} Z).

Specialized implementation.

Reimplemented from Ipopt::Matrix.

3.18.3.8 virtual void Ipopt::CompoundMatrix::SinvBlrmZMTdBrImpl (Number *alpha*, const Vector & *S*, const Vector & *R*, const Vector & *Z*, const Vector & *D*, Vector & *X*) const [protected], [virtual]

X = S^{-1} (r + alpha∗Z∗M^Td).

Specialized implementation.

Reimplemented from Ipopt::Matrix.

3.18.3.9 virtual bool Ipopt::CompoundMatrix::HasValidNumbersImpl ( ) const [protected], [virtual]

Method for determining if all stored numbers are valid (i.e., no Inf or Nan).

Reimplemented from Ipopt::Matrix.

3.18.3.10 virtual void Ipopt::CompoundMatrix::ComputeRowAMaxImpl (Vector & *rows_norms,* bool *init*) const [protected], [virtual]

Compute the max-norm of the rows in the matrix.

The result is stored in rows_norms. The vector is assumed to be initialized.

Implements Ipopt::Matrix.

3.18.3.11 virtual void Ipopt::CompoundMatrix::ComputeColAMaxImpl (Vector & *cols_norms,* bool *init*) const [protected], [virtual]

Compute the max-norm of the columns in the matrix.

The result is stored in cols_norms. The vector is assumed to be initialized.

Implements Ipopt::Matrix.

3.18.3.12 virtual void Ipopt::CompoundMatrix::PrintImpl (const Journalist & *jnlst,* EJournalLevel *level,* EJournalCategory *category,* const std::string & *name,* Index *indent,* const std::string & *prefix*) const [protected], [virtual]

Print detailed information about the matrix.

Implements Ipopt::Matrix.

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

- IpCompoundMatrix.hpp

3.19 Ipopt::CompoundMatrixSpace Class Reference

This is the matrix space for CompoundMatrix.

`#include <IpCompoundMatrix.hpp>`

Inheritance diagram for Ipopt::CompoundMatrixSpace:

Public Member Functions

- SmartPtr< const MatrixSpace > GetCompSpace (Index irow, Index jcol) const

  Obtain the component MatrixSpace in block row irow and block column jcol.
- CompoundMatrix ∗ MakeNewCompoundMatrix () const

  Method for creating a new matrix of this specific type.
- virtual Matrix ∗ MakeNew () const

  Overloaded MakeNew method for the MatrixSpace base class.

Constructors / Destructors

- CompoundMatrixSpace (Index ncomps_rows, Index ncomps_cols, Index total_nRows, Index total_nCols)

  Constructor, given the number of row and columns blocks, as well as the totel number of rows and columns.
- ∼CompoundMatrixSpace ()

  Destructor.

Methods for setting information about the components.

- void SetBlockRows (Index irow, Index nrows)
  Set the number nrows of rows in row-block number irow.
- void SetBlockCols (Index jcol, Index ncols)
  Set the number ncols of columns in column-block number jcol.
- Index GetBlockRows (Index irow) const
  Get the number nrows of rows in row-block number irow.
- Index GetBlockCols (Index jcol) const
  Set the number ncols of columns in column-block number jcol.
- void SetCompSpace (Index irow, Index jcol, const MatrixSpace &mat_space, bool auto_allocate=false)
  Set the component MatrixSpace.

Accessor methods

- Index NComps_Rows () const
  Number of block rows.
- Index NComps_Cols () const
  Number of block columns.
- bool Diagonal () const
  True if the blocks lie on the diagonal - can make some operations faster.

3.19.1 Detailed Description

This is the matrix space for CompoundMatrix.

Before a CompoundMatrix can be created, at least one MatrixSpace has to be set per block row and column. Individual component MatrixSpace's can be set with the SetComp method.

Definition at line 168 of file IpCompoundMatrix.hpp.

3.19.2 Member Function Documentation

3.19.2.1 void Ipopt::CompoundMatrixSpace::SetBlockRows (Index *irow*, Index *nrows*)

Set the number nrows of rows in row-block number irow.

3.19.2.2 void Ipopt::CompoundMatrixSpace::SetBlockCols (Index *jcol*, Index *ncols*)

Set the number ncols of columns in column-block number jcol.

3.19.2.3 Index Ipopt::CompoundMatrixSpace::GetBlockRows (Index *irow*) const

Get the number nrows of rows in row-block number irow.

3.19.2.4 Index Ipopt::CompoundMatrixSpace::GetBlockCols (Index *jcol*) const

Set the number ncols of columns in column-block number jcol.

3.19.2.5 void Ipopt::CompoundMatrixSpace::SetCompSpace (Index *irow*, Index *jcol*, const MatrixSpace & *mat_space*, bool *auto_allocate* = false)

Set the component MatrixSpace.

If auto_allocate is true, then a new CompoundMatrix created later with MakeNew will have this component automatically created with the Matrix's MakeNew. Otherwise, the corresponding component will be NULL and has to be set with the SetComp methods of the CompoundMatrix.

3.19.2.6 CompoundMatrix∗ Ipopt::CompoundMatrixSpace::MakeNewCompoundMatrix ( ) const

Method for creating a new matrix of this specific type.

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

- IpCompoundMatrix.hpp

3.20 Ipopt::CompoundSymMatrix Class Reference

Class for symmetric matrices consisting of other matrices.

`#include <IpCompoundSymMatrix.hpp>`

Inheritance diagram for Ipopt::CompoundSymMatrix:

Public Member Functions

- void SetComp (Index irow, Index jcol, const Matrix &matrix)

  Method for setting an individual component at position (irow, icol) in the compound matrix.
- void SetCompNonConst (Index irow, Index jcol, Matrix &matrix)

  Non const version of the same method.
- SmartPtr< const Matrix > GetComp (Index irow, Index jcol) const

  Method for retrieving one block from the compound matrix.
- SmartPtr< Matrix > GetCompNonConst (Index irow, Index jcol)

  Non const version of GetComp.
- SmartPtr< CompoundSymMatrix > MakeNewCompoundSymMatrix () const

  Method for creating a new matrix of this specific type.
- Index NComps_Dim () const

  Number of block rows and columns.

Constructors / Destructors

- CompoundSymMatrix (const CompoundSymMatrixSpace ∗owner_space)

  Constructor, taking only the number for block components into the row and column direction.
- ∼CompoundSymMatrix ()

  Destructor.

Protected Member Functions

Methods overloaded from matrix

- virtual void MultVectorImpl (Number alpha, const Vector &x, Number beta, Vector &y) const
  Matrix-vector multiply.
- virtual bool HasValidNumbersImpl () const
  Method for determining if all stored numbers are valid (i.e., no Inf or Nan).
- virtual void ComputeRowAMaxImpl (Vector &rows_norms, bool init) const
  Compute the max-norm of the rows in the matrix.
- virtual void PrintImpl (const Journalist &jnlst, EJournalLevel level, EJournalCategory category, const std::string &name, Index indent, const std::string &prefix) const
  Print detailed information about the matrix.

Additional Inherited Members

3.20.1 Detailed Description

Class for symmetric matrices consisting of other matrices.

Here, the lower left block of the matrix is stored.

Definition at line 24 of file IpCompoundSymMatrix.hpp.

3.20.2 Constructor & Destructor Documentation

3.20.2.1 Ipopt::CompoundSymMatrix::CompoundSymMatrix (const CompoundSymMatrixSpace ∗ *owner_space*)

Constructor, taking only the number for block components into the row and column direction.

The owner_space has to be defined, so that at each block row and column contain at least one non-NULL component.

3.20.3 Member Function Documentation

3.20.3.1 void Ipopt::CompoundSymMatrix::SetComp (Index *irow,* Index *jcol,* const Matrix & *matrix*)

Method for setting an individual component at position (irow, icol) in the compound matrix.

The counting of indices starts at 0. Since this only the lower left components are stored, we need to have jcol<=irow, and if irow==jcol, the matrix must be a SymMatrix

3.20.3.2 SmartPtr<const Matrix> Ipopt::CompoundSymMatrix::GetComp (Index *irow,* Index *jcol*) const [inline]

Method for retrieving one block from the compound matrix.

Since this only the lower left components are stored, we need to have jcol<=irow

Definition at line 54 of file IpCompoundSymMatrix.hpp.

3.20.3.3 SmartPtr<Matrix> Ipopt::CompoundSymMatrix::GetCompNonConst (Index *irow,* Index *jcol*) [inline]

Non const version of GetComp.

You should only use this method if you are intending to change the matrix you receive, since this CompoundSymMatrix will be marked as changed.

Definition at line 62 of file IpCompoundSymMatrix.hpp.

3.20.3.4 SmartPtr< CompoundSymMatrix > Ipopt::CompoundSymMatrix::MakeNewCompoundSymMatrix ( ) const [inline]

Method for creating a new matrix of this specific type.

Definition at line 277 of file IpCompoundSymMatrix.hpp.

3.20.3.5 virtual void Ipopt::CompoundSymMatrix::MultVectorImpl (Number *alpha*, const Vector & *x*, Number *beta*, Vector & *y*) const [protected], [virtual]

Matrix-vector multiply.

Computes y = alpha ∗ Matrix ∗ x + beta ∗ y

Implements Ipopt::Matrix.

3.20.3.6 virtual bool Ipopt::CompoundSymMatrix::HasValidNumbersImpl ( ) const [protected], [virtual]

Method for determining if all stored numbers are valid (i.e., no Inf or Nan).

Reimplemented from Ipopt::Matrix.

3.20.3.7 virtual void Ipopt::CompoundSymMatrix::ComputeRowAMaxImpl (Vector & *rows_norms*, bool *init*) const [protected], [virtual]

Compute the max-norm of the rows in the matrix.

The result is stored in rows_norms. The vector is assumed to be initialized.

Implements Ipopt::Matrix.

3.20.3.8 virtual void Ipopt::CompoundSymMatrix::PrintImpl (const Journalist & *jnlst*, EJournalLevel *level*, EJournalCategory *category*, const std::string & *name*, Index *indent*, const std::string & *prefix*) const [protected], [virtual]

Print detailed information about the matrix.

Implements Ipopt::Matrix.

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

- IpCompoundSymMatrix.hpp

3.21 Ipopt::CompoundSymMatrixSpace Class Reference

This is the matrix space for CompoundSymMatrix.

`#include <IpCompoundSymMatrix.hpp>`

Inheritance diagram for Ipopt::CompoundSymMatrixSpace:

Public Member Functions

- SmartPtr< const MatrixSpace > GetCompSpace (Index irow, Index jcol) const

  Obtain the component MatrixSpace in block row irow and block column jcol.
- CompoundSymMatrix * MakeNewCompoundSymMatrix () const

  Method for creating a new matrix of this specific type.
- virtual SymMatrix * MakeNewSymMatrix () const

  Overloaded MakeNew method for the SymMatrixSpace base class.

Constructors / Destructors

- CompoundSymMatrixSpace (Index ncomp_spaces, Index total_dim)

  Constructor, given the number of blocks (same for rows and columns), as well as the total dimension of the matrix.
- ∼CompoundSymMatrixSpace ()

  Destructor.

Methods for setting information about the components.

- void SetBlockDim (Index irow_jcol, Index dim)

  Set the dimension dim for block row (or column) irow_jcol.
- Index GetBlockDim (Index irow_jcol) const

  Get the dimension dim for block row (or column) irow_jcol.
- void SetCompSpace (Index irow, Index jcol, const MatrixSpace &mat_space, bool auto_allocate=false)

  Set the component SymMatrixSpace.

Accessor methods

- Index **NComps_Dim** () const

3.21.1 Detailed Description

This is the matrix space for CompoundSymMatrix.

Before a CompoundSymMatrix can be created, at least one SymMatrixSpace has to be set per block row and column. Individual component SymMatrixSpace's can be set with the SetComp method.

Definition at line 171 of file IpCompoundSymMatrix.hpp.

3.21.2 Member Function Documentation

3.21.2.1 void Ipopt::CompoundSymMatrixSpace::SetCompSpace (Index *irow*, Index *jcol*, const **MatrixSpace** & *mat_space*, bool *auto_allocate* = `false`)

Set the component SymMatrixSpace.

If auto_allocate is true, then a new CompoundSymMatrix created later with MakeNew will have this component automatically created with the SymMatrix's MakeNew. Otherwise, the corresponding component will be NULL and has to be set with the SetComp methods of the CompoundSymMatrix.

3.21.2.2 CompoundSymMatrix* Ipopt::CompoundSymMatrixSpace::MakeNewCompoundSymMatrix ( ) const

Method for creating a new matrix of this specific type.

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

- IpCompoundSymMatrix.hpp

3.22 Ipopt::CompoundVector Class Reference

Class of Vectors consisting of other vectors.

`#include <IpCompoundVector.hpp>`

Inheritance diagram for Ipopt::CompoundVector:

Public Member Functions

- void SetComp (Index icomp, const Vector &vec)

    Method for setting the pointer for a component that is a const Vector.
- void SetCompNonConst (Index icomp, Vector &vec)

    Method for setting the pointer for a component that is a non-const Vector.
- Index NComps () const

    Number of components of this compound vector.
- bool IsCompConst (Index i) const

    Check if a particular component is const or not.
- bool IsCompNull (Index i) const

    Check if a particular component is null or not.
- SmartPtr< const Vector > GetComp (Index i) const

    Return a particular component (const version)
- SmartPtr< Vector > GetCompNonConst (Index i)

    Return a particular component (non-const version).

Constructors/Destructors

- CompoundVector (const CompoundVectorSpace *owner_space, bool create_new)

    Constructor, given the corresponding CompoundVectorSpace.
- virtual ~CompoundVector ()

    Default destructor.

Protected Member Functions

- virtual bool HasValidNumbersImpl () const

    Method for determining if all stored numbers are valid (i.e., no Inf or Nan).

Overloaded methods from Vector base class

- virtual void CopyImpl (const Vector &x)

    Copy the data of the vector x into this vector (DCOPY).
- virtual void ScalImpl (Number alpha)

Scales the vector by scalar alpha (DSCAL)
- virtual void AxpyImpl (Number alpha, const Vector &x)
  Add the multiple alpha of vector x to this vector (DAXPY)
- virtual Number DotImpl (const Vector &x) const
  Computes inner product of vector x with this (DDOT)
- virtual Number Nrm2Impl () const
  Computes the 2-norm of this vector (DNRM2)
- virtual Number AsumImpl () const
  Computes the 1-norm of this vector (DASUM)
- virtual Number AmaxImpl () const
  Computes the max-norm of this vector (based on IDAMAX)
- virtual void SetImpl (Number value)
  Set each element in the vector to the scalar alpha.
- virtual void ElementWiseDivideImpl (const Vector &x)
  Element-wise division $y_i \leftarrow y_i/x_i$.
- virtual void ElementWiseMultiplyImpl (const Vector &x)
  Element-wise multiplication $y_i \leftarrow y_i * x_i$.
- virtual void ElementWiseMaxImpl (const Vector &x)
  Element-wise max against entries in x.
- virtual void ElementWiseMinImpl (const Vector &x)
  Element-wise min against entries in x.
- virtual void ElementWiseReciprocalImpl ()
  Element-wise reciprocal.
- virtual void ElementWiseAbsImpl ()
  Element-wise absolute values.
- virtual void ElementWiseSqrtImpl ()
  Element-wise square-root.
- virtual void ElementWiseSgnImpl ()
  Replaces entries with sgn of the entry.
- virtual void AddScalarImpl (Number scalar)
  Add scalar to every component of the vector.
- virtual Number MaxImpl () const
  Max value in the vector.
- virtual Number MinImpl () const
  Min value in the vector.
- virtual Number SumImpl () const
  Computes the sum of the lements of vector.
- virtual Number SumLogsImpl () const
  Computes the sum of the logs of the elements of vector.

Implemented specialized functions

- void AddTwoVectorsImpl (Number a, const Vector &v1, Number b, const Vector &v2, Number c)
  Add two vectors (a ∗ v1 + b ∗ v2).
- Number FracToBoundImpl (const Vector &delta, Number tau) const
  Fraction to the boundary parameter.
- void AddVectorQuotientImpl (Number a, const Vector &z, const Vector &s, Number c)
  Add the quotient of two vectors, y = a ∗ z/s + c ∗ y.

Output methods

- virtual void PrintImpl (const Journalist &jnlst, EJournalLevel level, EJournalCategory category, const std::string &name, Index indent, const std::string &prefix) const
  Print the entire vector.

Additional Inherited Members

3.22.1 Detailed Description

Class of Vectors consisting of other vectors.

This vector is a vector that consists of zero, one or more Vector's which are stacked on each others: $x_{\rm compound} = \left(\begin{array}{c} x_0 \\ \dots \\ x_{\rm ncomps-1} \end{array}\right)$. The individual components can be associated to different VectorSpaces. The individual components can also be const and non-const Vectors.

Definition at line 30 of file IpCompoundVector.hpp.

3.22.2 Constructor & Destructor Documentation

3.22.2.1 Ipopt::CompoundVector::CompoundVector (const **CompoundVectorSpace** ∗ *owner_space,* bool *create_new*)

Constructor, given the corresponding CompoundVectorSpace.

Before this constructor can be called, all components of the CompoundVectorSpace have to be set, so that the constructors for the individual components can be called. If the flag create_new is true, then the individual components of the new CompoundVector are initialized with the MakeNew methods of each VectorSpace (and are non-const). Otherwise, the individual components can later be set using the SetComp and SetCompNonConst method.

3.22.3 Member Function Documentation

3.22.3.1 SmartPtr<Vector> Ipopt::CompoundVector::GetCompNonConst (Index *i*) [inline]

Return a particular component (non-const version).

Note that calling this method with mark the CompoundVector as changed. Therefore, only use this method if you are intending to change the Vector that you receive.

Definition at line 96 of file IpCompoundVector.hpp.

3.22.3.2 virtual void Ipopt::CompoundVector::CopyImpl (const Vector & *x*) [protected], [virtual]

Copy the data of the vector x into this vector (DCOPY).

Implements Ipopt::Vector.

3.22.3.3 virtual void Ipopt::CompoundVector::SetImpl (Number *value*) [protected], [virtual]

Set each element in the vector to the scalar alpha.

Implements Ipopt::Vector.

3.22.3.4 virtual void Ipopt::CompoundVector::ElementWiseDivideImpl (const Vector & *x*) [protected], [virtual]

Element-wise division $y_i \leftarrow y_i / x_i$.

Implements Ipopt::Vector.

3.22.3.5 virtual void Ipopt::CompoundVector::ElementWiseMultiplyImpl (const Vector & *x*) [protected], [virtual]

Element-wise multiplication $y_i \leftarrow y_i * x_i$.

Implements Ipopt::Vector.

3.22.3.6 virtual void Ipopt::CompoundVector::AddScalarImpl (Number *scalar*) [protected], [virtual]

Add scalar to every component of the vector.

Implements Ipopt::Vector.

3.22.3.7 void Ipopt::CompoundVector::AddTwoVectorsImpl (Number *a,* const Vector & *v1,* Number *b,* const Vector & *v2,* Number *c*) [protected], [virtual]

Add two vectors (a $*$ v1 + b $*$ v2).

Result is stored in this vector.

Reimplemented from Ipopt::Vector.

3.22.3.8 Number Ipopt::CompoundVector::FracToBoundImpl (const Vector & *delta,* Number *tau*) const [protected], [virtual]

Fraction to the boundary parameter.

Reimplemented from Ipopt::Vector.

3.22.3.9 void Ipopt::CompoundVector::AddVectorQuotientImpl (Number *a,* const Vector & *z,* const Vector & *s,* Number *c*) [protected], [virtual]

Add the quotient of two vectors, y = a $*$ z/s + c $*$ y.

Reimplemented from Ipopt::Vector.

3.22.3.10 virtual bool Ipopt::CompoundVector::HasValidNumbersImpl ( ) const [protected], [virtual]

Method for determining if all stored numbers are valid (i.e., no Inf or Nan).

Reimplemented from Ipopt::Vector.

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

- IpCompoundVector.hpp

3.23 Ipopt::CompoundVectorSpace Class Reference

This vectors space is the vector space for CompoundVector.

`#include <IpCompoundVector.hpp>`

Inheritance diagram for Ipopt::CompoundVectorSpace:

Public Member Functions

- virtual void SetCompSpace (Index icomp, const VectorSpace &vec_space)

  Method for setting the individual component VectorSpaces.
- SmartPtr< const VectorSpace > GetCompSpace (Index icomp) const

  Method for obtaining an individual component VectorSpace.
- Index NCompSpaces () const

  Accessor method to obtain the number of components.
- virtual CompoundVector ∗ MakeNewCompoundVector (bool create_new=true) const

  Method for creating a new vector of this specific type.
- virtual Vector ∗ MakeNew () const

  Overloaded MakeNew method for the VectorSpace base class.

Constructors/Destructors.

- CompoundVectorSpace (Index ncomp_spaces, Index total_dim)

  Constructor, has to be given the number of components and the total dimension of all components combined.
- ∼CompoundVectorSpace ()

  Destructor.

3.23.1 Detailed Description

This vectors space is the vector space for CompoundVector.

Before a CompoundVector can be created, all components of this CompoundVectorSpace have to be set. When calling the constructor, the number of component has to be specified. The individual VectorSpaces can be set with the SetComp method.

Definition at line 239 of file IpCompoundVector.hpp.

3.23.2 Constructor & Destructor Documentation

3.23.2.1 Ipopt::CompoundVectorSpace::CompoundVectorSpace (Index *ncomp_spaces*, Index *total_dim*)

Constructor, has to be given the number of components and the total dimension of all components combined.

3.23.3 Member Function Documentation

3.23.3.1 virtual void Ipopt::CompoundVectorSpace::SetCompSpace (Index *icomp*, const VectorSpace & *vec_space*) [virtual]

Method for setting the individual component VectorSpaces.

Parameters

icomp	Number of the component to be set
vec_space	VectorSpace for component icomp

Reimplemented in Ipopt::IteratesVectorSpace.

3.23.3.2 virtual CompoundVector* Ipopt::CompoundVectorSpace::MakeNewCompoundVector (bool *create_new* = true) const [inline], [virtual]

Method for creating a new vector of this specific type.

Reimplemented in Ipopt::IteratesVectorSpace.

Definition at line 268 of file IpCompoundVector.hpp.

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

- IpCompoundVector.hpp

3.24 Ipopt::ConvergenceCheck Class Reference

Base class for checking the algorithm termination criteria.

`#include <IpConvCheck.hpp>`

Inheritance diagram for Ipopt::ConvergenceCheck:

Public Types

- enum ConvergenceStatus

  Convergence return enum.

Public Member Functions

- virtual bool InitializeImpl (const OptionsList &options, const std::string &prefix)=0

  overloaded from AlgorithmStrategyObject
- virtual ConvergenceStatus CheckConvergence (bool call_intermediate_callback=true)=0

  Pure virtual method for performing the convergence test.
- virtual bool CurrentIsAcceptable ()=0

  Method for testing if the current iterate is considered to satisfy the "accptable level" of accuracy.

Constructors/Destructors

- ConvergenceCheck ()

  Constructor.
- virtual ~ConvergenceCheck ()

  Default destructor.

Additional Inherited Members

3.24.1 Detailed Description

Base class for checking the algorithm termination criteria.

Definition at line 20 of file IpConvCheck.hpp.

3.24.2 Member Function Documentation

3.24.2.1 virtual ConvergenceStatus Ipopt::ConvergenceCheck::CheckConvergence (bool *call_intermediate_callback* = true) [pure virtual]

Pure virtual method for performing the convergence test.

If call_intermediate_callback is true, the user callback method in the NLP should be called in order to see if the user requests an early termination.

Implemented in Ipopt::RestoConvergenceCheck, and Ipopt::OptimalityErrorConvergenceCheck.

3.24.2.2 virtual bool Ipopt::ConvergenceCheck::CurrentIsAcceptable ( ) [pure virtual]

Method for testing if the current iterate is considered to satisfy the "accptable level" of accuracy.

The idea is that if the desired convergence tolerance cannot be achieved, the algorithm might stop after a number of acceptable points have been encountered.

Implemented in Ipopt::OptimalityErrorConvergenceCheck.

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

- IpConvCheck.hpp

3.25 Ipopt::DefaultIterateInitializer Class Reference

Class implementing the default initialization procedure (based on user options) for the iterates.

`#include <IpDefaultIterateInitializer.hpp>`

Inheritance diagram for Ipopt::DefaultIterateInitializer:

Public Types

Enums of option values

- enum **BoundMultInitMethod**

Public Member Functions

- virtual bool InitializeImpl (const OptionsList &options, const std::string &prefix)

    overloaded from AlgorithmStrategyObject
- virtual bool SetInitialIterates ()

    Compute the initial iterates and set the into the curr field of the ip_data object.

Constructors/Destructors

- DefaultIterateInitializer (const SmartPtr< EqMultiplierCalculator > &eq_mult_calculator, const SmartPtr< IterateInitializer > &warm_start_initializer, const SmartPtr< AugSystemSolver > aug_system_solver=NULL)

    Constructor.
- virtual ∼DefaultIterateInitializer ()

    Default destructor.

Static Public Member Functions

- static void push_variables (const Journalist &jnlst, Number bound_push, Number bound_frac, std::string name, const Vector &orig_x, SmartPtr< const Vector > &new_x, const Vector &x_L, const Vector &x_U, const Matrix &Px_L, const Matrix &Px_U)

  Auxilliary function for moving the initial point.
- static void least_square_mults (const Journalist &jnlst, IpoptNLP &ip_nlp, IpoptData &ip_data, IpoptCalculatedQuantities &ip_cq, const SmartPtr< EqMultiplierCalculator > &eq_mult_calculator, Number constr_mult_init_max)

  Auxilliary function for computing least_square multipliers.

- static void RegisterOptions (SmartPtr< RegisteredOptions > reg_options)

  Methods for IpoptType.

Additional Inherited Members

3.25.1 Detailed Description

Class implementing the default initialization procedure (based on user options) for the iterates.

It is used at the very beginning of the optimization for determine the starting point for all variables.

Definition at line 24 of file IpDefaultIterateInitializer.hpp.

3.25.2 Constructor & Destructor Documentation

3.25.2.1 Ipopt::DefaultIterateInitializer::DefaultIterateInitializer (const SmartPtr< EqMultiplierCalculator > & *eq_mult_calculator*, const SmartPtr< IterateInitializer > & *warm_start_initializer*, const SmartPtr< AugSystemSolver > *aug_system_solver* = NULL)

Constructor.

If eq_mult_calculator is not NULL, it will be used to compute the initial values for equality constraint multipliers. If warm_start_initializer is not NULL, it will be used to compute the initial values if the option warm_start_init_point is chosen.

3.25.3 Member Function Documentation

3.25.3.1 virtual bool Ipopt::DefaultIterateInitializer::SetInitialIterates ( ) [virtual]

Compute the initial iterates and set the into the curr field of the ip_data object.

Implements Ipopt::IterateInitializer.

3.25.3.2 static void Ipopt::DefaultIterateInitializer::push_variables (const Journalist & *jnlst*, Number *bound_push*, Number *bound_frac*, std::string *name*, const Vector & *orig_x*, SmartPtr< const Vector > & *new_x*, const Vector & *x_L*, const Vector & *x_U*, const Matrix & *Px_L*, const Matrix & *Px_U*) [static]

Auxilliary function for moving the initial point.

This is declared static so that it can also be used from WarmStartIterateInitializer.

3.25.3.3 static void Ipopt::DefaultIterateInitializer::least_square_mults (const Journalist & *jnlst*, IpoptNLP & *ip_nlp*, IpoptData & *ip_data*, IpoptCalculatedQuantities & *ip_cq*, const SmartPtr< EqMultiplierCalculator > & *eq_mult_calculator*, Number *constr_mult_init_max*) [static]

Auxilliary function for computing least_square multipliers.

The multipliers are computed based on the values in the trial fields (current is overwritten). On return, the multipliers are in the trial fields as well. The value of constr_mult_init_max determines if the computed least square estimate should be used, or if the initial multipliers are set to zero.

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

- IpDefaultIterateInitializer.hpp

3.26 Ipopt::DenseGenMatrix Class Reference

Class for dense general matrices.

`#include <IpDenseGenMatrix.hpp>`

Inheritance diagram for Ipopt::DenseGenMatrix:

Public Member Functions

- SmartPtr< DenseGenMatrix > MakeNewDenseGenMatrix () const

    Create a new DenseGenMatrix from same MatrixSpace.
- Number ∗ Values ()

    Retrieve the array for storing the matrix elements.
- const Number ∗ Values () const

    Retrieve the array that stores the matrix elements.
- void Copy (const DenseGenMatrix &M)

    Method for copying the content of another matrix into this matrix.
- void FillIdentity (Number factor=1.)

    Set this matrix to be a multiple of the identity matrix .
- void ScaleColumns (const DenseVector &scal_vec)

    Method for scaling the columns of the matrix.
- void AddMatrixProduct (Number alpha, const DenseGenMatrix &A, bool transA, const DenseGenMatrix &B, bool transB, Number beta)

    Method for adding the product of two matrices to this matrix.
- void HighRankUpdateTranspose (Number alpha, const MultiVectorMatrix &V1, const MultiVectorMatrix &V2, Number beta)

    Method for adding a high-rank update to this matrix.
- bool ComputeCholeskyFactor (const DenseSymMatrix &M)

    Method for computing the Cholesky factorization of a positive definite matrix.
- bool ComputeEigenVectors (const DenseSymMatrix &M, DenseVector &Evalues)

    Method for computing an eigenvalue decomposition of the given symmetrix matrix M.
- void CholeskyBackSolveMatrix (bool trans, Number alpha, DenseGenMatrix &B) const

    Method for performing one backsolve with an entire matrix on the right hand side, assuming that the this matrix is square and contains a lower triangular matrix.
- void CholeskySolveVector (DenseVector &b) const

Method for performing a solve of a linear system for one vector, assuming that this matrix contains the Cholesky factor for the linear system.

- void CholeskySolveMatrix (DenseGenMatrix &B) const

  Method for performing a solve of a linear system for one right-hand-side matrix, assuming that this matrix contains the Cholesky factor for the linear system.

- bool ComputeLUFactorInPlace ()

  Method for computing the LU factorization of an unsymmetric matrix.

- void LUSolveMatrix (DenseGenMatrix &B) const

  Method for using a previously computed LU factorization for a backsolve with a matrix on the rhs.

- void LUSolveVector (DenseVector &b) const

  Method for using a previously computed LU fatorization for a backsolve with a single vector.

Constructors / Destructors

- DenseGenMatrix (const DenseGenMatrixSpace ∗owner_space)

  Constructor, taking the owner_space.

- ∼DenseGenMatrix ()

  Destructor.

Protected Member Functions

Overloaded methods from Matrix base class

- virtual void MultVectorImpl (Number alpha, const Vector &x, Number beta, Vector &y) const

  Matrix-vector multiply.

- virtual void TransMultVectorImpl (Number alpha, const Vector &x, Number beta, Vector &y) const

  Matrix(transpose) vector multiply.

- virtual bool HasValidNumbersImpl () const

  Method for determining if all stored numbers are valid (i.e., no Inf or Nan).

- virtual void ComputeRowAMaxImpl (Vector &rows_norms, bool init) const

  Compute the max-norm of the rows in the matrix.

- virtual void ComputeColAMaxImpl (Vector &cols_norms, bool init) const

  Compute the max-norm of the columns in the matrix.

- virtual void PrintImpl (const Journalist &jnlst, EJournalLevel level, EJournalCategory category, const std::string &name, Index indent, const std::string &prefix) const

  Print detailed information about the matrix.

Additional Inherited Members

3.26.1 Detailed Description

Class for dense general matrices.

Matrix elements are stored in one array in "Fortran" format.

Definition at line 26 of file IpDenseGenMatrix.hpp.

3.26.2 Member Function Documentation

3.26.2.1 Number* Ipopt::DenseGenMatrix::Values ( ) [inline]

Retrieve the array for storing the matrix elements.

This is the non-const version, and it is assume that afterwards the calling method will set all matrix elements. The matrix elements are stored one column after each other.

Definition at line 48 of file IpDenseGenMatrix.hpp.

3.26.2.2 const Number* Ipopt::DenseGenMatrix::Values ( ) const [inline]

Retrieve the array that stores the matrix elements.

This is the const version, i.e., read-only. The matrix elements are stored one column after each other.

Definition at line 58 of file IpDenseGenMatrix.hpp.

3.26.2.3 void Ipopt::DenseGenMatrix::FillIdentity (Number *factor* = 1.)

Set this matrix to be a multiple of the identity matrix .

This assumes that this matrix is square.

3.26.2.4 void Ipopt::DenseGenMatrix::ScaleColumns (const DenseVector & *scal_vec*)

Method for scaling the columns of the matrix.

The scaling factors are given in form of a DenseVector

3.26.2.5 void Ipopt::DenseGenMatrix::AddMatrixProduct (Number *alpha*, const DenseGenMatrix & *A*, bool *transA*, const DenseGenMatrix & *B*, bool *transB*, Number *beta*)

Method for adding the product of two matrices to this matrix.

3.26.2.6 void Ipopt::DenseGenMatrix::HighRankUpdateTranspose (Number *alpha*, const MultiVectorMatrix & *V1*, const MultiVectorMatrix & *V2*, Number *beta*)

Method for adding a high-rank update to this matrix.

It computes M = alpha*V1^T V2 + beta*M, where V1 and V2 are MultiVectorMatrices.

3.26.2.7 bool Ipopt::DenseGenMatrix::ComputeCholeskyFactor (const DenseSymMatrix & *M*)

Method for computing the Cholesky factorization of a positive definite matrix.

The factor is stored in this matrix, as lower-triangular matrix, i.e., M = J * J^T. The return values is false if the factorization could not be done, e.g., when the matrix is not positive definite.

3.26.2.8 bool Ipopt::DenseGenMatrix::ComputeEigenVectors (const DenseSymMatrix & *M*, DenseVector & *Evalues*)

Method for computing an eigenvalue decomposition of the given symmetrix matrix M.

On return, this matrix contains the eigenvalues in its columns, and Evalues contains the eigenvalues. The return value is false, if there problems during the computation.

3.26.2.9 void Ipopt::DenseGenMatrix::CholeskyBackSolveMatrix (bool *trans,* Number *alpha,* DenseGenMatrix & *B*) const

Method for performing one backsolve with an entire matrix on the right hand side, assuming that the this matrix is square and contains a lower triangular matrix.

The incoming right hand side B is overwritten with the solution X of op(A)∗X = alpha∗B. op(A) = A or op(A) = A^T.

3.26.2.10 void Ipopt::DenseGenMatrix::CholeskySolveVector (DenseVector & *b*) const

Method for performing a solve of a linear system for one vector, assuming that this matrix contains the Cholesky factor for the linear system.

The vector b contains the right hand side on input, and contains the solution on output.

3.26.2.11 void Ipopt::DenseGenMatrix::CholeskySolveMatrix (DenseGenMatrix & *B*) const

Method for performing a solve of a linear system for one right-hand-side matrix, assuming that this matrix contains the Cholesky factor for the linear system.

The matrix B contains the right hand sides on input, and contains the solution on output.

3.26.2.12 bool Ipopt::DenseGenMatrix::ComputeLUFactorInPlace ( )

Method for computing the LU factorization of an unsymmetric matrix.

The factorization is done in place.

3.26.2.13 void Ipopt::DenseGenMatrix::LUSolveMatrix (DenseGenMatrix & *B*) const

Method for using a previously computed LU factorization for a backsolve with a matrix on the rhs.

3.26.2.14 void Ipopt::DenseGenMatrix::LUSolveVector (DenseVector & *b*) const

Method for using a previously computed LU fatorization for a backsolve with a single vector.

3.26.2.15 virtual void Ipopt::DenseGenMatrix::MultVectorImpl (Number *alpha,* const Vector & *x,* Number *beta,* Vector & *y*) const [protected], [virtual]

Matrix-vector multiply.

Computes y = alpha ∗ Matrix ∗ x + beta ∗ y

Implements Ipopt::Matrix.

3.26.2.16 virtual void Ipopt::DenseGenMatrix::TransMultVectorImpl (Number *alpha,* const Vector & *x,* Number *beta,* Vector & *y*) const [protected], [virtual]

Matrix(transpose) vector multiply.

Computes y = alpha * Matrix^T * x + beta * y

Implements Ipopt::Matrix.

3.26.2.17 virtual bool Ipopt::DenseGenMatrix::HasValidNumbersImpl ( ) const [protected], [virtual]

Method for determining if all stored numbers are valid (i.e., no Inf or Nan).

Reimplemented from Ipopt::Matrix.

3.26.2.18 virtual void Ipopt::DenseGenMatrix::ComputeRowAMaxImpl (Vector & *rows_norms*, bool *init*) const [protected], [virtual]

Compute the max-norm of the rows in the matrix.

The result is stored in rows_norms. The vector is assumed to be initialized.

Implements Ipopt::Matrix.

3.26.2.19 virtual void Ipopt::DenseGenMatrix::ComputeColAMaxImpl (Vector & *cols_norms*, bool *init*) const [protected], [virtual]

Compute the max-norm of the columns in the matrix.

The result is stored in cols_norms. The vector is assumed to be initialized.

Implements Ipopt::Matrix.

3.26.2.20 virtual void Ipopt::DenseGenMatrix::PrintImpl (const Journalist & *jnlst*, EJournalLevel *level*, EJournalCategory *category*, const std::string & *name*, Index *indent*, const std::string & *prefix*) const [protected], [virtual]

Print detailed information about the matrix.

Implements Ipopt::Matrix.

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

- IpDenseGenMatrix.hpp

3.27 Ipopt::DenseGenMatrixSpace Class Reference

This is the matrix space for DenseGenMatrix.

`#include <IpDenseGenMatrix.hpp>`

Inheritance diagram for Ipopt::DenseGenMatrixSpace:

Public Member Functions

- DenseGenMatrix * MakeNewDenseGenMatrix () const

    Method for creating a new matrix of this specific type.
- virtual Matrix * MakeNew () const

    Overloaded MakeNew method for the MatrixSpace base class.

Constructors / Destructors

- DenseGenMatrixSpace (Index nRows, Index nCols)
  Constructor for matrix space for DenseGenMatrices.
- ∼DenseGenMatrixSpace ()
  Destructor.

3.27.1 Detailed Description

This is the matrix space for DenseGenMatrix.

Definition at line 208 of file IpDenseGenMatrix.hpp.

3.27.2 Constructor & Destructor Documentation

3.27.2.1 Ipopt::DenseGenMatrixSpace::DenseGenMatrixSpace (Index *nRows*, Index *nCols*)

Constructor for matrix space for DenseGenMatrices.

Takes in dimension of the matrices.

3.27.3 Member Function Documentation

3.27.3.1 DenseGenMatrix∗ Ipopt::DenseGenMatrixSpace::MakeNewDenseGenMatrix ( ) const [inline]

Method for creating a new matrix of this specific type.

Definition at line 224 of file IpDenseGenMatrix.hpp.

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

- IpDenseGenMatrix.hpp

3.28 Ipopt::DenseSymMatrix Class Reference

Class for dense symetrix matrices.

`#include <IpDenseSymMatrix.hpp>`

Inheritance diagram for Ipopt::DenseSymMatrix:

Public Member Functions

- SmartPtr< DenseSymMatrix > MakeNewDenseSymMatrix () const
  Create a new DenseSymMatrix from same MatrixSpace.
- Number ∗ Values ()
  Retrieve the array for storing the matrix elements.
- const Number ∗ Values () const
  Retrieve the array that stores the matrix elements.
- void FillIdentity (Number factor=1.)
  Set this matrix to be a multiple of the identity matrix.

- void AddMatrix (Number alpha, const DenseSymMatrix &A, Number beta)

  Method for adding another matrix to this one.
- void HighRankUpdate (bool trans, Number alpha, const DenseGenMatrix &V, Number beta)

  Method for adding a high-rank update to this matrix.
- void HighRankUpdateTranspose (Number alpha, const MultiVectorMatrix &V1, const MultiVectorMatrix &V2, Number beta)

  Method for adding a high-rank update to this matrix.
- void SpecialAddForLMSR1 (const DenseVector &D, const DenseGenMatrix &L)

  Method for doing a specialized Add operation, required in the limited memory SR1 update.

Constructors / Destructors

- DenseSymMatrix (const DenseSymMatrixSpace ∗owner_space)

  Constructor, taking the owner_space.
- ∼DenseSymMatrix ()

  Destructor.

Protected Member Functions

Overloaded methods from Matrix base class

- virtual void MultVectorImpl (Number alpha, const Vector &x, Number beta, Vector &y) const

  Matrix-vector multiply.
- virtual bool HasValidNumbersImpl () const

  Method for determining if all stored numbers are valid (i.e., no Inf or Nan).
- virtual void ComputeRowAMaxImpl (Vector &rows_norms, bool init) const

  Compute the max-norm of the rows in the matrix.
- virtual void PrintImpl (const Journalist &jnlst, EJournalLevel level, EJournalCategory category, const std::string &name, Index indent, const std::string &prefix) const

  Print detailed information about the matrix.

Additional Inherited Members

3.28.1 Detailed Description

Class for dense symetrix matrices.

Matrix elements are stored in one array in "Fortran" format, using BLAS "lower triangular" storage (not packed).

Definition at line 31 of file IpDenseSymMatrix.hpp.

3.28.2 Member Function Documentation

3.28.2.1 Number∗ Ipopt::DenseSymMatrix::Values ( ) [inline]

Retrieve the array for storing the matrix elements.

This is the non-const version, and it is assume that afterwards the calling method will set all matrix elements. The matrix elements are stored one column after each other.

Definition at line 53 of file IpDenseSymMatrix.hpp.

3.28.2.2 const Number* Ipopt::DenseSymMatrix::Values ( ) const [inline]

Retrieve the array that stores the matrix elements.

This is the const version, i.e., read-only. The matrix elements are stored one column after each other.

Definition at line 63 of file IpDenseSymMatrix.hpp.

3.28.2.3 void Ipopt::DenseSymMatrix::FillIdentity (Number *factor* = 1.)

Set this matrix to be a multiple of the identity matrix.

3.28.2.4 void Ipopt::DenseSymMatrix::AddMatrix (Number *alpha*, const DenseSymMatrix & *A*, Number *beta*)

Method for adding another matrix to this one.

If B is this matrix, it becomes B = alpha * A + beta * B after this call.

3.28.2.5 void Ipopt::DenseSymMatrix::HighRankUpdate (bool *trans*, Number *alpha*, const DenseGenMatrix & *V*, Number *beta*)

Method for adding a high-rank update to this matrix.

It computes M = alpha*op(V) op(V)^T + beta*M, where V is a DenseGenMatrix, where op(V) is V^T trans is true.

3.28.2.6 void Ipopt::DenseSymMatrix::HighRankUpdateTranspose (Number *alpha*, const MultiVectorMatrix & *V1*, const MultiVectorMatrix & *V2*, Number *beta*)

Method for adding a high-rank update to this matrix.

It computes M = alpha*V1^T V2 + beta*M, where V1 and V2 are MultiVectorMatrices, so that V1^T V2 is symmetric.

3.28.2.7 void Ipopt::DenseSymMatrix::SpecialAddForLMSR1 (const DenseVector & *D*, const DenseGenMatrix & *L*)

Method for doing a specialized Add operation, required in the limited memory SR1 update.

if M is this matrix, it computes M = M + D + L + L^T, where D is a diagonal matrix (given as a DenseVector), and L is a matrix that is assumed to be strictly lower triangular.

3.28.2.8 virtual void Ipopt::DenseSymMatrix::MultVectorImpl (Number *alpha*, const Vector & *x*, Number *beta*, Vector & *y*) const [protected], [virtual]

Matrix-vector multiply.

Computes y = alpha * Matrix * x + beta * y

Implements Ipopt::Matrix.

3.28.2.9 virtual bool Ipopt::DenseSymMatrix::HasValidNumbersImpl ( ) const [protected], [virtual]

Method for determining if all stored numbers are valid (i.e., no Inf or Nan).

Reimplemented from Ipopt::Matrix.

3.28.2.10 virtual void Ipopt::DenseSymMatrix::ComputeRowAMaxImpl (Vector & *rows_norms*, bool *init*) const [protected], [virtual]

Compute the max-norm of the rows in the matrix.

The result is stored in rows_norms. The vector is assumed to be initialized.

Implements Ipopt::Matrix.

3.28.2.11 virtual void Ipopt::DenseSymMatrix::PrintImpl (const Journalist & *jnlst*, EJournalLevel *level*, EJournalCategory *category*, const std::string & *name*, Index *indent*, const std::string & *prefix*) const [protected], [virtual]

Print detailed information about the matrix.

Implements Ipopt::Matrix.

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

- IpDenseSymMatrix.hpp

3.29 Ipopt::DenseSymMatrixSpace Class Reference

This is the matrix space for DenseSymMatrix.

`#include <IpDenseSymMatrix.hpp>`

Inheritance diagram for Ipopt::DenseSymMatrixSpace:

Public Member Functions

- DenseSymMatrix * MakeNewDenseSymMatrix () const

  Method for creating a new matrix of this specific type.
- virtual SymMatrix * MakeNewSymMatrix () const

  Overloaded MakeNew method for the MatrixSpace base class.

Constructors / Destructors

- DenseSymMatrixSpace (Index nDim)

  Constructor for matrix space for DenseSymMatrices.
- ∼DenseSymMatrixSpace ()

  Destructor.

3.29.1 Detailed Description

This is the matrix space for DenseSymMatrix.

Definition at line 149 of file IpDenseSymMatrix.hpp.

3.29.2 Constructor & Destructor Documentation

3.29.2.1 Ipopt::DenseSymMatrixSpace::DenseSymMatrixSpace (Index *nDim*)

Constructor for matrix space for DenseSymMatrices.

Takes in dimension of the matrices.

3.29.3 Member Function Documentation

3.29.3.1 DenseSymMatrix* Ipopt::DenseSymMatrixSpace::MakeNewDenseSymMatrix ( ) const [inline]

Method for creating a new matrix of this specific type.

Definition at line 165 of file IpDenseSymMatrix.hpp.

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

- IpDenseSymMatrix.hpp

3.30 Ipopt::DenseVector Class Reference

Dense Vector Implementation.

`#include <IpDenseVector.hpp>`

Inheritance diagram for Ipopt::DenseVector:

Public Member Functions

Constructors / Destructors

- DenseVector (const DenseVectorSpace *owner_space)

  Default Constructor.
- virtual ~DenseVector ()

  Destructor.

Additional public methods not in Vector base class.

- SmartPtr< DenseVector > MakeNewDenseVector () const

  Create a new DenseVector from same VectorSpace.
- void SetValues (const Number *x)

  Set elements in the vector to the Number array x.
- Number * Values ()

  Obtain pointer to the internal Number array with vector elements with the indention to change the vector data (USE WITH CARE!).
- const Number * Values () const

  Obtain pointer to the internal Number array with vector elements without the intention to change the vector data (USE WITH CARE!).
- const Number * ExpandedValues () const

  The same as the const version of Values, but we ensure that we always return a valid array, even if IsHomogeneous returns true.
- Number * ExpandedValues ()

  This is the same as Values, but we add it here so that ExpandedValues can also be used for the non-const case.
- bool IsHomogeneous () const

  Indicates if the vector is homogeneous (i.e., all entries have the value Scalar()
- Number Scalar () const

  Scalar value of all entries in a homogeneous vector.

Modifying subranges of the vector.

- void CopyToPos (Index Pos, const Vector &x)

  Copy the data in x into the subrange of this vector starting at position Pos in this vector.
- void CopyFromPos (Index Pos, const Vector &x)

  Copy a subrange of x, starting at Pos, into the full data of this vector.

Protected Member Functions

Overloaded methods from Vector base class

- virtual void CopyImpl (const Vector &x)

  Copy the data of the vector x into this vector (DCOPY).
- virtual void ScalImpl (Number alpha)

  Scales the vector by scalar alpha (DSCAL)
- virtual void AxpyImpl (Number alpha, const Vector &x)

  Add the multiple alpha of vector x to this vector (DAXPY)
- virtual Number DotImpl (const Vector &x) const

  Computes inner product of vector x with this (DDOT)
- virtual Number Nrm2Impl () const

  Computes the 2-norm of this vector (DNRM2)
- virtual Number AsumImpl () const

  Computes the 1-norm of this vector (DASUM)
- virtual Number AmaxImpl () const

  Computes the max-norm of this vector (based on IDAMAX)
- virtual void SetImpl (Number value)

  Set each element in the vector to the scalar alpha.
- virtual void ElementWiseDivideImpl (const Vector &x)

  Element-wise division $y_i \leftarrow y_i/x_i$.
- virtual void ElementWiseMultiplyImpl (const Vector &x)

  Element-wise multiplication $y_i \leftarrow y_i * x_i$.
- virtual void ElementWiseMaxImpl (const Vector &x)

  Set entry to max of itself and the corresponding element in x.
- virtual void ElementWiseMinImpl (const Vector &x)

  Set entry to min of itself and the corresponding element in x.
- virtual void ElementWiseReciprocalImpl ()

  reciprocates the elements of the vector
- virtual void ElementWiseAbsImpl ()

  take abs of the elements of the vector
- virtual void ElementWiseSqrtImpl ()

  take square-root of the elements of the vector
- virtual void ElementWiseSgnImpl ()

  Changes each entry in the vector to its sgn value.
- virtual void AddScalarImpl (Number scalar)

  Add scalar to every component of the vector.
- virtual Number MaxImpl () const

  Max value in the vector.
- virtual Number MinImpl () const

  Min value in the vector.
- virtual Number SumImpl () const

  Computes the sum of the lements of vector.
- virtual Number SumLogsImpl () const

Computes the sum of the logs of the elements of vector.

Implemented specialized functions

- void AddTwoVectorsImpl (Number a, const Vector &v1, Number b, const Vector &v2, Number c)
  Add two vectors (a ∗ v1 + b ∗ v2).
- Number FracToBoundImpl (const Vector &delta, Number tau) const
  Fraction to the boundary parameter.
- void AddVectorQuotientImpl (Number a, const Vector &z, const Vector &s, Number c)
  Add the quotient of two vectors, y = a ∗ z/s + c ∗ y.

Output methods

- virtual void PrintImpl (const Journalist &jnlst, EJournalLevel level, EJournalCategory category, const std::string &name, Index indent, const std::string &prefix) const
  Print the entire vector.
- void **PrintImplOffset** (const Journalist &jnlst, EJournalLevel level, EJournalCategory category, const std::string &name, Index indent, const std::string &prefix, Index offset) const

Additional Inherited Members

3.30.1 Detailed Description

Dense Vector Implementation.

This is the default Vector class in Ipopt. It stores vectors in contiguous Number arrays, unless the vector has the same value in all entires. In the latter case, we call the vector "homogeneous", and we store only the values that is repeated in all elements. If you want to obtain the values of vector, use the IsHomogeneous() method to find out what status the vector is in, and then use either Values() const or Scalar() const methods to get the values. To set the values of a homogeneous method, use the Set method. To set the values of a non-homogeneous vector, use the SetValues method, or use the non-const Values method to get an array that you can overwrite. In the latter case, storage is ensured.

Definition at line 40 of file IpDenseVector.hpp.

3.30.2 Member Function Documentation

3.30.2.1 void Ipopt::DenseVector::SetValues (const Number ∗ *x*)

Set elements in the vector to the Number array x.

3.30.2.2 Number ∗ Ipopt::DenseVector::Values ( ) [inline]

Obtain pointer to the internal Number array with vector elements with the indention to change the vector data (USE WITH CARE!).

This does not produce a copy, and lifetime is not guaranteed!.

Definition at line 393 of file IpDenseVector.hpp.

3.30.2.3 const Number ∗ Ipopt::DenseVector::Values ( ) const [inline]

Obtain pointer to the internal Number array with vector elements without the intention to change the vector data (USE WITH CARE!).

This does not produce a copy, and lifetime is not guaranteed! IMPORTANT: If this method is currently homogeneous (i.e. IsHomogeneous returns true), then you cannot call this method. Instead, you need to use the Scalar() method.

Definition at line 410 of file IpDenseVector.hpp.

3.30.2.4 const Number∗ Ipopt::DenseVector::ExpandedValues ( ) const

The same as the const version of Values, but we ensure that we always return a valid array, even if IsHomogeneous returns true.

3.30.2.5 Number∗ Ipopt::DenseVector::ExpandedValues ( ) [inline]

This is the same as Values, but we add it here so that ExpandedValues can also be used for the non-const case.

Definition at line 87 of file IpDenseVector.hpp.

3.30.2.6 void Ipopt::DenseVector::CopyToPos (Index *Pos*, const Vector & *x*)

Copy the data in x into the subrange of this vector starting at position Pos in this vector.

Position count starts at 0.

3.30.2.7 void Ipopt::DenseVector::CopyFromPos (Index *Pos*, const Vector & *x*)

Copy a subrange of x, starting at Pos, into the full data of this vector.

Position count starts at 0.

3.30.2.8 virtual void Ipopt::DenseVector::CopyImpl (const Vector & *x*) [protected], [virtual]

Copy the data of the vector x into this vector (DCOPY).

Implements Ipopt::Vector.

3.30.2.9 virtual void Ipopt::DenseVector::SetImpl (Number *value*) [protected], [virtual]

Set each element in the vector to the scalar alpha.

Implements Ipopt::Vector.

3.30.2.10 virtual void Ipopt::DenseVector::ElementWiseDivideImpl (const Vector & *x*) [protected], [virtual]

Element-wise division $y_i \leftarrow y_i/x_i$.

Implements Ipopt::Vector.

3.30.2.11 virtual void Ipopt::DenseVector::ElementWiseMultiplyImpl (const Vector & *x*) [protected], [virtual]

Element-wise multiplication $y_i \leftarrow y_i * x_i$.

Implements Ipopt::Vector.

3.30.2.12 virtual void Ipopt::DenseVector::AddScalarImpl (Number *scalar*) [protected], [virtual]

Add scalar to every component of the vector.

Implements Ipopt::Vector.

3.30.2.13 void Ipopt::DenseVector::AddTwoVectorsImpl (Number *a*, const Vector & *v1*, Number *b*, const Vector & *v2*, Number *c*) [protected], [virtual]

Add two vectors (a $*$ v1 + b $*$ v2).

Result is stored in this vector.

Reimplemented from Ipopt::Vector.

3.30.2.14 Number Ipopt::DenseVector::FracToBoundImpl (const Vector & *delta*, Number *tau*) const [protected], [virtual]

Fraction to the boundary parameter.

Reimplemented from Ipopt::Vector.

3.30.2.15 void Ipopt::DenseVector::AddVectorQuotientImpl (Number *a*, const Vector & *z*, const Vector & *s*, Number *c*) [protected], [virtual]

Add the quotient of two vectors, y = a $*$ z/s + c $*$ y.

Reimplemented from Ipopt::Vector.

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

- IpDenseVector.hpp

3.31 Ipopt::DenseVectorSpace Class Reference

This vectors space is the vector space for DenseVector.

`#include <IpDenseVector.hpp>`

Inheritance diagram for Ipopt::DenseVectorSpace:

Public Member Functions

- DenseVector $*$ MakeNewDenseVector () const

  Method for creating a new vector of this specific type.
- virtual Vector $*$ MakeNew () const

  Instantiation of the generate MakeNew method for the VectorSpace base class.

Constructors/Destructors.

- DenseVectorSpace (Index dim)

  Constructor, requires dimension of all vector for this VectorSpace.
- ∼DenseVectorSpace ()

Destructor.

Methods called by DenseVector for memory management.

This could allow to have sophisticated memory management in the VectorSpace.

- Number ∗ AllocateInternalStorage () const
  Allocate internal storage for the DenseVector.
- void FreeInternalStorage (Number ∗values) const
  Deallocate internal storage for the DenseVector.

Methods for dealing with meta data on the vector

- bool HasStringMetaData (const std::string tag) const
  Check if string meta exists for tag.
- bool HasIntegerMetaData (const std::string tag) const
  Check if Integer meta exists for tag.
- bool HasNumericMetaData (const std::string tag) const
  Check if Numeric meta exists for tag.
- const std::vector< std::string > & GetStringMetaData (const std::string &tag) const
  Get meta data of type std::string by tag.
- const std::vector< Index > & GetIntegerMetaData (const std::string &tag) const
  Get meta data of type Index by tag.
- const std::vector< Number > & GetNumericMetaData (const std::string &tag) const
  Get meta data of type Number by tag.
- void SetStringMetaData (std::string tag, std::vector< std::string > meta_data)
  Set meta data of type std::string by tag.
- void SetIntegerMetaData (std::string tag, std::vector< Index > meta_data)
  Set meta data of type Index by tag.
- void SetNumericMetaData (std::string tag, std::vector< Number > meta_data)
  Set meta data of type Number by tag.
- const StringMetaDataMapType & GetStringMetaData () const
  Get map of meta data of type Number.
- const IntegerMetaDataMapType & GetIntegerMetaData () const
  Get map of meta data of type Number.
- const NumericMetaDataMapType & GetNumericMetaData () const
  Get map of meta data of type Number.

3.31.1 Detailed Description

This vectors space is the vector space for DenseVector.

Definition at line 285 of file IpDenseVector.hpp.

3.31.2 Member Function Documentation

3.31.2.1 DenseVector∗ Ipopt::DenseVectorSpace::MakeNewDenseVector ( ) const [inline]

Method for creating a new vector of this specific type.

Definition at line 305 of file IpDenseVector.hpp.

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

- IpDenseVector.hpp

3.32 Ipopt::DependentResult< T > Class Template Reference

Templated class which stores one entry for the CachedResult class.

`#include <IpCachedResults.hpp>`

Inheritance diagram for Ipopt::DependentResult< T >:

Public Member Functions

- bool DependentsIdentical (const std::vector< const TaggedObject ∗ > &dependents, const std::vector< Number > &scalar_dependents) const

    This method returns true if the dependencies provided to this function are identical to the ones stored with the DependentResult.
- void DebugPrint () const

    Print information about this DependentResults.

Constructor, Destructors

- DependentResult (const T &result, const std::vector< const TaggedObject ∗ > &dependents, const std::vector< Number > &scalar_dependents)

    Constructor, given all information about the result.
- ∼DependentResult ()

    Destructor.

Accessor method.

- bool IsStale () const

    This returns true, if the DependentResult is no longer valid.
- void Invalidate ()

    Invalidates the cached result.
- const T & GetResult () const

    Returns the cached result.

Protected Member Functions

- virtual void RecieveNotification (NotifyType notify_type, const Subject ∗subject)

    This method is overloading the pure virtual method from the Observer base class.

Additional Inherited Members

3.32.1 Detailed Description

template< class T > class Ipopt::DependentResult< T >

Templated class which stores one entry for the CachedResult class.

It stores the result (of type T), together with its dependencies (vector of TaggedObjects and vector of Numbers). It also stores a priority.

Definition at line 31 of file IpCachedResults.hpp.

3.32.2 Constructor & Destructor Documentation

3.32.2.1 template<class T> Ipopt::DependentResult< T >::DependentResult (const T & *result*, const std::vector< const TaggedObject * > & *dependents*, const std::vector< Number > & *scalar_dependents*)

Constructor, given all information about the result.

Definition at line 348 of file IpCachedResults.hpp.

3.32.2.2 template<class T > Ipopt::DependentResult< T >::~DependentResult ( )

Destructor.

Definition at line 381 of file IpCachedResults.hpp.

3.32.3 Member Function Documentation

3.32.3.1 template<class T > bool Ipopt::DependentResult< T >::IsStale ( ) const

This returns true, if the DependentResult is no longer valid.

Definition at line 393 of file IpCachedResults.hpp.

3.32.3.2 template<class T > void Ipopt::DependentResult< T >::Invalidate ( )

Invalidates the cached result.

Definition at line 399 of file IpCachedResults.hpp.

3.32.3.3 template<class T > const T & Ipopt::DependentResult< T >::GetResult ( ) const

Returns the cached result.

Definition at line 456 of file IpCachedResults.hpp.

3.32.3.4 template<class T > void Ipopt::DependentResult< T >::DebugPrint ( ) const

Print information about this DependentResults.

Definition at line 467 of file IpCachedResults.hpp.

3.32.3.5 template<class T > void Ipopt::DependentResult< T >::RecieveNotification (NotifyType *notify_type*, const Subject * *subject*) [protected], [virtual]

This method is overloading the pure virtual method from the Observer base class.

This method is called when a Subject registered for this Observer sends a notification. In this particular case, if this method is called with notify_type==NT_Changed or NT_BeingDeleted, then this results is marked as stale.

Implements Ipopt::Observer.

Definition at line 405 of file IpCachedResults.hpp.

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

- IpCachedResults.hpp

3.33 Ipopt::DiagMatrix Class Reference

Class for diagonal matrices.

`#include <IpDiagMatrix.hpp>`

Inheritance diagram for Ipopt::DiagMatrix:

Public Member Functions

- void SetDiag (const Vector &diag)

  Method for setting the diagonal elements (as a Vector).
- SmartPtr< const Vector > GetDiag () const

  Method for setting the diagonal elements.

Constructors / Destructors

- DiagMatrix (const SymMatrixSpace ∗owner_space)

  Constructor, given the corresponding matrix space.
- ∼DiagMatrix ()

  Destructor.

Protected Member Functions

Methods overloaded from matrix

- virtual void MultVectorImpl (Number alpha, const Vector &x, Number beta, Vector &y) const

  Matrix-vector multiply.
- virtual bool HasValidNumbersImpl () const

  Method for determining if all stored numbers are valid (i.e., no Inf or Nan).
- virtual void ComputeRowAMaxImpl (Vector &rows_norms, bool init) const

  Compute the max-norm of the rows in the matrix.
- virtual void PrintImpl (const Journalist &jnlst, EJournalLevel level, EJournalCategory category, const std::string &name, Index indent, const std::string &prefix) const

  Print detailed information about the matrix.

Additional Inherited Members

3.33.1 Detailed Description

Class for diagonal matrices.

The diagonal is stored as a Vector.

Definition at line 20 of file IpDiagMatrix.hpp.

3.33.2 Constructor & Destructor Documentation

3.33.2.1 Ipopt::DiagMatrix::DiagMatrix (const SymMatrixSpace * *owner_space*)

Constructor, given the corresponding matrix space.

3.33.3 Member Function Documentation

3.33.3.1 void Ipopt::DiagMatrix::SetDiag (const Vector & *diag*) [inline]

Method for setting the diagonal elements (as a Vector).

Definition at line 35 of file IpDiagMatrix.hpp.

3.33.3.2 SmartPtr<const Vector> Ipopt::DiagMatrix::GetDiag ( ) const [inline]

Method for setting the diagonal elements.

Definition at line 41 of file IpDiagMatrix.hpp.

3.33.3.3 virtual void Ipopt::DiagMatrix::MultVectorImpl (Number *alpha*, const Vector & *x*, Number *beta*, Vector & *y*) const [protected], [virtual]

Matrix-vector multiply.

Computes y = alpha * Matrix * x + beta * y

Implements Ipopt::Matrix.

3.33.3.4 virtual bool Ipopt::DiagMatrix::HasValidNumbersImpl ( ) const [protected], [virtual]

Method for determining if all stored numbers are valid (i.e., no Inf or Nan).

Reimplemented from Ipopt::Matrix.

3.33.3.5 virtual void Ipopt::DiagMatrix::ComputeRowAMaxImpl (Vector & *rows_norms*, bool *init*) const [protected], [virtual]

Compute the max-norm of the rows in the matrix.

The result is stored in rows_norms. The vector is assumed to be initialized.

Implements Ipopt::Matrix.

3.33.3.6 virtual void Ipopt::DiagMatrix::PrintImpl (const Journalist & *jnlst*, EJournalLevel *level*, EJournalCategory *category*, const std::string & *name*, Index *indent*, const std::string & *prefix*) const [protected], [virtual]

Print detailed information about the matrix.

Implements Ipopt::Matrix.

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

- IpDiagMatrix.hpp

3.34 Ipopt::DiagMatrixSpace Class Reference

This is the matrix space for DiagMatrix.

`#include <IpDiagMatrix.hpp>`

Inheritance diagram for Ipopt::DiagMatrixSpace:

Public Member Functions

- virtual SymMatrix ∗ MakeNewSymMatrix () const

  Overloaded MakeNew method for the SymMatrixSpace base class.
- DiagMatrix ∗ MakeNewDiagMatrix () const

  Method for creating a new matrix of this specific type.

Constructors / Destructors

- DiagMatrixSpace (Index dim)

  Constructor, given the dimension of the matrix.
- virtual ∼DiagMatrixSpace ()

  Destructor.

3.34.1 Detailed Description

This is the matrix space for DiagMatrix.

Definition at line 90 of file IpDiagMatrix.hpp.

3.34.2 Constructor & Destructor Documentation

3.34.2.1 Ipopt::DiagMatrixSpace::DiagMatrixSpace (Index *dim*) [inline]

Constructor, given the dimension of the matrix.

Definition at line 96 of file IpDiagMatrix.hpp.

3.34.3 Member Function Documentation

3.34.3.1 DiagMatrix∗ Ipopt::DiagMatrixSpace::MakeNewDiagMatrix ( ) const [inline]

Method for creating a new matrix of this specific type.

Definition at line 114 of file IpDiagMatrix.hpp.

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

- IpDiagMatrix.hpp

3.35 Ipopt::EqMultiplierCalculator Class Reference

Base Class for objects that compute estimates for the equality constraint multipliers y_c and y_d.

```
#include <IpEqMultCalculator.hpp>
```

Inheritance diagram for Ipopt::EqMultiplierCalculator:

Public Member Functions

- virtual bool InitializeImpl (const OptionsList &options, const std::string &prefix)=0

  overloaded from AlgorithmStrategyObject
- virtual bool CalculateMultipliers (Vector &y_c, Vector &y_d)=0

  This method computes the estimates for y_c and y_d at the current point.

Constructors/Destructors

- EqMultiplierCalculator ()

  Default Constructor.
- virtual ∼EqMultiplierCalculator ()

  Default destructor.

Additional Inherited Members

3.35.1 Detailed Description

Base Class for objects that compute estimates for the equality constraint multipliers y_c and y_d.

For example, this is the base class for objects for computing least square multipliers or coordinate multipliers.

Definition at line 21 of file IpEqMultCalculator.hpp.

3.35.2 Constructor & Destructor Documentation

3.35.2.1 Ipopt::EqMultiplierCalculator::EqMultiplierCalculator ( ) [inline]

Default Constructor.

Definition at line 27 of file IpEqMultCalculator.hpp.

3.35.3 Member Function Documentation

3.35.3.1 virtual bool Ipopt::EqMultiplierCalculator::CalculateMultipliers (Vector & *y_c*, Vector & *y_d*) [pure virtual]

This method computes the estimates for y_c and y_d at the current point.

If the estimates cannot be computed (e.g. some linear system is singular), the return value of this method is false.

Implemented in Ipopt::LeastSquareMultipliers.

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

- IpEqMultCalculator.hpp

3.36 Ipopt::EquilibrationScaling Class Reference

This class does problem scaling by setting the scaling parameters based on the maximum of the gradient at the user provided initial point.

`#include <IpEquilibrationScaling.hpp>`

Inheritance diagram for Ipopt::EquilibrationScaling:

Public Member Functions

Constructors/Destructors

- **EquilibrationScaling** (const SmartPtr< NLP > &nlp)
- virtual ∼EquilibrationScaling ()

    Default destructor.

Static Public Member Functions

- static void RegisterOptions (const SmartPtr< RegisteredOptions > &roptions)

    Methods for IpoptType.

Protected Member Functions

- bool InitializeImpl (const OptionsList &options, const std::string &prefix)

    Initialize the object from the options.
- virtual void DetermineScalingParametersImpl (const SmartPtr< const VectorSpace > x_space, const SmartPtr< const VectorSpace > c_space, const SmartPtr< const VectorSpace > d_space, const SmartPtr< const MatrixSpace > jac_c_space, const SmartPtr< const MatrixSpace > jac_d_space, const SmartPtr< const SymMatrixSpace > h_space, const Matrix &Px_L, const Vector &x_L, const Matrix &Px_U, const Vector &x_U, Number &df, SmartPtr< Vector > &dx, SmartPtr< Vector > &dc, SmartPtr< Vector > &dd)

    This is the method that has to be overloaded by a particular scaling method that somehow computes the scaling vectors dx, dc, and dd.

3.36.1 Detailed Description

This class does problem scaling by setting the scaling parameters based on the maximum of the gradient at the user provided initial point.

Definition at line 21 of file IpEquilibrationScaling.hpp.

3.36.2 Member Function Documentation

3.36.2.1 static void Ipopt::EquilibrationScaling::RegisterOptions (const SmartPtr< RegisteredOptions > & *roptions*) `[static]`

Methods for IpoptType.

Register the options for this class

3.36.2.2 virtual void Ipopt::EquilibrationScaling::DetermineScalingParametersImpl (const SmartPtr< const VectorSpace > *x_space*, const SmartPtr< const VectorSpace > *c_space*, const SmartPtr< const VectorSpace > *d_space*, const SmartPtr< const MatrixSpace > *jac_c_space*, const SmartPtr< const MatrixSpace > *jac_d_space*, const SmartPtr< const SymMatrixSpace > *h_space*, const Matrix & *Px_L*, const Vector & *x_L*, const Matrix & *Px_U*, const Vector & *x_U*, Number & *df*, SmartPtr< Vector > & *dx*, SmartPtr< Vector > & *dc*, SmartPtr< Vector > & *dd*) [protected], [virtual]

This is the method that has to be overloaded by a particular scaling method that somehow computes the scaling vectors dx, dc, and dd.

The pointers to those vectors can be NULL, in which case no scaling for that item will be done later.

Implements Ipopt::StandardScalingBase.

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

- IpEquilibrationScaling.hpp

3.37 Ipopt::ExactHessianUpdater Class Reference

Implementation of the HessianUpdater for the use of exact second derivatives.

`#include <IpExactHessianUpdater.hpp>`

Inheritance diagram for Ipopt::ExactHessianUpdater:

Public Member Functions

- virtual bool InitializeImpl (const OptionsList &options, const std::string &prefix)

  overloaded from AlgorithmStrategyObject
- virtual void UpdateHessian ()

  Update the Hessian based on the current information in IpData.

Constructors/Destructors

- ExactHessianUpdater ()

  Default Constructor.
- virtual ∼ExactHessianUpdater ()

  Default destructor.

Additional Inherited Members

3.37.1 Detailed Description

Implementation of the HessianUpdater for the use of exact second derivatives.

Definition at line 20 of file IpExactHessianUpdater.hpp.

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

- IpExactHessianUpdater.hpp

3.38 Ipopt::ExpandedMultiVectorMatrix Class Reference

Class for Matrices with few rows that consists of Vectors, together with a premultiplied Expansion matrix.

```
#include <IpExpandedMultiVectorMatrix.hpp>
```

Inheritance diagram for Ipopt::ExpandedMultiVectorMatrix:

Public Member Functions

- void SetVector (Index i, SmartPtr< const Vector > vec)

  Set a particular Vector at a given row position, replacing another vector if there has been one.
- SmartPtr< const Vector > GetVector (Index i) const

  Get a Vector in a particular row as a const Vector.
- SmartPtr< const VectorSpace > RowVectorSpace () const

  Vector space for the rows.
- SmartPtr< const ExpandedMultiVectorMatrixSpace > ExpandedMultiVectorMatrixOwnerSpace () const

  Return the ExpandedMultiVectorMatrixSpace.
- SmartPtr< const ExpansionMatrix > GetExpansionMatrix () const

  Return the Expansion matrix.

Constructors / Destructors

- ExpandedMultiVectorMatrix (const ExpandedMultiVectorMatrixSpace ∗owner_space)

  Constructor, taking the owner_space.
- virtual ∼ExpandedMultiVectorMatrix ()

  Destructor.

Protected Member Functions

Overloaded methods from Matrix base class

- virtual void MultVectorImpl (Number alpha, const Vector &x, Number beta, Vector &y) const

  Matrix-vector multiply.
- virtual void TransMultVectorImpl (Number alpha, const Vector &x, Number beta, Vector &y) const

  Matrix(transpose) vector multiply.
- virtual bool HasValidNumbersImpl () const

  Method for determining if all stored numbers are valid (i.e., no Inf or Nan).
- virtual void ComputeRowAMaxImpl (Vector &rows_norms, bool init) const

  Compute the max-norm of the rows in the matrix.
- virtual void ComputeColAMaxImpl (Vector &cols_norms, bool init) const

  Compute the max-norm of the columns in the matrix.
- virtual void PrintImpl (const Journalist &jnlst, EJournalLevel level, EJournalCategory category, const std::string &name, Index indent, const std::string &prefix) const

  Print detailed information about the matrix.

Additional Inherited Members

3.38.1 Detailed Description

Class for Matrices with few rows that consists of Vectors, together with a premultiplied Expansion matrix.

So, the matrix is V^T∗P^T. If P is NULL, it is assumed to be the identity matrix. If a row vector of V is NULL, it is assumed to be all zero. This is used to construct the KKT system with low-rank Hessian approximation.

Definition at line 29 of file IpExpandedMultiVectorMatrix.hpp.

3.38.2 Member Function Documentation

3.38.2.1 void Ipopt::ExpandedMultiVectorMatrix::SetVector (Index *i*, SmartPtr< const Vector > *vec*)

Set a particular Vector at a given row position, replacing another vector if there has been one.

3.38.2.2 SmartPtr< const ExpansionMatrix > Ipopt::ExpandedMultiVectorMatrix::GetExpansionMatrix ( ) const [inline]

Return the Expansion matrix.

If NULL, there is no expansion, the vector is used as is.

Definition at line 182 of file IpExpandedMultiVectorMatrix.hpp.

3.38.2.3 virtual void Ipopt::ExpandedMultiVectorMatrix::MultVectorImpl (Number *alpha*, const Vector & *x*, Number *beta*, Vector & *y*) const [protected], [virtual]

Matrix-vector multiply.

Computes y = alpha ∗ Matrix ∗ x + beta ∗ y

Implements Ipopt::Matrix.

3.38.2.4 virtual void Ipopt::ExpandedMultiVectorMatrix::TransMultVectorImpl (Number *alpha*, const Vector & *x*, Number *beta*, Vector & *y*) const [protected], [virtual]

Matrix(transpose) vector multiply.

Computes y = alpha ∗ Matrix^T ∗ x + beta ∗ y

Implements Ipopt::Matrix.

3.38.2.5 virtual bool Ipopt::ExpandedMultiVectorMatrix::HasValidNumbersImpl ( ) const [protected], [virtual]

Method for determining if all stored numbers are valid (i.e., no Inf or Nan).

Reimplemented from Ipopt::Matrix.

3.38.2.6 virtual void Ipopt::ExpandedMultiVectorMatrix::ComputeRowAMaxImpl (Vector & *rows_norms*, bool *init*) const [protected], [virtual]

Compute the max-norm of the rows in the matrix.

The result is stored in rows_norms. The vector is assumed to be initialized.

Implements Ipopt::Matrix.

3.38.2.7 virtual void Ipopt::ExpandedMultiVectorMatrix::ComputeColAMaxImpl (Vector & *cols_norms,* bool *init*) const [protected], [virtual]

Compute the max-norm of the columns in the matrix.

The result is stored in cols_norms. The vector is assumed to be initialized.

Implements Ipopt::Matrix.

3.38.2.8 virtual void Ipopt::ExpandedMultiVectorMatrix::PrintImpl (const Journalist & *jnlst,* EJournalLevel *level,* EJournalCategory *category,* const std::string & *name,* Index *indent,* const std::string & *prefix*) const [protected], [virtual]

Print detailed information about the matrix.

Implements Ipopt::Matrix.

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

- IpExpandedMultiVectorMatrix.hpp

3.39 Ipopt::ExpandedMultiVectorMatrixSpace Class Reference

This is the matrix space for ExpandedMultiVectorMatrix.

`#include <IpExpandedMultiVectorMatrix.hpp>`

Inheritance diagram for Ipopt::ExpandedMultiVectorMatrixSpace:

Public Member Functions

- ExpandedMultiVectorMatrix ∗ MakeNewExpandedMultiVectorMatrix () const

  Method for creating a new matrix of this specific type.
- virtual Matrix ∗ MakeNew () const

  Overloaded MakeNew method for the MatrixSpace base class.
- SmartPtr< const VectorSpace > RowVectorSpace () const

  Accessor method for the VectorSpace for the rows.

Constructors / Destructors

- ExpandedMultiVectorMatrixSpace (Index nrows, const VectorSpace &vec_space, SmartPtr< const ExpansionMatrix > exp_matrix)

  Constructor, given the number of rows (i.e., Vectors to be stored) and given the VectorSpace for the Vectors.
- virtual ∼ExpandedMultiVectorMatrixSpace ()

  Destructor.

3.39.1 Detailed Description

This is the matrix space for ExpandedMultiVectorMatrix.

Definition at line 123 of file IpExpandedMultiVectorMatrix.hpp.

3.39.2 Member Function Documentation

3.39.2.1 ExpandedMultiVectorMatrix∗ Ipopt::ExpandedMultiVectorMatrixSpace::MakeNewExpandedMultiVectorMatrix ( ) const [inline]

Method for creating a new matrix of this specific type.

Definition at line 140 of file IpExpandedMultiVectorMatrix.hpp.

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

- IpExpandedMultiVectorMatrix.hpp

3.40 Ipopt::ExpansionMatrix Class Reference

Class for expansion/projection matrices.

`#include <IpExpansionMatrix.hpp>`

Inheritance diagram for Ipopt::ExpansionMatrix:

Public Member Functions

- const Index ∗ ExpandedPosIndices () const

  Return the vector of indices marking the expanded position.
- const Index ∗ CompressedPosIndices () const

  Return the vector of indices marking the compressed position.

Constructors / Destructors

- ExpansionMatrix (const ExpansionMatrixSpace ∗owner_space)

  Constructor, taking the owner_space.
- ∼ExpansionMatrix ()

  Destructor.

Protected Member Functions

Overloaded methods from Matrix base class

- virtual void MultVectorImpl (Number alpha, const Vector &x, Number beta, Vector &y) const

  Matrix-vector multiply.
- virtual void TransMultVectorImpl (Number alpha, const Vector &x, Number beta, Vector &y) const

  Matrix(transpose) vector multiply.
- virtual void AddMSinvZImpl (Number alpha, const Vector &S, const Vector &Z, Vector &X) const

  X = beta∗X + alpha∗(Matrix S^{-1} Z).

- virtual void SinvBlrmZMTdBrImpl (Number alpha, const Vector &S, const Vector &R, const Vector &Z, const Vector &D, Vector &X) const

  X = S^{-1} (r + alpha∗Z∗M^Td).
- virtual void ComputeRowAMaxImpl (Vector &rows_norms, bool init) const

  Compute the max-norm of the rows in the matrix.
- virtual void ComputeColAMaxImpl (Vector &cols_norms, bool init) const

  Compute the max-norm of the columns in the matrix.
- virtual void PrintImpl (const Journalist &jnlst, EJournalLevel level, EJournalCategory category, const std::string &name, Index indent, const std::string &prefix) const

  Print detailed information about the matrix.

Additional Inherited Members

3.40.1 Detailed Description

Class for expansion/projection matrices.

These matrices allow to lift a vector to a vector with larger dimension, keeping some elements of the larger vector zero. This operation is achieved by the MultVector operation. The transpose operation then filters some elements from a large vector into a smaller vector.

Definition at line 27 of file IpExpansionMatrix.hpp.

3.40.2 Member Function Documentation

3.40.2.1 const Index ∗ Ipopt::ExpansionMatrix::ExpandedPosIndices ( ) const [inline]

Return the vector of indices marking the expanded position.

The result is the Index array (of length NSmallVec=NCols()) that stores the mapping from the small vector to the large vector. For each element i=0,..,NSmallVec in the small vector, ExpandedPosIndices()[i] give the corresponding index in the large vector.

Definition at line 200 of file IpExpansionMatrix.hpp.

3.40.2.2 const Index ∗ Ipopt::ExpansionMatrix::CompressedPosIndices ( ) const [inline]

Return the vector of indices marking the compressed position.

The result is the Index array (of length NLargeVec=NRows()) that stores the mapping from the large vector to the small vector. For each element i=0,..,NLargeVec in the large vector, CompressedPosIndices()[i] gives the corresponding index in the small vector, unless CompressedPosIndices()[i] is negative.

Definition at line 206 of file IpExpansionMatrix.hpp.

3.40.2.3 virtual void Ipopt::ExpansionMatrix::MultVectorImpl (Number *alpha*, const Vector & *x*, Number *beta*, Vector & *y*) const [protected], [virtual]

Matrix-vector multiply.

Computes y = alpha ∗ Matrix ∗ x + beta ∗ y

Implements Ipopt::Matrix.

3.40.2.4 virtual void Ipopt::ExpansionMatrix::TransMultVectorImpl (Number *alpha*, const Vector & *x*, Number *beta*, Vector & *y*) const [protected], [virtual]

Matrix(transpose) vector multiply.

Computes y = alpha * Matrix^T * x + beta * y

Implements Ipopt::Matrix.

3.40.2.5 virtual void Ipopt::ExpansionMatrix::AddMSinvZImpl (Number *alpha*, const Vector & *S*, const Vector & *Z*, Vector & *X*) const [protected], [virtual]

X = beta*X + alpha*(Matrix S^{-1} Z).

Specialized implementation.

Reimplemented from Ipopt::Matrix.

3.40.2.6 virtual void Ipopt::ExpansionMatrix::SinvBlrmZMTdBrImpl (Number *alpha*, const Vector & *S*, const Vector & *R*, const Vector & *Z*, const Vector & *D*, Vector & *X*) const [protected], [virtual]

X = S^{-1} (r + alpha*Z*M^Td).

Specialized implementation.

Reimplemented from Ipopt::Matrix.

3.40.2.7 virtual void Ipopt::ExpansionMatrix::ComputeRowAMaxImpl (Vector & *rows_norms*, bool *init*) const [protected], [virtual]

Compute the max-norm of the rows in the matrix.

The result is stored in rows_norms. The vector is assumed to be initialized.

Implements Ipopt::Matrix.

3.40.2.8 virtual void Ipopt::ExpansionMatrix::ComputeColAMaxImpl (Vector & *cols_norms*, bool *init*) const [protected], [virtual]

Compute the max-norm of the columns in the matrix.

The result is stored in cols_norms. The vector is assumed to be initialized.

Implements Ipopt::Matrix.

3.40.2.9 virtual void Ipopt::ExpansionMatrix::PrintImpl (const Journalist & *jnlst*, EJournalLevel *level*, EJournalCategory *category*, const std::string & *name*, Index *indent*, const std::string & *prefix*) const [inline], [protected], [virtual]

Print detailed information about the matrix.

Implements Ipopt::Matrix.

Definition at line 85 of file IpExpansionMatrix.hpp.

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

- IpExpansionMatrix.hpp

3.41 Ipopt::ExpansionMatrixSpace Class Reference

This is the matrix space for ExpansionMatrix.

`#include <IpExpansionMatrix.hpp>`

Inheritance diagram for Ipopt::ExpansionMatrixSpace:

Public Member Functions

- ExpansionMatrix ∗ MakeNewExpansionMatrix () const

  Method for creating a new matrix of this specific type.
- virtual Matrix ∗ MakeNew () const

  Overloaded MakeNew method for the MatrixSpace base class.
- const Index ∗ ExpandedPosIndices () const

  Accessor Method to obtain the Index array (of length NSmallVec=NCols()) that stores the mapping from the small vector to the large vector.
- const Index ∗ CompressedPosIndices () const

  Accessor Method to obtain the Index array (of length NLargeVec=NRows()) that stores the mapping from the large vector to the small vector.

Constructors / Destructors

- ExpansionMatrixSpace (Index NLargeVec, Index NSmallVec, const Index ∗ExpPos, const int offset=0)

  Constructor, given the list of elements of the large vector (of size NLargeVec) to be filtered into the small vector (of size NSmallVec).
- ∼ExpansionMatrixSpace ()

  Destructor.

3.41.1 Detailed Description

This is the matrix space for ExpansionMatrix.

Definition at line 132 of file IpExpansionMatrix.hpp.

3.41.2 Constructor & Destructor Documentation

3.41.2.1 Ipopt::ExpansionMatrixSpace::ExpansionMatrixSpace (Index *NLargeVec*, Index *NSmallVec*, const Index ∗ *ExpPos*, const int *offset* = 0)

Constructor, given the list of elements of the large vector (of size NLargeVec) to be filtered into the small vector (of size NSmallVec).

For each i=0..NSmallVec-1 the i-th element of the small vector will be put into the ExpPos[i] position of the large vector. The position counting in the vector is assumed to start at 0 (C-like array notation).

3.41.3 Member Function Documentation

3.41.3.1 ExpansionMatrix∗ Ipopt::ExpansionMatrixSpace::MakeNewExpansionMatrix ( ) const [inline]

Method for creating a new matrix of this specific type.

Definition at line 158 of file IpExpansionMatrix.hpp.

3.41.3.2 const Index∗ Ipopt::ExpansionMatrixSpace::ExpandedPosIndices ( ) const [inline]

Accessor Method to obtain the Index array (of length NSmallVec=NCols()) that stores the mapping from the small vector to the large vector.

For each element i=0,..,NSmallVec in the small vector, ExpandedPosIndices()[i] give the corresponding index in the large vector.

Definition at line 176 of file IpExpansionMatrix.hpp.

3.41.3.3 const Index∗ Ipopt::ExpansionMatrixSpace::CompressedPosIndices ( ) const [inline]

Accessor Method to obtain the Index array (of length NLargeVec=NRows()) that stores the mapping from the large vector to the small vector.

For each element i=0,..,NLargeVec in the large vector, CompressedPosIndices()[i] gives the corresponding index in the small vector, unless CompressedPosIndices()[i] is negative.

Definition at line 188 of file IpExpansionMatrix.hpp.

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

- IpExpansionMatrix.hpp

3.42 Ipopt::FileJournal Class Reference

FileJournal class.

```
#include <IpJournalist.hpp>
```

Inheritance diagram for Ipopt::FileJournal:

Public Member Functions

- FileJournal (const std::string &name, EJournalLevel default_level)

  Constructor.
- virtual ∼FileJournal ()

  Destructor.
- virtual bool Open (const char ∗fname)

  Open a new file for the output location.

Protected Member Functions

Implementation version of Print methods - Overloaded from

Journal base class.

- virtual void PrintImpl (EJournalCategory category, EJournalLevel level, const char ∗str)
  Print to the designated output location.
- virtual void PrintfImpl (EJournalCategory category, EJournalLevel level, const char ∗pformat, va_list ap)
  Printf to the designated output location.
- virtual void FlushBufferImpl ()
  Flush output buffer.

3.42.1 Detailed Description

FileJournal class.

This is a particular Journal implementation that writes to a file for output. It can write to (stdout, stderr, or disk) by using "stdout" and "stderr" as filenames.

Definition at line 379 of file IpJournalist.hpp.

3.42.2 Constructor & Destructor Documentation

3.42.2.1 Ipopt::FileJournal::FileJournal (const std::string & *name,* EJournalLevel *default_level*)

Constructor.

3.42.2.2 virtual Ipopt::FileJournal::∼FileJournal ( ) [virtual]

Destructor.

3.42.3 Member Function Documentation

3.42.3.1 virtual bool Ipopt::FileJournal::Open (const char ∗ *fname*) [virtual]

Open a new file for the output location.

Special Names: stdout means stdout, : stderr means stderr.

Return code is false only if the file with the given name could not be opened.

3.42.3.2 virtual void Ipopt::FileJournal::FlushBufferImpl ( ) [protected], [virtual]

Flush output buffer.

Implements Ipopt::Journal.

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

- IpJournalist.hpp

3.43 Ipopt::Filter Class Reference

Class for the filter.

`#include <IpFilter.hpp>`

Public Member Functions

- bool Acceptable (std::vector< Number > vals) const
    Check acceptability of given coordinates with respect to the filter.
- void AddEntry (std::vector< Number > vals, Index iteration)
    Add filter entry for given coordinates.
- void Clear ()
    Delete all filter entries.
- void Print (const Journalist &jnlst)
    Print current filter entries.

Constructors/Destructors

- Filter (Index dim)
    Default Constructor.
- ∼Filter ()
    Default Destructor.

Wrappers for 2-dimensional filter.

- bool **Acceptable** (Number val1, Number val2) const
- void **AddEntry** (Number val1, Number val2, Index iteration)

3.43.1 Detailed Description

Class for the filter.

This class contains all filter entries. The entries are stored as the corner point, including the margin.

Definition at line 111 of file IpFilter.hpp.

3.43.2 Member Function Documentation

3.43.2.1 bool Ipopt::Filter::Acceptable (std::vector< Number > *vals*) const

Check acceptability of given coordinates with respect to the filter.

Returns true, if pair is acceptable

3.43.2.2 void Ipopt::Filter::AddEntry (std::vector< Number > *vals,* Index *iteration*)

Add filter entry for given coordinates.

This will also delete all dominated entries in the current filter.

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

- IpFilter.hpp

3.44 Ipopt::FilterEntry Class Reference

Class for one filter entry.

`#include <IpFilter.hpp>`

Public Member Functions

- bool Acceptable (std::vector< Number > vals) const

  Check acceptability of pair (phi,theta) with respect to this filter entry.
- bool Dominated (std::vector< Number > vals) const

  Check if this entry is dominated by given coordinates.

Constructors/Destructors

- FilterEntry (std::vector< Number > vals, Index iter)

  Constructor with the two components and the current iteration count.
- ∼FilterEntry ()

  Default Destructor.

Accessor functions

- Number **val** (Index i) const
- Index **iter** () const

3.44.1 Detailed Description

Class for one filter entry.

Definition at line 21 of file IpFilter.hpp.

3.44.2 Member Function Documentation

3.44.2.1 bool Ipopt::FilterEntry::Acceptable (std::vector< Number > *vals*) const [inline]

Check acceptability of pair (phi,theta) with respect to this filter entry.

Returns true, if pair is acceptable.

Definition at line 36 of file IpFilter.hpp.

3.44.2.2 bool Ipopt::FilterEntry::Dominated (std::vector< Number > *vals*) const [inline]

Check if this entry is dominated by given coordinates.

Returns true, if this entry is dominated.

Definition at line 56 of file IpFilter.hpp.

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

- IpFilter.hpp

3.45 Ipopt::FilterLSAcceptor Class Reference

Filter line search.

`#include <IpFilterLSAcceptor.hpp>`

Inheritance diagram for Ipopt::FilterLSAcceptor:

Public Member Functions

- virtual bool InitializeImpl (const OptionsList &options, const std::string &prefix)

    InitializeImpl - overloaded from AlgorithmStrategyObject.
- virtual void Reset ()

    Reset the acceptor.
- virtual void InitThisLineSearch (bool in_watchdog)

    Initialization for the next line search.
- virtual void PrepareRestoPhaseStart ()

    Method that is called before the restoration phase is called.
- virtual Number CalculateAlphaMin ()

    Method returning the lower bound on the trial step sizes.
- virtual bool CheckAcceptabilityOfTrialPoint (Number alpha_primal)

    Method for checking if current trial point is acceptable.
- virtual bool TrySecondOrderCorrection (Number alpha_primal_test, Number &alpha_primal, SmartPtr< IteratesVector > &actual_delta)

    Try a second order correction for the constraints.
- virtual bool TryCorrector (Number alpha_primal_test, Number &alpha_primal, SmartPtr< IteratesVector > &actual_delta)

    Try higher order corrector (for fast local convergence).
- virtual char UpdateForNextIteration (Number alpha_primal_test)

    Method for ending the current line search.
- virtual void StartWatchDog ()

    Method for setting internal data if the watchdog procedure is started.
- virtual void StopWatchDog ()

    Method for setting internal data if the watchdog procedure is stopped.

Constructors/Destructors

- FilterLSAcceptor (const SmartPtr< PDSystemSolver > &pd_solver)

    Constructor.
- virtual ∼FilterLSAcceptor ()

    Default destructor.

Trial Point Accepting Methods. Used internally to check certain

acceptability criteria and used externally (by the restoration phase convergence check object, for instance)

- bool IsAcceptableToCurrentIterate (Number trial_barr, Number trial_theta, bool called_from_restoration=false) const

    Checks if a trial point is acceptable to the current iterate.
- bool IsAcceptableToCurrentFilter (Number trial_barr, Number trial_theta) const

    Checks if a trial point is acceptable to the current filter.

Static Public Member Functions

- static void RegisterOptions (SmartPtr< RegisteredOptions > roptions)

    Methods for OptionsList.

Additional Inherited Members

3.45.1 Detailed Description

Filter line search.

This class implements the filter line search procedure.

Definition at line 23 of file IpFilterLSAcceptor.hpp.

3.45.2 Constructor & Destructor Documentation

3.45.2.1 Ipopt::FilterLSAcceptor::FilterLSAcceptor (const SmartPtr< PDSystemSolver > & *pd_solver*)

Constructor.

The PDSystemSolver object only needs to be provided (i.e. not NULL) if second order correction or corrector steps are to be used.

3.45.3 Member Function Documentation

3.45.3.1 virtual void Ipopt::FilterLSAcceptor::Reset ( ) [virtual]

Reset the acceptor.

This function should be called if all previous information should be discarded when the line search is performed the next time. For example, this method should be called if the barrier parameter is changed.

Implements Ipopt::BacktrackingLSAcceptor.

3.45.3.2 virtual void Ipopt::FilterLSAcceptor::InitThisLineSearch (bool *in_watchdog*) [virtual]

Initialization for the next line search.

The flag in_watchdog indicates if we are currently in an active watchdog procedure.

Implements Ipopt::BacktrackingLSAcceptor.

3.45.3.3 virtual void Ipopt::FilterLSAcceptor::PrepareRestoPhaseStart ( ) [virtual]

Method that is called before the restoration phase is called.

Here, we can set up things that are required in the termination test for the restoration phase, such as augmenting a filter.

Implements Ipopt::BacktrackingLSAcceptor.

3.45.3.4 virtual Number Ipopt::FilterLSAcceptor::CalculateAlphaMin ( ) [virtual]

Method returning the lower bound on the trial step sizes.

Implements Ipopt::BacktrackingLSAcceptor.

3.45.3.5 virtual bool Ipopt::FilterLSAcceptor::CheckAcceptabilityOfTrialPoint (Number *alpha_primal*) [virtual]

Method for checking if current trial point is acceptable.

It is assumed that the delta information in ip_data is the search direction used in criteria. The primal trial point has to be set before the call.

Implements Ipopt::BacktrackingLSAcceptor.

3.45.3.6 virtual bool Ipopt::FilterLSAcceptor::TrySecondOrderCorrection (Number *alpha_primal_test,* Number & *alpha_primal,* SmartPtr< IteratesVector > & *actual_delta*) [virtual]

Try a second order correction for the constraints.

If the first trial step (with incoming alpha_primal) has been reject, this tries up to max_soc_ second order corrections for the constraints. Here, alpha_primal_test is the step size that has to be used in the filter acceptance tests. On output actual_delta_ has been set to the step including the second order correction if it has been accepted, otherwise it is unchanged. If the SOC step has been accepted, alpha_primal has the fraction-to-the-boundary value for the SOC step on output. The return value is true, if a SOC step has been accepted.

Implements Ipopt::BacktrackingLSAcceptor.

3.45.3.7 virtual bool Ipopt::FilterLSAcceptor::TryCorrector (Number *alpha_primal_test,* Number & *alpha_primal,* SmartPtr< IteratesVector > & *actual_delta*) [virtual]

Try higher order corrector (for fast local convergence).

In contrast to a second order correction step, which tries to make an unacceptable point acceptable by improving constraint violation, this corrector step is tried even if the regular primal-dual step is acceptable.

Implements Ipopt::BacktrackingLSAcceptor.

3.45.3.8 virtual char Ipopt::FilterLSAcceptor::UpdateForNextIteration (Number *alpha_primal_test*) [virtual]

Method for ending the current line search.

When it is called, the internal data should be updates, e.g., the filter might be augmented. alpha_primal_test is the value of alpha that has been used for in the acceptence test ealier.

Implements Ipopt::BacktrackingLSAcceptor.

3.45.3.9 virtual void Ipopt::FilterLSAcceptor::StartWatchDog ( ) [virtual]

Method for setting internal data if the watchdog procedure is started.

Implements Ipopt::BacktrackingLSAcceptor.

3.45.3.10 virtual void Ipopt::FilterLSAcceptor::StopWatchDog ( ) [virtual]

Method for setting internal data if the watchdog procedure is stopped.

Implements Ipopt::BacktrackingLSAcceptor.

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

- IpFilterLSAcceptor.hpp

3.46 Ipopt::GenAugSystemSolver Class Reference

Solver for the augmented system using GenKKTSolverInterfaces.

`#include <IpGenAugSystemSolver.hpp>`

Inheritance diagram for Ipopt::GenAugSystemSolver:

Public Member Functions

- bool InitializeImpl (const OptionsList &options, const std::string &prefix)

  overloaded from AlgorithmStrategyObject
- virtual ESymSolverStatus MultiSolve (const SymMatrix ∗W, double W_factor, const Vector ∗D_x, double delta_x, const Vector ∗D_s, double delta_s, const Matrix ∗J_c, const Vector ∗D_c, double delta_c, const Matrix ∗J_d, const Vector ∗D_d, double delta_d, std::vector< SmartPtr< const Vector > > &rhs_xV, std::vector< SmartPtr< const Vector > > &rhs_sV, std::vector< SmartPtr< const Vector > > &rhs_cV, std::vector< SmartPtr< const Vector > > &rhs_dV, std::vector< SmartPtr< Vector > > &sol_xV, std::vector< SmartPtr< Vector > > &sol_sV, std::vector< SmartPtr< Vector > > &sol_cV, std::vector< SmartPtr< Vector > > &sol_dV, bool check_NegEVals, Index numberOfNegEVals)

  Set up the augmented system and solve it for a set of given right hand side - implementation for GenTMatrices and SymTMatrices.
- virtual Index NumberOfNegEVals () const

  Number of negative eigenvalues detected during last solve.
- virtual bool ProvidesInertia () const

  Query whether inertia is computed by linear solver.
- virtual bool IncreaseQuality ()

  Request to increase quality of solution for next solve.

Constructors/Destructors

- GenAugSystemSolver (GenKKTSolverInterface &SolverInterface)

  Constructor using only a linear solver object.
- virtual ∼GenAugSystemSolver ()

  Default destructor.

Additional Inherited Members

3.46.1 Detailed Description

Solver for the augmented system using GenKKTSolverInterfaces.

This takes any Vector values out and provides Number∗'s, but Matrices are provided as given from the NLP.

Definition at line 22 of file IpGenAugSystemSolver.hpp.

3.46.2 Member Function Documentation

3.46.2.1 virtual Index Ipopt::GenAugSystemSolver::NumberOfNegEVals ( ) const [virtual]

Number of negative eigenvalues detected during last solve.

Returns the number of negative eigenvalues of the most recent factorized matrix. This must not be called if the linear solver does not compute this quantities (see ProvidesInertia).

Implements Ipopt::AugSystemSolver.

3.46.2.2 virtual bool Ipopt::GenAugSystemSolver::ProvidesInertia ( ) const [virtual]

Query whether inertia is computed by linear solver.

Returns true, if linear solver provides inertia.

Implements Ipopt::AugSystemSolver.

3.46.2.3 virtual bool Ipopt::GenAugSystemSolver::IncreaseQuality ( ) [virtual]

Request to increase quality of solution for next solve.

Ask underlying linear solver to increase quality of solution for the next solve (e.g. increase pivot tolerance). Returns false, if this is not possible (e.g. maximal pivot tolerance already used.)

Implements Ipopt::AugSystemSolver.

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

- IpGenAugSystemSolver.hpp

3.47 Ipopt::GenKKTSolverInterface Class Reference

Base class for interfaces to symmetric indefinite linear solvers for generic matrices.

`#include <IpGenKKTSolverInterface.hpp>`

Inheritance diagram for Ipopt::GenKKTSolverInterface:

Public Member Functions

- virtual bool InitializeImpl (const OptionsList &options, const std::string &prefix)=0

  overloaded from AlgorithmStrategyObject

Constructor/Destructor

- **GenKKTSolverInterface** ()
- virtual ∼**GenKKTSolverInterface** ()

Methods for requesting solution of the linear system.

- virtual ESymSolverStatus MultiSolve (bool new_matrix, Index n_x, Index n_c, Index n_d, SmartPtr< const SymMatrix > W, SmartPtr< const Matrix > Jac_c, SmartPtr< const Matrix > Jac_d, const Number ∗D_x, const Number ∗D_s, const Number ∗D_c, const Number ∗D_d, Number delta_x, Number delta_s, Number delta_c, Number delta_d, Index n_rhs, Number ∗rhssol, bool check_NegEVals, Index numberOfNegEVals)=0

  Solve operation for multiple right hand sides.
- virtual Index NumberOfNegEVals () const =0

  Number of negative eigenvalues detected during last factorization.

- virtual bool IncreaseQuality ()=0

  Request to increase quality of solution for next solve.

- virtual bool ProvidesInertia () const =0

  Query whether inertia is computed by linear solver.

Additional Inherited Members

3.47.1 Detailed Description

Base class for interfaces to symmetric indefinite linear solvers for generic matrices.

Definition at line 20 of file IpGenKKTSolverInterface.hpp.

3.47.2 Member Function Documentation

3.47.2.1 virtual ESymSolverStatus Ipopt::GenKKTSolverInterface::MultiSolve (bool *new_matrix*, Index *n_x*, Index *n_c*, Index *n_d*, SmartPtr< const SymMatrix > *W*, SmartPtr< const Matrix > *Jac_c*, SmartPtr< const Matrix > *Jac_d*, const Number ∗ *D_x*, const Number ∗ *D_s*, const Number ∗ *D_c*, const Number ∗ *D_d*, Number *delta_x*, Number *delta_s*, Number *delta_c*, Number *delta_d*, Index *n_rhs*, Number ∗ *rhssol*, bool *check_NegEVals*, Index *numberOfNegEVals*) [pure virtual]

Solve operation for multiple right hand sides.

The linear system is of the form

$$\left[\begin{array}{cccc} W + D_x + \delta_x I & 0 & J_c^T & J_d^T \\ 0 & D_s + \delta_s I & 0 & -I \\ J_c & 0 & D_c - \delta_c I & 0 \\ J_d & -I & 0 & D_d - \delta_d I \end{array}\right] \left(\begin{array}{c} sol_x \\ sol_s \\ sol_c \\ sol_d \end{array}\right) = \left(\begin{array}{c} rhs_x \\ rhs_s \\ rhs_c \\ rhs_d \end{array}\right)$$

(see also AugSystemSolver).

The return code is SYMSOLV_SUCCESS if the factorization and solves were successful, SYMSOLV_SINGULAR if the linear system is singular, and SYMSOLV_WRONG_INERTIA if check_NegEVals is true and the number of negative eigenvalues in the matrix does not match numberOfNegEVals. If SYMSOLV_CALL_AGAIN is returned, then the calling function will request the pointer for the array for storing a again (with GetValuesPtr), write the values of the nonzero elements into it, and call this MultiSolve method again with the same right-hand sides. (This can be done, for example, if the linear solver realized it does not have sufficient memory and needs to redo the factorization; e.g., for MA27.)

The number of right-hand sides is given by nrhs, the values of the right-hand sides are given in rhs_vals (one full right-hand side stored immediately after the other), and solutions are to be returned in the same array.

check_NegEVals will not be chosen true, if ProvidesInertia() returns false.

Parameters

new_matrix	If this flag is false, the same matrix as in the most recent call is given to the solver again
n_x	Dimension of D_x
n_c	Dimension of D_s and D_c
n_d	Dimension of D_d
W	Hessian of Lagrangian (as given by NLP)

Jac_c	Jacobian of equality constraints (as given by NLP)
Jac_d	Jacobian of inequality constraints (as given by NLP)
D_x	Array with the elements D_x (if NULL, assume all zero)
D_s	Array with the elements D_s (if NULL, assume all zero)
D_c	Array with the elements D_c (if NULL, assume all zero)
D_d	Array with the elements D_d (if NULL, assume all zero)
delta_x	δ_x
delta_s	δ_s
delta_c	δ_c
delta_d	δ_d
n_rhs	Number of right hand sides
rhssol	On input, this containts the right hand sides, and on successful termination of the solver, the solutions are expected in there on return. At the moment, the order is x,d,c,s, but this can be made flexible and chosen according to an option.
check_NegEVals	if true, we want to ensure that the inertia is correct
numberOfNeg←EVals	Required number of negative eigenvalues if check_NegEVals is true

3.47.2.2 virtual Index Ipopt::GenKKTSolverInterface::NumberOfNegEVals ( ) const `[pure virtual]`

Number of negative eigenvalues detected during last factorization.

Returns the number of negative eigenvalues of the most recent factorized matrix. This must not be called if the linear solver does not compute this quantities (see ProvidesInertia).

3.47.2.3 virtual bool Ipopt::GenKKTSolverInterface::IncreaseQuality ( ) `[pure virtual]`

Request to increase quality of solution for next solve.

The calling class asks linear solver to increase quality of solution for the next solve (e.g. increase pivot tolerance). Returns false, if this is not possible (e.g. maximal pivot tolerance already used.)

3.47.2.4 virtual bool Ipopt::GenKKTSolverInterface::ProvidesInertia ( ) const `[pure virtual]`

Query whether inertia is computed by linear solver.

Returns true, if linear solver provides inertia.

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

- IpGenKKTSolverInterface.hpp

3.48 Ipopt::GenTMatrix Class Reference

Class for general matrices stored in triplet format.

`#include <IpGenTMatrix.hpp>`

Inheritance diagram for Ipopt::GenTMatrix:

Public Member Functions

Constructors / Destructors

- GenTMatrix (const GenTMatrixSpace ∗owner_space)

  Constructor, taking the owner_space.
- ∼GenTMatrix ()

  Destructor.

Changing the Values.

- void SetValues (const Number ∗Values)

  Set values of nonzero elements.

Accessor Methods

- Index Nonzeros () const

  Number of nonzero entries.
- const Index ∗ Irows () const

  Array with Row indices (counting starts at 1)
- const Index ∗ Jcols () const

  Array with Column indices (counting starts at 1)
- const Number ∗ Values () const

  Array with nonzero values (const version).
- Number ∗ Values ()

  Array with the nonzero values of this matrix (non-const version).

Protected Member Functions

Overloaded methods from Matrix base class

- virtual void MultVectorImpl (Number alpha, const Vector &x, Number beta, Vector &y) const

  Matrix-vector multiply.
- virtual void TransMultVectorImpl (Number alpha, const Vector &x, Number beta, Vector &y) const

  Matrix(transpose) vector multiply.
- virtual bool HasValidNumbersImpl () const

  Method for determining if all stored numbers are valid (i.e., no Inf or Nan).
- virtual void ComputeRowAMaxImpl (Vector &rows_norms, bool init) const

  Compute the max-norm of the rows in the matrix.
- virtual void ComputeColAMaxImpl (Vector &cols_norms, bool init) const

  Compute the max-norm of the columns in the matrix.
- virtual void PrintImpl (const Journalist &jnlst, EJournalLevel level, EJournalCategory category, const std::string &name, Index indent, const std::string &prefix) const

  Print detailed information about the matrix.

Additional Inherited Members

3.48.1 Detailed Description

Class for general matrices stored in triplet format.

In the triplet format, the nonzeros elements of a general matrix is stored in three arrays, Irow, Jcol, and Values, all of length Nonzeros. The first two arrays indicate the location of a non-zero element (row and column indices), and the last array stores the value at that location. If nonzero elements are listed more than once, their values are added.

The structure of the nonzeros (i.e. the arrays Irow and Jcol) cannot be changed after the matrix can been initialized. Only the values of the nonzero elements can be modified.

Note that the first row and column of a matrix has index 1, not 0.

Definition at line 36 of file IpGenTMatrix.hpp.

3.48.2 Member Function Documentation

3.48.2.1 void Ipopt::GenTMatrix::SetValues (const Number ∗ *Values*)

Set values of nonzero elements.

The values of the nonzero elements are copied from the incoming Number array. Important: It is assume that the order of the values in Values corresponds to the one of Irn and Jcn given to one of the constructors above.

3.48.2.2 const Number∗ Ipopt::GenTMatrix::Values ( ) const [inline]

Array with nonzero values (const version).

Definition at line 73 of file IpGenTMatrix.hpp.

3.48.2.3 Number∗ Ipopt::GenTMatrix::Values ( ) [inline]

Array with the nonzero values of this matrix (non-const version).

Use this method only if you are intending to change the values, because the GenTMatrix will be marked as changed.

Definition at line 82 of file IpGenTMatrix.hpp.

3.48.2.4 virtual void Ipopt::GenTMatrix::MultVectorImpl (Number *alpha*, const Vector & *x*, Number *beta*, Vector & *y*) const [protected], [virtual]

Matrix-vector multiply.

Computes y = alpha ∗ Matrix ∗ x + beta ∗ y

Implements Ipopt::Matrix.

3.48.2.5 virtual void Ipopt::GenTMatrix::TransMultVectorImpl (Number *alpha*, const Vector & *x*, Number *beta*, Vector & *y*) const [protected], [virtual]

Matrix(transpose) vector multiply.

Computes y = alpha ∗ Matrix^T ∗ x + beta ∗ y

Implements Ipopt::Matrix.

3.48.2.6 virtual bool Ipopt::GenTMatrix::HasValidNumbersImpl ( ) const [protected], [virtual]

Method for determining if all stored numbers are valid (i.e., no Inf or Nan).

Reimplemented from Ipopt::Matrix.

3.48.2.7 virtual void Ipopt::GenTMatrix::ComputeRowAMaxImpl (Vector & *rows_norms,* bool *init*) const [protected], [virtual]

Compute the max-norm of the rows in the matrix.

The result is stored in rows_norms. The vector is assumed to be initialized.

Implements Ipopt::Matrix.

3.48.2.8 virtual void Ipopt::GenTMatrix::ComputeColAMaxImpl (Vector & *cols_norms,* bool *init*) const [protected], [virtual]

Compute the max-norm of the columns in the matrix.

The result is stored in cols_norms. The vector is assumed to be initialized.

Implements Ipopt::Matrix.

3.48.2.9 virtual void Ipopt::GenTMatrix::PrintImpl (const Journalist & *jnlst,* EJournalLevel *level,* EJournalCategory *category,* const std::string & *name,* Index *indent,* const std::string & *prefix*) const [inline], [protected], [virtual]

Print detailed information about the matrix.

Implements Ipopt::Matrix.

Definition at line 107 of file IpGenTMatrix.hpp.

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

- IpGenTMatrix.hpp

3.49 Ipopt::GenTMatrixSpace Class Reference

This is the matrix space for a GenTMatrix with fixed sparsity structure.

`#include <IpGenTMatrix.hpp>`

Inheritance diagram for Ipopt::GenTMatrixSpace:

Public Member Functions

- GenTMatrix ∗ MakeNewGenTMatrix () const

  Method for creating a new matrix of this specific type.
- virtual Matrix ∗ MakeNew () const

  Overloaded MakeNew method for the MatrixSpace base class.

Constructors / Destructors

- GenTMatrixSpace (Index nRows, Index nCols, Index nonZeros, const Index ∗iRows, const Index ∗jCols)

  Constructor, given the number of rows and columns, as well as the number of nonzeros and the position of the nonzero elements.

- ∼GenTMatrixSpace ()
  Destructor.

Methods describing Matrix structure

- Index Nonzeros () const
  Number of non-zeros in the sparse matrix.
- const Index ∗ Irows () const
  Row index of each non-zero element (counting starts at 1)
- const Index ∗ Jcols () const
  Column index of each non-zero element (counting starts at 1)

3.49.1 Detailed Description

This is the matrix space for a GenTMatrix with fixed sparsity structure.

The sparsity structure is stored here in the matrix space.

Definition at line 164 of file IpGenTMatrix.hpp.

3.49.2 Constructor & Destructor Documentation

3.49.2.1 Ipopt::GenTMatrixSpace::GenTMatrixSpace (Index *nRows,* Index *nCols,* Index *nonZeros,* const Index ∗ *iRows,* const Index ∗ *jCols*)

Constructor, given the number of rows and columns, as well as the number of nonzeros and the position of the nonzero elements.

Note that the counting of the nonzeros starts a 1, i.e., iRows[i]==1 and jCols[i]==1 refers to the first element in the first row. This is in accordance with the HSL data structure.

3.49.3 Member Function Documentation

3.49.3.1 GenTMatrix∗ Ipopt::GenTMatrixSpace::MakeNewGenTMatrix ( ) const [inline]

Method for creating a new matrix of this specific type.

Definition at line 189 of file IpGenTMatrix.hpp.

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

- IpGenTMatrix.hpp

3.50 Ipopt::GradientScaling Class Reference

This class does problem scaling by setting the scaling parameters based on the maximum of the gradient at the user provided initial point.

`#include <IpGradientScaling.hpp>`

Inheritance diagram for Ipopt::GradientScaling:

Public Member Functions

Constructors/Destructors

- **GradientScaling** (const SmartPtr< NLP > &nlp)
- virtual ~GradientScaling ()

  Default destructor.

Static Public Member Functions

- static void RegisterOptions (const SmartPtr< RegisteredOptions > &roptions)

  Methods for IpoptType.

Protected Member Functions

- bool InitializeImpl (const OptionsList &options, const std::string &prefix)

  Initialize the object from the options.
- virtual void DetermineScalingParametersImpl (const SmartPtr< const VectorSpace > x_space, const SmartPtr< const VectorSpace > c_space, const SmartPtr< const VectorSpace > d_space, const SmartPtr< const MatrixSpace > jac_c_space, const SmartPtr< const MatrixSpace > jac_d_space, const SmartPtr< const SymMatrixSpace > h_space, const Matrix &Px_L, const Vector &x_L, const Matrix &Px_U, const Vector &x_U, Number &df, SmartPtr< Vector > &dx, SmartPtr< Vector > &dc, SmartPtr< Vector > &dd)

  This is the method that has to be overloaded by a particular scaling method that somehow computes the scaling vectors dx, dc, and dd.

3.50.1 Detailed Description

This class does problem scaling by setting the scaling parameters based on the maximum of the gradient at the user provided initial point.

Definition at line 21 of file IpGradientScaling.hpp.

3.50.2 Member Function Documentation

3.50.2.1 static void Ipopt::GradientScaling::RegisterOptions (const SmartPtr< RegisteredOptions > & *roptions*) [static]

Methods for IpoptType.

Register the options for this class

3.50.2.2 virtual void Ipopt::GradientScaling::DetermineScalingParametersImpl (const SmartPtr< const VectorSpace > *x_space*, const SmartPtr< const VectorSpace > *c_space*, const SmartPtr< const VectorSpace > *d_space*, const SmartPtr< const MatrixSpace > *jac_c_space*, const SmartPtr< const MatrixSpace > *jac_d_space*, const SmartPtr< const SymMatrixSpace > *h_space*, const Matrix & *Px_L*, const Vector & *x_L*, const Matrix & *Px_U*, const Vector & *x_U*, Number & *df*, SmartPtr< Vector > & *dx*, SmartPtr< Vector > & *dc*, SmartPtr< Vector > & *dd*) [protected], [virtual]

This is the method that has to be overloaded by a particular scaling method that somehow computes the scaling vectors dx, dc, and dd.

The pointers to those vectors can be NULL, in which case no scaling for that item will be done later.

Implements Ipopt::StandardScalingBase.

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

- IpGradientScaling.hpp

3.51 Ipopt::HessianUpdater Class Reference

Abstract base class for objects responsible for updating the Hessian information.

`#include <IpHessianUpdater.hpp>`

Inheritance diagram for Ipopt::HessianUpdater:

Public Member Functions

- virtual bool InitializeImpl (const OptionsList &options, const std::string &prefix)=0

  overloaded from AlgorithmStrategyObject
- virtual void UpdateHessian ()=0

  Update the Hessian based on the current information in IpData, and possibly on information from previous calls.

Constructors/Destructors

- HessianUpdater ()

  Default Constructor.
- virtual ~HessianUpdater ()

  Default destructor.

Additional Inherited Members

3.51.1 Detailed Description

Abstract base class for objects responsible for updating the Hessian information.

This can be done using exact second derivatives from the NLP, or by a quasi-Newton Option. The result is put into the W field in IpData.

Definition at line 22 of file IpHessianUpdater.hpp.

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

- IpHessianUpdater.hpp

3.52 Ipopt::IdentityMatrix Class Reference

Class for Matrices which are multiples of the identity matrix.

`#include <IpIdentityMatrix.hpp>`

Inheritance diagram for Ipopt::IdentityMatrix:

Public Member Functions

- void SetFactor (Number factor)
  Method for setting the factor for the identity matrix.
- Number GetFactor () const
  Method for getting the factor for the identity matrix.
- Index Dim () const
  Method for obtaining the dimention of the matrix.

Constructors / Destructors

- IdentityMatrix (const SymMatrixSpace *owner_space)
  Constructor, initializing with dimensions of the matrix (true identity matrix).
- ~IdentityMatrix ()
  Destructor.

Protected Member Functions

Methods overloaded from matrix

- virtual void MultVectorImpl (Number alpha, const Vector &x, Number beta, Vector &y) const
  Matrix-vector multiply.
- virtual void AddMSinvZImpl (Number alpha, const Vector &S, const Vector &Z, Vector &X) const
  X = X + alpha(Matrix S^{-1} Z).*
- virtual bool HasValidNumbersImpl () const
  Method for determining if all stored numbers are valid (i.e., no Inf or Nan).
- virtual void ComputeRowAMaxImpl (Vector &rows_norms, bool init) const
  Compute the max-norm of the rows in the matrix.
- virtual void PrintImpl (const Journalist &jnlst, EJournalLevel level, EJournalCategory category, const std::string &name, Index indent, const std::string &prefix) const
  Print detailed information about the matrix.

Additional Inherited Members

3.52.1 Detailed Description

Class for Matrices which are multiples of the identity matrix.

Definition at line 21 of file IpIdentityMatrix.hpp.

3.52.2 Member Function Documentation

3.52.2.1 void Ipopt::IdentityMatrix::SetFactor (Number *factor*) [inline]

Method for setting the factor for the identity matrix.

Definition at line 38 of file IpIdentityMatrix.hpp.

3.52.2.2 Number Ipopt::IdentityMatrix::GetFactor ( ) const [inline]

Method for getting the factor for the identity matrix.

Definition at line 44 of file IpIdentityMatrix.hpp.

3.52.2.3 Index Ipopt::IdentityMatrix::Dim ( ) const

Method for obtaining the dimention of the matrix.

3.52.2.4 virtual void Ipopt::IdentityMatrix::MultVectorImpl (Number *alpha*, const Vector & *x*, Number *beta*, Vector & *y*) const [protected], [virtual]

Matrix-vector multiply.

Computes y = alpha ∗ Matrix ∗ x + beta ∗ y

Implements Ipopt::Matrix.

3.52.2.5 virtual void Ipopt::IdentityMatrix::AddMSinvZImpl (Number *alpha*, const Vector & *S*, const Vector & *Z*, Vector & *X*) const [protected], [virtual]

X = X + alpha∗(Matrix S^{-1} Z).

Prototype for this specialize method is provided, but for efficient implementation it should be overloaded for the expansion matrix.

Reimplemented from Ipopt::Matrix.

3.52.2.6 virtual bool Ipopt::IdentityMatrix::HasValidNumbersImpl ( ) const [protected], [virtual]

Method for determining if all stored numbers are valid (i.e., no Inf or Nan).

Reimplemented from Ipopt::Matrix.

3.52.2.7 virtual void Ipopt::IdentityMatrix::ComputeRowAMaxImpl (Vector & *rows_norms*, bool *init*) const [protected], [virtual]

Compute the max-norm of the rows in the matrix.

The result is stored in rows_norms. The vector is assumed to be initialized.

Implements Ipopt::Matrix.

3.52.2.8 virtual void Ipopt::IdentityMatrix::PrintImpl (const Journalist & *jnlst*, EJournalLevel *level*, EJournalCategory *category*, const std::string & *name*, Index *indent*, const std::string & *prefix*) const [protected], [virtual]

Print detailed information about the matrix.

Implements Ipopt::Matrix.

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

- IpIdentityMatrix.hpp

3.53 Ipopt::IdentityMatrixSpace Class Reference

This is the matrix space for IdentityMatrix.

`#include <IpIdentityMatrix.hpp>`

Inheritance diagram for Ipopt::IdentityMatrixSpace:

Public Member Functions

- virtual SymMatrix * MakeNewSymMatrix () const

  Overloaded MakeNew method for the SymMatrixSpace base class.
- IdentityMatrix * MakeNewIdentityMatrix () const

  Method for creating a new matrix of this specific type.

Constructors / Destructors

- IdentityMatrixSpace (Index dim)

  Constructor, given the dimension of the matrix.
- virtual ~IdentityMatrixSpace ()

  Destructor.

3.53.1 Detailed Description

This is the matrix space for IdentityMatrix.

Definition at line 99 of file IpIdentityMatrix.hpp.

3.53.2 Constructor & Destructor Documentation

3.53.2.1 Ipopt::IdentityMatrixSpace::IdentityMatrixSpace (Index *dim*) [inline]

Constructor, given the dimension of the matrix.

Definition at line 105 of file IpIdentityMatrix.hpp.

3.53.3 Member Function Documentation

3.53.3.1 IdentityMatrix* Ipopt::IdentityMatrixSpace::MakeNewIdentityMatrix ( ) const [inline]

Method for creating a new matrix of this specific type.

Definition at line 123 of file IpIdentityMatrix.hpp.

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

- IpIdentityMatrix.hpp

3.54 Ipopt::InexactAlgorithmBuilder Class Reference

Builder to create a complete IpoptAlg object for the inexact step computation version.

`#include <IpInexactAlgBuilder.hpp>`

Inheritance diagram for Ipopt::InexactAlgorithmBuilder:

Public Member Functions

Constructors/Destructors

- InexactAlgorithmBuilder ()

  Constructor.
- virtual ∼InexactAlgorithmBuilder ()

  Destructor.

Methods to build parts of the algorithm

- virtual void **BuildIpoptObjects** (const Journalist &jnlst, const OptionsList &options, const std::string &prefix, const SmartPtr< NLP > &nlp, SmartPtr< IpoptNLP > &ip_nlp, SmartPtr< IpoptData > &ip_data, SmartPtr< IpoptCalculatedQuantities > &ip_cq)
- virtual SmartPtr< IpoptAlgorithm > **BuildBasicAlgorithm** (const Journalist &jnlst, const OptionsList &options, const std::string &prefix)

Static Public Member Functions

- static void RegisterOptions (SmartPtr< RegisteredOptions > roptions)

  Methods for IpoptTypeInfo.

3.54.1 Detailed Description

Builder to create a complete IpoptAlg object for the inexact step computation version.

Definition at line 21 of file IpInexactAlgBuilder.hpp.

3.54.2 Member Function Documentation

3.54.2.1 static void Ipopt::InexactAlgorithmBuilder::RegisterOptions (SmartPtr< RegisteredOptions > *roptions*) [static]

Methods for IpoptTypeInfo.

register the options used by the algorithm builder

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

- IpInexactAlgBuilder.hpp

3.55 Ipopt::InexactCq Class Reference

Class for all Chen-Goldfarb penalty method specific calculated quantities.

`#include <IpInexactCq.hpp>`

Inheritance diagram for Ipopt::InexactCq:

Public Member Functions

- bool Initialize (const Journalist &jnlst, const OptionsList &options, const std::string &prefix)
  This method must be called to initialize the global algorithmic parameters.
- SmartPtr< const Vector > curr_jac_cdT_times_curr_cdminuss ()
  Gradient of infeasibility w.r.t.
- SmartPtr< const Vector > curr_scaling_slacks ()
  Vector of all inequality slacks for doing the slack-based scaling.
- SmartPtr< const Vector > curr_slack_scaled_d_minus_s ()
  Vector with the slack-scaled d minus s inequalities.
- Number curr_scaled_Ac_norm ()
  Scaled norm of Ac.
- Number curr_scaled_A_norm2 ()
  Scaled, squared norm of A.
- Number slack_scaled_norm (const Vector &x, const Vector &s)
  Compute the 2-norm of a slack-scaled vector with x and s component.
- SmartPtr< const Vector > curr_W_times_vec_x (const Vector &vec_x)
  Compute x component of the W∗vec product for the current Hessian and a vector.
- SmartPtr< const Vector > curr_W_times_vec_s (const Vector &vec_s)
  Compute s component of the W∗vec product for the current Hessian and a vector.
- SmartPtr< const Vector > curr_Wu_x ()
  Compute x component of the W∗u product for the current values.
- SmartPtr< const Vector > curr_Wu_s ()
  Compute s component of the W∗u product for the current values.
- Number curr_uWu ()
  Compute the u^T∗W∗u product for the current values.
- SmartPtr< const Vector > curr_jac_times_normal_c ()
  Compute the c-component of the product of the current constraint Jacobian with the current normal step.
- SmartPtr< const Vector > curr_jac_times_normal_d ()
  Compute the d-component of the product of the current constraint Jacobian with the current normal step.

Constructors/Destructors

- InexactCq (IpoptNLP ∗ip_nlp, IpoptData ∗ip_data, IpoptCalculatedQuantities ∗ip_cq)
  Constructor.
- virtual ~InexactCq ()
  Default destructor.

Static Public Member Functions

- static void RegisterOptions (const SmartPtr< RegisteredOptions > &roptions)
  Methods for IpoptType.

3.55.1 Detailed Description

Class for all Chen-Goldfarb penalty method specific calculated quantities.

Definition at line 22 of file IpInexactCq.hpp.

3.55.2 Member Function Documentation

3.55.2.1 bool Ipopt::InexactCq::Initialize (const Journalist & *jnlst*, const OptionsList & *options*, const std::string & *prefix*) [virtual]

This method must be called to initialize the global algorithmic parameters.

The parameters are taken from the OptionsList object.

Implements Ipopt::IpoptAdditionalCq.

3.55.2.2 SmartPtr<const Vector> Ipopt::InexactCq::curr_jac_cdT_times_curr_cdminuss ( )

Gradient of infeasibility w.r.t.

x. Jacobian of equality constraints transpose times the equality constraints plus Jacobian of the inequality constraints transpose times the inequality constraints (including slacks).

3.55.2.3 SmartPtr<const Vector> Ipopt::InexactCq::curr_Wu_x ( )

Compute x component of the W∗u product for the current values.

u here is the tangential step.

3.55.2.4 SmartPtr<const Vector> Ipopt::InexactCq::curr_Wu_s ( )

Compute s component of the W∗u product for the current values.

u here is the tangential step.

3.55.2.5 Number Ipopt::InexactCq::curr_uWu ( )

Compute the u^T∗W∗u product for the current values.

u here is the tangential step.

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

- IpInexactCq.hpp

3.56 Ipopt::InexactData Class Reference

Class to organize all the additional data required by the Chen-Goldfarb penalty function algorithm.

`#include <IpInexactData.hpp>`

Inheritance diagram for Ipopt::InexactData:

Public Member Functions

Constructors/Destructors

- InexactData ()

Constructor.
- ∼InexactData ()

  Default destructor.

Methods overloaded from IpoptAdditionalData

- bool Initialize (const Journalist &jnlst, const OptionsList &options, const std::string &prefix)

  This method must be called to initialize the global algorithmic parameters.
- bool InitializeDataStructures ()

  Initialize Data Structures at the beginning.
- void AcceptTrialPoint ()

  Do whatever is necessary to accept a trial point as current iterate.

Normal step set and accessor methods

- void **set_normal_x** (SmartPtr< Vector > &normal_x)
- void **set_normal_s** (SmartPtr< Vector > &normal_s)
- SmartPtr< const Vector > **normal_x** ()
- SmartPtr< const Vector > **normal_s** ()

Tangential step set and accessor methods

- void **set_tangential_x** (SmartPtr< const Vector > &tangential_x)
- void **set_tangential_s** (SmartPtr< const Vector > &tangential_s)
- SmartPtr< const Vector > **tangential_x** ()
- SmartPtr< const Vector > **tangential_s** ()

Flag indicating if most recent step has been fully

accepted.

This is used to determine if the trust region radius should be increased.

- void **set_full_step_accepted** (bool full_step_accepted)
- bool **full_step_accepted** ()

Current value of penalty parameter

- void **set_curr_nu** (Number nu)
- Number **curr_nu** ()

Current normal step computation flag

- void **set_compute_normal** (bool compute_normal)
- bool **compute_normal** ()

Next iteration normal step computation flag

- void **set_next_compute_normal** (bool next_compute_normal)
- bool **next_compute_normal** ()

3.56.1 Detailed Description

Class to organize all the additional data required by the Chen-Goldfarb penalty function algorithm.

Definition at line 19 of file IpInexactData.hpp.

3.56.2 Member Function Documentation

3.56.2.1 bool Ipopt::InexactData::Initialize (const Journalist & *jnlst*, const OptionsList & *options*, const std::string & *prefix*) [virtual]

This method must be called to initialize the global algorithmic parameters.

The parameters are taken from the OptionsList object.

Implements Ipopt::IpoptAdditionalData.

3.56.2.2 bool Ipopt::InexactData::InitializeDataStructures ( ) [virtual]

Initialize Data Structures at the beginning.

Implements Ipopt::IpoptAdditionalData.

3.56.2.3 void Ipopt::InexactData::AcceptTrialPoint ( ) [virtual]

Do whatever is necessary to accept a trial point as current iterate.

This is also used to finish an iteration, i.e., to release memory, and to reset any flags for a new iteration.

Implements Ipopt::IpoptAdditionalData.

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

- IpInexactData.hpp

3.57 Ipopt::InexactDoglegNormalStep Class Reference

Compute the normal step using a dogleg approach.

`#include <IpInexactDoglegNormal.hpp>`

Inheritance diagram for Ipopt::InexactDoglegNormalStep:

Public Member Functions

- virtual bool InitializeImpl (const OptionsList &options, const std::string &prefix)

  overloaded from AlgorithmStrategyObject
- virtual bool ComputeNormalStep (SmartPtr< Vector > &normal_x, SmartPtr< Vector > &normal_s)

  Method for computing the normal step.

Constructors/Destructors

- InexactDoglegNormalStep (SmartPtr< InexactNewtonNormalStep > newton_step, SmartPtr< InexactNormalTerminationTester > normal_tester=NULL)

  Default onstructor.
- virtual ~InexactDoglegNormalStep ()

  Default destructor.

Static Public Member Functions

- static void RegisterOptions (SmartPtr< RegisteredOptions > roptions)

    Methods for IpoptType.

Additional Inherited Members

3.57.1 Detailed Description

Compute the normal step using a dogleg approach.

Definition at line 20 of file IpInexactDoglegNormal.hpp.

3.57.2 Member Function Documentation

3.57.2.1 virtual bool Ipopt::InexactDoglegNormalStep::ComputeNormalStep (SmartPtr< Vector > & *normal_x*, SmartPtr< Vector > & *normal_s*) [virtual]

Method for computing the normal step.

The computed step is returned as normal_x and normal_s, for the x and s variables, respectively. These quantities are not slack-scaled. If the step cannot be computed, this method returns false.

Implements Ipopt::InexactNormalStepCalculator.

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

- IpInexactDoglegNormal.hpp

3.58 Ipopt::InexactLSAcceptor Class Reference

Penalty function line search for the inexact step algorithm version.

`#include <IpInexactLSAcceptor.hpp>`

Inheritance diagram for Ipopt::InexactLSAcceptor:

Public Member Functions

- virtual bool InitializeImpl (const OptionsList &options, const std::string &prefix)

    InitializeImpl - overloaded from AlgorithmStrategyObject.
- virtual void Reset ()

    Reset the acceptor.
- virtual void InitThisLineSearch (bool in_watchdog)

    Initialization for the next line search.
- virtual void PrepareRestoPhaseStart ()

    Method that is called before the restoration phase is called.
- virtual Number CalculateAlphaMin ()

    Method returning the lower bound on the trial step sizes.
- virtual bool CheckAcceptabilityOfTrialPoint (Number alpha_primal)

    Method for checking if current trial point is acceptable.

- virtual bool TrySecondOrderCorrection (Number alpha_primal_test, Number &alpha_primal, SmartPtr< IteratesVector > &actual_delta)

  Try a second order correction for the constraints.
- virtual bool TryCorrector (Number alpha_primal_test, Number &alpha_primal, SmartPtr< IteratesVector > &actual_delta)

  Try higher order corrector (for fast local convergence).
- virtual char UpdateForNextIteration (Number alpha_primal_test)

  Method for ending the current line search.
- virtual void StartWatchDog ()

  Method for setting internal data if the watchdog procedure is started.
- virtual void StopWatchDog ()

  Method for setting internal data if the watchdog procedure is stopped.
- virtual Number ComputeAlphaForY (Number alpha_primal, Number alpha_dual, SmartPtr< IteratesVector > &delta)

  Method for updating the equality constraint multipliers.
- virtual bool HasComputeAlphaForY () const

  Method returning true of ComputeAlphaForY is implemented for this acceptor.

Constructors/Destructors

- InexactLSAcceptor ()

  Constructor.
- virtual ∼InexactLSAcceptor ()

  Default destructor.

Trial Point Accepting Methods. Used internally to check certain

acceptability criteria and used externally (by the restoration phase convergence check object, for instance)

- bool IsAcceptableToCurrentIterate (Number trial_barr, Number trial_theta, bool called_from_restoration=false) const

  Checks if a trial point is acceptable to the current iterate.

Static Public Member Functions

- static void RegisterOptions (SmartPtr< RegisteredOptions > roptions)

  Methods for OptionsList.

Protected Member Functions

- InexactData & InexData ()

  Method to easily access Inexact data.
- InexactCq & InexCq ()

  Method to easily access Inexact calculated quantities.

3.58.1 Detailed Description

Penalty function line search for the inexact step algorithm version.

Definition at line 22 of file IpInexactLSAcceptor.hpp.

3.58.2 Constructor & Destructor Documentation

3.58.2.1 Ipopt::InexactLSAcceptor::InexactLSAcceptor ( )

Constructor.

The PDSystemSolver object only needs to be provided (i.e. not NULL) if second order correction or corrector steps are to be used.

3.58.3 Member Function Documentation

3.58.3.1 virtual void Ipopt::InexactLSAcceptor::Reset ( ) [virtual]

Reset the acceptor.

This function should be called if all previous information should be discarded when the line search is performed the next time. For example, this method should be called if the barrier parameter is changed.

Implements Ipopt::BacktrackingLSAcceptor.

3.58.3.2 virtual void Ipopt::InexactLSAcceptor::InitThisLineSearch (bool *in_watchdog*) [virtual]

Initialization for the next line search.

The flag in_watchdog indicates if we are currently in an active watchdog procedure. Here is where the penalty parameter is updated.

Implements Ipopt::BacktrackingLSAcceptor.

3.58.3.3 virtual void Ipopt::InexactLSAcceptor::PrepareRestoPhaseStart ( ) [virtual]

Method that is called before the restoration phase is called.

For now, we just terminate if this is called.

Implements Ipopt::BacktrackingLSAcceptor.

3.58.3.4 virtual Number Ipopt::InexactLSAcceptor::CalculateAlphaMin ( ) [virtual]

Method returning the lower bound on the trial step sizes.

Implements Ipopt::BacktrackingLSAcceptor.

3.58.3.5 virtual bool Ipopt::InexactLSAcceptor::CheckAcceptabilityOfTrialPoint (Number *alpha_primal*) [virtual]

Method for checking if current trial point is acceptable.

It is assumed that the delta information in ip_data is the search direction used in criteria. The primal trial point has to be set before the call.

Implements Ipopt::BacktrackingLSAcceptor.

3.58.3.6 virtual bool Ipopt::InexactLSAcceptor::TrySecondOrderCorrection (Number *alpha_primal_test,* Number & *alpha_primal,* SmartPtr< IteratesVector > & *actual_delta*) [virtual]

Try a second order correction for the constraints.

For the inexact version, this always returns false because a second order step is too expensive.

Implements Ipopt::BacktrackingLSAcceptor.

3.58.3.7 virtual bool Ipopt::InexactLSAcceptor::TryCorrector (Number *alpha_primal_test,* Number & *alpha_primal,* SmartPtr< IteratesVector > & *actual_delta*) [virtual]

Try higher order corrector (for fast local convergence).

In contrast to a second order correction step, which tries to make an unacceptable point acceptable by improving constraint violation, this corrector step is tried even if the regular primal-dual step is acceptable.

Implements Ipopt::BacktrackingLSAcceptor.

3.58.3.8 virtual char Ipopt::InexactLSAcceptor::UpdateForNextIteration (Number *alpha_primal_test*) [virtual]

Method for ending the current line search.

When it is called, the internal data should be updates. alpha_primal_test is the value of alpha that has been used for in the acceptence test ealier.

Implements Ipopt::BacktrackingLSAcceptor.

3.58.3.9 virtual void Ipopt::InexactLSAcceptor::StartWatchDog ( ) [virtual]

Method for setting internal data if the watchdog procedure is started.

Implements Ipopt::BacktrackingLSAcceptor.

3.58.3.10 virtual void Ipopt::InexactLSAcceptor::StopWatchDog ( ) [virtual]

Method for setting internal data if the watchdog procedure is stopped.

Implements Ipopt::BacktrackingLSAcceptor.

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

- IpInexactLSAcceptor.hpp

3.59 Ipopt::InexactNewtonNormalStep Class Reference

Compute the "Newton" normal step from the (slack-scaled) augmented system.

`#include <IpInexactNewtonNormal.hpp>`

Inheritance diagram for Ipopt::InexactNewtonNormalStep:

Public Member Functions

- virtual bool InitializeImpl (const OptionsList &options, const std::string &prefix)

overloaded from AlgorithmStrategyObject

- virtual bool ComputeNewtonNormalStep (Vector &newton_x, Vector &newton_s)

  Method for computing the normal step.

Constructors/Destructors

- InexactNewtonNormalStep (SmartPtr< AugSystemSolver > aug_solver)

  Default onstructor.
- virtual ∼InexactNewtonNormalStep ()

  Default destructor.

Static Public Member Functions

- static void RegisterOptions (SmartPtr< RegisteredOptions > roptions)

  Methods for IpoptType.

Protected Member Functions

- InexactData & InexData ()

  Method to easily access Inexact data.
- InexactCq & InexCq ()

  Method to easily access Inexact calculated quantities.

3.59.1 Detailed Description

Compute the "Newton" normal step from the (slack-scaled) augmented system.

Definition at line 21 of file IpInexactNewtonNormal.hpp.

3.59.2 Member Function Documentation

3.59.2.1 virtual bool Ipopt::InexactNewtonNormalStep::ComputeNewtonNormalStep (Vector & *newton_x*, Vector & *newton_s*) `[virtual]`

Method for computing the normal step.

The computed step is returned as normal_x and normal_s, for the x and s variables, respectively. These quantities are not in the original space, but in the space scaled by the slacks. If the step cannot be computed, this method returns false.

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

- IpInexactNewtonNormal.hpp

3.60 Ipopt::InexactNormalStepCalculator Class Reference

Base class for computing the normal step for the inexact step calculation algorithm.

`#include <IpInexactNormalStepCalc.hpp>`

Inheritance diagram for Ipopt::InexactNormalStepCalculator:

Public Member Functions

- virtual bool InitializeImpl (const OptionsList &options, const std::string &prefix)=0

  overloaded from AlgorithmStrategyObject
- virtual bool ComputeNormalStep (SmartPtr< Vector > &normal_x, SmartPtr< Vector > &normal_s)=0

  Method for computing the normal step.

Constructors/Destructors

- InexactNormalStepCalculator ()

  Default onstructor.
- virtual ∼InexactNormalStepCalculator ()

  Default destructor.

Protected Member Functions

- InexactData & InexData ()

  Method to easily access Inexact data.
- InexactCq & InexCq ()

  Method to easily access Inexact calculated quantities.

3.60.1 Detailed Description

Base class for computing the normal step for the inexact step calculation algorithm.

Definition at line 20 of file IpInexactNormalStepCalc.hpp.

3.60.2 Member Function Documentation

3.60.2.1 virtual bool Ipopt::InexactNormalStepCalculator::ComputeNormalStep (SmartPtr< Vector > & *normal_x*, SmartPtr< Vector > & *normal_s*) [pure virtual]

Method for computing the normal step.

The computed step is returned as normal_x and normal_s, for the x and s variables, respectively. These quantities are not slack-scaled. If the step cannot be computed, this method returns false.

Implemented in Ipopt::InexactDoglegNormalStep.

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

- IpInexactNormalStepCalc.hpp

3.61 Ipopt::InexactNormalTerminationTester Class Reference

This class implements the termination tests for the primal-dual system.

`#include <IpInexactNormalTerminationTester.hpp>`

Inheritance diagram for Ipopt::InexactNormalTerminationTester:

Public Member Functions

- virtual bool InitializeImpl (const OptionsList &options, const std::string &prefix)
  Implementation of the initialization method that has to be overloaded by for each derived class.
- virtual bool InitializeSolve ()
  Method for initializing for the next iterative solve.
- virtual ETerminationTest TestTermination (Index ndim, const Number ∗sol, const Number ∗resid, Index iter, Number norm2_rhs)
  This method checks if the current soltion of the iterative linear solver is good enough (by returning the corresponding satisfied termination test), or if the Hessian should be modified.
- virtual void Clear ()
  This method can be called after the Solve is over and we can delete anything that has been allocated to free memory.
- virtual Index GetSolverIterations () const
  Return the number of iterative solver iteration from the most recent solve.
- void Set_c_Avc_norm_cauchy (Number c_Avc_norm_cauchy)
  Method for setting the normal problem objective function value at the Cauchy step.

/Destructor

- InexactNormalTerminationTester ()
  Default constructor.
- virtual ∼InexactNormalTerminationTester ()
  Default destructor.

Static Public Member Functions

- static void RegisterOptions (SmartPtr< RegisteredOptions > roptions)
  Methods for IpoptType.

Additional Inherited Members

3.61.1 Detailed Description

This class implements the termination tests for the primal-dual system.

Definition at line 20 of file IpInexactNormalTerminationTester.hpp.

3.61.2 Member Function Documentation

3.61.2.1 virtual bool Ipopt::InexactNormalTerminationTester::InitializeImpl (const OptionsList & *options,* const std::string & *prefix*) [virtual]

Implementation of the initialization method that has to be overloaded by for each derived class.

Implements Ipopt::IterativeSolverTerminationTester.

3.61.2.2 virtual bool Ipopt::InexactNormalTerminationTester::InitializeSolve ( ) [virtual]

Method for initializing for the next iterative solve.

This must be call before the test methods are called.

Implements Ipopt::IterativeSolverTerminationTester.

3.61.2.3 virtual ETerminationTest Ipopt::InexactNormalTerminationTester::TestTermination (Index *ndim*, const Number ∗ *sol*, const Number ∗ *resid*, Index *iter*, Number *norm2_rhs*) [virtual]

This method checks if the current soltion of the iterative linear solver is good enough (by returning the corresponding satisfied termination test), or if the Hessian should be modified.

The input is the dimension of the augmented system, the current solution vector of the augmented system, the current residual vector.

Implements Ipopt::IterativeSolverTerminationTester.

3.61.2.4 virtual void Ipopt::InexactNormalTerminationTester::Clear ( ) [virtual]

This method can be called after the Solve is over and we can delete anything that has been allocated to free memory.

Implements Ipopt::IterativeSolverTerminationTester.

3.61.2.5 void Ipopt::InexactNormalTerminationTester::Set_c_Avc_norm_cauchy (Number *c_Avc_norm_cauchy*) [inline]

Method for setting the normal problem objective function value at the Cauchy step.

This must be called by the Dogleg object.

Definition at line 71 of file IpInexactNormalTerminationTester.hpp.

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

- IpInexactNormalTerminationTester.hpp

3.62 Ipopt::InexactPDSolver Class Reference

This is the implemetation of the Primal-Dual System, allowing the usage of an inexact linear solver.

`#include <IpInexactPDSolver.hpp>`

Inheritance diagram for Ipopt::InexactPDSolver:

Public Member Functions

- bool InitializeImpl (const OptionsList &options, const std::string &prefix)

  Implementation of the initialization method that has to be overloaded by for each derived class.
- virtual bool Solve (const IteratesVector &rhs, IteratesVector &sol)

  Solve the primal dual system, given one right hand side.

/Destructor

- InexactPDSolver (AugSystemSolver &augSysSolver, PDPerturbationHandler &perturbHandler)

  Constructor that takes in the Augmented System solver that is to be used inside.
- virtual ∼InexactPDSolver ()

  Default destructor.

Static Public Member Functions

- static void RegisterOptions (SmartPtr< RegisteredOptions > roptions)
  Methods for IpoptType.

Additional Inherited Members

3.62.1 Detailed Description

This is the implemetation of the Primal-Dual System, allowing the usage of an inexact linear solver.

The step computed is usually for the tangential step.

Definition at line 24 of file IpInexactPDSolver.hpp.

3.62.2 Member Function Documentation

3.62.2.1 bool Ipopt::InexactPDSolver::InitializeImpl (const OptionsList & *options*, const std::string & *prefix*) [virtual]

Implementation of the initialization method that has to be overloaded by for each derived class.

Implements Ipopt::AlgorithmStrategyObject.

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

- IpInexactPDSolver.hpp

3.63 Ipopt::InexactPDTerminationTester Class Reference

This class implements the termination tests for the primal-dual system.

`#include <IpInexactPDTerminationTester.hpp>`

Inheritance diagram for Ipopt::InexactPDTerminationTester:

Public Member Functions

- virtual bool InitializeImpl (const OptionsList &options, const std::string &prefix)
  Implementation of the initialization method that has to be overloaded by for each derived class.
- virtual bool InitializeSolve ()
  Method for initializing for the next iterative solve.
- virtual ETerminationTest TestTermination (Index ndim, const Number *sol, const Number *resid, Index iter, Number norm2_rhs)
  This method checks if the current soltion of the iterative linear solver is good enough (by returning the corresponding satisfied termination test), or if the Hessian should be modified.
- virtual void Clear ()
  This method can be called after the Solve is over and we can delete anything that has been allocated to free memory.
- virtual Index GetSolverIterations () const
  Return the number of iterative solver iteration from the most recent solve.

/Destructor

- InexactPDTerminationTester ()
  Default constructor.
- virtual ∼InexactPDTerminationTester ()
  Default destructor.

Static Public Member Functions

- static void RegisterOptions (SmartPtr< RegisteredOptions > roptions)
  Methods for IpoptType.

Additional Inherited Members

3.63.1 Detailed Description

This class implements the termination tests for the primal-dual system.

Definition at line 20 of file IpInexactPDTerminationTester.hpp.

3.63.2 Member Function Documentation

3.63.2.1 virtual bool Ipopt::InexactPDTerminationTester::InitializeImpl (const OptionsList & *options*, const std::string & *prefix*) [virtual]

Implementation of the initialization method that has to be overloaded by for each derived class.

Implements Ipopt::IterativeSolverTerminationTester.

3.63.2.2 virtual bool Ipopt::InexactPDTerminationTester::InitializeSolve ( ) [virtual]

Method for initializing for the next iterative solve.

This must be call before the test methods are called.

Implements Ipopt::IterativeSolverTerminationTester.

3.63.2.3 virtual ETerminationTest Ipopt::InexactPDTerminationTester::TestTermination (Index *ndim*, const Number ∗ *sol*, const Number ∗ *resid*, Index *iter*, Number *norm2_rhs*) [virtual]

This method checks if the current soltion of the iterative linear solver is good enough (by returning the corresponding satisfied termination test), or if the Hessian should be modified.

The input is the dimension of the augmented system, the current solution vector of the augmented system, the current residual vector.

Implements Ipopt::IterativeSolverTerminationTester.

3.63.2.4 virtual void Ipopt::InexactPDTerminationTester::Clear ( ) [virtual]

This method can be called after the Solve is over and we can delete anything that has been allocated to free memory.

Implements Ipopt::IterativeSolverTerminationTester.

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

- IpInexactPDTerminationTester.hpp

3.64 Ipopt::InexactSearchDirCalculator Class Reference

Implementation of the search direction calculator that computes the search direction using iterative linear solvers.

`#include <IpInexactSearchDirCalc.hpp>`

Inheritance diagram for Ipopt::InexactSearchDirCalculator:

Public Member Functions

- virtual bool InitializeImpl (const OptionsList &options, const std::string &prefix)

  overloaded from AlgorithmStrategyObject
- virtual bool ComputeSearchDirection ()

  Method for computing the search direction.

Constructors/Destructors

- InexactSearchDirCalculator (SmartPtr< InexactNormalStepCalculator > normal_step_calculator, SmartPtr< InexactPDSolver > inexact_pd_solver)

  Constructor.
- virtual ∼InexactSearchDirCalculator ()

  Default destructor.

Static Public Member Functions

- static void RegisterOptions (SmartPtr< RegisteredOptions > roptions)

  Methods for IpoptType.

Additional Inherited Members

3.64.1 Detailed Description

Implementation of the search direction calculator that computes the search direction using iterative linear solvers.

Those steps do not necessarily satisfy the linearized KKT conditions with high accuracy.

Definition at line 24 of file IpInexactSearchDirCalc.hpp.

3.64.2 Member Function Documentation

3.64.2.1 virtual bool Ipopt::InexactSearchDirCalculator::ComputeSearchDirection ( ) [virtual]

Method for computing the search direction.

In this version, we compute a normal and a tangential component, which are stored in the InexactData object. The overall step is still stored in the IpoptData object.

Implements Ipopt::SearchDirectionCalculator.

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

- IpInexactSearchDirCalc.hpp

3.65 Ipopt::InexactTSymScalingMethod Class Reference

Class for the method for computing scaling factors for symmetric matrices in triplet format, specifically for the inexaxct algorithm.

`#include <IpInexactTSymScalingMethod.hpp>`

Inheritance diagram for Ipopt::InexactTSymScalingMethod:

Public Member Functions

- virtual bool InitializeImpl (const OptionsList &options, const std::string &prefix)

    overloaded from AlgorithmStrategyObject
- virtual bool ComputeSymTScalingFactors (Index n, Index nnz, const ipfint ∗airn, const ipfint ∗ajcn, const double ∗a, double ∗scaling_factors)

    Method for computing the symmetric scaling factors, given the symmtric matrix in triplet (MA27) format.

Constructor/Destructor

- **InexactTSymScalingMethod** ()
- virtual ∼**InexactTSymScalingMethod** ()

Additional Inherited Members

3.65.1 Detailed Description

Class for the method for computing scaling factors for symmetric matrices in triplet format, specifically for the inexaxct algorithm.

The scaling is only considering the current slacks.

Definition at line 24 of file IpInexactTSymScalingMethod.hpp.

3.65.2 Member Function Documentation

3.65.2.1 virtual bool Ipopt::InexactTSymScalingMethod::ComputeSymTScalingFactors (Index *n*, Index *nnz*, const ipfint ∗ *airn*, const ipfint ∗ *ajcn*, const double ∗ *a*, double ∗ *scaling_factors*) `[virtual]`

Method for computing the symmetric scaling factors, given the symmtric matrix in triplet (MA27) format.

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

- IpInexactTSymScalingMethod.hpp

3.66 Ipopt::IpoptAdditionalCq Class Reference

Base class for additional calculated quantities that is special to a particular type of algorithm, such as the CG penalty function, or using iterative linear solvers.

`#include <IpIpoptCalculatedQuantities.hpp>`

Inheritance diagram for Ipopt::IpoptAdditionalCq:

Public Member Functions

- virtual bool Initialize (const Journalist &jnlst, const OptionsList &options, const std::string &prefix)=0

  This method is called to initialize the global algorithmic parameters.

Constructors/Destructors

- IpoptAdditionalCq ()

  Default Constructor.
- virtual ∼IpoptAdditionalCq ()

  Default destructor.

3.66.1 Detailed Description

Base class for additional calculated quantities that is special to a particular type of algorithm, such as the CG penalty function, or using iterative linear solvers.

The regular IpoptCalculatedQuantities object should be given a derivation of this base class when it is created.

Definition at line 40 of file IpIpoptCalculatedQuantities.hpp.

3.66.2 Member Function Documentation

3.66.2.1 virtual bool Ipopt::IpoptAdditionalCq::Initialize (const Journalist & *jnlst*, const OptionsList & *options*, const std::string & *prefix*) [pure virtual]

This method is called to initialize the global algorithmic parameters.

The parameters are taken from the OptionsList object.

Implemented in Ipopt::InexactCq, and Ipopt::CGPenaltyCq.

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

- IpIpoptCalculatedQuantities.hpp

3.67 Ipopt::IpoptAdditionalData Class Reference

Base class for additional data that is special to a particular type of algorithm, such as the CG penalty function, or using iterative linear solvers.

`#include <IpIpoptData.hpp>`

Inheritance diagram for Ipopt::IpoptAdditionalData:

Public Member Functions

- virtual bool Initialize (const Journalist &jnlst, const OptionsList &options, const std::string &prefix)=0

This method is called to initialize the global algorithmic parameters.

- virtual bool InitializeDataStructures ()=0

  Initialize Data Structures at the beginning.
- virtual void AcceptTrialPoint ()=0

  Do whatever is necessary to accept a trial point as current iterate.

Constructors/Destructors

- IpoptAdditionalData ()

  Default Constructor.
- virtual ∼IpoptAdditionalData ()

  Default destructor.

3.67.1 Detailed Description

Base class for additional data that is special to a particular type of algorithm, such as the CG penalty function, or using iterative linear solvers.

The regular IpoptData object should be given a derivation of this base class when it is created.

Definition at line 28 of file IpIpoptData.hpp.

3.67.2 Member Function Documentation

3.67.2.1 virtual bool Ipopt::IpoptAdditionalData::Initialize (const Journalist & *jnlst*, const OptionsList & *options*, const std::string & *prefix*) [pure virtual]

This method is called to initialize the global algorithmic parameters.

The parameters are taken from the OptionsList object.

Implemented in Ipopt::CGPenaltyData, and Ipopt::InexactData.

3.67.2.2 virtual bool Ipopt::IpoptAdditionalData::InitializeDataStructures ( ) [pure virtual]

Initialize Data Structures at the beginning.

Implemented in Ipopt::CGPenaltyData, and Ipopt::InexactData.

3.67.2.3 virtual void Ipopt::IpoptAdditionalData::AcceptTrialPoint ( ) [pure virtual]

Do whatever is necessary to accept a trial point as current iterate.

This is also used to finish an iteration, i.e., to release memory, and to reset any flags for a new iteration.

Implemented in Ipopt::CGPenaltyData, and Ipopt::InexactData.

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

- IpIpoptData.hpp

3.68 Ipopt::IpoptAlgorithm Class Reference

The main ipopt algorithm class.

```
#include <IpIpoptAlg.hpp>
```

Inheritance diagram for Ipopt::IpoptAlgorithm:

Public Member Functions

- virtual bool InitializeImpl (const OptionsList &options, const std::string &prefix)

  overloaded from AlgorithmStrategyObject
- SolverReturn Optimize (bool isResto=false)

  Main solve method.

Constructors/Destructors

- IpoptAlgorithm (const SmartPtr< SearchDirectionCalculator > &search_dir_calculator, const SmartPtr< LineSearch > &line_search, const SmartPtr< MuUpdate > &mu_update, const SmartPtr< ConvergenceCheck > &conv_check, const SmartPtr< IterateInitializer > &iterate_initializer, const SmartPtr< IterationOutput > &iter_output, const SmartPtr< HessianUpdater > &hessian_updater, const SmartPtr< EqMultiplierCalculator > &eq_multiplier_calculator=NULL)

  Constructor.
- virtual ∼IpoptAlgorithm ()

  Default destructor.

Access to internal strategy objects

- SmartPtr< SearchDirectionCalculator > **SearchDirCalc** ()

Static Public Member Functions

- static void RegisterOptions (SmartPtr< RegisteredOptions > roptions)

  Methods for IpoptType.

Additional Inherited Members

3.68.1 Detailed Description

The main ipopt algorithm class.

Main Ipopt algorithm class, contains the main optimize method, handles the execution of the optimization. The constructor initializes the data structures through the nlp, and the Optimize method then assumes that everything is initialized and ready to go. After an optimization is complete, the user can access the solution through the passed in ip_data structure. Multiple calls to the Optimize method are allowed as long as the structure of the problem remains the same (i.e. starting point or nlp parameter changes only).

Definition at line 45 of file IpIpoptAlg.hpp.

3.68.2 Constructor & Destructor Documentation

3.68.2.1 Ipopt::IpoptAlgorithm::IpoptAlgorithm (const SmartPtr< SearchDirectionCalculator > & *search_dir_calculator*, const SmartPtr< LineSearch > & *line_search*, const SmartPtr< MuUpdate > & *mu_update*, const SmartPtr< ConvergenceCheck > & *conv_check*, const SmartPtr< IterateInitializer > & *iterate_initializer*, const SmartPtr< IterationOutput > & *iter_output*, const SmartPtr< HessianUpdater > & *hessian_updater*, const SmartPtr< EqMultiplierCalculator > & *eq_multiplier_calculator* = `NULL`)

Constructor.

(The IpoptAlgorithm uses smart pointers for these passed-in pieces to make sure that a user of IpoptAlgoroithm cannot pass in an object created on the stack!)

3.68.3 Member Function Documentation

3.68.3.1 SolverReturn Ipopt::IpoptAlgorithm::Optimize (bool *isResto* = `false`)

Main solve method.

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

- IpIpoptAlg.hpp

3.69 Ipopt::IpoptApplication Class Reference

This is the main application class for making calls to Ipopt.

`#include <IpIpoptApplication.hpp>`

Inheritance diagram for Ipopt::IpoptApplication:

Public Member Functions

- IpoptApplication (SmartPtr< RegisteredOptions > reg_options, SmartPtr< OptionsList > options, SmartPtr< Journalist > jnlst)

  Another constructor that assumes that the code in the (default) constructor has already been executed.
- virtual SmartPtr< IpoptApplication > clone ()

  Method for creating a new IpoptApplication that uses the same journalist and registered options, and a copy of the options list.
- virtual ApplicationReturnStatus Initialize (std::istream &is)

  Initialization method.
- virtual ApplicationReturnStatus Initialize (std::string params_file)

  Initialization method.
- virtual ApplicationReturnStatus Initialize ()

  Initialize method.
- virtual bool OpenOutputFile (std::string file_name, EJournalLevel print_level)

  Method for opening an output file with given print_level.
- void PrintCopyrightMessage ()

  Method for printing Ipopt copyright message now instead of just before the optimization.
- void RethrowNonIpoptException (bool dorethrow)

Method to set whether non-ipopt non-bad_alloc exceptions are rethrown by Ipopt.

Solve methods

- virtual ApplicationReturnStatus OptimizeTNLP (const SmartPtr< TNLP > &tnlp)
  Solve a problem that inherits from TNLP.
- virtual ApplicationReturnStatus OptimizeNLP (const SmartPtr< NLP > &nlp)
  Solve a problem that inherits from NLP.
- virtual ApplicationReturnStatus OptimizeNLP (const SmartPtr< NLP > &nlp, SmartPtr< AlgorithmBuilder > &alg_builder)
  Solve a problem that inherits from NLP.
- virtual ApplicationReturnStatus ReOptimizeTNLP (const SmartPtr< TNLP > &tnlp)
  Solve a problem (that inherits from TNLP) for a repeated time.
- virtual ApplicationReturnStatus ReOptimizeNLP (const SmartPtr< NLP > &nlp)
  Solve a problem (that inherits from NLP) for a repeated time.

Accessor methods

- virtual SmartPtr< Journalist > Jnlst ()
  Get the Journalist for printing output.
- virtual SmartPtr< RegisteredOptions > RegOptions ()
  Get a pointer to RegisteredOptions object to add new options.
- virtual SmartPtr< OptionsList > Options ()
  Get the options list for setting options.
- virtual SmartPtr< const OptionsList > Options () const
  Get the options list for setting options (const version)
- virtual SmartPtr< SolveStatistics > Statistics ()
  Get the object with the statistics about the most recent optimization run.
- virtual SmartPtr< IpoptNLP > IpoptNLPObject ()
  Get the IpoptNLP Object.
- SmartPtr< IpoptData > IpoptDataObject ()
  Get the IpoptData Object.
- virtual SmartPtr< IpoptCalculatedQuantities > IpoptCQObject ()
  Get the IpoptCQ Object.
- SmartPtr< IpoptAlgorithm > AlgorithmObject ()
  Get the Algorithm Object.

Static Public Member Functions

- static void RegisterAllIpoptOptions (const SmartPtr< RegisteredOptions > &roptions)
  Method to registering all Ipopt options.

Methods for IpoptTypeInfo

- static void **RegisterOptions** (SmartPtr< RegisteredOptions > roptions)

3.69.1 Detailed Description

This is the main application class for making calls to Ipopt.

Definition at line 47 of file IpIpoptApplication.hpp.

3.69.2 Member Function Documentation

3.69.2.1 virtual SmartPtr<IpoptApplication> Ipopt::IpoptApplication::clone ( ) [virtual]

Method for creating a new IpoptApplication that uses the same journalist and registered options, and a copy of the options list.

3.69.2.2 virtual ApplicationReturnStatus Ipopt::IpoptApplication::Initialize (std::istream & *is*) [virtual]

Initialization method.

This method reads options from the input stream and initializes the journalists. It returns something other than Solve_Succeeded if there was a problem in the initialization (such as an invalid option). You should call one of the initialization methods at some point before the first optimize call.

3.69.2.3 virtual ApplicationReturnStatus Ipopt::IpoptApplication::Initialize (std::string *params_file*) [virtual]

Initialization method.

This method reads options from the params file and initializes the journalists. It returns something other than Solve_Succeeded if there was a problem in the initialization (such as an invalid option). You should call one of the initialization methods at some point before the first optimize call. Note: You can skip the processing of a params file by setting params_file to "".

3.69.2.4 virtual ApplicationReturnStatus Ipopt::IpoptApplication::Initialize ( ) [virtual]

Initialize method.

This method reads the options file specified by the option_file_name option and initializes the journalists. You should call this method at some point before the first optimize call. It returns something other than Solve_Succeeded if there was a problem in the initialization (such as an invalid option).

3.69.2.5 virtual ApplicationReturnStatus Ipopt::IpoptApplication::ReOptimizeTNLP (const SmartPtr< TNLP > & *tnlp*) [virtual]

Solve a problem (that inherits from TNLP) for a repeated time.

The OptimizeTNLP method must have been called before. The TNLP must be the same object, and the structure (number of variables and constraints and position of nonzeros in Jacobian and Hessian must be the same).

3.69.2.6 virtual ApplicationReturnStatus Ipopt::IpoptApplication::ReOptimizeNLP (const SmartPtr< NLP > & *nlp*) [virtual]

Solve a problem (that inherits from NLP) for a repeated time.

The OptimizeNLP method must have been called before. The NLP must be the same object, and the structure (number of variables and constraints and position of nonzeros in Jacobian and Hessian must be the same).

3.69.2.7 virtual bool Ipopt::IpoptApplication::OpenOutputFile (std::string *file_name*, EJournalLevel *print_level*) [virtual]

Method for opening an output file with given print_level.

Returns false if there was a problem.

3.69.2.8 virtual SmartPtr<SolveStatistics> Ipopt::IpoptApplication::Statistics ( ) [virtual]

Get the object with the statistics about the most recent optimization run.

3.69.2.9 void Ipopt::IpoptApplication::PrintCopyrightMessage ( )

Method for printing Ipopt copyright message now instead of just before the optimization.

If you want to have the copy right message printed earlier than by default, call this method at the convenient time.

3.69.2.10 void Ipopt::IpoptApplication::RethrowNonIpoptException (bool *dorethrow*) [inline]

Method to set whether non-ipopt non-bad_alloc exceptions are rethrown by Ipopt.

By default, non-Ipopt and non-std::bad_alloc exceptions are caught by Ipopts initialization and optimization methods and the status NonIpopt_Exception_Thrown is returned. This function allows to enable rethrowing of such exceptions.

Definition at line 180 of file IpIpoptApplication.hpp.

3.69.2.11 static void Ipopt::IpoptApplication::RegisterAllIpoptOptions (const SmartPtr< RegisteredOptions > & *roptions*) [static]

Method to registering all Ipopt options.

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

- IpIpoptApplication.hpp

3.70 Ipopt::IpoptCalculatedQuantities Class Reference

Class for all IPOPT specific calculated quantities.

`#include <IpIpoptCalculatedQuantities.hpp>`

Inheritance diagram for Ipopt::IpoptCalculatedQuantities:

Public Member Functions

- void SetAddCq (SmartPtr< IpoptAdditionalCq > add_cq)

  Method for setting pointer for additional calculated quantities.
- bool HaveAddCq ()

  Method detecting if additional object for calculated quantities has already been set.
- bool Initialize (const Journalist &jnlst, const OptionsList &options, const std::string &prefix)

  This method must be called to initialize the global algorithmic parameters.
- Number curr_avrg_compl ()

  average of current values of the complementarities
- Number trial_avrg_compl ()

  average of trial values of the complementarities
- Number curr_gradBarrTDelta ()

  inner_product of current barrier obj.

- Number CalcNormOfType (ENormType NormType, std::vector< SmartPtr< const Vector > > vecs)

  Compute the norm of a specific type of a set of vectors (uncached)
- Number CalcNormOfType (ENormType NormType, const Vector &vec1, const Vector &vec2)

  Compute the norm of a specific type of two vectors (uncached)
- ENormType constr_viol_normtype () const

  Norm type used for calculating constraint violation.
- bool IsSquareProblem () const

  Method returning true if this is a square problem.
- SmartPtr< IpoptNLP > & GetIpoptNLP ()

  Method returning the IpoptNLP object.

Constructors/Destructors

- IpoptCalculatedQuantities (const SmartPtr< IpoptNLP > &ip_nlp, const SmartPtr< IpoptData > &ip_data)

  Constructor.
- virtual ∼IpoptCalculatedQuantities ()

  Default destructor.

Slacks

- SmartPtr< const Vector > curr_slack_x_L ()

  Slacks for x_L (at current iterate)
- SmartPtr< const Vector > curr_slack_x_U ()

  Slacks for x_U (at current iterate)
- SmartPtr< const Vector > curr_slack_s_L ()

  Slacks for s_L (at current iterate)
- SmartPtr< const Vector > curr_slack_s_U ()

  Slacks for s_U (at current iterate)
- SmartPtr< const Vector > trial_slack_x_L ()

  Slacks for x_L (at trial point)
- SmartPtr< const Vector > trial_slack_x_U ()

  Slacks for x_U (at trial point)
- SmartPtr< const Vector > trial_slack_s_L ()

  Slacks for s_L (at trial point)
- SmartPtr< const Vector > trial_slack_s_U ()

  Slacks for s_U (at trial point)
- Index AdjustedTrialSlacks ()

  Indicating whether or not we "fudged" the slacks.
- void ResetAdjustedTrialSlacks ()

  Reset the flags for "fudged" slacks.

Objective function

- virtual Number curr_f ()

  Value of objective function (at current point)
- virtual Number unscaled_curr_f ()

  Unscaled value of the objective function (at the current point)
- virtual Number trial_f ()

  Value of objective function (at trial point)
- virtual Number unscaled_trial_f ()

  Unscaled value of the objective function (at the trial point)

- SmartPtr< const Vector > curr_grad_f ()
  Gradient of objective function (at current point)
- SmartPtr< const Vector > trial_grad_f ()
  Gradient of objective function (at trial point)

Barrier Objective Function

- virtual Number curr_barrier_obj ()
  Barrier Objective Function Value (at current iterate with current mu)
- virtual Number trial_barrier_obj ()
  Barrier Objective Function Value (at trial point with current mu)
- SmartPtr< const Vector > curr_grad_barrier_obj_x ()
  Gradient of barrier objective function with respect to x (at current point with current mu)
- SmartPtr< const Vector > curr_grad_barrier_obj_s ()
  Gradient of barrier objective function with respect to s (at current point with current mu)
- SmartPtr< const Vector > grad_kappa_times_damping_x ()
  Gradient of the damping term with respect to x (times kappa_d)
- SmartPtr< const Vector > grad_kappa_times_damping_s ()
  Gradient of the damping term with respect to s (times kappa_d)

Constraints

- SmartPtr< const Vector > curr_c ()
  c(x) (at current point)
- SmartPtr< const Vector > unscaled_curr_c ()
  unscaled c(x) (at current point)
- SmartPtr< const Vector > trial_c ()
  c(x) (at trial point)
- SmartPtr< const Vector > unscaled_trial_c ()
  unscaled c(x) (at trial point)
- SmartPtr< const Vector > curr_d ()
  d(x) (at current point)
- SmartPtr< const Vector > unscaled_curr_d ()
  unscaled d(x) (at current point)
- SmartPtr< const Vector > trial_d ()
  d(x) (at trial point)
- SmartPtr< const Vector > curr_d_minus_s ()
  d(x) - s (at current point)
- SmartPtr< const Vector > trial_d_minus_s ()
  d(x) - s (at trial point)
- SmartPtr< const Matrix > curr_jac_c ()
  Jacobian of c (at current point)
- SmartPtr< const Matrix > trial_jac_c ()
  Jacobian of c (at trial point)
- SmartPtr< const Matrix > curr_jac_d ()
  Jacobian of d (at current point)
- SmartPtr< const Matrix > trial_jac_d ()
  Jacobian of d (at trial point)
- SmartPtr< const Vector > curr_jac_cT_times_vec (const Vector &vec)
  Product of Jacobian (evaluated at current point) of C transpose with general vector.
- SmartPtr< const Vector > trial_jac_cT_times_vec (const Vector &vec)
  Product of Jacobian (evaluated at trial point) of C transpose with general vector.

- SmartPtr< const Vector > curr_jac_dT_times_vec (const Vector &vec)

  Product of Jacobian (evaluated at current point) of D transpose with general vector.
- SmartPtr< const Vector > trial_jac_dT_times_vec (const Vector &vec)

  Product of Jacobian (evaluated at trial point) of D transpose with general vector.
- SmartPtr< const Vector > curr_jac_cT_times_curr_y_c ()

  Product of Jacobian (evaluated at current point) of C transpose with current y_c.
- SmartPtr< const Vector > trial_jac_cT_times_trial_y_c ()

  Product of Jacobian (evaluated at trial point) of C transpose with trial y_c.
- SmartPtr< const Vector > curr_jac_dT_times_curr_y_d ()

  Product of Jacobian (evaluated at current point) of D transpose with current y_d.
- SmartPtr< const Vector > trial_jac_dT_times_trial_y_d ()

  Product of Jacobian (evaluated at trial point) of D transpose with trial y_d.
- SmartPtr< const Vector > curr_jac_c_times_vec (const Vector &vec)

  Product of Jacobian (evaluated at current point) of C with general vector.
- SmartPtr< const Vector > curr_jac_d_times_vec (const Vector &vec)

  Product of Jacobian (evaluated at current point) of D with general vector.
- virtual Number curr_constraint_violation ()

  Constraint Violation (at current iterate).
- virtual Number trial_constraint_violation ()

  Constraint Violation (at trial point).
- virtual Number curr_nlp_constraint_violation (ENormType NormType)

  Real constraint violation in a given norm (at current iterate).
- virtual Number unscaled_curr_nlp_constraint_violation (ENormType NormType)

  Unscaled real constraint violation in a given norm (at current iterate).
- virtual Number unscaled_trial_nlp_constraint_violation (ENormType NormType)

  Unscaled real constraint violation in a given norm (at trial iterate).

Hessian matrices

- SmartPtr< const SymMatrix > curr_exact_hessian ()

  exact Hessian at current iterate (uncached)

primal-dual error and its components

- SmartPtr< const Vector > curr_grad_lag_x ()

  x-part of gradient of Lagrangian function (at current point)
- SmartPtr< const Vector > trial_grad_lag_x ()

  x-part of gradient of Lagrangian function (at trial point)
- SmartPtr< const Vector > curr_grad_lag_s ()

  s-part of gradient of Lagrangian function (at current point)
- SmartPtr< const Vector > trial_grad_lag_s ()

  s-part of gradient of Lagrangian function (at trial point)
- SmartPtr< const Vector > curr_grad_lag_with_damping_x ()

  x-part of gradient of Lagrangian function (at current point) including linear damping term
- SmartPtr< const Vector > curr_grad_lag_with_damping_s ()

  s-part of gradient of Lagrangian function (at current point) including linear damping term
- SmartPtr< const Vector > curr_compl_x_L ()

  Complementarity for x_L (for current iterate)
- SmartPtr< const Vector > curr_compl_x_U ()

  Complementarity for x_U (for current iterate)
- SmartPtr< const Vector > curr_compl_s_L ()

  Complementarity for s_L (for current iterate)

- SmartPtr< const Vector > curr_compl_s_U ()

  Complementarity for s_U (for current iterate)
- SmartPtr< const Vector > trial_compl_x_L ()

  Complementarity for x_L (for trial iterate)
- SmartPtr< const Vector > trial_compl_x_U ()

  Complementarity for x_U (for trial iterate)
- SmartPtr< const Vector > trial_compl_s_L ()

  Complementarity for s_L (for trial iterate)
- SmartPtr< const Vector > trial_compl_s_U ()

  Complementarity for s_U (for trial iterate)
- SmartPtr< const Vector > curr_relaxed_compl_x_L ()

  Relaxed complementarity for x_L (for current iterate and current mu)
- SmartPtr< const Vector > curr_relaxed_compl_x_U ()

  Relaxed complementarity for x_U (for current iterate and current mu)
- SmartPtr< const Vector > curr_relaxed_compl_s_L ()

  Relaxed complementarity for s_L (for current iterate and current mu)
- SmartPtr< const Vector > curr_relaxed_compl_s_U ()

  Relaxed complementarity for s_U (for current iterate and current mu)
- virtual Number curr_primal_infeasibility (ENormType NormType)

  Primal infeasibility in a given norm (at current iterate).
- virtual Number trial_primal_infeasibility (ENormType NormType)

  Primal infeasibility in a given norm (at trial point)
- virtual Number curr_dual_infeasibility (ENormType NormType)

  Dual infeasibility in a given norm (at current iterate)
- virtual Number trial_dual_infeasibility (ENormType NormType)

  Dual infeasibility in a given norm (at trial iterate)
- virtual Number unscaled_curr_dual_infeasibility (ENormType NormType)

  Unscaled dual infeasibility in a given norm (at current iterate)
- virtual Number curr_complementarity (Number mu, ENormType NormType)

  Complementarity (for all complementarity conditions together) in a given norm (at current iterate)
- virtual Number trial_complementarity (Number mu, ENormType NormType)

  Complementarity (for all complementarity conditions together) in a given norm (at trial iterate)
- virtual Number unscaled_curr_complementarity (Number mu, ENormType NormType)

  Complementarity (for all complementarity conditions together) in a given norm (at current iterate) without NLP scaling.
- Number CalcCentralityMeasure (const Vector &compl_x_L, const Vector &compl_x_U, const Vector &compl_s_L, const Vector &compl_s_U)

  Centrality measure (in spirit of the -infinity-neighborhood.
- virtual Number curr_centrality_measure ()

  Centrality measure at current point.
- virtual Number curr_nlp_error ()

  Total optimality error for the original NLP at the current iterate, using scaling factors based on multipliers.
- virtual Number unscaled_curr_nlp_error ()

  Total optimality error for the original NLP at the current iterate, but using no scaling based on multipliers, and no scaling for the NLP.
- virtual Number curr_barrier_error ()

  Total optimality error for the barrier problem at the current iterate, using scaling factors based on multipliers.
- virtual Number curr_primal_dual_system_error (Number mu)

  Norm of the primal-dual system for a given mu (at current iterate).
- virtual Number trial_primal_dual_system_error (Number mu)

  Norm of the primal-dual system for a given mu (at trial iterate).

Computing fraction-to-the-boundary step sizes

- Number primal_frac_to_the_bound (Number tau, const Vector &delta_x, const Vector &delta_s)

  Fraction to the boundary from (current) primal variables x and s for a given step.
- Number curr_primal_frac_to_the_bound (Number tau)

  Fraction to the boundary from (current) primal variables x and s for internal (current) step.
- Number dual_frac_to_the_bound (Number tau, const Vector &delta_z_L, const Vector &delta_z_U, const Vector &delta_v_L, const Vector &delta_v_U)

  Fraction to the boundary from (current) dual variables z and v for a given step.
- Number uncached_dual_frac_to_the_bound (Number tau, const Vector &delta_z_L, const Vector &delta_z_U, const Vector &delta_v_L, const Vector &delta_v_U)

  Fraction to the boundary from (current) dual variables z and v for a given step, without caching.
- Number curr_dual_frac_to_the_bound (Number tau)

  Fraction to the boundary from (current) dual variables z and v for internal (current) step.
- Number uncached_slack_frac_to_the_bound (Number tau, const Vector &delta_x_L, const Vector &delta_x_U, const Vector &delta_s_L, const Vector &delta_s_U)

  Fraction to the boundary from (current) slacks for a given step in the slacks.

Sigma matrices

- SmartPtr< const Vector > **curr_sigma_x** ()
- SmartPtr< const Vector > **curr_sigma_s** ()

Static Public Member Functions

- static void RegisterOptions (SmartPtr< RegisteredOptions > roptions)

  Methods for IpoptType.

3.70.1 Detailed Description

Class for all IPOPT specific calculated quantities.

Definition at line 81 of file IpIpoptCalculatedQuantities.hpp.

3.70.2 Member Function Documentation

3.70.2.1 void Ipopt::IpoptCalculatedQuantities::SetAddCq (SmartPtr< IpoptAdditionalCq > *add_cq*) [inline]

Method for setting pointer for additional calculated quantities.

This needs to be called before Initialized.

Definition at line 96 of file IpIpoptCalculatedQuantities.hpp.

3.70.2.2 bool Ipopt::IpoptCalculatedQuantities::Initialize (const Journalist & *jnlst*, const OptionsList & *options*, const std::string & *prefix*)

This method must be called to initialize the global algorithmic parameters.

The parameters are taken from the OptionsList object.

3.70.2.3 virtual Number Ipopt::IpoptCalculatedQuantities::curr_constraint_violation ( ) [virtual]

Constraint Violation (at current iterate).

This value should be used in the line search, and not curr_primal_infeasibility(). What type of norm is used depends on constr_viol_normtype

3.70.2.4 virtual Number Ipopt::IpoptCalculatedQuantities::trial_constraint_violation ( ) [virtual]

Constraint Violation (at trial point).

This value should be used in the line search, and not curr_primal_infeasibility(). What type of norm is used depends on constr_viol_normtype

3.70.2.5 virtual Number Ipopt::IpoptCalculatedQuantities::curr_nlp_constraint_violation (ENormType *NormType*) [virtual]

Real constraint violation in a given norm (at current iterate).

This considers the inequality constraints without slacks.

3.70.2.6 virtual Number Ipopt::IpoptCalculatedQuantities::unscaled_curr_nlp_constraint_violation (ENormType *NormType*) [virtual]

Unscaled real constraint violation in a given norm (at current iterate).

This considers the inequality constraints without slacks.

3.70.2.7 virtual Number Ipopt::IpoptCalculatedQuantities::unscaled_trial_nlp_constraint_violation (ENormType *NormType*) [virtual]

Unscaled real constraint violation in a given norm (at trial iterate).

This considers the inequality constraints without slacks.

3.70.2.8 virtual Number Ipopt::IpoptCalculatedQuantities::curr_primal_infeasibility (ENormType *NormType*) [virtual]

Primal infeasibility in a given norm (at current iterate).

3.70.2.9 virtual Number Ipopt::IpoptCalculatedQuantities::unscaled_curr_complementarity (Number *mu*, ENormType *NormType*) [virtual]

Complementarity (for all complementarity conditions together) in a given norm (at current iterate) without NLP scaling.

3.70.2.10 Number Ipopt::IpoptCalculatedQuantities::CalcCentralityMeasure (const Vector & *compl_x_L*, const Vector & *compl_x_U*, const Vector & *compl_s_L*, const Vector & *compl_s_U*)

Centrality measure (in spirit of the -infinity-neighborhood.

3.70.2.11 virtual Number Ipopt::IpoptCalculatedQuantities::curr_nlp_error ( ) [virtual]

Total optimality error for the original NLP at the current iterate, using scaling factors based on multipliers.

Note that here the constraint violation is measured without slacks (nlp_constraint_violation)

3.70.2.12 virtual Number Ipopt::IpoptCalculatedQuantities::unscaled_curr_nlp_error ( ) [virtual]

Total optimality error for the original NLP at the current iterate, but using no scaling based on multipliers, and no scaling for the NLP.

Note that here the constraint violation is measured without slacks (nlp_constraint_violation)

3.70.2.13 virtual Number Ipopt::IpoptCalculatedQuantities::curr_barrier_error ( ) [virtual]

Total optimality error for the barrier problem at the current iterate, using scaling factors based on multipliers.

3.70.2.14 virtual Number Ipopt::IpoptCalculatedQuantities::curr_primal_dual_system_error (Number *mu*) [virtual]

Norm of the primal-dual system for a given mu (at current iterate).

The norm is defined as the sum of the 1-norms of dual infeasibiliy, primal infeasibility, and complementarity, all divided by the number of elements of the vectors of which the norm is taken.

3.70.2.15 virtual Number Ipopt::IpoptCalculatedQuantities::trial_primal_dual_system_error (Number *mu*) [virtual]

Norm of the primal-dual system for a given mu (at trial iterate).

The norm is defined as the sum of the 1-norms of dual infeasibiliy, primal infeasibility, and complementarity, all divided by the number of elements of the vectors of which the norm is taken.

3.70.2.16 Number Ipopt::IpoptCalculatedQuantities::uncached_slack_frac_to_the_bound (Number *tau*, const Vector & *delta_x_L*, const Vector & *delta_x_U*, const Vector & *delta_s_L*, const Vector & *delta_s_U*)

Fraction to the boundary from (current) slacks for a given step in the slacks.

Usually, one will use the primal_frac_to_the_bound method to compute the primal fraction to the boundary step size, but if it is cheaper to provide the steps in the slacks directly (e.g. when the primal step sizes are only temporary), the this method is more efficient. This method does not cache computations.

3.70.2.17 Number Ipopt::IpoptCalculatedQuantities::curr_gradBarrTDelta ( )

inner_product of current barrier obj.

fn. gradient with current search direction

3.70.2.18 SmartPtr<IpoptNLP>& Ipopt::IpoptCalculatedQuantities::GetIpoptNLP ( ) [inline]

Method returning the IpoptNLP object.

This should only be used with care!

Definition at line 447 of file IpIpoptCalculatedQuantities.hpp.

3.70.2.19 static void Ipopt::IpoptCalculatedQuantities::RegisterOptions (SmartPtr< RegisteredOptions > *roptions*) [static]

Methods for IpoptType.

Called by IpoptType to register the options

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

- IpIpoptCalculatedQuantities.hpp

3.71 Ipopt::IpoptData Class Reference

Class to organize all the data required by the algorithm.

`#include <IpIpoptData.hpp>`

Inheritance diagram for Ipopt::IpoptData:

Public Member Functions

- bool InitializeDataStructures (IpoptNLP &ip_nlp, bool want_x, bool want_y_c, bool want_y_d, bool want_z_L, bool want_z_U)

  Initialize Data Structures.
- bool Initialize (const Journalist &jnlst, const OptionsList &options, const std::string &prefix)

  This method must be called to initialize the global algorithmic parameters.
- Number cpu_time_start () const

  Cpu time counter at the beginning of the optimization.
- TimingStatistics & TimingStats ()

  Return Timing Statistics Object.
- bool HaveAddData ()

  Check if additional data has been set.
- IpoptAdditionalData & AdditionalData ()

  Get access to additional data object.
- void SetAddData (SmartPtr< IpoptAdditionalData > add_data)

  Set a new pointer for additional Ipopt data.
- void setPDPert (Number pd_pert_x, Number pd_pert_s, Number pd_pert_c, Number pd_pert_d)

  Set the perturbation of the primal-dual system.
- void getPDPert (Number &pd_pert_x, Number &pd_pert_s, Number &pd_pert_c, Number &pd_pert_d)

  Get the current perturbation of the primal-dual system.

Constructors/Destructors

- IpoptData (SmartPtr< IpoptAdditionalData > add_data=NULL, Number cpu_time_start=-1.)

  Constructor.
- virtual ~IpoptData ()

  Default destructor.

Get Methods for Iterates

- SmartPtr< const IteratesVector > curr () const

Current point.
- SmartPtr< const IteratesVector > trial () const

  Get the current point in a copied container that is non-const.
- void set_trial (SmartPtr< IteratesVector > &trial)

  Get Trial point in a copied container that is non-const.
- void SetTrialPrimalVariablesFromStep (Number alpha, const Vector &delta_x, const Vector &delta_s)

  Set the values of the primal trial variables (x and s) from provided Step with step length alpha.
- void SetTrialEqMultipliersFromStep (Number alpha, const Vector &delta_y_c, const Vector &delta_y_d)

  Set the values of the trial values for the equality constraint multipliers (y_c and y_d) from provided step with step length alpha.
- void SetTrialBoundMultipliersFromStep (Number alpha, const Vector &delta_z_L, const Vector &delta_z_U, const Vector &delta_v_L, const Vector &delta_v_U)

  Set the value of the trial values for the bound multipliers (z_L, z_U, v_L, v_U) from provided step with step length alpha.
- SmartPtr< const IteratesVector > delta () const

  ToDo: I may need to add versions of set_trial like the following, but I am not sure.
- void set_delta (SmartPtr< IteratesVector > &delta)

  Set the current delta - like the trial point, this method copies the pointer for efficiency (no copy and to keep cache tags the same) so after you call set, you cannot modify the data.
- void set_delta (SmartPtr< const IteratesVector > &delta)

  Set the current delta - like the trial point, this method copies the pointer for efficiency (no copy and to keep cache tags the same) so after you call set, you cannot modify the data.
- SmartPtr< const IteratesVector > delta_aff () const

  Affine Delta.
- void set_delta_aff (SmartPtr< IteratesVector > &delta_aff)

  Set the affine delta - like the trial point, this method copies the pointer for efficiency (no copy and to keep cache tags the same) so after you call set, you cannot modify the data.
- SmartPtr< const SymMatrix > W ()

  Hessian or Hessian approximation (do not hold on to it, it might be changed)
- void Set_W (SmartPtr< const SymMatrix > W)

  Set Hessian approximation.

("Main") Primal-dual search direction. Those fields are

used to store the search directions computed from solving the primal-dual system, and can be used in the line search.

They are overwritten in every iteration, so do not hold on to the pointers (make copies instead)

- bool HaveDeltas () const

  Returns true, if the primal-dual step have been already computed for the current iteration.
- void SetHaveDeltas (bool have_deltas)

  Method for setting the HaveDeltas flag.

Affine-scaling step. Those fields can be used to store

the affine scaling step.

For example, if the method for computing the current barrier parameter computes the affine scaling steps, then the corrector step in the line search does not have to recompute those solutions of the linear system.

- bool HaveAffineDeltas () const

  Returns true, if the affine-scaling step have been already computed for the current iteration.
- void SetHaveAffineDeltas (bool have_affine_deltas)

  Method for setting the HaveDeltas flag.

Public Methods for updating iterates

- void CopyTrialToCurrent ()

  Copy the trial values to the current values.
- void AcceptTrialPoint ()

  Set the current iterate values from the trial values.

General algorithmic data

- Index **iter_count** () const
- void **Set_iter_count** (Index iter_count)
- Number **curr_mu** () const
- void **Set_mu** (Number mu)
- bool **MuInitialized** () const
- Number **curr_tau** () const
- void **Set_tau** (Number tau)
- bool **TauInitialized** () const
- void **SetFreeMuMode** (bool free_mu_mode)
- bool **FreeMuMode** () const
- void Set_tiny_step_flag (bool flag)

  Setting the flag that indicates if a tiny step (below machine precision) has been detected.
- bool **tiny_step_flag** ()

- Number tol () const

  Overall convergence tolerance.
- void Set_tol (Number tol)

  Set a new value for the tolerance.

Information gathered for iteration output

- Number **info_regu_x** () const
- void **Set_info_regu_x** (Number regu_x)
- Number **info_alpha_primal** () const
- void **Set_info_alpha_primal** (Number alpha_primal)
- char **info_alpha_primal_char** () const
- void **Set_info_alpha_primal_char** (char info_alpha_primal_char)
- Number **info_alpha_dual** () const
- void **Set_info_alpha_dual** (Number alpha_dual)
- Index **info_ls_count** () const
- void **Set_info_ls_count** (Index ls_count)
- bool **info_skip_output** () const
- void **Append_info_string** (const std::string &add_str)
- const std::string & **info_string** () const
- void Set_info_skip_output (bool info_skip_output)

  Set this to true, if the next time when output is written, the summary line should not be printed.
- Number info_last_output ()

  gives time when the last summary output line was printed
- void Set_info_last_output (Number info_last_output)

  sets time when the last summary output line was printed
- int info_iters_since_header ()

  gives number of iteration summaries actually printed since last summary header was printed
- void Inc_info_iters_since_header ()

  increases number of iteration summaries actually printed since last summary header was printed
- void Set_info_iters_since_header (int info_iters_since_header)

  sets number of iteration summaries actually printed since last summary header was printed
- void ResetInfo ()

  Reset all info fields.

Static Public Member Functions

- static void RegisterOptions (const SmartPtr< RegisteredOptions > &roptions)

  Methods for IpoptType.

3.71.1 Detailed Description

Class to organize all the data required by the algorithm.

Internally, once this Data object has been initialized, all internal curr_ vectors must always be set (so that prototyes are available). The current values can only be set from the trial values. The trial values can be set by copying from a vector or by adding some fraction of a step to the current values. This object also stores steps, which allows to easily communicate the step from the step computation object to the line search object.

Definition at line 83 of file IpIpoptData.hpp.

3.71.2 Member Function Documentation

3.71.2.1 bool Ipopt::IpoptData::Initialize (const **Journalist** & *jnlst,* const **OptionsList** & *options,* const std::string & *prefix*)

This method must be called to initialize the global algorithmic parameters.

The parameters are taken from the OptionsList object.

3.71.2.2 **SmartPtr**< const **IteratesVector** > Ipopt::IpoptData::trial ( ) const [inline]

Get the current point in a copied container that is non-const.

The entries in the container cannot be modified, but the container can be modified to point to new entries.Get Trial point

Definition at line 698 of file IpIpoptData.hpp.

3.71.2.3 void Ipopt::IpoptData::set_trial (**SmartPtr**< **IteratesVector** > & *trial*) [inline]

Get Trial point in a copied container that is non-const.

The entries in the container can not be modified, but the container can be modified to point to new entries. Set the trial point - this method copies the pointer for efficiency (no copy and to keep cache tags the same) so after you call set you cannot modify the data again

Definition at line 740 of file IpIpoptData.hpp.

3.71.2.4 **SmartPtr**< const **IteratesVector** > Ipopt::IpoptData::delta ( ) const [inline]

ToDo: I may need to add versions of set_trial like the following, but I am not sure.

get the current delta

Definition at line 706 of file IpIpoptData.hpp.

3.71.2.5 void Ipopt::IpoptData::set_delta (**SmartPtr**< const **IteratesVector** > & *delta*) [inline]

Set the current delta - like the trial point, this method copies the pointer for efficiency (no copy and to keep cache tags the same) so after you call set, you cannot modify the data.

This is the version that is happy with a pointer to const IteratesVector.

Definition at line 780 of file IpIpoptData.hpp.

3.71.2.6 bool Ipopt::IpoptData::HaveDeltas ( ) const [inline]

Returns true, if the primal-dual step have been already computed for the current iteration.

This flag is reset after every call of AcceptTrialPoint(). If the search direction is computed during the computation of the barrier parameter, the method computing the barrier parameter should call SetHaveDeltas(true) to tell the IpoptAlgorithm object that it doesn't need to recompute the primal-dual step.

Definition at line 227 of file IpIpoptData.hpp.

3.71.2.7 void Ipopt::IpoptData::SetHaveDeltas (bool *have_deltas*) [inline]

Method for setting the HaveDeltas flag.

This method should be called if some method computes the primal-dual step (and stores it in the delta_ fields of Ipopt←Data) at an early part of the iteration. If that flag is set to true, the IpoptAlgorithm object will not recompute the step.

Definition at line 237 of file IpIpoptData.hpp.

3.71.2.8 bool Ipopt::IpoptData::HaveAffineDeltas ( ) const [inline]

Returns true, if the affine-scaling step have been already computed for the current iteration.

This flag is reset after every call of AcceptTrialPoint(). If the search direction is computed during the computation of the barrier parameter, the method computing the barrier parameter should call SetHaveDeltas(true) to tell the line search does not have to recompute them in case it wants to do a corrector step.

Definition at line 257 of file IpIpoptData.hpp.

3.71.2.9 void Ipopt::IpoptData::SetHaveAffineDeltas (bool *have_affine_deltas*) [inline]

Method for setting the HaveDeltas flag.

This method should be called if some method computes the primal-dual step (and stores it in the delta_ fields of Ipopt←Data) at an early part of the iteration. If that flag is set to true, the IpoptAlgorithm object will not recompute the step.

Definition at line 267 of file IpIpoptData.hpp.

3.71.2.10 void Ipopt::IpoptData::AcceptTrialPoint ( )

Set the current iterate values from the trial values.

3.71.2.11 Number Ipopt::IpoptData::tol ( ) const [inline]

Overall convergence tolerance.

It is used in the convergence test, but also in some other parts of the algorithm that depend on the specified tolerance, such as the minimum value for the barrier parameter. Obtain the tolerance.

Definition at line 352 of file IpIpoptData.hpp.

3.71.2.12 void Ipopt::IpoptData::Set_tol (Number *tol*) [inline]

Set a new value for the tolerance.

One should be very careful when using this, since changing the predefined tolerance might have unexpected consequences. This method is for example used in the restoration convergence checker to tighten the restoration phase convergence tolerance, if the restoration phase converged to a point that has not a large value for the constraint violation.

Definition at line 364 of file IpIpoptData.hpp.

3.71.2.13 Number Ipopt::IpoptData::cpu_time_start ( ) const [inline]

Cpu time counter at the beginning of the optimization.

This is useful to see how much CPU time has been spent in this optimization run.

Definition at line 373 of file IpIpoptData.hpp.

3.71.2.14 void Ipopt::IpoptData::Set_info_skip_output (bool *info_skip_output*) [inline]

Set this to true, if the next time when output is written, the summary line should not be printed.

Definition at line 434 of file IpIpoptData.hpp.

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

- IpIpoptData.hpp

3.72 Ipopt::IpoptException Class Reference

This is the base class for all exceptions.

`#include <IpException.hpp>`

Public Member Functions

- void ReportException (const Journalist &jnlst, EJournalLevel level=J_ERROR) const
  Method to report the exception to a journalist.

Constructors/Destructors

- IpoptException (std::string msg, std::string file_name, Index line_number, std::string type="IpoptException")
  Constructor.
- IpoptException (const IpoptException &copy)
  Copy Constructor.
- virtual ~IpoptException ()
  Default destructor.

3.72.1 Detailed Description

This is the base class for all exceptions.

The easiest way to use this class is by means of the following macros:

```
DECLARE_STD_EXCEPTION(ExceptionType);
```

This macro defines a new class with the name ExceptionType, inherited from the base class IpoptException. After this, exceptions of this type can be thrown using

```
THROW_EXCEPTION(ExceptionType, Message);
```

where Message is a std::string with a message that gives an indication of what caused the exception. Exceptions can also be thrown using the macro

```
ASSERT_EXCEPTION(Condition, ExceptionType, Message);
```

where Conditions is an expression. If Condition evaluates to false, then the exception of the type ExceptionType is thrown with Message.

When an exception is caught, the method ReportException can be used to write the information about the exception to the Journalist, using the level J_ERROR and the category J_MAIN.

Definition at line 57 of file IpException.hpp.

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

- IpException.hpp

3.73 Ipopt::IpoptNLP Class Reference

This is the abstract base class for classes that map the traditional NLP into something that is more useful by Ipopt.

`#include <IpIpoptNLP.hpp>`

Inheritance diagram for Ipopt::IpoptNLP:

Public Member Functions

- virtual bool Initialize (const Journalist &jnlst, const OptionsList &options, const std::string &prefix)

    Initialization method.
- virtual bool InitializeStructures (SmartPtr< Vector > &x, bool init_x, SmartPtr< Vector > &y_c, bool init_y_c, SmartPtr< Vector > &y_d, bool init_y_d, SmartPtr< Vector > &z_L, bool init_z_L, SmartPtr< Vector > &z_U, bool init_z_U, SmartPtr< Vector > &v_L, SmartPtr< Vector > &v_U)=0

    Initialize (create) structures for the iteration data.
- virtual bool GetWarmStartIterate (IteratesVector &warm_start_iterate)=0

    Method accessing the GetWarmStartIterate of the NLP.
- virtual void GetSpaces (SmartPtr< const VectorSpace > &x_space, SmartPtr< const VectorSpace > &c_space, SmartPtr< const VectorSpace > &d_space, SmartPtr< const VectorSpace > &x_l_space, SmartPtr< const MatrixSpace > &px_l_space, SmartPtr< const VectorSpace > &x_u_space, SmartPtr< const MatrixSpace > &px_u_space, SmartPtr< const VectorSpace > &d_l_space, SmartPtr< const MatrixSpace > &pd_l_space, SmartPtr< const VectorSpace > &d_u_space, SmartPtr< const MatrixSpace > &pd_u_space, SmartPtr< const MatrixSpace > &Jac_c_space, SmartPtr< const MatrixSpace > &Jac_d_space, SmartPtr< const SymMatrix←Space > &Hess_lagrangian_space)=0

    Accessor method for vector/matrix spaces pointers.
- virtual void AdjustVariableBounds (const Vector &new_x_L, const Vector &new_x_U, const Vector &new_d_L, const Vector &new_d_U)=0

Method for adapting the variable bounds.
- SmartPtr< NLPScalingObject > NLP_scaling () const

  Returns the scaling strategy object.

Constructors/Destructors

- **IpoptNLP** (const SmartPtr< NLPScalingObject > nlp_scaling)
- virtual ~IpoptNLP ()

  Default destructor.

Possible Exceptions

- DECLARE_STD_EXCEPTION (Eval_Error)

  thrown if there is any error evaluating values from the nlp

- virtual Number f (const Vector &x)=0

  Accessor methods for model data.
- virtual SmartPtr< const Vector > grad_f (const Vector &x)=0

  Gradient of the objective.
- virtual SmartPtr< const Vector > c (const Vector &x)=0

  Equality constraint residual.
- virtual SmartPtr< const Matrix > jac_c (const Vector &x)=0

  Jacobian Matrix for equality constraints.
- virtual SmartPtr< const Vector > d (const Vector &x)=0

  Inequality constraint residual (reformulated as equalities with slacks.
- virtual SmartPtr< const Matrix > jac_d (const Vector &x)=0

  Jacobian Matrix for inequality constraints.
- virtual SmartPtr< const SymMatrix > h (const Vector &x, Number obj_factor, const Vector &yc, const Vector &yd)=0

  Hessian of the Lagrangian.
- virtual SmartPtr< const Vector > x_L () const =0

  Lower bounds on x.
- virtual SmartPtr< const Matrix > Px_L () const =0

  Permutation matrix (x_L_ -> x)
- virtual SmartPtr< const Vector > x_U () const =0

  Upper bounds on x.
- virtual SmartPtr< const Matrix > Px_U () const =0

  Permutation matrix (x_U_ -> x.
- virtual SmartPtr< const Vector > d_L () const =0

  Lower bounds on d.
- virtual SmartPtr< const Matrix > Pd_L () const =0

  Permutation matrix (d_L_ -> d)
- virtual SmartPtr< const Vector > d_U () const =0

  Upper bounds on d.
- virtual SmartPtr< const Matrix > Pd_U () const =0

  Permutation matrix (d_U_ -> d.
- virtual SmartPtr< const VectorSpace > x_space () const =0

  x_space
- virtual SmartPtr< const SymMatrixSpace > HessianMatrixSpace () const =0

Accessor method to obtain the MatrixSpace for the Hessian matrix (or it's approximation)

Counters for the number of function evaluations.

- virtual Index **f_evals** () const =0
- virtual Index **grad_f_evals** () const =0
- virtual Index **c_evals** () const =0
- virtual Index **jac_c_evals** () const =0
- virtual Index **d_evals** () const =0
- virtual Index **jac_d_evals** () const =0
- virtual Index **h_evals** () const =0

Special method for dealing with the fact that the

restoration phase objective function depends on the barrier parameter

- virtual bool objective_depends_on_mu () const

  Method for telling the IpoptCalculatedQuantities class whether the objective function depends on the barrier function.
- virtual Number f (const Vector &x, Number mu)=0

  Replacement for the default objective function method which knows about the barrier parameter.
- virtual SmartPtr< const Vector > grad_f (const Vector &x, Number mu)=0

  Replacement for the default objective gradient method which knows about the barrier parameter.
- virtual SmartPtr< const SymMatrix > h (const Vector &x, Number obj_factor, const Vector &yc, const Vector &yd, Number mu)=0

  Replacement for the default Lagrangian Hessian method which knows about the barrier parameter.
- virtual SmartPtr< const SymMatrix > uninitialized_h ()=0

  Provides a Hessian matrix from the correct matrix space with uninitialized values.

solution routines

- virtual void **FinalizeSolution** (SolverReturn status, const Vector &x, const Vector &z_L, const Vector &z_U, const Vector &c, const Vector &d, const Vector &y_c, const Vector &y_d, Number obj_value, const IpoptData ∗ip_data, IpoptCalculatedQuantities ∗ip_cq)=0
- virtual bool **IntermediateCallBack** (AlgorithmMode mode, Index iter, Number obj_value, Number inf_pr, Number inf_du, Number mu, Number d_norm, Number regularization_size, Number alpha_du, Number alpha_pr, Index ls_trials, SmartPtr< const IpoptData > ip_data, SmartPtr< IpoptCalculatedQuantities > ip_cq)=0

3.73.1 Detailed Description

This is the abstract base class for classes that map the traditional NLP into something that is more useful by Ipopt.

This class takes care of storing the calculated model results, handles cacheing, and (some day) takes care of addition of slacks.

Definition at line 28 of file IpIpoptNLP.hpp.

3.73.2 Member Function Documentation

3.73.2.1 virtual bool Ipopt::IpoptNLP::Initialize (const **Journalist** & *jnlst,* const **OptionsList** & *options,* const std::string & *prefix*) [inline], [virtual]

Initialization method.

Set the internal options and initialize internal data structures.

Reimplemented in Ipopt::OrigIpoptNLP, and Ipopt::RestoIpoptNLP.

Definition at line 45 of file IpIpoptNLP.hpp.

3.73.2.2 virtual Number Ipopt::IpoptNLP::f (const Vector & *x*) [pure virtual]

Accessor methods for model data.

Objective value

Implemented in Ipopt::RestoIpoptNLP, and Ipopt::OrigIpoptNLP.

3.73.2.3 virtual void Ipopt::IpoptNLP::GetSpaces (SmartPtr< const VectorSpace > & *x_space*, SmartPtr< const VectorSpace > & *c_space*, SmartPtr< const VectorSpace > & *d_space*, SmartPtr< const VectorSpace > & *x_l_space*, SmartPtr< const MatrixSpace > & *px_l_space*, SmartPtr< const VectorSpace > & *x_u_space*, SmartPtr< const MatrixSpace > & *px_u_space*, SmartPtr< const VectorSpace > & *d_l_space*, SmartPtr< const MatrixSpace > & *pd_l_space*, SmartPtr< const VectorSpace > & *d_u_space*, SmartPtr< const MatrixSpace > & *pd_u_space*, SmartPtr< const MatrixSpace > & *Jac_c_space*, SmartPtr< const MatrixSpace > & *Jac_d_space*, SmartPtr< const SymMatrixSpace > & *Hess_lagrangian_space*) [pure virtual]

Accessor method for vector/matrix spaces pointers.

Implemented in Ipopt::RestoIpoptNLP, and Ipopt::OrigIpoptNLP.

3.73.2.4 virtual void Ipopt::IpoptNLP::AdjustVariableBounds (const Vector & *new_x_L*, const Vector & *new_x_U*, const Vector & *new_d_L*, const Vector & *new_d_U*) [pure virtual]

Method for adapting the variable bounds.

This is called if slacks are becoming too small

Implemented in Ipopt::RestoIpoptNLP, and Ipopt::OrigIpoptNLP.

3.73.2.5 virtual bool Ipopt::IpoptNLP::objective_depends_on_mu ( ) const [inline], [virtual]

Method for telling the IpoptCalculatedQuantities class whether the objective function depends on the barrier function.

This is only used for the restoration phase NLP formulation. Probably only RestoIpoptNLP should overwrite this.

Reimplemented in Ipopt::RestoIpoptNLP.

Definition at line 183 of file IpIpoptNLP.hpp.

3.73.2.6 virtual SmartPtr<const SymMatrix> Ipopt::IpoptNLP::uninitialized_h ( ) [pure virtual]

Provides a Hessian matrix from the correct matrix space with uninitialized values.

This can be used in LeastSquareMults to obtain a "zero Hessian".

Implemented in Ipopt::RestoIpoptNLP, and Ipopt::OrigIpoptNLP.

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

- IpIpoptNLP.hpp

3.74 Ipopt::IterateInitializer Class Reference

Base class for all methods for initializing the iterates.

```
#include <IpIterateInitializer.hpp>
```

Inheritance diagram for Ipopt::IterateInitializer:

Public Member Functions

- virtual bool InitializeImpl (const OptionsList &options, const std::string &prefix)=0

  overloaded from AlgorithmStrategyObject
- virtual bool SetInitialIterates ()=0

  Compute the initial iterates and set the into the curr field of the ip_data object.

Constructors/Destructors

- IterateInitializer ()

  Default Constructor.
- virtual ∼IterateInitializer ()

  Default destructor.

Additional Inherited Members

3.74.1 Detailed Description

Base class for all methods for initializing the iterates.

Definition at line 22 of file IpIterateInitializer.hpp.

3.74.2 Member Function Documentation

3.74.2.1 virtual bool Ipopt::IterateInitializer::SetInitialIterates ( ) [pure virtual]

Compute the initial iterates and set the into the curr field of the ip_data object.

Implemented in Ipopt::DefaultIterateInitializer, Ipopt::RestoIterateInitializer, and Ipopt::WarmStartIterateInitializer.

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

- IpIterateInitializer.hpp

3.75 Ipopt::IteratesVector Class Reference

Specialized CompoundVector class specifically for the algorithm iterates.

`#include <IpIteratesVector.hpp>`

Inheritance diagram for Ipopt::IteratesVector:

Public Member Functions

- IteratesVector (const IteratesVectorSpace ∗owner_space, bool create_new)

  Constructors / Destructors.

- SmartPtr< IteratesVector > MakeNewIteratesVector (bool create_new=true) const

Make New methods.
- SmartPtr< IteratesVector > MakeNewIteratesVectorCopy () const

  Use this method to create a new iterates vector with a copy of all the data.
- SmartPtr< IteratesVector > MakeNewContainer () const

  Use this method to create a new iterates vector container.

- SmartPtr< const Vector > x () const

  Iterates Set/Get Methods.
- SmartPtr< Vector > x_NonConst ()

  Get the x iterate (non-const) - this can only be called if the vector was created intenally, or the Set_x_NonConst method was used.
- SmartPtr< Vector > create_new_x ()

  Create a new vector in the x entry.
- SmartPtr< Vector > create_new_x_copy ()

  Create a new vector in the x entry and copy the current values into it.
- void Set_x (const Vector &vec)

  Set the x iterate (const).
- void Set_x_NonConst (Vector &vec)

  Set the x iterate (non-const).
- SmartPtr< const Vector > s () const

  Get the s iterate (const)
- SmartPtr< Vector > s_NonConst ()

  Get the s iterate (non-const) - this can only be called if the vector was created intenally, or the Set_s_NonConst method was used.
- SmartPtr< Vector > create_new_s ()

  Create a new vector in the s entry.
- SmartPtr< Vector > create_new_s_copy ()

  Create a new vector in the s entry and copy the current values into it.
- void Set_s (const Vector &vec)

  Set the s iterate (const).
- void Set_s_NonConst (Vector &vec)

  Set the s iterate (non-const).
- SmartPtr< const Vector > y_c () const

  Get the y_c iterate (const)
- SmartPtr< Vector > y_c_NonConst ()

  Get the y_c iterate (non-const) - this can only be called if the vector was created intenally, or the Set_y_c_NonConst method was used.
- SmartPtr< Vector > create_new_y_c ()

  Create a new vector in the y_c entry.
- SmartPtr< Vector > create_new_y_c_copy ()

  Create a new vector in the y_c entry and copy the current values into it.
- void Set_y_c (const Vector &vec)

  Set the y_c iterate (const).
- void Set_y_c_NonConst (Vector &vec)

  Set the y_c iterate (non-const).
- SmartPtr< const Vector > y_d () const

  Get the y_d iterate (const)
- SmartPtr< Vector > y_d_NonConst ()

Get the y_d iterate (non-const) - this can only be called if the vector was created intenally, or the Set_y_d_NonConst method was used.

- SmartPtr< Vector > create_new_y_d ()

  Create a new vector in the y_d entry.
- SmartPtr< Vector > create_new_y_d_copy ()

  Create a new vector in the y_d entry and copy the current values into it.
- void Set_y_d (const Vector &vec)

  Set the y_d iterate (const).
- void Set_y_d_NonConst (Vector &vec)

  Set the y_d iterate (non-const).
- SmartPtr< const Vector > z_L () const

  Get the z_L iterate (const)
- SmartPtr< Vector > z_L_NonConst ()

  Get the z_L iterate (non-const) - this can only be called if the vector was created intenally, or the Set_z_L_NonConst method was used.
- SmartPtr< Vector > create_new_z_L ()

  Create a new vector in the z_L entry.
- SmartPtr< Vector > create_new_z_L_copy ()

  Create a new vector in the z_L entry and copy the current values into it.
- void Set_z_L (const Vector &vec)

  Set the z_L iterate (const).
- void Set_z_L_NonConst (Vector &vec)

  Set the z_L iterate (non-const).
- SmartPtr< const Vector > z_U () const

  Get the z_U iterate (const)
- SmartPtr< Vector > z_U_NonConst ()

  Get the z_U iterate (non-const) - this can only be called if the vector was created intenally, or the Set_z_U_NonConst method was used.
- SmartPtr< Vector > create_new_z_U ()

  Create a new vector in the z_U entry.
- SmartPtr< Vector > create_new_z_U_copy ()

  Create a new vector in the z_U entry and copy the current values into it.
- void Set_z_U (const Vector &vec)

  Set the z_U iterate (const).
- void Set_z_U_NonConst (Vector &vec)

  Set the z_U iterate (non-const).
- SmartPtr< const Vector > v_L () const

  Get the v_L iterate (const)
- SmartPtr< Vector > v_L_NonConst ()

  Get the v_L iterate (non-const) - this can only be called if the vector was created intenally, or the Set_v_L_NonConst method was used.
- SmartPtr< Vector > create_new_v_L ()

  Create a new vector in the v_L entry.
- SmartPtr< Vector > create_new_v_L_copy ()

  Create a new vector in the v_L entry and copy the current values into it.
- void Set_v_L (const Vector &vec)

  Set the v_L iterate (const).
- void Set_v_L_NonConst (Vector &vec)

Set the v_L iterate (non-const).

- SmartPtr< const Vector > v_U () const

  Get the v_U iterate (const)

- SmartPtr< Vector > v_U_NonConst ()

  Get the v_U iterate (non-const) - this can only be called if the vector was created intenally, or the Set_v_U_NonConst method was used.

- SmartPtr< Vector > create_new_v_U ()

  Create a new vector in the v_U entry.

- SmartPtr< Vector > create_new_v_U_copy ()

  Create a new vector in the v_U entry and copy the current values into it.

- void Set_v_U (const Vector &vec)

  Set the v_U iterate (const).

- void Set_v_U_NonConst (Vector &vec)

  Set the v_U iterate (non-const).

- void Set_primal (const Vector &x, const Vector &s)

  Set the primal variables all in one shot.

- void Set_eq_mult (const Vector &y_c, const Vector &y_d)

  Set the eq multipliers all in one shot.

- void Set_bound_mult (const Vector &z_L, const Vector &z_U, const Vector &v_L, const Vector &v_U)

  Set the bound multipliers all in one shot.

- TaggedObject::Tag GetTagSum () const

  Get a sum of the tags of the contained items.

Additional Inherited Members

3.75.1 Detailed Description

Specialized CompoundVector class specifically for the algorithm iterates.

This class inherits from CompoundVector and is a specialized class for handling the iterates of the Ipopt Algorithm, that is, x, s, y_c, y_d, z_L, z_U, v_L, and v_U. It inherits from CompoundVector so it can behave like a CV in most calculations, but it has fixed dimensions and cannot be customized

Definition at line 27 of file IpIteratesVector.hpp.

3.75.2 Member Function Documentation

3.75.2.1 SmartPtr<IteratesVector> Ipopt::IteratesVector::MakeNewIteratesVector (bool *create_new* = `true`) const

Make New methods.

Use this method to create a new iterates vector. The MakeNew method on the Vector class also works, but it does not give the create_new option.

3.75.2.2 SmartPtr<IteratesVector> Ipopt::IteratesVector::MakeNewContainer ( ) const

Use this method to create a new iterates vector container.

This creates a new NonConst container, but the elements inside the iterates vector may be const. Therefore, the container can be modified to point to new entries, but the existing entries may or may not be modifiable.

3.75.2.3 SmartPtr<const Vector> Ipopt::IteratesVector::x ( ) const [inline]

Iterates Set/Get Methods.

Get the x iterate (const)

Definition at line 67 of file IpIteratesVector.hpp.

3.75.2.4 SmartPtr<Vector> Ipopt::IteratesVector::x_NonConst ( ) [inline]

Get the x iterate (non-const) - this can only be called if the vector was created intenally, or the Set_x_NonConst method was used.

Definition at line 75 of file IpIteratesVector.hpp.

3.75.2.5 SmartPtr<Vector> Ipopt::IteratesVector::create_new_x_copy ( ) [inline]

Create a new vector in the x entry and copy the current values into it.

Definition at line 86 of file IpIteratesVector.hpp.

3.75.2.6 void Ipopt::IteratesVector::Set_x (const Vector & *vec*) [inline]

Set the x iterate (const).

Sets the pointer, does NOT copy data.

Definition at line 96 of file IpIteratesVector.hpp.

3.75.2.7 void Ipopt::IteratesVector::Set_x_NonConst (Vector & *vec*) [inline]

Set the x iterate (non-const).

Sets the pointer, does NOT copy data.

Definition at line 103 of file IpIteratesVector.hpp.

3.75.2.8 SmartPtr<Vector> Ipopt::IteratesVector::s_NonConst ( ) [inline]

Get the s iterate (non-const) - this can only be called if the vector was created intenally, or the Set_s_NonConst method was used.

Definition at line 117 of file IpIteratesVector.hpp.

3.75.2.9 SmartPtr<Vector> Ipopt::IteratesVector::create_new_s_copy ( ) [inline]

Create a new vector in the s entry and copy the current values into it.

Definition at line 128 of file IpIteratesVector.hpp.

3.75.2.10 void Ipopt::IteratesVector::Set_s (const Vector & *vec*) [inline]

Set the s iterate (const).

Sets the pointer, does NOT copy data.

Definition at line 138 of file IpIteratesVector.hpp.

3.75.2.11 void Ipopt::IteratesVector::Set_s_NonConst (Vector & *vec*) [inline]

Set the s iterate (non-const).

Sets the pointer, does NOT copy data.

Definition at line 145 of file IpIteratesVector.hpp.

3.75.2.12 SmartPtr<Vector> Ipopt::IteratesVector::y_c_NonConst ( ) [inline]

Get the y_c iterate (non-const) - this can only be called if the vector was created intenally, or the Set_y_c_NonConst method was used.

Definition at line 159 of file IpIteratesVector.hpp.

3.75.2.13 SmartPtr<Vector> Ipopt::IteratesVector::create_new_y_c_copy ( ) [inline]

Create a new vector in the y_c entry and copy the current values into it.

Definition at line 170 of file IpIteratesVector.hpp.

3.75.2.14 void Ipopt::IteratesVector::Set_y_c (const Vector & *vec*) [inline]

Set the y_c iterate (const).

Sets the pointer, does NOT copy data.

Definition at line 180 of file IpIteratesVector.hpp.

3.75.2.15 void Ipopt::IteratesVector::Set_y_c_NonConst (Vector & *vec*) [inline]

Set the y_c iterate (non-const).

Sets the pointer, does NOT copy data.

Definition at line 187 of file IpIteratesVector.hpp.

3.75.2.16 SmartPtr<Vector> Ipopt::IteratesVector::y_d_NonConst ( ) [inline]

Get the y_d iterate (non-const) - this can only be called if the vector was created intenally, or the Set_y_d_NonConst method was used.

Definition at line 201 of file IpIteratesVector.hpp.

3.75.2.17 SmartPtr<Vector> Ipopt::IteratesVector::create_new_y_d_copy ( ) [inline]

Create a new vector in the y_d entry and copy the current values into it.

Definition at line 212 of file IpIteratesVector.hpp.

3.75.2.18 void Ipopt::IteratesVector::Set_y_d (const Vector & *vec*) [inline]

Set the y_d iterate (const).

Sets the pointer, does NOT copy data.

Definition at line 222 of file IpIteratesVector.hpp.

3.75.2.19 void Ipopt::IteratesVector::Set_y_d_NonConst (Vector & *vec*) [inline]

Set the y_d iterate (non-const).

Sets the pointer, does NOT copy data.

Definition at line 229 of file IpIteratesVector.hpp.

3.75.2.20 SmartPtr<Vector> Ipopt::IteratesVector::z_L_NonConst ( ) [inline]

Get the z_L iterate (non-const) - this can only be called if the vector was created intenally, or the Set_z_L_NonConst method was used.

Definition at line 243 of file IpIteratesVector.hpp.

3.75.2.21 SmartPtr<Vector> Ipopt::IteratesVector::create_new_z_L_copy ( ) [inline]

Create a new vector in the z_L entry and copy the current values into it.

Definition at line 254 of file IpIteratesVector.hpp.

3.75.2.22 void Ipopt::IteratesVector::Set_z_L (const Vector & *vec*) [inline]

Set the z_L iterate (const).

Sets the pointer, does NOT copy data.

Definition at line 264 of file IpIteratesVector.hpp.

3.75.2.23 void Ipopt::IteratesVector::Set_z_L_NonConst (Vector & *vec*) [inline]

Set the z_L iterate (non-const).

Sets the pointer, does NOT copy data.

Definition at line 271 of file IpIteratesVector.hpp.

3.75.2.24 SmartPtr<Vector> Ipopt::IteratesVector::z_U_NonConst ( ) [inline]

Get the z_U iterate (non-const) - this can only be called if the vector was created intenally, or the Set_z_U_NonConst method was used.

Definition at line 285 of file IpIteratesVector.hpp.

3.75.2.25 SmartPtr<Vector> Ipopt::IteratesVector::create_new_z_U_copy ( ) [inline]

Create a new vector in the z_U entry and copy the current values into it.

Definition at line 296 of file IpIteratesVector.hpp.

3.75.2.26 void Ipopt::IteratesVector::Set_z_U (const Vector & *vec*) [inline]

Set the z_U iterate (const).

Sets the pointer, does NOT copy data.

Definition at line 306 of file IpIteratesVector.hpp.

3.75.2.27 void Ipopt::IteratesVector::Set_z_U_NonConst (Vector & *vec*) [inline]

Set the z_U iterate (non-const).

Sets the pointer, does NOT copy data.

Definition at line 313 of file IpIteratesVector.hpp.

3.75.2.28 SmartPtr<Vector> Ipopt::IteratesVector::v_L_NonConst ( ) [inline]

Get the v_L iterate (non-const) - this can only be called if the vector was created intenally, or the Set_v_L_NonConst method was used.

Definition at line 327 of file IpIteratesVector.hpp.

3.75.2.29 SmartPtr<Vector> Ipopt::IteratesVector::create_new_v_L_copy ( ) [inline]

Create a new vector in the v_L entry and copy the current values into it.

Definition at line 338 of file IpIteratesVector.hpp.

3.75.2.30 void Ipopt::IteratesVector::Set_v_L (const Vector & *vec*) [inline]

Set the v_L iterate (const).

Sets the pointer, does NOT copy data.

Definition at line 348 of file IpIteratesVector.hpp.

3.75.2.31 void Ipopt::IteratesVector::Set_v_L_NonConst (Vector & *vec*) [inline]

Set the v_L iterate (non-const).

Sets the pointer, does NOT copy data.

Definition at line 355 of file IpIteratesVector.hpp.

3.75.2.32 SmartPtr<Vector> Ipopt::IteratesVector::v_U_NonConst ( ) [inline]

Get the v_U iterate (non-const) - this can only be called if the vector was created intenally, or the Set_v_U_NonConst method was used.

Definition at line 369 of file IpIteratesVector.hpp.

3.75.2.33 SmartPtr<Vector> Ipopt::IteratesVector::create_new_v_U_copy ( ) [inline]

Create a new vector in the v_U entry and copy the current values into it.

Definition at line 380 of file IpIteratesVector.hpp.

3.75.2.34 void Ipopt::IteratesVector::Set_v_U (const Vector & *vec*) [inline]

Set the v_U iterate (const).

Sets the pointer, does NOT copy data.

Definition at line 390 of file IpIteratesVector.hpp.

3.75.2.35 void Ipopt::IteratesVector::Set_v_U_NonConst (Vector & *vec*) [inline]

Set the v_U iterate (non-const).

Sets the pointer, does NOT copy data.

Definition at line 397 of file IpIteratesVector.hpp.

3.75.2.36 void Ipopt::IteratesVector::Set_primal (const Vector & *x*, const Vector & *s*) [inline]

Set the primal variables all in one shot.

Sets the pointers, does NOT copy data

Definition at line 404 of file IpIteratesVector.hpp.

3.75.2.37 void Ipopt::IteratesVector::Set_eq_mult (const Vector & *y_c*, const Vector & *y_d*) [inline]

Set the eq multipliers all in one shot.

Sets the pointers, does not copy data.

Definition at line 417 of file IpIteratesVector.hpp.

3.75.2.38 void Ipopt::IteratesVector::Set_bound_mult (const Vector & *z_L*, const Vector & *z_U*, const Vector & *v_L*, const Vector & *v_U*) [inline]

Set the bound multipliers all in one shot.

Sets the pointers, does not copy data.

Definition at line 430 of file IpIteratesVector.hpp.

3.75.2.39 TaggedObject::Tag Ipopt::IteratesVector::GetTagSum ( ) const [inline]

Get a sum of the tags of the contained items.

There is no guarantee that this is unique, but there is a high chance it is unique and it can be used for debug checks relatively reliably.

Definition at line 450 of file IpIteratesVector.hpp.

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

- IpIteratesVector.hpp

3.76 Ipopt::IteratesVectorSpace Class Reference

Vector Space for the IteratesVector class.

`#include <IpIteratesVector.hpp>`

Inheritance diagram for Ipopt::IteratesVectorSpace:

Public Member Functions

- virtual void SetCompSpace (Index icomp, const VectorSpace &vec_space)

  This method hides the CompoundVectorSpace::SetCompSpace method since the components of the Iterates are fixed at construction.

Constructors/Destructors.

- IteratesVectorSpace (const VectorSpace &x_space, const VectorSpace &s_space, const VectorSpace &y_c_space, const VectorSpace &y_d_space, const VectorSpace &z_L_space, const VectorSpace &z_U_space, const VectorSpace &v_L_space, const VectorSpace &v_U_space)

  Constructor that takes the spaces for each of the iterates.
- virtual ∼**IteratesVectorSpace** ()

- virtual IteratesVector ∗ MakeNewIteratesVector (bool create_new=true) const

  Method for creating vectors .
- const SmartPtr< const IteratesVector > MakeNewIteratesVector (const Vector &x, const Vector &s, const Vector &y_c, const Vector &y_d, const Vector &z_L, const Vector &z_U, const Vector &v_L, const Vector &v_U)

  Use this method to create a new const IteratesVector.
- virtual CompoundVector ∗ MakeNewCompoundVector (bool create_new=true) const

  This method overloads ComooundVectorSpace::MakeNewCompoundVector to make sure that we get a vector of the correct type.
- virtual Vector ∗ MakeNew () const

  This method creates a new vector (and allocates space in all the contained vectors.

3.76.1 Detailed Description

Vector Space for the IteratesVector class.

This is a specialized vector space for the IteratesVector class.

Definition at line 532 of file IpIteratesVector.hpp.

3.76.2 Constructor & Destructor Documentation

3.76.2.1 Ipopt::IteratesVectorSpace::IteratesVectorSpace (const VectorSpace & *x_space*, const VectorSpace & *s_space*, const VectorSpace & *y_c_space*, const VectorSpace & *y_d_space*, const VectorSpace & *z_L_space*, const VectorSpace & *z_U_space*, const VectorSpace & *v_L_space*, const VectorSpace & *v_U_space*)

Constructor that takes the spaces for each of the iterates.

Warning! None of these can be NULL !

3.76.3 Member Function Documentation

3.76.3.1 virtual IteratesVector* Ipopt::IteratesVectorSpace::MakeNewIteratesVector (bool *create_new* = true) const [inline], [virtual]

Method for creating vectors .

Use this to create a new IteratesVector. You can pass-in create_new = false if you only want a container and do not want vectors allocated.

Definition at line 555 of file IpIteratesVector.hpp.

3.76.3.2 const SmartPtr<const IteratesVector> Ipopt::IteratesVectorSpace::MakeNewIteratesVector (const Vector & *x*, const Vector & *s*, const Vector & *y_c*, const Vector & *y_d*, const Vector & *z_L*, const Vector & *z_U*, const Vector & *v_L*, const Vector & *v_U*) [inline]

Use this method to create a new const IteratesVector.

You must pass in valid pointers for all of the entries.

Definition at line 563 of file IpIteratesVector.hpp.

3.76.3.3 virtual Vector* Ipopt::IteratesVectorSpace::MakeNew ( ) const [inline], [virtual]

This method creates a new vector (and allocates space in all the contained vectors.

This is really only used for code that does not know what type of vector it is dealing with - for example, this method is called from Vector::MakeNew()

Reimplemented from Ipopt::CompoundVectorSpace.

Definition at line 595 of file IpIteratesVector.hpp.

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

- IpIteratesVector.hpp

3.77 Ipopt::IterationOutput Class Reference

Base class for objects that do the output summary per iteration.

#include <IpIterationOutput.hpp>

Inheritance diagram for Ipopt::IterationOutput:

Public Member Functions

- virtual bool InitializeImpl (const OptionsList &options, const std::string &prefix)=0

  overloaded from AlgorithmStrategyObject
- virtual void WriteOutput ()=0

  Method to do all the summary output per iteration.

Constructors/Destructors

- IterationOutput ()

Default Constructor.
- virtual ~IterationOutput ()

  Default destructor.

Protected Types

- enum InfPrOutput

  enumeration for different inf_pr output options

Additional Inherited Members

3.77.1 Detailed Description

Base class for objects that do the output summary per iteration.

Definition at line 22 of file IpIterationOutput.hpp.

3.77.2 Member Function Documentation

3.77.2.1 virtual void Ipopt::IterationOutput::WriteOutput ( ) [pure virtual]

Method to do all the summary output per iteration.

This include the one-line summary output as well as writing the details about the iterates if desired

Implemented in Ipopt::RestoIterationOutput, and Ipopt::OrigIterationOutput.

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

- IpIterationOutput.hpp

3.78 Ipopt::IterativePardisoSolverInterface Class Reference

Interface to the linear solver Pardiso, derived from SparseSymLinearSolverInterface.

`#include <IpIterativePardisoSolverInterface.hpp>`

Inheritance diagram for Ipopt::IterativePardisoSolverInterface:

Public Member Functions

- bool InitializeImpl (const OptionsList &options, const std::string &prefix)

  overloaded from AlgorithmStrategyObject

Constructor/Destructor

- IterativePardisoSolverInterface (IterativeSolverTerminationTester &normal_tester, IterativeSolverTermination←Tester &pd_tester)

  Constructor.
- virtual ~IterativePardisoSolverInterface ()

  Destructor.

Methods for requesting solution of the linear system.

- virtual ESymSolverStatus InitializeStructure (Index dim, Index nonzeros, const Index ∗ia, const Index ∗ja)
  Method for initializing internal stuctures.
- virtual double ∗ GetValuesArrayPtr ()
  Method returing an internal array into which the nonzero elements are to be stored.
- virtual ESymSolverStatus MultiSolve (bool new_matrix, const Index ∗ia, const Index ∗ja, Index nrhs, double ∗rhs_vals, bool check_NegEVals, Index numberOfNegEVals)
  Solve operation for multiple right hand sides.
- virtual Index NumberOfNegEVals () const
  Number of negative eigenvalues detected during last factorization.

- virtual bool IncreaseQuality ()
  Request to increase quality of solution for next solve.
- virtual bool ProvidesInertia () const
  Query whether inertia is computed by linear solver.
- EMatrixFormat MatrixFormat () const
  Query of requested matrix type that the linear solver understands.

Static Public Member Functions

- static void RegisterOptions (SmartPtr< RegisteredOptions > roptions)
  Methods for IpoptType.

Additional Inherited Members

3.78.1 Detailed Description

Interface to the linear solver Pardiso, derived from SparseSymLinearSolverInterface.

For details, see description of SparseSymLinearSolverInterface base class.

Definition at line 25 of file IpIterativePardisoSolverInterface.hpp.

3.78.2 Member Function Documentation

3.78.2.1 virtual ESymSolverStatus Ipopt::IterativePardisoSolverInterface::InitializeStructure (Index *dim*, Index *nonzeros*, const Index ∗ *ia*, const Index ∗ *ja*) [virtual]

Method for initializing internal stuctures.

Implements Ipopt::SparseSymLinearSolverInterface.

3.78.2.2 virtual double∗ Ipopt::IterativePardisoSolverInterface::GetValuesArrayPtr ( ) [virtual]

Method returing an internal array into which the nonzero elements are to be stored.

Implements Ipopt::SparseSymLinearSolverInterface.

3.78.2.3 virtual ESymSolverStatus Ipopt::IterativePardisoSolverInterface::MultiSolve (bool *new_matrix,* const Index ∗ *ia,* const Index ∗ *ja,* Index *nrhs,* double ∗ *rhs_vals,* bool *check_NegEVals,* Index *numberOfNegEVals*) [virtual]

Solve operation for multiple right hand sides.

Implements Ipopt::SparseSymLinearSolverInterface.

3.78.2.4 virtual bool Ipopt::IterativePardisoSolverInterface::ProvidesInertia ( ) const [inline], [virtual]

Query whether inertia is computed by linear solver.

Returns true, if linear solver provides inertia.

Implements Ipopt::SparseSymLinearSolverInterface.

Definition at line 78 of file IpIterativePardisoSolverInterface.hpp.

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

- IpIterativePardisoSolverInterface.hpp

3.79 Ipopt::IterativeSolverTerminationTester Class Reference

This base class is for the termination tests for the iterative linear solver in the inexact version of Ipopt.

`#include <IpIterativeSolverTerminationTester.hpp>`

Inheritance diagram for Ipopt::IterativeSolverTerminationTester:

Public Types

- enum ETerminationTest {
CONTINUE, TEST_1_SATISFIED, TEST_2_SATISFIED, TEST_3_SATISFIED,
MODIFY_HESSIAN, OTHER_SATISFIED }

  Enum to report result of termination test.

Public Member Functions

- virtual bool InitializeImpl (const OptionsList &options, const std::string &prefix)=0

  Implementation of the initialization method that has to be overloaded by for each derived class.
- virtual bool InitializeSolve ()=0

  Method for initializing for the next iterative solve.
- virtual ETerminationTest TestTermination (Index ndim, const Number ∗sol, const Number ∗resid, Index iter, Number norm2_rhs)=0

  This method checks if the current soltion of the iterative linear solver is good enough (by returning the corresponding satisfied termination test), or if the Hessian should be modified.
- virtual void Clear ()=0

  This method can be called after the Solve is over and we can delete anything that has been allocated to free memory.
- const Journalist & GetJnlst () const

  An easy way to get the journalist if accessed from the outside.
- virtual Index GetSolverIterations () const =0

  Return the number of iterative solver iteration from the most recent solve.

/Destructor

- IterativeSolverTerminationTester ()

    Default constructor.
- virtual ∼IterativeSolverTerminationTester ()

    Default destructor.

Protected Member Functions

- void GetVectors (Index ndim, const Number ∗array, SmartPtr< const Vector > &comp_x, SmartPtr< const Vector > &comp_s, SmartPtr< const Vector > &comp_c, SmartPtr< const Vector > &comp_d)

    Method for copying a long augmented system array into Vectors in Ipopt notation.
- InexactData & InexData ()

    Method to easily access Inexact data.
- InexactCq & InexCq ()

    Method to easily access Inexact calculated quantities.

3.79.1 Detailed Description

This base class is for the termination tests for the iterative linear solver in the inexact version of Ipopt.

Definition at line 21 of file IpIterativeSolverTerminationTester.hpp.

3.79.2 Member Enumeration Documentation

3.79.2.1 enum Ipopt::IterativeSolverTerminationTester::ETerminationTest

Enum to report result of termination test.

Enumerator

CONTINUE The current solution is not yet good enough.

TEST_1_SATISFIED Termination Test 1 is satisfied.

TEST_2_SATISFIED Termination Test 2 is satisfied.

TEST_3_SATISFIED Termination Test 3 is satisfied.

MODIFY_HESSIAN Hessian matrix should be modified.

OTHER_SATISFIED Some other termination criterion satisfied.

Definition at line 25 of file IpIterativeSolverTerminationTester.hpp.

3.79.3 Member Function Documentation

3.79.3.1 virtual bool Ipopt::IterativeSolverTerminationTester::InitializeImpl (const OptionsList & *options*, const std::string & *prefix*) [pure virtual]

Implementation of the initialization method that has to be overloaded by for each derived class.

Implements Ipopt::AlgorithmStrategyObject.

Implemented in Ipopt::InexactNormalTerminationTester, and Ipopt::InexactPDTerminationTester.

3.79.3.2 virtual bool Ipopt::IterativeSolverTerminationTester::InitializeSolve ( ) [pure virtual]

Method for initializing for the next iterative solve.

This must be call before the test methods are called.

Implemented in Ipopt::InexactNormalTerminationTester, and Ipopt::InexactPDTerminationTester.

3.79.3.3 virtual ETerminationTest Ipopt::IterativeSolverTerminationTester::TestTermination (Index *ndim*, const Number ∗ *sol*, const Number ∗ *resid*, Index *iter*, Number *norm2_rhs*) [pure virtual]

This method checks if the current soltion of the iterative linear solver is good enough (by returning the corresponding satisfied termination test), or if the Hessian should be modified.

The input is the dimension of the augmented system, the current solution vector of the augmented system, the current residual vector.

Implemented in Ipopt::InexactNormalTerminationTester, and Ipopt::InexactPDTerminationTester.

3.79.3.4 virtual void Ipopt::IterativeSolverTerminationTester::Clear ( ) [pure virtual]

This method can be called after the Solve is over and we can delete anything that has been allocated to free memory.

Implemented in Ipopt::InexactNormalTerminationTester, and Ipopt::InexactPDTerminationTester.

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

- IpIterativeSolverTerminationTester.hpp

3.80 Ipopt::IterativeWsmpSolverInterface Class Reference

Interface to the linear solver WISMP, derived from SparseSymLinearSolverInterface.

`#include <IpIterativeWsmpSolverInterface.hpp>`

Inheritance diagram for Ipopt::IterativeWsmpSolverInterface:

Public Member Functions

- bool InitializeImpl (const OptionsList &options, const std::string &prefix)

  overloaded from AlgorithmStrategyObject

Constructor/Destructor

- IterativeWsmpSolverInterface ()

  Constructor.
- virtual ∼IterativeWsmpSolverInterface ()

  Destructor.

Methods for requesting solution of the linear system.

- virtual ESymSolverStatus InitializeStructure (Index dim, Index nonzeros, const Index ∗ia, const Index ∗ja)

  Method for initializing internal stuctures.
- virtual double ∗ GetValuesArrayPtr ()

Method returing an internal array into which the nonzero elements are to be stored.
  - virtual ESymSolverStatus MultiSolve (bool new_matrix, const Index *ia, const Index *ja, Index nrhs, double *rhs_vals, bool check_NegEVals, Index numberOfNegEVals)

    Solve operation for multiple right hand sides.
  - virtual Index NumberOfNegEVals () const

    Number of negative eigenvalues detected during last factorization.

- virtual bool IncreaseQuality ()

  Request to increase quality of solution for next solve.
- virtual bool ProvidesInertia () const

  Query whether inertia is computed by linear solver.
- EMatrixFormat MatrixFormat () const

  Query of requested matrix type that the linear solver understands.

Static Public Member Functions

- static void RegisterOptions (SmartPtr< RegisteredOptions > roptions)

  Methods for IpoptType.

Additional Inherited Members

3.80.1 Detailed Description

Interface to the linear solver WISMP, derived from SparseSymLinearSolverInterface.

For details, see description of SparseSymLinearSolverInterface base class.

Definition at line 23 of file IpIterativeWsmpSolverInterface.hpp.

3.80.2 Member Function Documentation

3.80.2.1 virtual ESymSolverStatus Ipopt::IterativeWsmpSolverInterface::InitializeStructure (Index *dim,* Index *nonzeros,* const Index * *ia,* const Index * *ja*) [virtual]

Method for initializing internal stuctures.

Implements Ipopt::SparseSymLinearSolverInterface.

3.80.2.2 virtual double* Ipopt::IterativeWsmpSolverInterface::GetValuesArrayPtr ( ) [virtual]

Method returing an internal array into which the nonzero elements are to be stored.

Implements Ipopt::SparseSymLinearSolverInterface.

3.80.2.3 virtual ESymSolverStatus Ipopt::IterativeWsmpSolverInterface::MultiSolve (bool *new_matrix,* const Index * *ia,* const Index * *ja,* Index *nrhs,* double * *rhs_vals,* bool *check_NegEVals,* Index *numberOfNegEVals*) [virtual]

Solve operation for multiple right hand sides.

Implements Ipopt::SparseSymLinearSolverInterface.

3.80.2.4 virtual bool Ipopt::IterativeWsmpSolverInterface::ProvidesInertia ( ) const [inline], [virtual]

Query whether inertia is computed by linear solver.

Returns true, if linear solver provides inertia.

Implements Ipopt::SparseSymLinearSolverInterface.

Definition at line 75 of file IpIterativeWsmpSolverInterface.hpp.

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

- IpIterativeWsmpSolverInterface.hpp

3.81 Ipopt::Journal Class Reference

Journal class (part of the Journalist implementation.).

`#include <IpJournalist.hpp>`

Inheritance diagram for Ipopt::Journal:

Public Member Functions

- Journal (const std::string &name, EJournalLevel default_level)

  Constructor.
- virtual ∼Journal ()

  Destructor.
- virtual std::string Name ()

  Get the name of the Journal.
- virtual void SetPrintLevel (EJournalCategory category, EJournalLevel level)

  Set the print level for a particular category.
- virtual void SetAllPrintLevels (EJournalLevel level)

  Set the print level for all category.

Journal Output Methods. These methods are called by the

Journalist who first checks if the output print level and category are acceptable.

Calling the Print methods explicitly (instead of through the Journalist will output the message regardless of print level and category. You should use the Journalist to print & flush instead

- virtual bool IsAccepted (EJournalCategory category, EJournalLevel level) const

  Ask if a particular print level/category is accepted by the journal.
- virtual void Print (EJournalCategory category, EJournalLevel level, const char ∗str)

  Print to the designated output location.
- virtual void Printf (EJournalCategory category, EJournalLevel level, const char ∗pformat, va_list ap)

  Printf to the designated output location.
- virtual void FlushBuffer ()

  Flush output buffer.

Protected Member Functions

Implementation version of Print methods. Derived classes

should overload the Impl methods.

- virtual void PrintImpl (EJournalCategory category, EJournalLevel level, const char ∗str)=0

  Print to the designated output location.
- virtual void PrintfImpl (EJournalCategory category, EJournalLevel level, const char ∗pformat, va_list ap)=0

  Printf to the designated output location.
- virtual void FlushBufferImpl ()=0

  Flush output buffer.

3.81.1 Detailed Description

Journal class (part of the Journalist implementation.).

This class is the base class for all Journals. It controls the acceptance criteria for print statements etc. Derived classes like the FileJournal - output those messages to specific locations

Definition at line 273 of file IpJournalist.hpp.

3.81.2 Constructor & Destructor Documentation

3.81.2.1 Ipopt::Journal::Journal (const std::string & *name*, EJournalLevel *default_level*)

Constructor.

3.81.2.2 virtual Ipopt::Journal::∼Journal ( ) [virtual]

Destructor.

3.81.3 Member Function Documentation

3.81.3.1 virtual void Ipopt::Journal::SetPrintLevel (EJournalCategory *category*, EJournalLevel *level*) [virtual]

Set the print level for a particular category.

3.81.3.2 virtual void Ipopt::Journal::SetAllPrintLevels (EJournalLevel *level*) [virtual]

Set the print level for all category.

3.81.3.3 virtual void Ipopt::Journal::FlushBuffer ( ) [inline], [virtual]

Flush output buffer.

Definition at line 325 of file IpJournalist.hpp.

3.81.3.4 virtual void Ipopt::Journal::FlushBufferImpl ( ) [protected], [pure virtual]

Flush output buffer.

Implemented in Ipopt::StreamJournal, and Ipopt::FileJournal.

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

- IpJournalist.hpp

3.82 Ipopt::Journalist Class Reference

Class responsible for all message output.

`#include <IpJournalist.hpp>`

Inheritance diagram for Ipopt::Journalist:

Public Member Functions

Constructor / Desructor.

- Journalist ()
  Constructor.
- virtual ∼Journalist ()
  Destructor...

Author Methods.

These methods are used by authoring code, or code that wants to report some information.

- virtual void Printf (EJournalLevel level, EJournalCategory category, const char ∗format,...) const
  Method to print a formatted string.
- virtual void PrintStringOverLines (EJournalLevel level, EJournalCategory category, Index indent_spaces, Index max_length, const std::string &line) const
  Method to print a long string including indentation.
- virtual void PrintfIndented (EJournalLevel level, EJournalCategory category, Index indent_level, const char ∗format,...) const
  Method to print a formatted string with indentation.
- virtual void VPrintf (EJournalLevel level, EJournalCategory category, const char ∗pformat, va_list ap) const
  Method to print a formatted string using the va_list argument.
- virtual void VPrintfIndented (EJournalLevel level, EJournalCategory category, Index indent_level, const char ∗pformat, va_list ap) const
  Method to print a formatted string with indentation, using the va_list argument.
- virtual bool ProduceOutput (EJournalLevel level, EJournalCategory category) const
  Method that returns true if there is a Journal that would write output for the given JournalLevel and JournalCategory.
- virtual void FlushBuffer () const
  Method that flushes the current buffer for all Journalists.

Reader Methods.

These methods are used by the reader.

The reader will setup the journalist with each output file and the acceptance criteria for that file.

Use these methods to setup the journals (files or other output). These are the internal objects that keep track of the print levels for each category. Then use the internal Journal objects to set specific print levels for each category (or keep defaults).

- virtual bool AddJournal (const SmartPtr< Journal > jrnl)

  Add a new journal.
- virtual SmartPtr< Journal > AddFileJournal (const std::string &location_name, const std::string &fname, E↩ JournalLevel default_level=J_WARNING)

  Add a new FileJournal.
- virtual SmartPtr< Journal > GetJournal (const std::string &location_name)

  Get an existing journal.
- virtual void DeleteAllJournals ()

  Delete all journals curently known by the journalist.

3.82.1 Detailed Description

Class responsible for all message output.

This class is responsible for all messaging and output. The "printing" code or "author" should send ALL messages to the Journalist, indicating an appropriate category and print level. The journalist then decides, based on reader specified acceptance criteria, which message is actually printed in which journals. This allows the printing code to send everything, while the "reader" can decide what they really want to see.

Authors: Authors use the Journals: You can add as many Journals as you like to the Journalist with the AddJournal or the AddFileJournal methods. Each one represents a different printing location (or file). Then, you can call the "print" methods of the Journalist to output information to each of the journals.

Acceptance Criteria: Each print message should be flagged appropriately with an EJournalCategory and EJournalLevel.

The AddFileJournal method returns a pointer to the newly created Journal object (if successful) so you can set Acceptance criteria for that particular location.

Definition at line 134 of file IpJournalist.hpp.

3.82.2 Constructor & Destructor Documentation

3.82.2.1 Ipopt::Journalist::Journalist ( )

Constructor.

3.82.2.2 virtual Ipopt::Journalist::∼Journalist ( ) [virtual]

Destructor...

3.82.3 Member Function Documentation

3.82.3.1 virtual void Ipopt::Journalist::PrintStringOverLines (EJournalLevel *level,* EJournalCategory *category,* Index *indent_spaces,* Index *max_length,* const std::string & *line*) const [virtual]

Method to print a long string including indentation.

The string is printed starting at the current position. If the position (counting started at the current position) exceeds max_length, a new line is inserted, and indent_spaces many spaces are printed before the string is continued. This is for example used during the printing of the option documentation.

3.82.3.2 virtual void Ipopt::Journalist::VPrintf (EJournalLevel *level,* EJournalCategory *category,* const char * *pformat,* va_list *ap*) const [virtual]

Method to print a formatted string using the va_list argument.

3.82.3.3 virtual void Ipopt::Journalist::VPrintfIndented (EJournalLevel *level,* EJournalCategory *category,* Index *indent_level,* const char * *pformat,* va_list *ap*) const [virtual]

Method to print a formatted string with indentation, using the va_list argument.

3.82.3.4 virtual bool Ipopt::Journalist::ProduceOutput (EJournalLevel *level,* EJournalCategory *category*) const [virtual]

Method that returns true if there is a Journal that would write output for the given JournalLevel and JournalCategory.

This is useful if expensive computation would be required for a particular output. The author code can check with this method if the computations are indeed required.

3.82.3.5 virtual void Ipopt::Journalist::FlushBuffer ( ) const [virtual]

Method that flushes the current buffer for all Journalists.

Calling this method after one optimization run helps to avoid cluttering output with that produced by other parts of the program (e.g. written in Fortran)

3.82.3.6 virtual bool Ipopt::Journalist::AddJournal (const SmartPtr< Journal > *jrnl*) [virtual]

Add a new journal.

The location_name is a string identifier, which can be used to obtain the pointer to the new Journal at a later point using the GetJournal method. The default_level is used to initialize the * printing level for all categories.

3.82.3.7 virtual SmartPtr<Journal> Ipopt::Journalist::AddFileJournal (const std::string & *location_name,* const std::string & *fname,* EJournalLevel *default_level* = J_WARNING) [virtual]

Add a new FileJournal.

fname is the name of the * file to which this Journal corresponds. Use fname="stdout" * for stdout, and use fname="stderr" for stderr. This method * returns the Journal pointer so you can set specific acceptance criteria. It returns NULL if there was a problem creating a new Journal.

Parameters

location_name	journal identifier
fname	file name
default_level	default journal level

3.82.3.8 virtual SmartPtr<Journal> Ipopt::Journalist::GetJournal (const std::string & *location_name*) [virtual]

Get an existing journal.

You can use this method to change the acceptance criteria at runtime.

3.82.3.9 virtual void Ipopt::Journalist::DeleteAllJournals ( ) [virtual]

Delete all journals curently known by the journalist.

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

- IpJournalist.hpp

3.83 Ipopt::LeastSquareMultipliers Class Reference

Class for calculator for the least-square equality constraint multipliers.

`#include <IpLeastSquareMults.hpp>`

Inheritance diagram for Ipopt::LeastSquareMultipliers:

Public Member Functions

- virtual bool InitializeImpl (const OptionsList &options, const std::string &prefix)

  overloaded from AlgorithmStrategyObject
- virtual bool CalculateMultipliers (Vector &y_c, Vector &y_d)

  This method computes the least-square estimates for y_c and y_d at the current point.

Constructors/Destructors

- LeastSquareMultipliers (AugSystemSolver &augSysSolver)

  Constructor.
- virtual ∼LeastSquareMultipliers ()

  Default destructor.

Additional Inherited Members

3.83.1 Detailed Description

Class for calculator for the least-square equality constraint multipliers.

The Calculate method of this class computes the least-square estimate for the y_c and y_d multiplers, based on the current values of the gradient of the Lagrangian.

Definition at line 23 of file IpLeastSquareMults.hpp.

3.83.2 Constructor & Destructor Documentation

3.83.2.1 Ipopt::LeastSquareMultipliers::LeastSquareMultipliers (AugSystemSolver & *augSysSolver*)

Constructor.

It needs to be given the strategy object for solving the augmented system.

3.83.3 Member Function Documentation

3.83.3.1 virtual bool Ipopt::LeastSquareMultipliers::CalculateMultipliers (Vector & *y_c*, Vector & *y_d*) [virtual]

This method computes the least-square estimates for y_c and y_d at the current point.

The return value is false, if the least square system could not be solved (the linear system is singular).

Implements Ipopt::EqMultiplierCalculator.

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

- IpLeastSquareMults.hpp

3.84 Ipopt::LimMemQuasiNewtonUpdater Class Reference

Implementation of the HessianUpdater for limit-memory quasi-Newton approximation of the Lagrangian Hessian.

`#include <IpLimMemQuasiNewtonUpdater.hpp>`

Inheritance diagram for Ipopt::LimMemQuasiNewtonUpdater:

Public Member Functions

- virtual bool InitializeImpl (const OptionsList &options, const std::string &prefix)

  overloaded from AlgorithmStrategyObject
- virtual void UpdateHessian ()

  Update the Hessian based on the current information in IpData.

Constructors/Destructors

- LimMemQuasiNewtonUpdater (bool update_for_resto)

  Default Constructor.
- virtual ∼LimMemQuasiNewtonUpdater ()

  Default destructor.

Static Public Member Functions

- static void RegisterOptions (SmartPtr< RegisteredOptions > roptions)

  Methods for OptionsList.

Additional Inherited Members

3.84.1 Detailed Description

Implementation of the HessianUpdater for limit-memory quasi-Newton approximation of the Lagrangian Hessian.

Definition at line 25 of file IpLimMemQuasiNewtonUpdater.hpp.

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

- IpLimMemQuasiNewtonUpdater.hpp

3.85 Ipopt::LineSearch Class Reference

Base class for line search objects.

`#include <IpLineSearch.hpp>`

Inheritance diagram for Ipopt::LineSearch:

Public Member Functions

- virtual void FindAcceptableTrialPoint ()=0

  Perform the line search.
- virtual void Reset ()=0

  Reset the line search.
- virtual void SetRigorousLineSearch (bool rigorous)=0

  Set flag indicating whether a very rigorous line search should be performed.
- virtual bool CheckSkippedLineSearch ()=0

  Check if the line search procedure didn't accept a new iterate during the last call of FindAcceptableTrialPoint().
- virtual bool ActivateFallbackMechanism ()=0

  This method should be called if the optimization process requires the line search object to switch to some fallback mechanism (like the restoration phase), when the regular optimization procedure cannot be continued (for example, because the search direction could not be computed).

Constructors/Destructors

- LineSearch ()

  Default Constructor.
- virtual ∼LineSearch ()

  Default destructor.

Additional Inherited Members

3.85.1 Detailed Description

Base class for line search objects.

Definition at line 20 of file IpLineSearch.hpp.

3.85.2 Member Function Documentation

3.85.2.1 virtual void Ipopt::LineSearch::FindAcceptableTrialPoint ( ) [pure virtual]

Perform the line search.

As search direction the delta in the data object is used

Implemented in Ipopt::BacktrackingLineSearch.

3.85.2.2 virtual void Ipopt::LineSearch::Reset ( ) [pure virtual]

Reset the line search.

This function should be called if all previous information should be discarded when the line search is performed the next time. For example, this method should be called after the barrier parameter is changed.

Implemented in Ipopt::BacktrackingLineSearch.

3.85.2.3 virtual void Ipopt::LineSearch::SetRigorousLineSearch (bool *rigorous*) [pure virtual]

Set flag indicating whether a very rigorous line search should be performed.

If this flag is set to true, the line search algorithm might decide to abort the line search and not to accept a new iterate. If the line search decided not to accept a new iterate, the return value of CheckSkippedLineSearch() is true at the next call. For example, in the non-monotone barrier parameter update procedure, the filter algorithm should not switch to the restoration phase in the free mode; instead, the algorithm should swtich to the fixed mode.

Implemented in Ipopt::BacktrackingLineSearch.

3.85.2.4 virtual bool Ipopt::LineSearch::ActivateFallbackMechanism ( ) [pure virtual]

This method should be called if the optimization process requires the line search object to switch to some fallback mechanism (like the restoration phase), when the regular optimization procedure cannot be continued (for example, because the search direction could not be computed).

This will cause the line search object to immediately proceed with this mechanism when FindAcceptableTrialPoint() is call. This method returns false if no fallback mechanism is available.

Implemented in Ipopt::BacktrackingLineSearch.

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

- IpLineSearch.hpp

3.86 Ipopt::LoqoMuOracle Class Reference

Implementation of the LOQO formula for computing the barrier parameter.

`#include <IpLoqoMuOracle.hpp>`

Inheritance diagram for Ipopt::LoqoMuOracle:

Public Member Functions

- virtual bool InitializeImpl (const OptionsList &options, const std::string &prefix)

  Initialize method - overloaded from AlgorithmStrategyObject.
- virtual bool CalculateMu (Number mu_min, Number mu_max, Number &new_mu)

  Method for computing the value of the barrier parameter that could be used in the current iteration (using the LOQO formula).

Constructors/Destructors

- LoqoMuOracle ()

Default Constructor.
- virtual ∼LoqoMuOracle ()

  Default destructor.

Additional Inherited Members

3.86.1 Detailed Description

Implementation of the LOQO formula for computing the barrier parameter.

Definition at line 20 of file IpLoqoMuOracle.hpp.

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

- IpLoqoMuOracle.hpp

3.87 Ipopt::LowRankAugSystemSolver Class Reference

Solver for the augmented system with LowRankUpdateSymMatrix Hessian matrices.

`#include <IpLowRankAugSystemSolver.hpp>`

Inheritance diagram for Ipopt::LowRankAugSystemSolver:

Public Member Functions

- bool InitializeImpl (const OptionsList &options, const std::string &prefix)

  overloaded from AlgorithmStrategyObject
- virtual ESymSolverStatus Solve (const SymMatrix ∗W, double W_factor, const Vector ∗D_x, double delta_x, const Vector ∗D_s, double delta_s, const Matrix ∗J_c, const Vector ∗D_c, double delta_c, const Matrix ∗J_d, const Vector ∗D_d, double delta_d, const Vector &rhs_x, const Vector &rhs_s, const Vector &rhs_c, const Vector &rhs_d, Vector &sol_x, Vector &sol_s, Vector &sol_c, Vector &sol_d, bool check_NegEVals, Index numberOfNegEVals)

  Set up the augmented system and solve it for a given right hand side.
- virtual Index NumberOfNegEVals () const

  Number of negative eigenvalues detected during last solve.
- virtual bool ProvidesInertia () const

  Query whether inertia is computed by linear solver.
- virtual bool IncreaseQuality ()

  Request to increase quality of solution for next solve.

Constructors/Destructors

- LowRankAugSystemSolver (AugSystemSolver &aug_system_solver)

  Constructor using only a linear solver object.
- virtual ∼LowRankAugSystemSolver ()

  Default destructor.

Additional Inherited Members

3.87.1 Detailed Description

Solver for the augmented system with LowRankUpdateSymMatrix Hessian matrices.

This version works with the Sherman-Morrison formula and multiple backsolves.

Definition at line 24 of file IpLowRankAugSystemSolver.hpp.

3.87.2 Member Function Documentation

3.87.2.1 virtual Index Ipopt::LowRankAugSystemSolver::NumberOfNegEVals ( ) const [virtual]

Number of negative eigenvalues detected during last solve.

Returns the number of negative eigenvalues of the most recent factorized matrix. This must not be called if the linear solver does not compute this quantities (see ProvidesInertia).

Implements Ipopt::AugSystemSolver.

3.87.2.2 virtual bool Ipopt::LowRankAugSystemSolver::ProvidesInertia ( ) const [virtual]

Query whether inertia is computed by linear solver.

Returns true, if linear solver provides inertia.

Implements Ipopt::AugSystemSolver.

3.87.2.3 virtual bool Ipopt::LowRankAugSystemSolver::IncreaseQuality ( ) [virtual]

Request to increase quality of solution for next solve.

Ask underlying linear solver to increase quality of solution for the next solve (e.g. increase pivot tolerance). Returns false, if this is not possible (e.g. maximal pivot tolerance already used.)

Implements Ipopt::AugSystemSolver.

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

- IpLowRankAugSystemSolver.hpp

3.88 Ipopt::LowRankSSAugSystemSolver Class Reference

Solver for the augmented system with LowRankUpdateSymMatrix Hessian matrices.

`#include <IpLowRankSSAugSystemSolver.hpp>`

Inheritance diagram for Ipopt::LowRankSSAugSystemSolver:

Public Member Functions

- bool InitializeImpl (const OptionsList &options, const std::string &prefix)
  overloaded from AlgorithmStrategyObject

- virtual ESymSolverStatus Solve (const SymMatrix ∗W, double W_factor, const Vector ∗D_x, double delta_x, const Vector ∗D_s, double delta_s, const Matrix ∗J_c, const Vector ∗D_c, double delta_c, const Matrix ∗J_d, const Vector ∗D_d, double delta_d, const Vector &rhs_x, const Vector &rhs_s, const Vector &rhs_c, const Vector &rhs_d, Vector &sol_x, Vector &sol_s, Vector &sol_c, Vector &sol_d, bool check_NegEVals, Index numberOfNegEVals)

    Set up the augmented system and solve it for a given right hand side.
- virtual Index NumberOfNegEVals () const

    Number of negative eigenvalues detected during last solve.
- virtual bool ProvidesInertia () const

    Query whether inertia is computed by linear solver.
- virtual bool IncreaseQuality ()

    Request to increase quality of solution for next solve.

Constructors/Destructors

- LowRankSSAugSystemSolver (AugSystemSolver &aug_system_solver, Index max_rank)

    Constructor using an existing augmented system solver.
- virtual ∼LowRankSSAugSystemSolver ()

    Default destructor.

Additional Inherited Members

3.88.1 Detailed Description

Solver for the augmented system with LowRankUpdateSymMatrix Hessian matrices.

This version works with only one backsolve (so it is better for iterative linear solvers), by augmenting the regular augmented system.

Definition at line 27 of file IpLowRankSSAugSystemSolver.hpp.

3.88.2 Constructor & Destructor Documentation

3.88.2.1 Ipopt::LowRankSSAugSystemSolver::LowRankSSAugSystemSolver (AugSystemSolver & *aug_system_solver,* Index *max_rank*)

Constructor using an existing augmented system solver.

the max_rank argument is the maximal rank that can appear.

3.88.3 Member Function Documentation

3.88.3.1 virtual Index Ipopt::LowRankSSAugSystemSolver::NumberOfNegEVals ( ) const [virtual]

Number of negative eigenvalues detected during last solve.

Returns the number of negative eigenvalues of the most recent factorized matrix. This must not be called if the linear solver does not compute this quantities (see ProvidesInertia).

Implements Ipopt::AugSystemSolver.

3.88.3.2 virtual bool Ipopt::LowRankSSAugSystemSolver::ProvidesInertia ( ) const [virtual]

Query whether inertia is computed by linear solver.

Returns true, if linear solver provides inertia.

Implements Ipopt::AugSystemSolver.

3.88.3.3 virtual bool Ipopt::LowRankSSAugSystemSolver::IncreaseQuality ( ) [virtual]

Request to increase quality of solution for next solve.

Ask underlying linear solver to increase quality of solution for the next solve (e.g. increase pivot tolerance). Returns false, if this is not possible (e.g. maximal pivot tolerance already used.)

Implements Ipopt::AugSystemSolver.

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

- IpLowRankSSAugSystemSolver.hpp

3.89 Ipopt::LowRankUpdateSymMatrix Class Reference

Class for symmetric matrices, represented as low-rank updates.

`#include <IpLowRankUpdateSymMatrix.hpp>`

Inheritance diagram for Ipopt::LowRankUpdateSymMatrix:

Public Member Functions

- void SetDiag (const Vector &D)

  Method for setting the diagonal elements (as a Vector).
- SmartPtr< const Vector > GetDiag () const

  Method for getting the diagonal elements.
- void SetV (const MultiVectorMatrix &V)

  Method for setting the positive low-rank update part.
- SmartPtr< const MultiVectorMatrix > GetV () const

  Method for getting the positive low-rank update part.
- void SetU (const MultiVectorMatrix &U)

  Method for setting the negative low-rank update part.
- SmartPtr< const MultiVectorMatrix > GetU () const

  Method for getting the negative low-rank update part.
- SmartPtr< const Matrix > P_LowRank () const

  Return the expansion matrix to lift the low-rank update to the higher-dimensional space.
- SmartPtr< const VectorSpace > LowRankVectorSpace () const

  Return the vector space in with the low-rank update vectors live.
- bool ReducedDiag () const

  Flag indicating whether the diagonal term lives in the smaller space (from P_LowRank) or in the full space.

Constructors / Destructors

- LowRankUpdateSymMatrix (const LowRankUpdateSymMatrixSpace ∗owner_space)

    Constructor, given the corresponding matrix space.
- ∼LowRankUpdateSymMatrix ()

    Destructor.

Protected Member Functions

Methods overloaded from matrix

- virtual void MultVectorImpl (Number alpha, const Vector &x, Number beta, Vector &y) const

    Matrix-vector multiply.
- virtual bool HasValidNumbersImpl () const

    Method for determining if all stored numbers are valid (i.e., no Inf or Nan).
- virtual void ComputeRowAMaxImpl (Vector &rows_norms, bool init) const

    Compute the max-norm of the rows in the matrix.
- virtual void ComputeColAMaxImpl (Vector &cols_norms, bool init) const

    Since the matrix is symmetric, the row and column max norms are identical.
- virtual void PrintImpl (const Journalist &jnlst, EJournalLevel level, EJournalCategory category, const std::string &name, Index indent, const std::string &prefix) const

    Print detailed information about the matrix.

Additional Inherited Members

3.89.1 Detailed Description

Class for symmetric matrices, represented as low-rank updates.

The matrix M is represented as M = P_LR(D + V V^T - U U^T)P_LR^T (if reduced_diag is true), or M = D + P_LR(V V^T - U U^T)P_LR^T (if reduced_diag is false). D is a diagonal matrix, and V and U are MultiVectorMatrices, and P_LR is an ExpansionMatrix. The vectors in the low-rank update (before expansion) live in the LowRankVectorSpace. If P_LR is NULL, P_LR is assumed to be the identity matrix. If V or U is NULL, it is assume to be a matrix of zero columns.

Definition at line 31 of file IpLowRankUpdateSymMatrix.hpp.

3.89.2 Constructor & Destructor Documentation

3.89.2.1 Ipopt::LowRankUpdateSymMatrix::LowRankUpdateSymMatrix (const LowRankUpdateSymMatrixSpace ∗ *owner_space*)

Constructor, given the corresponding matrix space.

3.89.3 Member Function Documentation

3.89.3.1 void Ipopt::LowRankUpdateSymMatrix::SetDiag (const Vector & *D*) [inline]

Method for setting the diagonal elements (as a Vector).

Definition at line 46 of file IpLowRankUpdateSymMatrix.hpp.

3.89.3.2 SmartPtr<const Vector> Ipopt::LowRankUpdateSymMatrix::GetDiag ( ) const [inline]

Method for getting the diagonal elements.

Definition at line 53 of file IpLowRankUpdateSymMatrix.hpp.

3.89.3.3 void Ipopt::LowRankUpdateSymMatrix::SetV (const MultiVectorMatrix & *V*) [inline]

Method for setting the positive low-rank update part.

Definition at line 59 of file IpLowRankUpdateSymMatrix.hpp.

3.89.3.4 SmartPtr<const MultiVectorMatrix> Ipopt::LowRankUpdateSymMatrix::GetV ( ) const [inline]

Method for getting the positive low-rank update part.

Definition at line 66 of file IpLowRankUpdateSymMatrix.hpp.

3.89.3.5 void Ipopt::LowRankUpdateSymMatrix::SetU (const MultiVectorMatrix & *U*) [inline]

Method for setting the negative low-rank update part.

Definition at line 72 of file IpLowRankUpdateSymMatrix.hpp.

3.89.3.6 SmartPtr<const MultiVectorMatrix> Ipopt::LowRankUpdateSymMatrix::GetU ( ) const [inline]

Method for getting the negative low-rank update part.

Definition at line 79 of file IpLowRankUpdateSymMatrix.hpp.

3.89.3.7 SmartPtr< const Matrix > Ipopt::LowRankUpdateSymMatrix::P_LowRank ( ) const [inline]

Return the expansion matrix to lift the low-rank update to the higher-dimensional space.

Definition at line 237 of file IpLowRankUpdateSymMatrix.hpp.

3.89.3.8 SmartPtr< const VectorSpace > Ipopt::LowRankUpdateSymMatrix::LowRankVectorSpace ( ) const [inline]

Return the vector space in with the low-rank update vectors live.

Definition at line 243 of file IpLowRankUpdateSymMatrix.hpp.

3.89.3.9 bool Ipopt::LowRankUpdateSymMatrix::ReducedDiag ( ) const [inline]

Flag indicating whether the diagonal term lives in the smaller space (from P_LowRank) or in the full space.

Definition at line 249 of file IpLowRankUpdateSymMatrix.hpp.

3.89.3.10 virtual void Ipopt::LowRankUpdateSymMatrix::MultVectorImpl (Number *alpha*, const Vector & *x*, Number *beta*, Vector & *y*) const [protected], [virtual]

Matrix-vector multiply.

Computes y = alpha ∗ Matrix ∗ x + beta ∗ y

Implements Ipopt::Matrix.

3.89.3.11 virtual bool Ipopt::LowRankUpdateSymMatrix::HasValidNumbersImpl ( ) const [protected], [virtual]

Method for determining if all stored numbers are valid (i.e., no Inf or Nan).

Reimplemented from Ipopt::Matrix.

3.89.3.12 virtual void Ipopt::LowRankUpdateSymMatrix::ComputeRowAMaxImpl (Vector & *rows_norms*, bool *init*) const [protected], [virtual]

Compute the max-norm of the rows in the matrix.

The result is stored in rows_norms. The vector is assumed to be initialized.

Implements Ipopt::Matrix.

3.89.3.13 virtual void Ipopt::LowRankUpdateSymMatrix::PrintImpl (const Journalist & *jnlst*, EJournalLevel *level*, EJournalCategory *category*, const std::string & *name*, Index *indent*, const std::string & *prefix*) const [protected], [virtual]

Print detailed information about the matrix.

Implements Ipopt::Matrix.

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

- IpLowRankUpdateSymMatrix.hpp

3.90 Ipopt::LowRankUpdateSymMatrixSpace Class Reference

This is the matrix space for LowRankUpdateSymMatrix.

`#include <IpLowRankUpdateSymMatrix.hpp>`

Inheritance diagram for Ipopt::LowRankUpdateSymMatrixSpace:

Public Member Functions

- virtual SymMatrix ∗ MakeNewSymMatrix () const

    Overloaded MakeNew method for the SymMatrixSpace base class.
- LowRankUpdateSymMatrix ∗ MakeNewLowRankUpdateSymMatrix () const

    Method for creating a new matrix of this specific type.

Constructors / Destructors

- LowRankUpdateSymMatrixSpace (Index dim, SmartPtr< const Matrix > P_LowRank, SmartPtr< const VectorSpace > LowRankVectorSpace, bool reduced_diag)

    Constructor, given the dimension of the matrix.
- virtual ∼LowRankUpdateSymMatrixSpace ()

    Destructor.

3.90.1 Detailed Description

This is the matrix space for LowRankUpdateSymMatrix.

Definition at line 151 of file IpLowRankUpdateSymMatrix.hpp.

3.90.2 Constructor & Destructor Documentation

3.90.2.1 Ipopt::LowRankUpdateSymMatrixSpace::LowRankUpdateSymMatrixSpace (Index *dim*, SmartPtr< const Matrix > *P_LowRank*, SmartPtr< const VectorSpace > *LowRankVectorSpace*, bool *reduced_diag*) [inline]

Constructor, given the dimension of the matrix.

Definition at line 157 of file IpLowRankUpdateSymMatrix.hpp.

3.90.3 Member Function Documentation

3.90.3.1 LowRankUpdateSymMatrix* Ipopt::LowRankUpdateSymMatrixSpace::MakeNewLowRankUpdateSymMatrix ( ) const [inline]

Method for creating a new matrix of this specific type.

Definition at line 183 of file IpLowRankUpdateSymMatrix.hpp.

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

- IpLowRankUpdateSymMatrix.hpp

3.91 Ipopt::Ma27TSolverInterface Class Reference

Interface to the symmetric linear solver MA27, derived from SparseSymLinearSolverInterface.

```
#include <IpMa27TSolverInterface.hpp>
```

Inheritance diagram for Ipopt::Ma27TSolverInterface:

Public Member Functions

- bool InitializeImpl (const OptionsList &options, const std::string &prefix)

  overloaded from AlgorithmStrategyObject

Constructor/Destructor

- Ma27TSolverInterface ()

  Constructor.
- virtual ∼Ma27TSolverInterface ()

  Destructor.

Methods for requesting solution of the linear system.

- virtual ESymSolverStatus InitializeStructure (Index dim, Index nonzeros, const Index ∗airn, const Index ∗ajcn)

  Method for initializing internal stuctures.

- virtual double $*$ GetValuesArrayPtr ()

    Method returing an internal array into which the nonzero elements (in the same order as airn and ajcn) are to be stored by the calling routine before a call to MultiSolve with a new_matrix=true.
- virtual ESymSolverStatus MultiSolve (bool new_matrix, const Index $*$airn, const Index $*$ajcn, Index nrhs, double $*$rhs_vals, bool check_NegEVals, Index numberOfNegEVals)

    Solve operation for multiple right hand sides.
- virtual Index NumberOfNegEVals () const

    Number of negative eigenvalues detected during last factorization.

- virtual bool IncreaseQuality ()

    Request to increase quality of solution for next solve.
- virtual bool ProvidesInertia () const

    Query whether inertia is computed by linear solver.
- EMatrixFormat MatrixFormat () const

    Query of requested matrix type that the linear solver understands.

Static Public Member Functions

- static void RegisterOptions (SmartPtr$<$ RegisteredOptions $>$ roptions)

    Methods for IpoptType.

Additional Inherited Members

3.91.1 Detailed Description

Interface to the symmetric linear solver MA27, derived from SparseSymLinearSolverInterface.

Definition at line 19 of file IpMa27TSolverInterface.hpp.

3.91.2 Member Function Documentation

3.91.2.1 virtual ESymSolverStatus Ipopt::Ma27TSolverInterface::InitializeStructure (Index *dim*, Index *nonzeros*, const Index $*$ *airn*, const Index $*$ *ajcn*) [virtual]

Method for initializing internal stuctures.

Here, ndim gives the number of rows and columns of the matrix, nonzeros give the number of nonzero elements, and airn and acjn give the positions of the nonzero elements.

Implements Ipopt::SparseSymLinearSolverInterface.

3.91.2.2 virtual double$*$ Ipopt::Ma27TSolverInterface::GetValuesArrayPtr ( ) [virtual]

Method returing an internal array into which the nonzero elements (in the same order as airn and ajcn) are to be stored by the calling routine before a call to MultiSolve with a new_matrix=true.

The returned array must have space for at least nonzero elements.

Implements Ipopt::SparseSymLinearSolverInterface.

3.91.2.3 virtual ESymSolverStatus Ipopt::Ma27TSolverInterface::MultiSolve (bool *new_matrix*, const Index * *airn*, const Index * *ajcn*, Index *nrhs*, double * *rhs_vals*, bool *check_NegEVals*, Index *numberOfNegEVals*) [virtual]

Solve operation for multiple right hand sides.

Overloaded from SparseSymLinearSolverInterface.

Implements Ipopt::SparseSymLinearSolverInterface.

3.91.2.4 virtual Index Ipopt::Ma27TSolverInterface::NumberOfNegEVals ( ) const [virtual]

Number of negative eigenvalues detected during last factorization.

Returns the number of negative eigenvalues of the most recent factorized matrix. This must not be called if the linear solver does not compute this quantities (see ProvidesInertia).

Implements Ipopt::SparseSymLinearSolverInterface.

3.91.2.5 virtual bool Ipopt::Ma27TSolverInterface::IncreaseQuality ( ) [virtual]

Request to increase quality of solution for next solve.

Ask linear solver to increase quality of solution for the next solve (e.g. increase pivot tolerance). Returns false, if this is not possible (e.g. maximal pivot tolerance already used.)

Implements Ipopt::SparseSymLinearSolverInterface.

3.91.2.6 virtual bool Ipopt::Ma27TSolverInterface::ProvidesInertia ( ) const [inline], [virtual]

Query whether inertia is computed by linear solver.

Returns true, if linear solver provides inertia.

Implements Ipopt::SparseSymLinearSolverInterface.

Definition at line 86 of file IpMa27TSolverInterface.hpp.

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

- IpMa27TSolverInterface.hpp

3.92 Ipopt::Ma28TDependencyDetector Class Reference

Base class for all derived algorithms for detecting linearly dependent rows in the constraint Jacobian.

`#include <IpMa28TDependencyDetector.hpp>`

Inheritance diagram for Ipopt::Ma28TDependencyDetector:

Public Member Functions

- virtual bool InitializeImpl (const OptionsList &options, const std::string &prefix)

  Has to be called to initialize and reset these objects.
- virtual bool DetermineDependentRows (Index n_rows, Index n_cols, Index n_jac_nz, Number *jac_c_vals, Index *jac_c_iRow, Index *jac_c_jCol, std::list< Index > &c_deps)

Method determining the number of linearly dependent rows in the matrix and the indices of those rows.

Constructor/Destructor

- **Ma28TDependencyDetector** ()
- virtual ∼**Ma28TDependencyDetector** ()

Static Public Member Functions

- static void RegisterOptions (SmartPtr< RegisteredOptions > roptions)

  This must be called to make the options for this class known.

Additional Inherited Members

3.92.1 Detailed Description

Base class for all derived algorithms for detecting linearly dependent rows in the constraint Jacobian.

Definition at line 19 of file IpMa28TDependencyDetector.hpp.

3.92.2 Member Function Documentation

3.92.2.1 virtual bool Ipopt::Ma28TDependencyDetector::InitializeImpl (const OptionsList & *options,* const std::string & *prefix*) [virtual]

Has to be called to initialize and reset these objects.

Implements Ipopt::TDependencyDetector.

3.92.2.2 virtual bool Ipopt::Ma28TDependencyDetector::DetermineDependentRows (Index *n_rows,* Index *n_cols,* Index *n_jac_nz,* Number ∗ *jac_c_vals,* Index ∗ *jac_c_iRow,* Index ∗ *jac_c_jCol,* std::list< Index > & *c_deps*) [virtual]

Method determining the number of linearly dependent rows in the matrix and the indices of those rows.

We assume that the matrix is available in "Triplet" format (MA28 format), and that the arrays given to this method can be modified internally, i.e., they are not used by the calling program anymore after this call. This method returns false if there was a problem with the underlying linear solver.

Implements Ipopt::TDependencyDetector.

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

- IpMa28TDependencyDetector.hpp

3.93 Ipopt::Ma57TSolverInterface Class Reference

Interface to the symmetric linear solver MA57, derived from SparseSymLinearSolverInterface.

`#include <IpMa57TSolverInterface.hpp>`

Inheritance diagram for Ipopt::Ma57TSolverInterface:

Public Member Functions

- bool InitializeImpl (const OptionsList &options, const std::string &prefix)

    overloaded from AlgorithmStrategyObject

Constructor/Destructor

- Ma57TSolverInterface ()

    Constructor.
- virtual ∼Ma57TSolverInterface ()

    Destructor.

Methods for requesting solution of the linear system.

- virtual ESymSolverStatus InitializeStructure (Index dim, Index nonzeros, const Index ∗airn, const Index ∗ajcn)

    Method for initializing internal stuctures.
- virtual double ∗ GetValuesArrayPtr ()

    Method returing an internal array into which the nonzero elements (in the same order as airn and ajcn) are to be stored by the calling routine before a call to MultiSolve with a new_matrix=true.
- virtual ESymSolverStatus MultiSolve (bool new_matrix, const Index ∗airn, const Index ∗ajcn, Index nrhs, double ∗rhs_vals, bool check_NegEVals, Index numberOfNegEVals)

    Solve operation for multiple right hand sides.
- virtual Index NumberOfNegEVals () const

    Number of negative eigenvalues detected during last factorization.

- virtual bool IncreaseQuality ()

    Request to increase quality of solution for next solve.
- virtual bool ProvidesInertia () const

    Query whether inertia is computed by linear solver.
- EMatrixFormat MatrixFormat () const

    Query of requested matrix type that the linear solver understands.

Static Public Member Functions

- static void RegisterOptions (SmartPtr< RegisteredOptions > roptions)

    Methods for IpoptType.

Additional Inherited Members

3.93.1 Detailed Description

Interface to the symmetric linear solver MA57, derived from SparseSymLinearSolverInterface.

Definition at line 27 of file IpMa57TSolverInterface.hpp.

3.93.2 Member Function Documentation

3.93.2.1 virtual ESymSolverStatus Ipopt::Ma57TSolverInterface::InitializeStructure (Index *dim*, Index *nonzeros*, const Index * *airn*, const Index * *ajcn*) [virtual]

Method for initializing internal stuctures.

Here, ndim gives the number of rows and columns of the matrix, nonzeros give the number of nonzero elements, and airn and acjn give the positions of the nonzero elements.

Implements Ipopt::SparseSymLinearSolverInterface.

3.93.2.2 virtual double* Ipopt::Ma57TSolverInterface::GetValuesArrayPtr ( ) [virtual]

Method returing an internal array into which the nonzero elements (in the same order as airn and ajcn) are to be stored by the calling routine before a call to MultiSolve with a new_matrix=true.

The returned array must have space for at least nonzero elements.

Implements Ipopt::SparseSymLinearSolverInterface.

3.93.2.3 virtual ESymSolverStatus Ipopt::Ma57TSolverInterface::MultiSolve (bool *new_matrix*, const Index * *airn*, const Index * *ajcn*, Index *nrhs*, double * *rhs_vals*, bool *check_NegEVals*, Index *numberOfNegEVals*) [virtual]

Solve operation for multiple right hand sides.

Overloaded from SparseSymLinearSolverInterface.

Implements Ipopt::SparseSymLinearSolverInterface.

3.93.2.4 virtual Index Ipopt::Ma57TSolverInterface::NumberOfNegEVals ( ) const [virtual]

Number of negative eigenvalues detected during last factorization.

Returns the number of negative eigenvalues of the most recent factorized matrix. This must not be called if the linear solver does not compute this quantities (see ProvidesInertia).

Implements Ipopt::SparseSymLinearSolverInterface.

3.93.2.5 virtual bool Ipopt::Ma57TSolverInterface::IncreaseQuality ( ) [virtual]

Request to increase quality of solution for next solve.

Ask linear solver to increase quality of solution for the next solve (e.g. increase pivot tolerance). Returns false, if this is not possible (e.g. maximal pivot tolerance already used.)

Implements Ipopt::SparseSymLinearSolverInterface.

3.93.2.6 virtual bool Ipopt::Ma57TSolverInterface::ProvidesInertia ( ) const [inline], [virtual]

Query whether inertia is computed by linear solver.

Returns true, if linear solver provides inertia.

Implements Ipopt::SparseSymLinearSolverInterface.

Definition at line 96 of file IpMa57TSolverInterface.hpp.

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

- IpMa57TSolverInterface.hpp

3.94 ma77_control_d Struct Reference

3.94.1 Detailed Description

Definition at line 38 of file hsl_ma77d.h.

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

- hsl_ma77d.h

3.95 ma77_info_d Struct Reference

3.95.1 Detailed Description

Definition at line 98 of file hsl_ma77d.h.

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

- hsl_ma77d.h

3.96 Ipopt::Ma77SolverInterface Class Reference

Base class for interfaces to symmetric indefinite linear solvers for sparse matrices.

`#include <IpMa77SolverInterface.hpp>`

Inheritance diagram for Ipopt::Ma77SolverInterface:

Public Member Functions

- bool InitializeImpl (const OptionsList &options, const std::string &prefix)

  overloaded from AlgorithmStrategyObject

Methods for requesting solution of the linear system.

- ESymSolverStatus InitializeStructure (Index dim, Index nonzeros, const Index ∗ia, const Index ∗ja)

  Method for initializing internal stuctures.
- double ∗ GetValuesArrayPtr ()

  Method returing an internal array into which the nonzero elements (in the same order as ja) will be stored by the calling routine before a call to MultiSolve with a new_matrix=true (or after a return of MultiSolve with SYMSOLV_CALL_AG←AIN).
- ESymSolverStatus MultiSolve (bool new_matrix, const Index ∗ia, const Index ∗ja, Index nrhs, double ∗rhs_vals, bool check_NegEVals, Index numberOfNegEVals)

  Solve operation for multiple right hand sides.
- Index NumberOfNegEVals () const

Number of negative eigenvalues detected during last factorization.

- bool IncreaseQuality ()

    Request to increase quality of solution for next solve.
- bool ProvidesInertia () const

    Query whether inertia is computed by linear solver.
- EMatrixFormat MatrixFormat () const

    Query of requested matrix type that the linear solver understands.

Methods related to the detection of linearly dependent

rows in a matrix

- bool ProvidesDegeneracyDetection () const

    Query whether the indices of linearly dependent rows/columns can be determined by this linear solver.
- ESymSolverStatus DetermineDependentRows (const Index ∗ia, const Index ∗ja, std::list< Index > &c_deps)

    This method determines the list of row indices of the linearly dependent rows.

Additional Inherited Members

3.96.1 Detailed Description

Base class for interfaces to symmetric indefinite linear solvers for sparse matrices.

This defines the general interface to linear solvers for sparse symmetric indefinite matrices. The matrices can be provided either in "triplet format" (like for Harwell's MA27 solver), or in compressed sparse row (CSR) format for the lower triangular part of the symmetric matrix.

The solver should be able to compute the interia of the matrix, or more specifically, the number of negative eigenvalues in the factorized matrix.

This interface is used by the calling objective in the following way:

1. The InitializeImpl method is called at the very beginning (for every optimization run), which allows the linear solver object to retrieve options given in the OptionsList (such as pivot tolerances etc). At this point, some internal data can also be initialized.

2. The calling class calls MatrixFormat to find out which matrix representation the linear solver requires. The possible options are Triplet_Format, as well as CSR_Format_0_Offset and CSR_Format_1_Offset. The difference between the last two is that for CSR_Format_0_Offset the couning of the element position in the ia and ja arrays starts are 0 (C-style numbering), whereas for the other one it starts at 1 (Fortran-style numbering).

3. After this, the InitializeStructure method is called (once). Here, the structure of the matrix is provided. If the linear solver requires a symbolic preprocessing phase that can be done without knowledge of the matrix element values, it can be done here.

4. The calling class will request an array for storing the actual values for a matrix using the GetValuesArrayPtr method. This array must be at least as large as the number of nonzeros in the matrix (as given to this class by the InitializeStructure method call). After a call of this method, the calling class will fill this array with the actual values of the matrix.

5. Every time lateron, when actual solves of a linear system is requested, the calling class will call the MultiSolve to request the solve, possibly for mulitple right-hand sides. The flag new_matrix then indicates if the values of the matrix have changed and if a factorization is required, or if an old factorization can be used to do the solve.

Note that the GetValuesArrayPtr method will be called before every call of MultiSolve with new_matrix=true, or before a renewed call of MultiSolve if the most previous return value was SYMSOLV_CALL_AGAIN.

1. The calling class might request with NumberOfNegEVals the number of the negative eigenvalues for the original matrix that were detected during the most recently performed factorization.
2. The calling class might ask the linear solver to increase the quality of the solution. For example, if the linear solver uses a pivot tolerance, a larger value should be used for the next solve (which might require a refactorization).
3. Finally, when the destructor is called, the internal storage, also in the linear solver, should be released.

Note, if the matrix is given in triplet format, entries might be listed multiple times, in which case the corresponsing elements have to be added.

A note for warm starts: If the option "warm_start_same_structure" is specified with "yes", the algorithm assumes that a problem with the same sparsity structure is solved for a repeated time. In that case, the linear solver might reuse information from the previous optimization. See Ma27TSolverInterface for an example.

Definition at line 103 of file IpMa77SolverInterface.hpp.

3.96.2 Member Function Documentation

3.96.2.1 ESymSolverStatus Ipopt::Ma77SolverInterface::InitializeStructure (Index *dim,* Index *nonzeros,* const Index ∗ *ia,* const Index ∗ *ja*) [virtual]

Method for initializing internal stuctures.

Here, ndim gives the number of rows and columns of the matrix, nonzeros give the number of nonzero elements, and ia and ja give the positions of the nonzero elements, given in the matrix format determined by MatrixFormat.

Implements Ipopt::SparseSymLinearSolverInterface.

3.96.2.2 double∗ Ipopt::Ma77SolverInterface::GetValuesArrayPtr ( ) [inline], [virtual]

Method returing an internal array into which the nonzero elements (in the same order as ja) will be stored by the calling routine before a call to MultiSolve with a new_matrix=true (or after a return of MultiSolve with SYMSOLV_CALL_AGAIN).

The returned array must have space for at least nonzero elements.

Implements Ipopt::SparseSymLinearSolverInterface.

Definition at line 152 of file IpMa77SolverInterface.hpp.

3.96.2.3 ESymSolverStatus Ipopt::Ma77SolverInterface::MultiSolve (bool *new_matrix,* const Index ∗ *ia,* const Index ∗ *ja,* Index *nrhs,* double ∗ *rhs_vals,* bool *check_NegEVals,* Index *numberOfNegEVals*) [virtual]

Solve operation for multiple right hand sides.

Solves the linear system A ∗ x = b with multiple right hand sides, where A is the symmtric indefinite matrix. Here, ia and ja give the positions of the values (in the required matrix data format). The actual values of the matrix will have been given to this object by copying them into the array provided by GetValuesArrayPtr. ia and ja are identical to the ones given to InitializeStructure. The flag new_matrix is set to true, if the values of the matrix has changed, and a refactorzation is required.

The return code is SYMSOLV_SUCCESS if the factorization and solves were successful, SYMSOLV_SINGULAR if the linear system is singular, and SYMSOLV_WRONG_INERTIA if check_NegEVals is true and the number of negative eigenvalues in the matrix does not match numberOfNegEVals. If SYMSOLV_CALL_AGAIN is returned, then the calling function will request the pointer for the array for storing a again (with GetValuesPtr), write the values of the nonzero elements into it, and call this MultiSolve method again with the same right-hand sides. (This can be done, for example, if the linear solver realized it does not have sufficient memory and needs to redo the factorization; e.g., for MA27.)

The number of right-hand sides is given by nrhs, the values of the right-hand sides are given in rhs_vals (one full right-hand side stored immediately after the other), and solutions are to be returned in the same array.

check_NegEVals will not be chosen true, if ProvidesInertia() returns false.

Implements Ipopt::SparseSymLinearSolverInterface.

3.96.2.4 Index Ipopt::Ma77SolverInterface::NumberOfNegEVals ( ) const [inline], [virtual]

Number of negative eigenvalues detected during last factorization.

Returns the number of negative eigenvalues of the most recent factorized matrix. This must not be called if the linear solver does not compute this quantities (see ProvidesInertia).

Implements Ipopt::SparseSymLinearSolverInterface.

Definition at line 203 of file IpMa77SolverInterface.hpp.

3.96.2.5 bool Ipopt::Ma77SolverInterface::IncreaseQuality ( ) [virtual]

Request to increase quality of solution for next solve.

The calling class asks linear solver to increase quality of solution for the next solve (e.g. increase pivot tolerance). Returns false, if this is not possible (e.g. maximal pivot tolerance already used.)

Implements Ipopt::SparseSymLinearSolverInterface.

3.96.2.6 bool Ipopt::Ma77SolverInterface::ProvidesInertia ( ) const [inline], [virtual]

Query whether inertia is computed by linear solver.

Returns true, if linear solver provides inertia.

Implements Ipopt::SparseSymLinearSolverInterface.

Definition at line 222 of file IpMa77SolverInterface.hpp.

3.96.2.7 bool Ipopt::Ma77SolverInterface::ProvidesDegeneracyDetection ( ) const [inline], [virtual]

Query whether the indices of linearly dependent rows/columns can be determined by this linear solver.

Reimplemented from Ipopt::SparseSymLinearSolverInterface.

Definition at line 241 of file IpMa77SolverInterface.hpp.

3.96.2.8 ESymSolverStatus Ipopt::Ma77SolverInterface::DetermineDependentRows (const Index * *ia,* const Index * *ja,* std::list< Index > & *c_deps*) [inline], [virtual]

This method determines the list of row indices of the linearly dependent rows.

Reimplemented from Ipopt::SparseSymLinearSolverInterface.

Definition at line 247 of file IpMa77SolverInterface.hpp.

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

- IpMa77SolverInterface.hpp

3.97 ma86_control_d Struct Reference

3.97.1 Detailed Description

Definition at line 27 of file hsl_ma86d.h.

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

- hsl_ma86d.h

3.98 ma86_info_d Struct Reference

3.98.1 Detailed Description

Definition at line 70 of file hsl_ma86d.h.

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

- hsl_ma86d.h

3.99 Ipopt::Ma86SolverInterface Class Reference

Base class for interfaces to symmetric indefinite linear solvers for sparse matrices.

`#include <IpMa86SolverInterface.hpp>`

Inheritance diagram for Ipopt::Ma86SolverInterface:

Public Member Functions

- bool InitializeImpl (const OptionsList &options, const std::string &prefix)

  overloaded from AlgorithmStrategyObject

Methods for requesting solution of the linear system.

- ESymSolverStatus InitializeStructure (Index dim, Index nonzeros, const Index ∗ia, const Index ∗ja)

  Method for initializing internal stuctures.
- double ∗ GetValuesArrayPtr ()

  Method returning an internal array into which the nonzero elements (in the same order as ja) will be stored by the calling routine before a call to MultiSolve with a new_matrix=true (or after a return of MultiSolve with SYMSOLV_CA↩ LL_AGAIN).
- ESymSolverStatus MultiSolve (bool new_matrix, const Index ∗ia, const Index ∗ja, Index nrhs, double ∗rhs_vals, bool check_NegEVals, Index numberOfNegEVals)

  Solve operation for multiple right hand sides.
- Index NumberOfNegEVals () const

  Number of negative eigenvalues detected during last factorization.

- bool IncreaseQuality ()

Request to increase quality of solution for next solve.
- bool ProvidesInertia () const

  Query whether inertia is computed by linear solver.
- EMatrixFormat MatrixFormat () const

  Query of requested matrix type that the linear solver understands.

Methods related to the detection of linearly dependent

rows in a matrix

- bool ProvidesDegeneracyDetection () const

  Query whether the indices of linearly dependent rows/columns can be determined by this linear solver.
- ESymSolverStatus DetermineDependentRows (const Index ∗ia, const Index ∗ja, std::list< Index > &c_deps)

  This method determines the list of row indices of the linearly dependent rows.

Additional Inherited Members

3.99.1 Detailed Description

Base class for interfaces to symmetric indefinite linear solvers for sparse matrices.

This defines the general interface to linear solvers for sparse symmetric indefinite matrices. The matrices can be provided either in "triplet format" (like for Harwell's MA27 solver), or in compressed sparse row (CSR) format for the lower triangular part of the symmetric matrix.

The solver should be able to compute the interia of the matrix, or more specifically, the number of negative eigenvalues in the factorized matrix.

This interface is used by the calling objective in the following way:

1. The InitializeImpl method is called at the very beginning (for every optimization run), which allows the linear solver object to retrieve options given in the OptionsList (such as pivot tolerances etc). At this point, some internal data can also be initialized.

2. The calling class calls MatrixFormat to find out which matrix representation the linear solver requires. The possible options are Triplet_Format, as well as CSR_Format_0_Offset and CSR_Format_1_Offset. The difference between the last two is that for CSR_Format_0_Offset the couning of the element position in the ia and ja arrays starts are 0 (C-style numbering), whereas for the other one it starts at 1 (Fortran-style numbering).

3. After this, the InitializeStructure method is called (once). Here, the structure of the matrix is provided. If the linear solver requires a symbolic preprocessing phase that can be done without knowledge of the matrix element values, it can be done here.

4. The calling class will request an array for storing the actual values for a matrix using the GetValuesArrayPtr method. This array must be at least as large as the number of nonzeros in the matrix (as given to this class by the InitializeStructure method call). After a call of this method, the calling class will fill this array with the actual values of the matrix.

5. Every time lateron, when actual solves of a linear system is requested, the calling class will call the MultiSolve to request the solve, possibly for mulitple right-hand sides. The flag new_matrix then indicates if the values of the matrix have changed and if a factorization is required, or if an old factorization can be used to do the solve.

Note that the GetValuesArrayPtr method will be called before every call of MultiSolve with new_matrix=true, or before a renewed call of MultiSolve if the most previous return value was SYMSOLV_CALL_AGAIN.

1. The calling class might request with NumberOfNegEVals the number of the negative eigenvalues for the original matrix that were detected during the most recently performed factorization.
2. The calling class might ask the linear solver to increase the quality of the solution. For example, if the linear solver uses a pivot tolerance, a larger value should be used for the next solve (which might require a refactorization).
3. Finally, when the destructor is called, the internal storage, also in the linear solver, should be released.

Note, if the matrix is given in triplet format, entries might be listed multiple times, in which case the corresponsing elements have to be added.

A note for warm starts: If the option "warm_start_same_structure" is specified with "yes", the algorithm assumes that a problem with the same sparsity structure is solved for a repeated time. In that case, the linear solver might reuse information from the previous optimization. See Ma27TSolverInterface for an example.

Definition at line 104 of file IpMa86SolverInterface.hpp.

3.99.2 Member Function Documentation

3.99.2.1 ESymSolverStatus Ipopt::Ma86SolverInterface::InitializeStructure (Index *dim*, Index *nonzeros*, const Index ∗ *ia*, const Index ∗ *ja*) [virtual]

Method for initializing internal stuctures.

Here, ndim gives the number of rows and columns of the matrix, nonzeros give the number of nonzero elements, and ia and ja give the positions of the nonzero elements, given in the matrix format determined by MatrixFormat.

Implements Ipopt::SparseSymLinearSolverInterface.

3.99.2.2 double∗ Ipopt::Ma86SolverInterface::GetValuesArrayPtr ( ) [inline], [virtual]

Method returning an internal array into which the nonzero elements (in the same order as ja) will be stored by the calling routine before a call to MultiSolve with a new_matrix=true (or after a return of MultiSolve with SYMSOLV_CALL_AGAIN).

The returned array must have space for at least nonzero elements.

Implements Ipopt::SparseSymLinearSolverInterface.

Definition at line 155 of file IpMa86SolverInterface.hpp.

3.99.2.3 ESymSolverStatus Ipopt::Ma86SolverInterface::MultiSolve (bool *new_matrix*, const Index ∗ *ia*, const Index ∗ *ja*, Index *nrhs*, double ∗ *rhs_vals*, bool *check_NegEVals*, Index *numberOfNegEVals*) [virtual]

Solve operation for multiple right hand sides.

Solves the linear system A ∗ x = b with multiple right hand sides, where A is the symmtric indefinite matrix. Here, ia and ja give the positions of the values (in the required matrix data format). The actual values of the matrix will have been given to this object by copying them into the array provided by GetValuesArrayPtr. ia and ja are identical to the ones given to InitializeStructure. The flag new_matrix is set to true, if the values of the matrix has changed, and a refactorzation is required.

The return code is SYMSOLV_SUCCESS if the factorization and solves were successful, SYMSOLV_SINGULAR if the linear system is singular, and SYMSOLV_WRONG_INERTIA if check_NegEVals is true and the number of negative eigenvalues in the matrix does not match numberOfNegEVals. If SYMSOLV_CALL_AGAIN is returned, then the calling function will request the pointer for the array for storing a again (with GetValuesPtr), write the values of the nonzero elements into it, and call this MultiSolve method again with the same right-hand sides. (This can be done, for example, if the linear solver realized it does not have sufficient memory and needs to redo the factorization; e.g., for MA27.)

The number of right-hand sides is given by nrhs, the values of the right-hand sides are given in rhs_vals (one full right-hand side stored immediately after the other), and solutions are to be returned in the same array.

check_NegEVals will not be chosen true, if ProvidesInertia() returns false.

Implements Ipopt::SparseSymLinearSolverInterface.

3.99.2.4 Index Ipopt::Ma86SolverInterface::NumberOfNegEVals ( ) const [inline], [virtual]

Number of negative eigenvalues detected during last factorization.

Returns the number of negative eigenvalues of the most recent factorized matrix. This must not be called if the linear solver does not compute this quantities (see ProvidesInertia).

Implements Ipopt::SparseSymLinearSolverInterface.

Definition at line 206 of file IpMa86SolverInterface.hpp.

3.99.2.5 bool Ipopt::Ma86SolverInterface::IncreaseQuality ( ) [virtual]

Request to increase quality of solution for next solve.

The calling class asks linear solver to increase quality of solution for the next solve (e.g. increase pivot tolerance). Returns false, if this is not possible (e.g. maximal pivot tolerance already used.)

Implements Ipopt::SparseSymLinearSolverInterface.

3.99.2.6 bool Ipopt::Ma86SolverInterface::ProvidesInertia ( ) const [inline], [virtual]

Query whether inertia is computed by linear solver.

Returns true, if linear solver provides inertia.

Implements Ipopt::SparseSymLinearSolverInterface.

Definition at line 225 of file IpMa86SolverInterface.hpp.

3.99.2.7 bool Ipopt::Ma86SolverInterface::ProvidesDegeneracyDetection ( ) const [inline], [virtual]

Query whether the indices of linearly dependent rows/columns can be determined by this linear solver.

Reimplemented from Ipopt::SparseSymLinearSolverInterface.

Definition at line 244 of file IpMa86SolverInterface.hpp.

3.99.2.8 ESymSolverStatus Ipopt::Ma86SolverInterface::DetermineDependentRows (const Index ∗ *ia,* const Index ∗ *ja,* std::list< Index > & *c_deps*) [inline], [virtual]

This method determines the list of row indices of the linearly dependent rows.

Reimplemented from Ipopt::SparseSymLinearSolverInterface.

Definition at line 250 of file IpMa86SolverInterface.hpp.

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

- IpMa86SolverInterface.hpp

3.100 ma97_control_d Struct Reference

3.100.1 Detailed Description

Definition at line 35 of file hsl_ma97d.h.

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

- hsl_ma97d.h

3.101 ma97_info Struct Reference

3.101.1 Detailed Description

Definition at line 66 of file hsl_ma97d.h.

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

- hsl_ma97d.h

3.102 Ipopt::Ma97SolverInterface Class Reference

Base class for interfaces to symmetric indefinite linear solvers for sparse matrices.

`#include <IpMa97SolverInterface.hpp>`

Inheritance diagram for Ipopt::Ma97SolverInterface:

Public Member Functions

- bool InitializeImpl (const OptionsList &options, const std::string &prefix)
  overloaded from AlgorithmStrategyObject

Methods for requesting solution of the linear system.

- ESymSolverStatus InitializeStructure (Index dim, Index nonzeros, const Index ∗ia, const Index ∗ja)
  Method for initializing internal stuctures.
- double ∗ GetValuesArrayPtr ()
  Method returing an internal array into which the nonzero elements (in the same order as ja) will be stored by the calling routine before a call to MultiSolve with a new_matrix=true (or after a return of MultiSolve with SYMSOLV_CALL_AG←AIN).
- ESymSolverStatus MultiSolve (bool new_matrix, const Index ∗ia, const Index ∗ja, Index nrhs, double ∗rhs_vals, bool check_NegEVals, Index numberOfNegEVals)
  Solve operation for multiple right hand sides.
- Index NumberOfNegEVals () const
  Number of negative eigenvalues detected during last factorization.

- bool IncreaseQuality ()
  Request to increase quality of solution for next solve.
- bool ProvidesInertia () const

Query whether inertia is computed by linear solver.

- EMatrixFormat MatrixFormat () const

  Query of requested matrix type that the linear solver understands.

Methods related to the detection of linearly dependent

rows in a matrix

- bool ProvidesDegeneracyDetection () const

  Query whether the indices of linearly dependent rows/columns can be determined by this linear solver.
- ESymSolverStatus DetermineDependentRows (const Index ∗ia, const Index ∗ja, std::list< Index > &c_deps)

  This method determines the list of row indices of the linearly dependent rows.
- static int ScaleNameToNum (const std::string &name)

  converts a scalign optoin name to its ma97 option number

Additional Inherited Members

3.102.1 Detailed Description

Base class for interfaces to symmetric indefinite linear solvers for sparse matrices.

This defines the general interface to linear solvers for sparse symmetric indefinite matrices. The matrices can be provided either in "triplet format" (like for Harwell's MA27 solver), or in compressed sparse row (CSR) format for the lower triangular part of the symmetric matrix.

The solver should be able to compute the interia of the matrix, or more specifically, the number of negative eigenvalues in the factorized matrix.

This interface is used by the calling objective in the following way:

1. The InitializeImpl method is called at the very beginning (for every optimization run), which allows the linear solver object to retrieve options given in the OptionsList (such as pivot tolerances etc). At this point, some internal data can also be initialized.

2. The calling class calls MatrixFormat to find out which matrix representation the linear solver requires. The possible options are Triplet_Format, as well as CSR_Format_0_Offset and CSR_Format_1_Offset. The difference between the last two is that for CSR_Format_0_Offset the couning of the element position in the ia and ja arrays starts are 0 (C-style numbering), whereas for the other one it starts at 1 (Fortran-style numbering).

3. After this, the InitializeStructure method is called (once). Here, the structure of the matrix is provided. If the linear solver requires a symbolic preprocessing phase that can be done without knowledge of the matrix element values, it can be done here.

4. The calling class will request an array for storing the actual values for a matrix using the GetValuesArrayPtr method. This array must be at least as large as the number of nonzeros in the matrix (as given to this class by the InitializeStructure method call). After a call of this method, the calling class will fill this array with the actual values of the matrix.

5. Every time lateron, when actual solves of a linear system is requested, the calling class will call the MultiSolve to request the solve, possibly for mulitple right-hand sides. The flag new_matrix then indicates if the values of the matrix have changed and if a factorization is required, or if an old factorization can be used to do the solve.

Note that the GetValuesArrayPtr method will be called before every call of MultiSolve with new_matrix=true, or before a renewed call of MultiSolve if the most previous return value was SYMSOLV_CALL_AGAIN.

1. The calling class might request with NumberOfNegEVals the number of the negative eigenvalues for the original matrix that were detected during the most recently performed factorization.
2. The calling class might ask the linear solver to increase the quality of the solution. For example, if the linear solver uses a pivot tolerance, a larger value should be used for the next solve (which might require a refactorization).
3. Finally, when the destructor is called, the internal storage, also in the linear solver, should be released.

Note, if the matrix is given in triplet format, entries might be listed multiple times, in which case the corresponsing elements have to be added.

A note for warm starts: If the option "warm_start_same_structure" is specified with "yes", the algorithm assumes that a problem with the same sparsity structure is solved for a repeated time. In that case, the linear solver might reuse information from the previous optimization. See Ma27TSolverInterface for an example.

Definition at line 104 of file IpMa97SolverInterface.hpp.

3.102.2 Member Function Documentation

3.102.2.1 ESymSolverStatus Ipopt::Ma97SolverInterface::InitializeStructure (Index *dim,* Index *nonzeros,* const Index ∗ *ia,* const Index ∗ *ja*) [virtual]

Method for initializing internal stuctures.

Here, ndim gives the number of rows and columns of the matrix, nonzeros give the number of nonzero elements, and ia and ja give the positions of the nonzero elements, given in the matrix format determined by MatrixFormat.

Implements Ipopt::SparseSymLinearSolverInterface.

3.102.2.2 double∗ Ipopt::Ma97SolverInterface::GetValuesArrayPtr ( ) [inline], [virtual]

Method returing an internal array into which the nonzero elements (in the same order as ja) will be stored by the calling routine before a call to MultiSolve with a new_matrix=true (or after a return of MultiSolve with SYMSOLV_CALL_AGAIN).

The returned array must have space for at least nonzero elements.

Implements Ipopt::SparseSymLinearSolverInterface.

Definition at line 181 of file IpMa97SolverInterface.hpp.

3.102.2.3 ESymSolverStatus Ipopt::Ma97SolverInterface::MultiSolve (bool *new_matrix,* const Index ∗ *ia,* const Index ∗ *ja,* Index *nrhs,* double ∗ *rhs_vals,* bool *check_NegEVals,* Index *numberOfNegEVals*) [virtual]

Solve operation for multiple right hand sides.

Solves the linear system A ∗ x = b with multiple right hand sides, where A is the symmtric indefinite matrix. Here, ia and ja give the positions of the values (in the required matrix data format). The actual values of the matrix will have been given to this object by copying them into the array provided by GetValuesArrayPtr. ia and ja are identical to the ones given to InitializeStructure. The flag new_matrix is set to true, if the values of the matrix has changed, and a refactorzation is required.

The return code is SYMSOLV_SUCCESS if the factorization and solves were successful, SYMSOLV_SINGULAR if the linear system is singular, and SYMSOLV_WRONG_INERTIA if check_NegEVals is true and the number of negative eigenvalues in the matrix does not match numberOfNegEVals. If SYMSOLV_CALL_AGAIN is returned, then the calling function will request the pointer for the array for storing a again (with GetValuesPtr), write the values of the nonzero elements into it, and call this MultiSolve method again with the same right-hand sides. (This can be done, for example, if the linear solver realized it does not have sufficient memory and needs to redo the factorization; e.g., for MA27.)

The number of right-hand sides is given by nrhs, the values of the right-hand sides are given in rhs_vals (one full right-hand side stored immediately after the other), and solutions are to be returned in the same array.

check_NegEVals will not be chosen true, if ProvidesInertia() returns false.

Implements Ipopt::SparseSymLinearSolverInterface.

3.102.2.4 Index Ipopt::Ma97SolverInterface::NumberOfNegEVals ( ) const [inline], [virtual]

Number of negative eigenvalues detected during last factorization.

Returns the number of negative eigenvalues of the most recent factorized matrix. This must not be called if the linear solver does not compute this quantities (see ProvidesInertia).

Implements Ipopt::SparseSymLinearSolverInterface.

Definition at line 232 of file IpMa97SolverInterface.hpp.

3.102.2.5 bool Ipopt::Ma97SolverInterface::IncreaseQuality ( ) [virtual]

Request to increase quality of solution for next solve.

The calling class asks linear solver to increase quality of solution for the next solve (e.g. increase pivot tolerance). Returns false, if this is not possible (e.g. maximal pivot tolerance already used.)

Implements Ipopt::SparseSymLinearSolverInterface.

3.102.2.6 bool Ipopt::Ma97SolverInterface::ProvidesInertia ( ) const [inline], [virtual]

Query whether inertia is computed by linear solver.

Returns true, if linear solver provides inertia.

Implements Ipopt::SparseSymLinearSolverInterface.

Definition at line 251 of file IpMa97SolverInterface.hpp.

3.102.2.7 bool Ipopt::Ma97SolverInterface::ProvidesDegeneracyDetection ( ) const [inline], [virtual]

Query whether the indices of linearly dependent rows/columns can be determined by this linear solver.

Reimplemented from Ipopt::SparseSymLinearSolverInterface.

Definition at line 270 of file IpMa97SolverInterface.hpp.

3.102.2.8 ESymSolverStatus Ipopt::Ma97SolverInterface::DetermineDependentRows (const Index * *ia*, const Index * *ja*, std::list< Index > & *c_deps*) [inline], [virtual]

This method determines the list of row indices of the linearly dependent rows.

Reimplemented from Ipopt::SparseSymLinearSolverInterface.

Definition at line 276 of file IpMa97SolverInterface.hpp.

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

- IpMa97SolverInterface.hpp

3.103 Ipopt::Matrix Class Reference

Matrix Base Class.

`#include <IpMatrix.hpp>`

Inheritance diagram for Ipopt::Matrix:

Public Member Functions

- bool HasValidNumbers () const
  Method for determining if all stored numbers are valid (i.e., no Inf or Nan).
- SmartPtr< const MatrixSpace > OwnerSpace () const
  Return the owner MatrixSpace.

Constructor/Destructor

- Matrix (const MatrixSpace ∗owner_space)
  Constructor.
- virtual ∼Matrix ()
  Destructor.

Operations of the Matrix on a Vector

- void MultVector (Number alpha, const Vector &x, Number beta, Vector &y) const
  Matrix-vector multiply.
- void TransMultVector (Number alpha, const Vector &x, Number beta, Vector &y) const
  Matrix(transpose) vector multiply.

Methods for specialized operations. A prototype

implementation is provided, but for efficient implementation those should be specially implemented.

- void AddMSinvZ (Number alpha, const Vector &S, const Vector &Z, Vector &X) const
  X = X + alpha∗(Matrix S^{-1} Z).
- void SinvBlrmZMTdBr (Number alpha, const Vector &S, const Vector &R, const Vector &Z, const Vector &D, Vector &X) const
  X = S^{-1} (r + alpha∗Z∗M^Td).

Information about the size of the matrix

- Index NRows () const
  Number of rows.
- Index NCols () const
  Number of columns.

Norms of the individual rows and columns

- void ComputeRowAMax (Vector &rows_norms, bool init=true) const
  Compute the max-norm of the rows in the matrix.
- void ComputeColAMax (Vector &cols_norms, bool init=true) const
  Compute the max-norm of the columns in the matrix.

- virtual void Print (SmartPtr< const Journalist > jnlst, EJournalLevel level, EJournalCategory category, const std::string &name, Index indent=0, const std::string &prefix="") const
  Print detailed information about the matrix.

Protected Member Functions

implementation methods (derived classes MUST

overload these pure virtual protected methods.

- virtual void MultVectorImpl (Number alpha, const Vector &x, Number beta, Vector &y) const =0
  Matrix-vector multiply.
- virtual void TransMultVectorImpl (Number alpha, const Vector &x, Number beta, Vector &y) const =0
  Matrix(transpose) vector multiply.
- virtual void AddMSinvZImpl (Number alpha, const Vector &S, const Vector &Z, Vector &X) const
  X = X + alpha∗(Matrix S^{-1} Z).
- virtual void SinvBlrmZMTdBrImpl (Number alpha, const Vector &S, const Vector &R, const Vector &Z, const Vector &D, Vector &X) const
  X = S^{-1} (r + alpha∗Z∗M^Td).
- virtual bool HasValidNumbersImpl () const
  Method for determining if all stored numbers are valid (i.e., no Inf or Nan).
- virtual void ComputeRowAMaxImpl (Vector &rows_norms, bool init) const =0
  Compute the max-norm of the rows in the matrix.
- virtual void ComputeColAMaxImpl (Vector &cols_norms, bool init) const =0
  Compute the max-norm of the columns in the matrix.
- virtual void PrintImpl (const Journalist &jnlst, EJournalLevel level, EJournalCategory category, const std::string &name, Index indent, const std::string &prefix) const =0
  Print detailed information about the matrix.

Additional Inherited Members

3.103.1 Detailed Description

Matrix Base Class.

This is the base class for all derived matrix types. All Matrices, such as Jacobian and Hessian matrices, as well as possibly the iteration matrices needed for the step computation, are of this type.

Deriving from Matrix: Overload the protected XXX_Impl method.

Definition at line 27 of file IpMatrix.hpp.

3.103.2 Constructor & Destructor Documentation

3.103.2.1 Ipopt::Matrix::Matrix (const MatrixSpace ∗ *owner_space*) [inline]

Constructor.

It has to be given a pointer to the corresponding MatrixSpace.

Definition at line 35 of file IpMatrix.hpp.

3.103.3 Member Function Documentation

3.103.3.1 void Ipopt::Matrix::MultVector (Number *alpha*, const Vector & *x*, Number *beta*, Vector & *y*) const [inline]

Matrix-vector multiply.

Computes y = alpha ∗ Matrix ∗ x + beta ∗ y. Do not overload. Overload MultVectorImpl instead.

Definition at line 52 of file IpMatrix.hpp.

3.103.3.2 void Ipopt::Matrix::TransMultVector (Number *alpha*, const Vector & *x*, Number *beta*, Vector & *y*) const [inline]

Matrix(transpose) vector multiply.

Computes y = alpha * Matrix^T * x + beta * y. Do not overload. Overload TransMultVectorImpl instead.

Definition at line 62 of file IpMatrix.hpp.

3.103.3.3 void Ipopt::Matrix::AddMSinvZ (Number *alpha*, const Vector & *S*, const Vector & *Z*, Vector & *X*) const

X = X + alpha*(Matrix S^{-1} Z).

Should be implemented efficiently for the ExansionMatrix

3.103.3.4 void Ipopt::Matrix::SinvBlrmZMTdBr (Number *alpha*, const Vector & *S*, const Vector & *R*, const Vector & *Z*, const Vector & *D*, Vector & *X*) const

X = S^{-1} (r + alpha*Z*M^Td).

Should be implemented efficiently for the ExansionMatrix

3.103.3.5 bool Ipopt::Matrix::HasValidNumbers ( ) const

Method for determining if all stored numbers are valid (i.e., no Inf or Nan).

3.103.3.6 void Ipopt::Matrix::ComputeRowAMax (Vector & *rows_norms*, bool *init* = true) const [inline]

Compute the max-norm of the rows in the matrix.

The result is stored in rows_norms. The vector is assumed to be initialized of init is false.

Definition at line 108 of file IpMatrix.hpp.

3.103.3.7 void Ipopt::Matrix::ComputeColAMax (Vector & *cols_norms*, bool *init* = true) const [inline]

Compute the max-norm of the columns in the matrix.

The result is stored in cols_norms The vector is assumed to be initialized of init is false.

Definition at line 117 of file IpMatrix.hpp.

3.103.3.8 virtual void Ipopt::Matrix::Print (SmartPtr< const Journalist > *jnlst*, EJournalLevel *level*, EJournalCategory *category*, const std::string & *name*, Index *indent* = 0, const std::string & *prefix* = "") const [virtual]

Print detailed information about the matrix.

Do not overload. Overload PrintImpl instead.

3.103.3.9 virtual void Ipopt::Matrix::MultVectorImpl (Number *alpha*, const Vector & *x*, Number *beta*, Vector & *y*) const [protected], [pure virtual]

Matrix-vector multiply.

Computes y = alpha * Matrix * x + beta * y

Implemented in Ipopt::DenseGenMatrix, Ipopt::MultiVectorMatrix, Ipopt::SymTMatrix, Ipopt::DenseSymMatrix, Ipopt↩ ::LowRankUpdateSymMatrix, Ipopt::GenTMatrix, Ipopt::CompoundMatrix, Ipopt::CompoundSymMatrix, Ipopt::↩ ExpandedMultiVectorMatrix, Ipopt::ExpansionMatrix, Ipopt::ScaledMatrix, Ipopt::SymScaledMatrix, Ipopt::Identity↩ Matrix, Ipopt::SumSymMatrix, Ipopt::SumMatrix, Ipopt::DiagMatrix, Ipopt::TransposeMatrix, Ipopt::ZeroMatrix, and Ipopt::ZeroSymMatrix.

3.103.3.10 virtual void Ipopt::Matrix::TransMultVectorImpl (Number *alpha*, const Vector & *x*, Number *beta*, Vector & *y*) const [protected], [pure virtual]

Matrix(transpose) vector multiply.

Computes y = alpha ∗ Matrix^T ∗ x + beta ∗ y

Implemented in Ipopt::DenseGenMatrix, Ipopt::MultiVectorMatrix, Ipopt::GenTMatrix, Ipopt::CompoundMatrix, Ipopt::↩ ExpandedMultiVectorMatrix, Ipopt::ExpansionMatrix, Ipopt::ScaledMatrix, Ipopt::SymMatrix, Ipopt::SumMatrix, Ipopt↩ ::TransposeMatrix, Ipopt::ZeroMatrix, and Ipopt::ZeroSymMatrix.

3.103.3.11 virtual void Ipopt::Matrix::AddMSinvZImpl (Number *alpha*, const Vector & *S*, const Vector & *Z*, Vector & *X*) const [protected], [virtual]

X = X + alpha∗(Matrix S^{-1} Z).

Prototype for this specialize method is provided, but for efficient implementation it should be overloaded for the expansion matrix.

Reimplemented in Ipopt::CompoundMatrix, Ipopt::ScaledMatrix, Ipopt::ExpansionMatrix, and Ipopt::IdentityMatrix.

3.103.3.12 virtual void Ipopt::Matrix::SinvBlrmZMTdBrImpl (Number *alpha*, const Vector & *S*, const Vector & *R*, const Vector & *Z*, const Vector & *D*, Vector & *X*) const [protected], [virtual]

X = S^{-1} (r + alpha∗Z∗M^Td).

Should be implemented efficiently for the ExpansionMatrix.

Reimplemented in Ipopt::CompoundMatrix, Ipopt::ScaledMatrix, and Ipopt::ExpansionMatrix.

3.103.3.13 virtual bool Ipopt::Matrix::HasValidNumbersImpl ( ) const [inline], [protected], [virtual]

Method for determining if all stored numbers are valid (i.e., no Inf or Nan).

A default implementation always returning true is provided, but if possible it should be implemented.

Reimplemented in Ipopt::DenseGenMatrix, Ipopt::MultiVectorMatrix, Ipopt::SymTMatrix, Ipopt::CompoundMatrix, Ipopt::DenseSymMatrix, Ipopt::LowRankUpdateSymMatrix, Ipopt::GenTMatrix, Ipopt::CompoundSymMatrix, Ipopt↩ ::ExpandedMultiVectorMatrix, Ipopt::ScaledMatrix, Ipopt::SymScaledMatrix, Ipopt::IdentityMatrix, Ipopt::Transpose↩ Matrix, Ipopt::SumMatrix, Ipopt::SumSymMatrix, and Ipopt::DiagMatrix.

Definition at line 179 of file IpMatrix.hpp.

3.103.3.14 virtual void Ipopt::Matrix::ComputeRowAMaxImpl (Vector & *rows_norms*, bool *init*) const [protected], [pure virtual]

Compute the max-norm of the rows in the matrix.

The result is stored in rows_norms. The vector is assumed to be initialized.

Implemented in Ipopt::DenseGenMatrix, Ipopt::MultiVectorMatrix, Ipopt::SymTMatrix, Ipopt::CompoundMatrix, Ipopt::DenseSymMatrix, Ipopt::LowRankUpdateSymMatrix, Ipopt::GenTMatrix, Ipopt::CompoundSymMatrix, Ipopt::ExpandedMultiVectorMatrix, Ipopt::ExpansionMatrix, Ipopt::ScaledMatrix, Ipopt::TransposeMatrix, Ipopt::SymScaledMatrix, Ipopt::IdentityMatrix, Ipopt::SumMatrix, Ipopt::SumSymMatrix, Ipopt::DiagMatrix, Ipopt::ZeroMatrix, and Ipopt::ZeroSymMatrix.

3.103.3.15 virtual void Ipopt::Matrix::ComputeColAMaxImpl (Vector & *cols_norms*, bool *init*) const [protected], [pure virtual]

Compute the max-norm of the columns in the matrix.

The result is stored in cols_norms. The vector is assumed to be initialized.

Implemented in Ipopt::DenseGenMatrix, Ipopt::MultiVectorMatrix, Ipopt::CompoundMatrix, Ipopt::LowRankUpdateSymMatrix, Ipopt::GenTMatrix, Ipopt::ExpandedMultiVectorMatrix, Ipopt::ExpansionMatrix, Ipopt::ScaledMatrix, Ipopt::TransposeMatrix, Ipopt::SymMatrix, Ipopt::SumMatrix, Ipopt::SumSymMatrix, Ipopt::ZeroMatrix, and Ipopt::ZeroSymMatrix.

3.103.3.16 virtual void Ipopt::Matrix::PrintImpl (const Journalist & *jnlst*, EJournalLevel *level*, EJournalCategory *category*, const std::string & *name*, Index *indent*, const std::string & *prefix*) const [protected], [pure virtual]

Print detailed information about the matrix.

Implemented in Ipopt::DenseGenMatrix, Ipopt::MultiVectorMatrix, Ipopt::SymTMatrix, Ipopt::CompoundMatrix, Ipopt::LowRankUpdateSymMatrix, Ipopt::DenseSymMatrix, Ipopt::GenTMatrix, Ipopt::CompoundSymMatrix, Ipopt::ExpandedMultiVectorMatrix, Ipopt::ExpansionMatrix, Ipopt::TransposeMatrix, Ipopt::ScaledMatrix, Ipopt::SymScaledMatrix, Ipopt::IdentityMatrix, Ipopt::SumMatrix, Ipopt::SumSymMatrix, Ipopt::DiagMatrix, Ipopt::ZeroMatrix, and Ipopt::ZeroSymMatrix.

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

- IpMatrix.hpp

3.104 Ipopt::MatrixSpace Class Reference

MatrixSpace base class, corresponding to the Matrix base class.

`#include <IpMatrix.hpp>`

Inheritance diagram for Ipopt::MatrixSpace:

Public Member Functions

- virtual Matrix * MakeNew () const =0

  Pure virtual method for creating a new Matrix of the corresponding type.
- Index NRows () const

  Accessor function for the number of rows.
- Index NCols () const

  Accessor function for the number of columns.
- bool IsMatrixFromSpace (const Matrix &matrix) const

  Method to test if a given matrix belongs to a particular matrix space.

Constructors/Destructors

- MatrixSpace (Index nRows, Index nCols)

    Constructor, given the number rows and columns of all matrices generated by this MatrixSpace.
- virtual ∼MatrixSpace ()

    Destructor.

3.104.1 Detailed Description

MatrixSpace base class, corresponding to the Matrix base class.

For each Matrix implementation, a corresponding MatrixSpace has to be implemented. A MatrixSpace is able to create new Matrices of a specific type. The MatrixSpace should also store information that is common to all Matrices of that type. For example, the dimensions of a Matrix is stored in the MatrixSpace base class.

Definition at line 239 of file IpMatrix.hpp.

3.104.2 Member Function Documentation

3.104.2.1 Index Ipopt::MatrixSpace::NRows ( ) const [inline]

Accessor function for the number of rows.

Definition at line 264 of file IpMatrix.hpp.

3.104.2.2 Index Ipopt::MatrixSpace::NCols ( ) const [inline]

Accessor function for the number of columns.

Definition at line 269 of file IpMatrix.hpp.

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

- IpMatrix.hpp

3.105 Ipopt::Mc19TSymScalingMethod Class Reference

Class for the method for computing scaling factors for symmetric matrices in triplet format, using MC19.

`#include <IpMc19TSymScalingMethod.hpp>`

Inheritance diagram for Ipopt::Mc19TSymScalingMethod:

Public Member Functions

- virtual bool InitializeImpl (const OptionsList &options, const std::string &prefix)

    overloaded from AlgorithmStrategyObject
- virtual bool ComputeSymTScalingFactors (Index n, Index nnz, const ipfint ∗airn, const ipfint ∗ajcn, const double ∗a, double ∗scaling_factors)

    Method for computing the symmetric scaling factors, given the symmtric matrix in triplet (MA27) format.

Constructor/Destructor

- **Mc19TSymScalingMethod** ()
- virtual ∼**Mc19TSymScalingMethod** ()

Additional Inherited Members

3.105.1 Detailed Description

Class for the method for computing scaling factors for symmetric matrices in triplet format, using MC19.

Definition at line 21 of file IpMc19TSymScalingMethod.hpp.

3.105.2 Member Function Documentation

3.105.2.1 virtual bool Ipopt::Mc19TSymScalingMethod::ComputeSymTScalingFactors (Index *n*, Index *nnz*, const ipfint ∗ *airn*, const ipfint ∗ *ajcn*, const double ∗ *a*, double ∗ *scaling_factors*) [virtual]

Method for computing the symmetric scaling factors, given the symmtric matrix in triplet (MA27) format.

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

- IpMc19TSymScalingMethod.hpp

3.106 mc68_control Struct Reference

3.106.1 Detailed Description

Definition at line 27 of file hsl_mc68i.h.

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

- hsl_mc68i.h

3.107 mc68_info Struct Reference

3.107.1 Detailed Description

Definition at line 48 of file hsl_mc68i.h.

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

- hsl_mc68i.h

3.108 Ipopt::MinC_1NrmRestorationPhase Class Reference

Restoration Phase that minimizes the 1-norm of the constraint violation - using the interior point method (Ipopt).

`#include <IpRestoMinC_1Nrm.hpp>`

Inheritance diagram for Ipopt::MinC_1NrmRestorationPhase:

Public Member Functions

- virtual bool InitializeImpl (const OptionsList &options, const std::string &prefix)

  Overloaded from AlgorithmStrategy case class.

Constructors/Destructors

- MinC_1NrmRestorationPhase (IpoptAlgorithm &resto_alg, const SmartPtr< EqMultiplierCalculator > &eq_↩ mult_calculator)

  Constructor, taking strategy objects.
- virtual ∼MinC_1NrmRestorationPhase ()

  Default destructor.

Static Public Member Functions

- static void RegisterOptions (SmartPtr< RegisteredOptions > roptions)

  Methods for IpoptType.

Protected Member Functions

- virtual bool PerformRestoration ()

  Overloaded method from RestorationPhase.

3.108.1 Detailed Description

Restoration Phase that minimizes the 1-norm of the constraint violation - using the interior point method (Ipopt).

Definition at line 22 of file IpRestoMinC_1Nrm.hpp.

3.108.2 Constructor & Destructor Documentation

3.108.2.1 Ipopt::MinC_1NrmRestorationPhase::MinC_1NrmRestorationPhase (IpoptAlgorithm & *resto_alg*, const SmartPtr< EqMultiplierCalculator > & *eq_mult_calculator*)

Constructor, taking strategy objects.

The resto_alg strategy object is the restoration phase Ipopt algorithm. The eq_mult_calculator is used to reinitialize the equality constraint multipliers after the restoration phase algorithm has finished - unless it is NULL, in which case the multipliers are set to 0.

3.108.3 Member Function Documentation

3.108.3.1 virtual bool Ipopt::MinC_1NrmRestorationPhase::PerformRestoration ( ) [protected], [virtual]

Overloaded method from RestorationPhase.

Implements Ipopt::RestorationPhase.

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

- IpRestoMinC_1Nrm.hpp

3.109 Ipopt::MonotoneMuUpdate Class Reference

Monotone Mu Update.

`#include <IpMonotoneMuUpdate.hpp>`

Inheritance diagram for Ipopt::MonotoneMuUpdate:

Public Member Functions

- virtual bool InitializeImpl (const OptionsList &options, const std::string &prefix)

  Initialize method - overloaded from AlgorithmStrategyObject.
- virtual bool UpdateBarrierParameter ()

  Method for determining the barrier parameter for the next iteration.

Constructors/Destructors

- MonotoneMuUpdate (const SmartPtr< LineSearch > &linesearch)

  Default Constructor.
- virtual ∼MonotoneMuUpdate ()

  Default destructor.

Static Public Member Functions

- static void RegisterOptions (const SmartPtr< RegisteredOptions > &roptions)

  Methods for IpoptType.

Additional Inherited Members

3.109.1 Detailed Description

Monotone Mu Update.

This class implements the standard monotone mu update approach.

Definition at line 22 of file IpMonotoneMuUpdate.hpp.

3.109.2 Member Function Documentation

3.109.2.1 virtual bool Ipopt::MonotoneMuUpdate::UpdateBarrierParameter ( ) [virtual]

Method for determining the barrier parameter for the next iteration.

When the optimality error for the current barrier parameter is less than a tolerance, the barrier parameter is reduced, and the Reset method of the LineSearch object linesearch is called.

Implements Ipopt::MuUpdate.

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

- IpMonotoneMuUpdate.hpp

3.110 Ipopt::MultiVectorMatrix Class Reference

Class for Matrices with few columns that consists of Vectors.

`#include <IpMultiVectorMatrix.hpp>`

Inheritance diagram for Ipopt::MultiVectorMatrix:

Public Member Functions

- SmartPtr< MultiVectorMatrix > MakeNewMultiVectorMatrix () const

    Create a new MultiVectorMatrix from same MatrixSpace.
- SmartPtr< const Vector > GetVector (Index i) const

    Get a Vector in a particular column as a const Vector.
- SmartPtr< Vector > GetVectorNonConst (Index i)

    Get a Vector in a particular column as a non-const Vector.
- void ScaleRows (const Vector &scal_vec)

    Method for scaling the rows of the matrix, using the ElementWiseMultiply method for each column vector.
- void ScaleColumns (const Vector &scal_vec)

    Method for scaling the columns of the matrix, using the Scal method for each column vector.
- void AddOneMultiVectorMatrix (Number a, const MultiVectorMatrix &mv1, Number c)

    Adding another MultiVectorMatrix, using the AddOneVector methods for the individual column vectors.
- void AddRightMultMatrix (Number a, const MultiVectorMatrix &U, const Matrix &C, Number b)

    Multiplying a Matrix C (for now assumed to be a DenseGenMatrix) from the right to a MultiVectorMatrix U and adding the result to this MultiVectorMatrix V.
- void FillWithNewVectors ()

    Method for initializing all Vectors with new (uninitialized) Vectors.
- void LRMultVector (Number alpha, const Vector &x, Number beta, Vector &y) const

    Method for adding the low-rank update matrix corresponding to this matrix to a vector.
- SmartPtr< const VectorSpace > ColVectorSpace () const

    Vector space for the columns.
- SmartPtr< const MultiVectorMatrixSpace > MultiVectorMatrixOwnerSpace () const

    Return the MultiVectorMatrixSpace.

Constructors / Destructors

- MultiVectorMatrix (const MultiVectorMatrixSpace ∗owner_space)

    Constructor, taking the owner_space.
- ∼MultiVectorMatrix ()

    Destructor.

- void SetVector (Index i, const Vector &vec)

    Set a particular Vector at a given column position, replacing another vector if there has been one.

Protected Member Functions

Overloaded methods from Matrix base class

- virtual void MultVectorImpl (Number alpha, const Vector &x, Number beta, Vector &y) const
  Matrix-vector multiply.
- virtual void TransMultVectorImpl (Number alpha, const Vector &x, Number beta, Vector &y) const
  Matrix(transpose) vector multiply.
- virtual bool HasValidNumbersImpl () const
  Method for determining if all stored numbers are valid (i.e., no Inf or Nan).
- virtual void ComputeRowAMaxImpl (Vector &rows_norms, bool init) const
  Compute the max-norm of the rows in the matrix.
- virtual void ComputeColAMaxImpl (Vector &cols_norms, bool init) const
  Compute the max-norm of the columns in the matrix.
- virtual void PrintImpl (const Journalist &jnlst, EJournalLevel level, EJournalCategory category, const std::string &name, Index indent, const std::string &prefix) const
  Print detailed information about the matrix.

Additional Inherited Members

3.110.1 Detailed Description

Class for Matrices with few columns that consists of Vectors.

Those matrices are for example useful in the implementation of limited memory quasi-Newton methods.

Definition at line 25 of file IpMultiVectorMatrix.hpp.

3.110.2 Member Function Documentation

3.110.2.1 void Ipopt::MultiVectorMatrix::SetVector (Index *i*, const Vector & *vec*)

Set a particular Vector at a given column position, replacing another vector if there has been one.

Depending on whether the Vector is const or not, it is stored in the const or non-const internal column.

3.110.2.2 SmartPtr<Vector> Ipopt::MultiVectorMatrix::GetVectorNonConst (Index *i*) [inline]

Get a Vector in a particular column as a non-const Vector.

This is fail if the column has currently only a non-const Vector stored.

Definition at line 64 of file IpMultiVectorMatrix.hpp.

3.110.2.3 void Ipopt::MultiVectorMatrix::ScaleRows (const Vector & *scal_vec*)

Method for scaling the rows of the matrix, using the ElementWiseMultiply method for each column vector.

3.110.2.4 void Ipopt::MultiVectorMatrix::ScaleColumns (const Vector & *scal_vec*)

Method for scaling the columns of the matrix, using the Scal method for each column vector.

3.110.2.5 void Ipopt::MultiVectorMatrix::AddRightMultMatrix (Number *a*, const MultiVectorMatrix & *U*, const Matrix & *C*, Number *b*)

Multiplying a Matrix C (for now assumed to be a DenseGenMatrix) from the right to a MultiVectorMatrix U and adding the result to this MultiVectorMatrix V.

V = a ∗ U ∗ C + b ∗ V.

3.110.2.6 void Ipopt::MultiVectorMatrix::FillWithNewVectors ( )

Method for initializing all Vectors with new (uninitialized) Vectors.

3.110.2.7 void Ipopt::MultiVectorMatrix::LRMultVector (Number *alpha*, const Vector & *x*, Number *beta*, Vector & *y*) const

Method for adding the low-rank update matrix corresponding to this matrix to a vector.

If V is this MultiVectorMatrix, the operation is y = beta∗y + alpha∗V∗V^T∗x.

3.110.2.8 virtual void Ipopt::MultiVectorMatrix::MultVectorImpl (Number *alpha*, const Vector & *x*, Number *beta*, Vector & *y*) const [protected], [virtual]

Matrix-vector multiply.

Computes y = alpha ∗ Matrix ∗ x + beta ∗ y

Implements Ipopt::Matrix.

3.110.2.9 virtual void Ipopt::MultiVectorMatrix::TransMultVectorImpl (Number *alpha*, const Vector & *x*, Number *beta*, Vector & *y*) const [protected], [virtual]

Matrix(transpose) vector multiply.

Computes y = alpha ∗ Matrix^T ∗ x + beta ∗ y

Implements Ipopt::Matrix.

3.110.2.10 virtual bool Ipopt::MultiVectorMatrix::HasValidNumbersImpl ( ) const [protected], [virtual]

Method for determining if all stored numbers are valid (i.e., no Inf or Nan).

Reimplemented from Ipopt::Matrix.

3.110.2.11 virtual void Ipopt::MultiVectorMatrix::ComputeRowAMaxImpl (Vector & *rows_norms*, bool *init*) const [protected], [virtual]

Compute the max-norm of the rows in the matrix.

The result is stored in rows_norms. The vector is assumed to be initialized.

Implements Ipopt::Matrix.

3.110.2.12 virtual void Ipopt::MultiVectorMatrix::ComputeColAMaxImpl (Vector & *cols_norms*, bool *init*) const [protected], [virtual]

Compute the max-norm of the columns in the matrix.

The result is stored in cols_norms. The vector is assumed to be initialized.

Implements Ipopt::Matrix.

3.110.2.13 virtual void Ipopt::MultiVectorMatrix::PrintImpl (const Journalist & *jnlst*, EJournalLevel *level*, EJournalCategory *category*, const std::string & *name*, Index *indent*, const std::string & *prefix*) const [protected], [virtual]

Print detailed information about the matrix.

Implements Ipopt::Matrix.

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

- IpMultiVectorMatrix.hpp

3.111 Ipopt::MultiVectorMatrixSpace Class Reference

This is the matrix space for MultiVectorMatrix.

`#include <IpMultiVectorMatrix.hpp>`

Inheritance diagram for Ipopt::MultiVectorMatrixSpace:

Public Member Functions

- MultiVectorMatrix * MakeNewMultiVectorMatrix () const

  Method for creating a new matrix of this specific type.
- virtual Matrix * MakeNew () const

  Overloaded MakeNew method for the MatrixSpace base class.
- SmartPtr< const VectorSpace > ColVectorSpace () const

  Accessor method for the VectorSpace for the columns.

Constructors / Destructors

- MultiVectorMatrixSpace (Index ncols, const VectorSpace &vec_space)

  Constructor, given the number of columns (i.e., Vectors to be stored) and given the VectorSpace for the Vectors.
- ~MultiVectorMatrixSpace ()

  Destructor.

3.111.1 Detailed Description

This is the matrix space for MultiVectorMatrix.

Definition at line 184 of file IpMultiVectorMatrix.hpp.

3.111.2 Member Function Documentation

3.111.2.1 MultiVectorMatrix∗ Ipopt::MultiVectorMatrixSpace::MakeNewMultiVectorMatrix ( ) const [inline]

Method for creating a new matrix of this specific type.

Definition at line 201 of file IpMultiVectorMatrix.hpp.

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

- IpMultiVectorMatrix.hpp

3.112 Ipopt::MumpsSolverInterface Class Reference

Interface to the linear solver Mumps, derived from SparseSymLinearSolverInterface.

`#include <IpMumpsSolverInterface.hpp>`

Inheritance diagram for Ipopt::MumpsSolverInterface:

Public Member Functions

- bool InitializeImpl (const OptionsList &options, const std::string &prefix)
  overloaded from AlgorithmStrategyObject
- virtual bool ProvidesDegeneracyDetection () const
  Query whether the indices of linearly dependent rows/columns can be determined by this linear solver.
- virtual ESymSolverStatus DetermineDependentRows (const Index ∗ia, const Index ∗ja, std::list< Index > &c_↩ deps)
  This method determines the list of row indices of the linearly dependent rows.

Constructor/Destructor

- MumpsSolverInterface ()
  Constructor.
- virtual ∼MumpsSolverInterface ()
  Destructor.

Methods for requesting solution of the linear system.

- virtual ESymSolverStatus InitializeStructure (Index dim, Index nonzeros, const Index ∗airn, const Index ∗ajcn)
  Method for initializing internal stuctures.
- virtual double ∗ GetValuesArrayPtr ()
  Method returing an internal array into which the nonzero elements (in the same order as airn and ajcn) are to be stored by the calling routine before a call to MultiSolve with a new_matrix=true.
- virtual ESymSolverStatus MultiSolve (bool new_matrix, const Index ∗airn, const Index ∗ajcn, Index nrhs, double ∗rhs_vals, bool check_NegEVals, Index numberOfNegEVals)
  Solve operation for multiple right hand sides.
- virtual Index NumberOfNegEVals () const
  Number of negative eigenvalues detected during last factorization.

- virtual bool IncreaseQuality ()
  Request to increase quality of solution for next solve.

- virtual bool ProvidesInertia () const

  Query whether inertia is computed by linear solver.
- EMatrixFormat MatrixFormat () const

  Query of requested matrix type that the linear solver understands.

Static Public Member Functions

- static void RegisterOptions (SmartPtr< RegisteredOptions > roptions)

  Methods for IpoptType.

Additional Inherited Members

3.112.1 Detailed Description

Interface to the linear solver Mumps, derived from SparseSymLinearSolverInterface.

For details, see description of SparseSymLinearSolverInterface base class.

Definition at line 26 of file IpMumpsSolverInterface.hpp.

3.112.2 Member Function Documentation

3.112.2.1 virtual ESymSolverStatus Ipopt::MumpsSolverInterface::InitializeStructure (Index *dim,* Index *nonzeros,* const Index ∗ *airn,* const Index ∗ *ajcn*) [virtual]

Method for initializing internal stuctures.

Here, ndim gives the number of rows and columns of the matrix, nonzeros give the number of nonzero elements, and airn and acjn give the positions of the nonzero elements.

Implements Ipopt::SparseSymLinearSolverInterface.

3.112.2.2 virtual double∗ Ipopt::MumpsSolverInterface::GetValuesArrayPtr ( ) [virtual]

Method returing an internal array into which the nonzero elements (in the same order as airn and ajcn) are to be stored by the calling routine before a call to MultiSolve with a new_matrix=true.

The returned array must have space for at least nonzero elements.

Implements Ipopt::SparseSymLinearSolverInterface.

3.112.2.3 virtual ESymSolverStatus Ipopt::MumpsSolverInterface::MultiSolve (bool *new_matrix,* const Index ∗ *airn,* const Index ∗ *ajcn,* Index *nrhs,* double ∗ *rhs_vals,* bool *check_NegEVals,* Index *numberOfNegEVals*) [virtual]

Solve operation for multiple right hand sides.

Overloaded from SparseSymLinearSolverInterface.

Implements Ipopt::SparseSymLinearSolverInterface.

3.112.2.4 virtual Index Ipopt::MumpsSolverInterface::NumberOfNegEVals ( ) const [virtual]

Number of negative eigenvalues detected during last factorization.

Returns the number of negative eigenvalues of the most recent factorized matrix. This must not be called if the linear solver does not compute this quantities (see ProvidesInertia).

Implements Ipopt::SparseSymLinearSolverInterface.

3.112.2.5 virtual bool Ipopt::MumpsSolverInterface::IncreaseQuality ( ) [virtual]

Request to increase quality of solution for next solve.

Ask linear solver to increase quality of solution for the next solve (e.g. increase pivot tolerance). Returns false, if this is not possible (e.g. maximal pivot tolerance already used.)

Implements Ipopt::SparseSymLinearSolverInterface.

3.112.2.6 virtual bool Ipopt::MumpsSolverInterface::ProvidesInertia ( ) const [inline], [virtual]

Query whether inertia is computed by linear solver.

Returns true, if linear solver provides inertia.

Implements Ipopt::SparseSymLinearSolverInterface.

Definition at line 92 of file IpMumpsSolverInterface.hpp.

3.112.2.7 virtual bool Ipopt::MumpsSolverInterface::ProvidesDegeneracyDetection ( ) const [virtual]

Query whether the indices of linearly dependent rows/columns can be determined by this linear solver.

Reimplemented from Ipopt::SparseSymLinearSolverInterface.

3.112.2.8 virtual ESymSolverStatus Ipopt::MumpsSolverInterface::DetermineDependentRows (const Index $*$ *ia*, const Index $*$ *ja*, std::list$<$ Index $>$ & *c_deps*) [virtual]

This method determines the list of row indices of the linearly dependent rows.

Reimplemented from Ipopt::SparseSymLinearSolverInterface.

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

- IpMumpsSolverInterface.hpp

3.113 Ipopt::MuOracle Class Reference

Abstract Base Class for classes that are able to compute a suggested value of the barrier parameter that can be used as an oracle in the NonmontoneMuUpdate class.

```
#include <IpMuOracle.hpp>
```

Inheritance diagram for Ipopt::MuOracle:

Public Member Functions

- virtual bool InitializeImpl (const OptionsList &options, const std::string &prefix)=0

  Initialize method - overloaded from AlgorithmStrategyObject.
- virtual bool CalculateMu (Number mu_min, Number mu_max, Number &new_mu)=0

  Method for computing the value of the barrier parameter that could be used in the current iteration.

Constructors/Destructors

- MuOracle ()

  Default Constructor.
- virtual ∼MuOracle ()

  Default destructor.

Additional Inherited Members

3.113.1 Detailed Description

Abstract Base Class for classes that are able to compute a suggested value of the barrier parameter that can be used as an oracle in the NonmontoneMuUpdate class.

Definition at line 21 of file IpMuOracle.hpp.

3.113.2 Member Function Documentation

3.113.2.1 virtual bool Ipopt::MuOracle::CalculateMu (Number *mu_min,* Number *mu_max,* Number & *new_mu*) [pure virtual]

Method for computing the value of the barrier parameter that could be used in the current iteration.

Here, mu_min and mu_max are the lower and upper bounds on acceptable values for the barrier parameter. The new value of mu is returned in new_mu, and the method returns false if a new value could not be determined (e.g., because the linear system could not be solved for a predictor step).

Implemented in Ipopt::ProbingMuOracle, Ipopt::QualityFunctionMuOracle, and Ipopt::LoqoMuOracle.

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

- IpMuOracle.hpp

3.114 Ipopt::MuUpdate Class Reference

Abstract Base Class for classes that implement methods for computing the barrier and fraction-to-the-boundary rule parameter for the current iteration.

`#include <IpMuUpdate.hpp>`

Inheritance diagram for Ipopt::MuUpdate:

Public Member Functions

- virtual bool InitializeImpl (const OptionsList &options, const std::string &prefix)=0

Initialize method - overloaded from AlgorithmStrategyObject.
- virtual bool UpdateBarrierParameter ()=0

  Method for determining the barrier parameter for the next iteration.

Constructors/Destructors

- MuUpdate ()

  Default Constructor.
- virtual ∼MuUpdate ()

  Default destructor.

Additional Inherited Members

3.114.1 Detailed Description

Abstract Base Class for classes that implement methods for computing the barrier and fraction-to-the-boundary rule parameter for the current iteration.

Definition at line 20 of file IpMuUpdate.hpp.

3.114.2 Member Function Documentation

3.114.2.1 virtual bool Ipopt::MuUpdate::UpdateBarrierParameter ( ) [pure virtual]

Method for determining the barrier parameter for the next iteration.

A LineSearch object is passed, so that this method can call the Reset method in the LineSearch object, for example when then barrier parameter is changed. This method is also responsible for setting the fraction-to-the-boundary parameter tau. This method returns false if the update could not be performed and the algorithm should revert to an emergency fallback mechanism.

Implemented in Ipopt::AdaptiveMuUpdate, and Ipopt::MonotoneMuUpdate.

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

- IpMuUpdate.hpp

3.115 Ipopt::NLP Class Reference

Brief Class Description.

#include <IpNLP.hpp>

Inheritance diagram for Ipopt::NLP:

Public Member Functions

- virtual void GetQuasiNewtonApproximationSpaces (SmartPtr< VectorSpace > &approx_space, SmartPtr< Matrix > &P_approx)

  Method for obtaining the subspace in which the limited-memory Hessian approximation should be done.

Constructors/Destructors

- NLP ()

  Default constructor.
- virtual ~NLP ()

  Default destructor.

- DECLARE_STD_EXCEPTION (USER_SCALING_NOT_IMPLEMENTED)

  Exceptions.

NLP Initialization (overload in

derived classes).

- virtual bool ProcessOptions (const OptionsList &options, const std::string &prefix)

  Overload if you want the chance to process options or parameters that may be specific to the NLP.
- virtual bool GetSpaces (SmartPtr< const VectorSpace > &x_space, SmartPtr< const VectorSpace > &c_space, SmartPtr< const VectorSpace > &d_space, SmartPtr< const VectorSpace > &x_l_space, SmartPtr< const MatrixSpace > &px_l_space, SmartPtr< const VectorSpace > &x_u_space, SmartPtr< const MatrixSpace > &px_u_space, SmartPtr< const VectorSpace > &d_l_space, SmartPtr< const MatrixSpace > &pd_l_space, SmartPtr< const VectorSpace > &d_u_space, SmartPtr< const MatrixSpace > &pd_u_space, SmartPtr< const MatrixSpace > &Jac_c_space, SmartPtr< const MatrixSpace > &Jac_d_space, SmartPtr< const SymMatrixSpace > &Hess_lagrangian_space)=0

  Method for creating the derived vector / matrix types.
- virtual bool GetBoundsInformation (const Matrix &Px_L, Vector &x_L, const Matrix &Px_U, Vector &x_U, const Matrix &Pd_L, Vector &d_L, const Matrix &Pd_U, Vector &d_U)=0

  Method for obtaining the bounds information.
- virtual bool GetStartingPoint (SmartPtr< Vector > x, bool need_x, SmartPtr< Vector > y_c, bool need_y_c, SmartPtr< Vector > y_d, bool need_y_d, SmartPtr< Vector > z_L, bool need_z_L, SmartPtr< Vector > z_U, bool need_z_U)=0

  Method for obtaining the starting point for all the iterates.
- virtual bool GetWarmStartIterate (IteratesVector &warm_start_iterate)

  Method for obtaining an entire iterate as a warmstart point.

NLP evaluation routines (overload

in derived classes.

- virtual bool **Eval_f** (const Vector &x, Number &f)=0
- virtual bool **Eval_grad_f** (const Vector &x, Vector &g_f)=0
- virtual bool **Eval_c** (const Vector &x, Vector &c)=0
- virtual bool **Eval_jac_c** (const Vector &x, Matrix &jac_c)=0
- virtual bool **Eval_d** (const Vector &x, Vector &d)=0
- virtual bool **Eval_jac_d** (const Vector &x, Matrix &jac_d)=0
- virtual bool **Eval_h** (const Vector &x, Number obj_factor, const Vector &yc, const Vector &yd, SymMatrix &h)=0

NLP solution routines. Have default dummy

implementations that can be overloaded.

- virtual void FinalizeSolution (SolverReturn status, const Vector &x, const Vector &z_L, const Vector &z_U, const Vector &c, const Vector &d, const Vector &y_c, const Vector &y_d, Number obj_value, const IpoptData *ip_data, IpoptCalculatedQuantities *ip_cq)

  This method is called at the very end of the optimization.
- virtual bool IntermediateCallBack (AlgorithmMode mode, Index iter, Number obj_value, Number inf_pr, Number inf_du, Number mu, Number d_norm, Number regularization_size, Number alpha_du, Number alpha_pr, Index ls_trials, const IpoptData *ip_data, IpoptCalculatedQuantities *ip_cq)

  This method is called once per iteration, after the iteration summary output has been printed.

- virtual void GetScalingParameters (const SmartPtr< const VectorSpace > x_space, const SmartPtr< const VectorSpace > c_space, const SmartPtr< const VectorSpace > d_space, Number &obj_scaling, SmartPtr< Vector > &x_scaling, SmartPtr< Vector > &c_scaling, SmartPtr< Vector > &d_scaling) const

  Routines to get the scaling parameters.

3.115.1 Detailed Description

Brief Class Description.

Detailed Class Description.

Definition at line 31 of file IpNLP.hpp.

3.115.2 Member Function Documentation

3.115.2.1 virtual bool Ipopt::NLP::GetSpaces (**SmartPtr< const VectorSpace > &** *x_space,* **SmartPtr< const VectorSpace > &** *c_space,* **SmartPtr< const VectorSpace > &** *d_space,* **SmartPtr< const VectorSpace > &** *x_l_space,* **SmartPtr< const MatrixSpace > &** *px_l_space,* **SmartPtr< const VectorSpace > &** *x_u_space,* **SmartPtr< const MatrixSpace > &** *px_u_space,* **SmartPtr< const VectorSpace > &** *d_l_space,* **SmartPtr< const MatrixSpace > &** *pd_l_space,* **SmartPtr< const VectorSpace > &** *d_u_space,* **SmartPtr< const MatrixSpace > &** *pd_u_space,* **SmartPtr< const MatrixSpace > &** *Jac_c_space,* **SmartPtr< const MatrixSpace > &** *Jac_d_space,* **SmartPtr< const SymMatrixSpace > &** *Hess_lagrangian_space*) [pure virtual]

Method for creating the derived vector / matrix types.

The Hess_lagrangian_space pointer can be NULL if a quasi-Newton options is chosen.

Implemented in Ipopt::TNLPAdapter, and Ipopt::NLPBoundsRemover.

3.115.2.2 virtual bool Ipopt::NLP::GetStartingPoint (**SmartPtr< Vector >** *x,* bool *need_x,* **SmartPtr< Vector >** *y_c,* bool *need_y_c,* **SmartPtr< Vector >** *y_d,* bool *need_y_d,* **SmartPtr< Vector >** *z_L,* bool *need_z_L,* **SmartPtr< Vector >** *z_U,* bool *need_z_U*) [pure virtual]

Method for obtaining the starting point for all the iterates.

ToDo it might not make sense to ask for initial values for v_L and v_U?

Implemented in Ipopt::TNLPAdapter, and Ipopt::NLPBoundsRemover.

3.115.2.3 virtual bool Ipopt::NLP::GetWarmStartIterate (**IteratesVector &** *warm_start_iterate*) [inline], [virtual]

Method for obtaining an entire iterate as a warmstart point.

The incoming IteratesVector has to be filled. The default dummy implementation returns false.

Reimplemented in Ipopt::TNLPAdapter, and Ipopt::NLPBoundsRemover.

Definition at line 109 of file IpNLP.hpp.

3.115.2.4 virtual void Ipopt::NLP::FinalizeSolution (**SolverReturn** *status,* **const Vector &** *x,* **const Vector &** *z_L,* **const Vector &** *z_U,* **const Vector &** *c,* **const Vector &** *d,* **const Vector &** *y_c,* **const Vector &** *y_d,* **Number** *obj_value,* **const IpoptData ∗** *ip_data,* **IpoptCalculatedQuantities ∗** *ip_cq*) [inline], [virtual]

This method is called at the very end of the optimization.

It provides the final iterate to the user, so that it can be stored as the solution. The status flag indicates the outcome of the optimization, where SolverReturn is defined in IpAlgTypes.hpp.

Reimplemented in Ipopt::NLPBoundsRemover, and Ipopt::TNLPAdapter.

Definition at line 145 of file IpNLP.hpp.

3.115.2.5 virtual bool Ipopt::NLP::IntermediateCallBack (AlgorithmMode *mode,* Index *iter,* Number *obj_value,* Number *inf_pr,* Number *inf_du,* Number *mu,* Number *d_norm,* Number *regularization_size,* Number *alpha_du,* Number *alpha_pr,* Index *ls_trials,* const IpoptData ∗ *ip_data,* IpoptCalculatedQuantities ∗ *ip_cq*) [inline], [virtual]

This method is called once per iteration, after the iteration summary output has been printed.

It provides the current information to the user to do with it anything she wants. It also allows the user to ask for a premature termination of the optimization by returning false, in which case Ipopt will terminate with a corresponding return status. The basic information provided in the argument list has the quantities values printed in the iteration summary line. If more information is required, a user can obtain it from the IpData and IpCalculatedQuantities objects. However, note that the provided quantities are all for the problem that Ipopt sees, i.e., the quantities might be scaled, fixed variables might be sorted out, etc. The status indicates things like whether the algorithm is in the restoration phase... In the restoration phase, the dual variables are probably not not changing.

Reimplemented in Ipopt::NLPBoundsRemover, and Ipopt::TNLPAdapter.

Definition at line 170 of file IpNLP.hpp.

3.115.2.6 virtual void Ipopt::NLP::GetScalingParameters (const SmartPtr< const VectorSpace > *x_space,* const SmartPtr< const VectorSpace > *c_space,* const SmartPtr< const VectorSpace > *d_space,* Number & *obj_scaling,* SmartPtr< Vector > & *x_scaling,* SmartPtr< Vector > & *c_scaling,* SmartPtr< Vector > & *d_scaling*) const [inline], [virtual]

Routines to get the scaling parameters.

These do not need to be overloaded unless the options are set for User scaling

Reimplemented in Ipopt::NLPBoundsRemover, and Ipopt::TNLPAdapter.

Definition at line 188 of file IpNLP.hpp.

3.115.2.7 virtual void Ipopt::NLP::GetQuasiNewtonApproximationSpaces (SmartPtr< VectorSpace > & *approx_space,* SmartPtr< Matrix > & *P_approx*) [inline], [virtual]

Method for obtaining the subspace in which the limited-memory Hessian approximation should be done.

This is only called if the limited-memory Hessian approximation is chosen. Since the Hessian is zero in the space of all variables that appear in the problem functions only linearly, this allows the user to provide a VectorSpace for all nonlinear variables, and an ExpansionMatrix to lift from this VectorSpace to the VectorSpace of the primal variables x. If the returned values are NULL, it is assumed that the Hessian is to be approximated in the space of all x variables. The default instantiation of this method returns NULL, and a user only has to overwrite this method if the approximation is to be done only in a subspace.

Reimplemented in Ipopt::NLPBoundsRemover, and Ipopt::TNLPAdapter.

Definition at line 217 of file IpNLP.hpp.

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

- IpNLP.hpp

3.116 Ipopt::NLPBoundsRemover Class Reference

This is an adaper for an NLP that converts variable bound constraints to inequality constraints.

`#include <IpNLPBoundsRemover.hpp>`

Inheritance diagram for Ipopt::NLPBoundsRemover:

Public Member Functions

- virtual void GetQuasiNewtonApproximationSpaces (SmartPtr< VectorSpace > &approx_space, SmartPtr< Matrix > &P_approx)

  Method for obtaining the subspace in which the limited-memory Hessian approximation should be done.
- SmartPtr< NLP > nlp ()

  Accessor method to the original NLP.

Constructors/Destructors

- NLPBoundsRemover (NLP &nlp, bool allow_twosided_inequalities=false)

  The constructor is given the NLP of which the bounds are to be replaced by inequality constraints.
- virtual ∼NLPBoundsRemover ()

  Default destructor.

NLP Initialization (overload in

derived classes).

- virtual bool ProcessOptions (const OptionsList &options, const std::string &prefix)

  Overload if you want the chance to process options or parameters that may be specific to the NLP.
- virtual bool GetSpaces (SmartPtr< const VectorSpace > &x_space, SmartPtr< const VectorSpace > &c_space, SmartPtr< const VectorSpace > &d_space, SmartPtr< const VectorSpace > &x_l_space, SmartPtr< const MatrixSpace > &px_l_space, SmartPtr< const VectorSpace > &x_u_space, SmartPtr< const MatrixSpace > &px_u_space, SmartPtr< const VectorSpace > &d_l_space, SmartPtr< const MatrixSpace > &pd_l_space, SmartPtr< const VectorSpace > &d_u_space, SmartPtr< const MatrixSpace > &pd_u_space, SmartPtr< const MatrixSpace > &Jac_c_space, SmartPtr< const MatrixSpace > &Jac_d_space, SmartPtr< const SymMatrixSpace > &Hess_lagrangian_space)

  Method for creating the derived vector / matrix types.
- virtual bool GetBoundsInformation (const Matrix &Px_L, Vector &x_L, const Matrix &Px_U, Vector &x_U, const Matrix &Pd_L, Vector &d_L, const Matrix &Pd_U, Vector &d_U)

  Method for obtaining the bounds information.
- virtual bool GetStartingPoint (SmartPtr< Vector > x, bool need_x, SmartPtr< Vector > y_c, bool need_y_c, SmartPtr< Vector > y_d, bool need_y_d, SmartPtr< Vector > z_L, bool need_z_L, SmartPtr< Vector > z_U, bool need_z_U)

  Method for obtaining the starting point for all the iterates.
- virtual bool GetWarmStartIterate (IteratesVector &warm_start_iterate)

  Method for obtaining an entire iterate as a warmstart point.

NLP evaluation routines (overload

in derived classes.

- virtual bool **Eval_f** (const Vector &x, Number &f)
- virtual bool **Eval_grad_f** (const Vector &x, Vector &g_f)
- virtual bool **Eval_c** (const Vector &x, Vector &c)
- virtual bool **Eval_jac_c** (const Vector &x, Matrix &jac_c)

- virtual bool **Eval_d** (const Vector &x, Vector &d)
- virtual bool **Eval_jac_d** (const Vector &x, Matrix &jac_d)
- virtual bool **Eval_h** (const Vector &x, Number obj_factor, const Vector &yc, const Vector &yd, SymMatrix &h)

NLP solution routines. Have default dummy

implementations that can be overloaded.

- virtual void FinalizeSolution (SolverReturn status, const Vector &x, const Vector &z_L, const Vector &z_U, const Vector &c, const Vector &d, const Vector &y_c, const Vector &y_d, Number obj_value, const IpoptData ∗ip_data, IpoptCalculatedQuantities ∗ip_cq)

    This method is called at the very end of the optimization.
- virtual bool IntermediateCallBack (AlgorithmMode mode, Index iter, Number obj_value, Number inf_pr, Number inf_du, Number mu, Number d_norm, Number regularization_size, Number alpha_du, Number alpha_pr, Index ls_trials, const IpoptData ∗ip_data, IpoptCalculatedQuantities ∗ip_cq)

    This method is called once per iteration, after the iteration summary output has been printed.

- virtual void GetScalingParameters (const SmartPtr< const VectorSpace > x_space, const SmartPtr< const VectorSpace > c_space, const SmartPtr< const VectorSpace > d_space, Number &obj_scaling, SmartPtr< Vector > &x_scaling, SmartPtr< Vector > &c_scaling, SmartPtr< Vector > &d_scaling) const

    Routines to get the scaling parameters.

3.116.1 Detailed Description

This is an adaper for an NLP that converts variable bound constraints to inequality constraints.

This is necessary for the version of Ipopt that uses iterative linear solvers. At this point, none of the original inequality constraints is allowed to have both lower and upper bounds. The NLP visible to Ipopt via this adapter will not have any bounds on variables, but have equivalent inequality constraints.

Definition at line 24 of file IpNLPBoundsRemover.hpp.

3.116.2 Constructor & Destructor Documentation

3.116.2.1 Ipopt::NLPBoundsRemover::NLPBoundsRemover (**NLP & *nlp*,** bool ***allow_twosided_inequalities* =** `false`)

The constructor is given the NLP of which the bounds are to be replaced by inequality constriants.

3.116.3 Member Function Documentation

3.116.3.1 virtual bool Ipopt::NLPBoundsRemover::GetSpaces (**SmartPtr< const VectorSpace > & *x_space*, SmartPtr< const VectorSpace > & *c_space*, SmartPtr< const VectorSpace > & *d_space*, SmartPtr< const VectorSpace > & *x_l_space*, SmartPtr< const MatrixSpace > & *px_l_space*, SmartPtr< const VectorSpace > & *x_u_space*, SmartPtr< const MatrixSpace > & *px_u_space*, SmartPtr< const VectorSpace > & *d_l_space*, SmartPtr< const MatrixSpace > & *pd_l_space*, SmartPtr< const VectorSpace > & *d_u_space*, SmartPtr< const MatrixSpace > & *pd_u_space*, SmartPtr< const MatrixSpace > & *Jac_c_space*, SmartPtr< const MatrixSpace > & *Jac_d_space*, SmartPtr< const SymMatrixSpace > & *Hess_lagrangian_space***) `[virtual]`

Method for creating the derived vector / matrix types.

The Hess_lagrangian_space pointer can be NULL if a quasi-Newton options is chosen.

Implements Ipopt::NLP.

3.116.3.2 virtual bool Ipopt::NLPBoundsRemover::GetStartingPoint (SmartPtr< Vector > *x*, bool *need_x*, SmartPtr< Vector > *y_c*, bool *need_y_c*, SmartPtr< Vector > *y_d*, bool *need_y_d*, SmartPtr< Vector > *z_L*, bool *need_z_L*, SmartPtr< Vector > *z_U*, bool *need_z_U*) [virtual]

Method for obtaining the starting point for all the iterates.

ToDo it might not make sense to ask for initial values for v_L and v_U?

Implements Ipopt::NLP.

3.116.3.3 virtual bool Ipopt::NLPBoundsRemover::GetWarmStartIterate (IteratesVector & *warm_start_iterate*) [inline], [virtual]

Method for obtaining an entire iterate as a warmstart point.

The incoming IteratesVector has to be filled. This has not yet been implemented for this adapter.

Reimplemented from Ipopt::NLP.

Definition at line 94 of file IpNLPBoundsRemover.hpp.

3.116.3.4 virtual void Ipopt::NLPBoundsRemover::FinalizeSolution (SolverReturn *status*, const Vector & *x*, const Vector & *z_L*, const Vector & *z_U*, const Vector & *c*, const Vector & *d*, const Vector & *y_c*, const Vector & *y_d*, Number *obj_value*, const IpoptData * *ip_data*, IpoptCalculatedQuantities * *ip_cq*) [virtual]

This method is called at the very end of the optimization.

It provides the final iterate to the user, so that it can be stored as the solution. The status flag indicates the outcome of the optimization, where SolverReturn is defined in IpAlgTypes.hpp.

Reimplemented from Ipopt::NLP.

3.116.3.5 virtual bool Ipopt::NLPBoundsRemover::IntermediateCallBack (AlgorithmMode *mode*, Index *iter*, Number *obj_value*, Number *inf_pr*, Number *inf_du*, Number *mu*, Number *d_norm*, Number *regularization_size*, Number *alpha_du*, Number *alpha_pr*, Index *ls_trials*, const IpoptData * *ip_data*, IpoptCalculatedQuantities * *ip_cq*) [inline], [virtual]

This method is called once per iteration, after the iteration summary output has been printed.

It provides the current information to the user to do with it anything she wants. It also allows the user to ask for a premature termination of the optimization by returning false, in which case Ipopt will terminate with a corresponding return status. The basic information provided in the argument list has the quantities values printed in the iteration summary line. If more information is required, a user can obtain it from the IpData and IpCalculatedQuantities objects. However, note that the provided quantities are all for the problem that Ipopt sees, i.e., the quantities might be scaled, fixed variables might be sorted out, etc. The status indicates things like whether the algorithm is in the restoration phase... In the restoration phase, the dual variables are probably not not changing.

Reimplemented from Ipopt::NLP.

Definition at line 166 of file IpNLPBoundsRemover.hpp.

3.116.3.6 virtual void Ipopt::NLPBoundsRemover::GetScalingParameters (const SmartPtr< const VectorSpace > *x_space*, const SmartPtr< const VectorSpace > *c_space*, const SmartPtr< const VectorSpace > *d_space*, Number & *obj_scaling*, SmartPtr< Vector > & *x_scaling*, SmartPtr< Vector > & *c_scaling*, SmartPtr< Vector > & *d_scaling*) const [virtual]

Routines to get the scaling parameters.

These do not need to be overloaded unless the options are set for User scaling

Reimplemented from Ipopt::NLP.

3.116.3.7 virtual void Ipopt::NLPBoundsRemover::GetQuasiNewtonApproximationSpaces (SmartPtr< VectorSpace > & *approx_space*, SmartPtr< Matrix > & *P_approx*) [inline], [virtual]

Method for obtaining the subspace in which the limited-memory Hessian approximation should be done.

This is only called if the limited-memory Hessian approximation is chosen. Since the Hessian is zero in the space of all variables that appear in the problem functions only linearly, this allows the user to provide a VectorSpace for all nonlinear variables, and an ExpansionMatrix to lift from this VectorSpace to the VectorSpace of the primal variables x. If the returned values are NULL, it is assumed that the Hessian is to be approximated in the space of all x variables. The default instantiation of this method returns NULL, and a user only has to overwrite this method if the approximation is to be done only in a subspace.

Reimplemented from Ipopt::NLP.

Definition at line 211 of file IpNLPBoundsRemover.hpp.

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

- IpNLPBoundsRemover.hpp

3.117 Ipopt::NLPScalingObject Class Reference

This is the abstract base class for problem scaling.

`#include <IpNLPScaling.hpp>`

Inheritance diagram for Ipopt::NLPScalingObject:

Public Member Functions

- bool Initialize (const Journalist &jnlst, const OptionsList &options, const std::string &prefix)

  Method to initialize the options.
- virtual void DetermineScaling (const SmartPtr< const VectorSpace > x_space, const SmartPtr< const VectorSpace > c_space, const SmartPtr< const VectorSpace > d_space, const SmartPtr< const MatrixSpace > jac_c_space, const SmartPtr< const MatrixSpace > jac_d_space, const SmartPtr< const SymMatrixSpace > h_space, SmartPtr< const MatrixSpace > &new_jac_c_space, SmartPtr< const MatrixSpace > &new_jac_d_space, SmartPtr< const SymMatrixSpace > &new_h_space, const Matrix &Px_L, const Vector &x_L, const Matrix &Px_U, const Vector &x_U)=0

  This method is called by the IpoptNLP's at a convenient time to compute and/or read scaling factors.

Constructors/Destructors

- **NLPScalingObject** ()

- virtual ~NLPScalingObject ()

  Default destructor.

- virtual Number apply_obj_scaling (const Number &f)=0

  Methods to map scaled and unscaled matrices.
- virtual Number unapply_obj_scaling (const Number &f)=0

  Returns an obj-unscaled version of the given scalar.
- virtual SmartPtr< Vector > apply_vector_scaling_x_NonConst (const SmartPtr< const Vector > &v)=0

  Returns an x-scaled version of the given vector.
- virtual SmartPtr< const Vector > apply_vector_scaling_x (const SmartPtr< const Vector > &v)=0

  Returns an x-scaled version of the given vector.
- virtual SmartPtr< Vector > unapply_vector_scaling_x_NonConst (const SmartPtr< const Vector > &v)=0

  Returns an x-unscaled version of the given vector.
- virtual SmartPtr< const Vector > unapply_vector_scaling_x (const SmartPtr< const Vector > &v)=0

  Returns an x-unscaled version of the given vector.
- virtual SmartPtr< const Vector > apply_vector_scaling_c (const SmartPtr< const Vector > &v)=0

  Returns an c-scaled version of the given vector.
- virtual SmartPtr< const Vector > unapply_vector_scaling_c (const SmartPtr< const Vector > &v)=0

  Returns an c-unscaled version of the given vector.
- virtual SmartPtr< Vector > apply_vector_scaling_c_NonConst (const SmartPtr< const Vector > &v)=0

  Returns an c-scaled version of the given vector.
- virtual SmartPtr< Vector > unapply_vector_scaling_c_NonConst (const SmartPtr< const Vector > &v)=0

  Returns an c-unscaled version of the given vector.
- virtual SmartPtr< const Vector > apply_vector_scaling_d (const SmartPtr< const Vector > &v)=0

  Returns an d-scaled version of the given vector.
- virtual SmartPtr< const Vector > unapply_vector_scaling_d (const SmartPtr< const Vector > &v)=0

  Returns an d-unscaled version of the given vector.
- virtual SmartPtr< Vector > apply_vector_scaling_d_NonConst (const SmartPtr< const Vector > &v)=0

  Returns an d-scaled version of the given vector.
- virtual SmartPtr< Vector > unapply_vector_scaling_d_NonConst (const SmartPtr< const Vector > &v)=0

  Returns an d-unscaled version of the given vector.
- virtual SmartPtr< const Matrix > apply_jac_c_scaling (SmartPtr< const Matrix > matrix)=0

  Returns a scaled version of the jacobian for c.
- virtual SmartPtr< const Matrix > apply_jac_d_scaling (SmartPtr< const Matrix > matrix)=0

  Returns a scaled version of the jacobian for d If the overloaded method does not create a new matrix, make sure to set the matrix ptr passed in to NULL.
- virtual SmartPtr< const SymMatrix > apply_hessian_scaling (SmartPtr< const SymMatrix > matrix)=0

  Returns a scaled version of the hessian of the lagrangian If the overloaded method does not create a new matrix, make sure to set the matrix ptr passed in to NULL.

- SmartPtr< Vector > apply_vector_scaling_x_LU_NonConst (const Matrix &Px_LU, const SmartPtr< const Vector > &lu, const VectorSpace &x_space)

  Methods for scaling bounds - these wrap those above.
- SmartPtr< const Vector > apply_vector_scaling_x_LU (const Matrix &Px_LU, const SmartPtr< const Vector > &lu, const VectorSpace &x_space)

  Returns an x-scaled vector in the x_L or x_U space.
- SmartPtr< Vector > apply_vector_scaling_d_LU_NonConst (const Matrix &Pd_LU, const SmartPtr< const Vector > &lu, const VectorSpace &d_space)

Returns an d-scaled vector in the d_L or d_U space.

- SmartPtr< const Vector > apply_vector_scaling_d_LU (const Matrix &Pd_LU, const SmartPtr< const Vector > &lu, const VectorSpace &d_space)

  Returns an d-scaled vector in the d_L or d_U space.

- SmartPtr< Vector > unapply_vector_scaling_d_LU_NonConst (const Matrix &Pd_LU, const SmartPtr< const Vector > &lu, const VectorSpace &d_space)

  Returns an d-unscaled vector in the d_L or d_U space.

- SmartPtr< const Vector > unapply_vector_scaling_d_LU (const Matrix &Pd_LU, const SmartPtr< const Vector > &lu, const VectorSpace &d_space)

  Returns an d-unscaled vector in the d_L or d_U space.

- virtual SmartPtr< Vector > apply_grad_obj_scaling_NonConst (const SmartPtr< const Vector > &v)

  Methods for scaling the gradient of the objective - wraps the virtual methods above.

- virtual SmartPtr< const Vector > apply_grad_obj_scaling (const SmartPtr< const Vector > &v)

  Returns a grad_f scaled version (d_f ∗ D_x^{-1}) of the given vector.

- virtual SmartPtr< Vector > unapply_grad_obj_scaling_NonConst (const SmartPtr< const Vector > &v)

  Returns a grad_f unscaled version (d_f ∗ D_x^{-1}) of the given vector.

- virtual SmartPtr< const Vector > unapply_grad_obj_scaling (const SmartPtr< const Vector > &v)

  Returns a grad_f unscaled version (d_f ∗ D_x^{-1}) of the given vector.

Methods for determining whether scaling for entities is

done

- virtual bool have_x_scaling ()=0

  Returns true if the primal x variables are scaled.
- virtual bool have_c_scaling ()=0

  Returns true if the equality constraints are scaled.
- virtual bool have_d_scaling ()=0

  Returns true if the inequality constraints are scaled.

Protected Member Functions

- virtual bool InitializeImpl (const OptionsList &options, const std::string &prefix)=0

  Implementation of the initialization method that has to be overloaded by for each derived class.

- const Journalist & Jnlst () const

  Accessor method for the journalist.

3.117.1 Detailed Description

This is the abstract base class for problem scaling.

It is repsonsible for determining the scaling factors and mapping quantities in and out of scaled and unscaled versions

Definition at line 32 of file IpNLPScaling.hpp.

3.117.2 Member Function Documentation

3.117.2.1 virtual Number Ipopt::NLPScalingObject::apply_obj_scaling (const Number & *f*) [pure virtual]

Methods to map scaled and unscaled matrices.

Returns an obj-scaled version of the given scalar

Implemented in Ipopt::StandardScalingBase.

3.117.2.2 virtual SmartPtr<const Matrix> Ipopt::NLPScalingObject::apply_jac_c_scaling (SmartPtr< const Matrix > *matrix*) [pure virtual]

Returns a scaled version of the jacobian for c.

If the overloaded method does not make a new matrix, make sure to set the matrix ptr passed in to NULL.

Implemented in Ipopt::StandardScalingBase.

3.117.2.3 SmartPtr<Vector> Ipopt::NLPScalingObject::apply_vector_scaling_x_LU_NonConst (const Matrix & *Px_LU*, const SmartPtr< const Vector > & *lu*, const VectorSpace & *x_space*)

Methods for scaling bounds - these wrap those above.

Returns an x-scaled vector in the x_L or x_U space

3.117.2.4 virtual SmartPtr<Vector> Ipopt::NLPScalingObject::apply_grad_obj_scaling_NonConst (const SmartPtr< const Vector > & *v*) [virtual]

Methods for scaling the gradient of the objective - wraps the virtual methods above.

Returns a grad_f scaled version (d_f $*$ D_x^{-1}) of the given vector

3.117.2.5 virtual bool Ipopt::NLPScalingObject::have_x_scaling ( ) [pure virtual]

Returns true if the primal x variables are scaled.

Implemented in Ipopt::StandardScalingBase.

3.117.2.6 virtual bool Ipopt::NLPScalingObject::have_c_scaling ( ) [pure virtual]

Returns true if the equality constraints are scaled.

Implemented in Ipopt::StandardScalingBase.

3.117.2.7 virtual bool Ipopt::NLPScalingObject::have_d_scaling ( ) [pure virtual]

Returns true if the inequality constraints are scaled.

Implemented in Ipopt::StandardScalingBase.

3.117.2.8 virtual bool Ipopt::NLPScalingObject::InitializeImpl (const OptionsList & *options*, const std::string & *prefix*) [protected], [pure virtual]

Implementation of the initialization method that has to be overloaded by for each derived class.

Implemented in Ipopt::StandardScalingBase, Ipopt::EquilibrationScaling, and Ipopt::GradientScaling.

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

- IpNLPScaling.hpp

3.118 Ipopt::NoNLPScalingObject Class Reference

Class implementing the scaling object that doesn't to any scaling.

`#include <IpNLPScaling.hpp>`

Inheritance diagram for Ipopt::NoNLPScalingObject:

Public Member Functions

Constructors/Destructors

- **NoNLPScalingObject** ()
- virtual ∼NoNLPScalingObject ()

    Default destructor.

Protected Member Functions

- virtual void DetermineScalingParametersImpl (const SmartPtr< const VectorSpace > x_space, const SmartPtr< const VectorSpace > c_space, const SmartPtr< const VectorSpace > d_space, const SmartPtr< const MatrixSpace > jac_c_space, const SmartPtr< const MatrixSpace > jac_d_space, const SmartPtr< const SymMatrixSpace > h_space, const Matrix &Px_L, const Vector &x_L, const Matrix &Px_U, const Vector &x_U, Number &df, SmartPtr< Vector > &dx, SmartPtr< Vector > &dc, SmartPtr< Vector > &dd)

    Overloaded from StandardScalingBase.

Additional Inherited Members

3.118.1 Detailed Description

Class implementing the scaling object that doesn't to any scaling.

Definition at line 400 of file IpNLPScaling.hpp.

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

- IpNLPScaling.hpp

3.119 Ipopt::Observer Class Reference

Slight Variation of the Observer Design Pattern.

`#include <IpObserver.hpp>`

Inheritance diagram for Ipopt::Observer:

Public Types

- enum NotifyType

  Enumeration specifying the type of notification.

Public Member Functions

Constructors/Destructors

- Observer ()

  Default Constructor.
- virtual ∼Observer ()

  Default destructor.

Protected Member Functions

- void RequestAttach (NotifyType notify_type, const Subject ∗subject)

  Derived classes should call this method to request an "Attach" to a Subject.
- void RequestDetach (NotifyType notify_type, const Subject ∗subject)

  Derived classes should call this method to request a "Detach" to a Subject.
- virtual void RecieveNotification (NotifyType notify_type, const Subject ∗subject)=0

  Derived classes should overload this method to recieve the requested notification from attached Subjects.

3.119.1 Detailed Description

Slight Variation of the Observer Design Pattern.

This class implements the Observer class of the Observer Design Pattern. An Observer "Attach"es to a Subject, indicating that it would like to be notified of changes in the Subject. Any derived class wishing to recieve notifications from a Subject should inherit off of Observer and overload the protected method, RecieveNotification_(...).

Definition at line 39 of file IpObserver.hpp.

3.119.2 Member Function Documentation

3.119.2.1 void Ipopt::Observer::RequestAttach (**NotifyType** *notify_type,* const **Subject** ∗ *subject*) `[inline]`, `[protected]`

Derived classes should call this method to request an "Attach" to a Subject.

Do not call "Attach" explicitly on the Subject since further processing is done here

Definition at line 219 of file IpObserver.hpp.

3.119.2.2 void Ipopt::Observer::RequestDetach (NotifyType *notify_type*, const Subject ∗ *subject*) [inline], [protected]

Derived classes should call this method to request a "Detach" to a Subject.

Do not call "Detach" explicitly on the Subject since further processing is done here

Definition at line 238 of file IpObserver.hpp.

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

- IpObserver.hpp

3.120 Ipopt::OptimalityErrorConvergenceCheck Class Reference

Brief Class Description.

`#include <IpOptErrorConvCheck.hpp>`

Inheritance diagram for Ipopt::OptimalityErrorConvergenceCheck:

Public Member Functions

- virtual bool InitializeImpl (const OptionsList &options, const std::string &prefix)

    overloaded from AlgorithmStrategyObject
- virtual ConvergenceStatus CheckConvergence (bool call_intermediate_callback=true)

    Overloaded convergence check.
- virtual bool CurrentIsAcceptable ()

    Auxilliary function for testing whether current iterate satisfies the acceptable level of optimality.

Constructors/Destructors

- OptimalityErrorConvergenceCheck ()

    Default Constructor.
- virtual ∼OptimalityErrorConvergenceCheck ()

    Default destructor.

Static Public Member Functions

- static void RegisterOptions (SmartPtr< RegisteredOptions > roptions)

    Methods for IpoptType.

Protected Attributes

Algorithmic parameters

- Index max_iterations_

    Maximal number of iterations.
- Number dual_inf_tol_

    Tolerance on unscaled dual infeasibility.
- Number constr_viol_tol_

Tolerance on unscaled constraint violation.
- Number compl_inf_tol_

  Tolerance on unscaled complementarity.
- Index acceptable_iter_

  Number of iterations with acceptable level of accuracy, after which the algorithm terminates.
- Number acceptable_tol_

  Acceptable tolerance for the problem to terminate earlier if algorithm seems stuck or cycling.
- Number acceptable_dual_inf_tol_

  Acceptable tolerance on unscaled dual infeasibility.
- Number acceptable_constr_viol_tol_

  Acceptable tolerance on unscaled constraint violation.
- Number acceptable_compl_inf_tol_

  Acceptable tolerance on unscaled complementarity.
- Number acceptable_obj_change_tol_

  Acceptable tolerance for relative objective function change from iteratoin to iteration.
- Number diverging_iterates_tol_

  Threshold for primal iterates for divergence test.
- Number mu_target_

  Desired value of the barrier parameter.
- Number max_cpu_time_

  Upper bound on CPU time.

Additional Inherited Members

3.120.1 Detailed Description

Brief Class Description.

Detailed Class Description.

Definition at line 20 of file IpOptErrorConvCheck.hpp.

3.120.2 Member Data Documentation

3.120.2.1 Index Ipopt::OptimalityErrorConvergenceCheck::acceptable_iter_ [protected]

Number of iterations with acceptable level of accuracy, after which the algorithm terminates.

If 0, this heuristic is disabled.

Definition at line 63 of file IpOptErrorConvCheck.hpp.

3.120.2.2 Number Ipopt::OptimalityErrorConvergenceCheck::acceptable_obj_change_tol_ [protected]

Acceptable tolerance for relative objective function change from iteratoin to iteration.

Definition at line 75 of file IpOptErrorConvCheck.hpp.

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

- IpOptErrorConvCheck.hpp

3.121 Ipopt::OptionsList Class Reference

This class stores a list of user set options.

`#include <IpOptionsList.hpp>`

Inheritance diagram for Ipopt::OptionsList:

Public Member Functions

- virtual void clear ()

  Method for clearing all previously set options.
- virtual void PrintList (std::string &list) const

  Get a string with the list of all options (tag, value, counter)
- virtual void PrintUserOptions (std::string &list) const

  Get a string with the list of all options set by the user (tag, value, use/notused).
- virtual bool ReadFromStream (const Journalist &jnlst, std::istream &is)

  Read options from the stream is.

Constructors/Destructors

- **OptionsList** (SmartPtr< RegisteredOptions > reg_options, SmartPtr< Journalist > jnlst)
- **OptionsList** ()
- OptionsList (const OptionsList &copy)

  Copy Constructor.
- virtual ∼OptionsList ()

  Default destructor.
- virtual void operator= (const OptionsList &source)

  Overloaded Equals Operator.

Get / Set Methods

- virtual void **SetRegisteredOptions** (const SmartPtr< RegisteredOptions > reg_options)
- virtual void **SetJournalist** (const SmartPtr< Journalist > jnlst)

Methods for setting options

- virtual bool **SetStringValue** (const std::string &tag, const std::string &value, bool allow_clobber=true, bool dont_print=false)
- virtual bool **SetNumericValue** (const std::string &tag, Number value, bool allow_clobber=true, bool dont_↩ print=false)
- virtual bool **SetIntegerValue** (const std::string &tag, Index value, bool allow_clobber=true, bool dont_↩ print=false)

Methods for setting options only if they have not been

set before

- virtual bool **SetStringValueIfUnset** (const std::string &tag, const std::string &value, bool allow_clobber=true, bool dont_print=false)
- virtual bool **SetNumericValueIfUnset** (const std::string &tag, Number value, bool allow_clobber=true, bool dont_print=false)
- virtual bool **SetIntegerValueIfUnset** (const std::string &tag, Index value, bool allow_clobber=true, bool dont↩ _print=false)

Methods for retrieving values from the options list. If

a tag is not found, the methods return false, and value is set to the default value defined in the registered options.

- virtual bool **GetStringValue** (const std::string &tag, std::string &value, const std::string &prefix) const
- virtual bool **GetEnumValue** (const std::string &tag, Index &value, const std::string &prefix) const
- virtual bool **GetBoolValue** (const std::string &tag, bool &value, const std::string &prefix) const
- virtual bool **GetNumericValue** (const std::string &tag, Number &value, const std::string &prefix) const
- virtual bool **GetIntegerValue** (const std::string &tag, Index &value, const std::string &prefix) const

3.121.1 Detailed Description

This class stores a list of user set options.

Each options is identified by a case-insensitive keyword (tag). Its value is stored internally as a string (always lower case), but for convenience set and get methods are provided to obtain Index and Number type values. For each keyword we also keep track of how often the value of an option has been requested by a get method.

Definition at line 32 of file IpOptionsList.hpp.

3.121.2 Member Function Documentation

3.121.2.1 virtual void Ipopt::OptionsList::PrintUserOptions (std::string & *list*) const [virtual]

Get a string with the list of all options set by the user (tag, value, use/notused).

Here, options with dont_print flag set to true are not printed.

3.121.2.2 virtual bool Ipopt::OptionsList::ReadFromStream (const Journalist & *jnlst,* std::istream & *is*) [virtual]

Read options from the stream is.

Returns false if an error was encountered.

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

- IpOptionsList.hpp

3.122 Ipopt::OrigIpoptNLP Class Reference

This class maps the traditional NLP into something that is more useful by Ipopt.

`#include <IpOrigIpoptNLP.hpp>`

Inheritance diagram for Ipopt::OrigIpoptNLP:

Public Member Functions

- virtual bool Initialize (const Journalist &jnlst, const OptionsList &options, const std::string &prefix)

  Initialize - overloaded from IpoptNLP.
- virtual bool InitializeStructures (SmartPtr< Vector > &x, bool init_x, SmartPtr< Vector > &y_c, bool init_y_c, SmartPtr< Vector > &y_d, bool init_y_d, SmartPtr< Vector > &z_L, bool init_z_L, SmartPtr< Vector > &z_U, bool init_z_U, SmartPtr< Vector > &v_L, SmartPtr< Vector > &v_U)

Initialize (create) structures for the iteration data.

- virtual bool GetWarmStartIterate (IteratesVector &warm_start_iterate)

    Method accessing the GetWarmStartIterate of the NLP.

- virtual void GetSpaces (SmartPtr< const VectorSpace > &x_space, SmartPtr< const VectorSpace > &c_space, SmartPtr< const VectorSpace > &d_space, SmartPtr< const VectorSpace > &x_l_space, SmartPtr< const MatrixSpace > &px_l_space, SmartPtr< const VectorSpace > &x_u_space, SmartPtr< const MatrixSpace > &px_u_space, SmartPtr< const VectorSpace > &d_l_space, SmartPtr< const MatrixSpace > &pd_l_space, SmartPtr< const VectorSpace > &d_u_space, SmartPtr< const MatrixSpace > &pd_u_space, SmartPtr< const MatrixSpace > &Jac_c_space, SmartPtr< const MatrixSpace > &Jac_d_space, SmartPtr< const SymMatrix↩ Space > &Hess_lagrangian_space)

    Accessor method for vector/matrix spaces pointers.

- virtual void AdjustVariableBounds (const Vector &new_x_L, const Vector &new_x_U, const Vector &new_d_L, const Vector &new_d_U)

    Method for adapting the variable bounds.

- SmartPtr< NLP > nlp ()

    Accessor method to the underlying NLP.

Constructors/Destructors

- **OrigIpoptNLP** (const SmartPtr< const Journalist > &jnlst, const SmartPtr< NLP > &nlp, const SmartPtr< NLPScalingObject > &nlp_scaling)
- virtual ~OrigIpoptNLP ()

    Default destructor.

- virtual Number f (const Vector &x)

    Accessor methods for model data.

- virtual Number f (const Vector &x, Number mu)

    Objective value (depending in mu) - incorrect version for OrigIpoptNLP.

- virtual SmartPtr< const Vector > grad_f (const Vector &x)

    Gradient of the objective.

- virtual SmartPtr< const Vector > grad_f (const Vector &x, Number mu)

    Gradient of the objective (depending in mu) - incorrect version for OrigIpoptNLP.

- virtual SmartPtr< const Vector > c (const Vector &x)

    Equality constraint residual.

- virtual SmartPtr< const Matrix > jac_c (const Vector &x)

    Jacobian Matrix for equality constraints.

- virtual SmartPtr< const Vector > d (const Vector &x)

    Inequality constraint residual (reformulated as equalities with slacks.

- virtual SmartPtr< const Matrix > jac_d (const Vector &x)

    Jacobian Matrix for inequality constraints.

- virtual SmartPtr< const SymMatrix > h (const Vector &x, Number obj_factor, const Vector &yc, const Vector &yd)

    Hessian of the Lagrangian.

- virtual SmartPtr< const SymMatrix > h (const Vector &x, Number obj_factor, const Vector &yc, const Vector &yd, Number mu)

    Hessian of the Lagrangian (depending in mu) - incorrect version for OrigIpoptNLP.

- virtual SmartPtr< const SymMatrix > uninitialized_h ()

    Provides a Hessian matrix from the correct matrix space with uninitialized values.

- virtual SmartPtr< const Vector > x_L () const

    Lower bounds on x.

- virtual SmartPtr< const Matrix > Px_L () const

    Permutation matrix (x_L_ -> x)
- virtual SmartPtr< const Vector > x_U () const

    Upper bounds on x.
- virtual SmartPtr< const Matrix > Px_U () const

    Permutation matrix (x_U_ -> x.
- virtual SmartPtr< const Vector > d_L () const

    Lower bounds on d.
- virtual SmartPtr< const Matrix > Pd_L () const

    Permutation matrix (d_L_ -> d)
- virtual SmartPtr< const Vector > d_U () const

    Upper bounds on d.
- virtual SmartPtr< const Matrix > Pd_U () const

    Permutation matrix (d_U_ -> d.
- virtual SmartPtr< const SymMatrixSpace > HessianMatrixSpace () const

    Accessor method to obtain the MatrixSpace for the Hessian matrix (or it's approximation)
- virtual SmartPtr< const VectorSpace > x_space () const

    x_space

Counters for the number of function evaluations.

- virtual Index **f_evals** () const
- virtual Index **grad_f_evals** () const
- virtual Index **c_evals** () const
- virtual Index **jac_c_evals** () const
- virtual Index **d_evals** () const
- virtual Index **jac_d_evals** () const
- virtual Index **h_evals** () const

- void FinalizeSolution (SolverReturn status, const Vector &x, const Vector &z_L, const Vector &z_U, const Vector &c, const Vector &d, const Vector &y_c, const Vector &y_d, Number obj_value, const IpoptData ∗ip_data, IpoptCalculatedQuantities ∗ip_cq)

    Solution Routines - overloaded from IpoptNLP.

Methods related to function evaluation timing.

- void ResetTimes ()

    Reset the timing statistics.
- void **PrintTimingStatistics** (Journalist &jnlst, EJournalLevel level, EJournalCategory category) const
- const TimedTask & **f_eval_time** () const
- const TimedTask & **grad_f_eval_time** () const
- const TimedTask & **c_eval_time** () const
- const TimedTask & **jac_c_eval_time** () const
- const TimedTask & **d_eval_time** () const
- const TimedTask & **jac_d_eval_time** () const
- const TimedTask & **h_eval_time** () const
- Number **TotalFunctionEvaluationCpuTime** () const
- Number **TotalFunctionEvaluationSysTime** () const
- Number **TotalFunctionEvaluationWallclockTime** () const

Static Public Member Functions

Methods for IpoptType

- static void RegisterOptions (SmartPtr< RegisteredOptions > roptions)
  Called by IpoptType to register the options.

3.122.1 Detailed Description

This class maps the traditional NLP into something that is more useful by Ipopt.

This class takes care of storing the calculated model results, handles caching, and (some day) takes care of addition of slacks.

Definition at line 37 of file IpOrigIpoptNLP.hpp.

3.122.2 Member Function Documentation

3.122.2.1 virtual Number Ipopt::OrigIpoptNLP::f (const Vector & *x*) `[virtual]`

Accessor methods for model data.

Objective value

Implements Ipopt::IpoptNLP.

3.122.2.2 virtual SmartPtr<const SymMatrix> Ipopt::OrigIpoptNLP::uninitialized_h ( ) `[virtual]`

Provides a Hessian matrix from the correct matrix space with uninitialized values.

This can be used in LeastSquareMults to obtain a "zero Hessian".

Implements Ipopt::IpoptNLP.

3.122.2.3 virtual void Ipopt::OrigIpoptNLP::AdjustVariableBounds (const Vector & *new_x_L*, const Vector & *new_x_U*, const Vector & *new_d_L*, const Vector & *new_d_U*) `[virtual]`

Method for adapting the variable bounds.

This is called if slacks are becoming too small

Implements Ipopt::IpoptNLP.

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

- IpOrigIpoptNLP.hpp

3.123 Ipopt::OrigIterationOutput Class Reference

Class for the iteration summary output for the original NLP.

`#include <IpOrigIterationOutput.hpp>`

Inheritance diagram for Ipopt::OrigIterationOutput:

Public Member Functions

- virtual bool InitializeImpl (const OptionsList &options, const std::string &prefix)

  overloaded from AlgorithmStrategyObject
- virtual void WriteOutput ()

  Method to do all the summary output per iteration.

Constructors/Destructors

- OrigIterationOutput ()

  Default Constructor.
- virtual ∼OrigIterationOutput ()

  Default destructor.

Static Public Member Functions

- static void RegisterOptions (SmartPtr< RegisteredOptions > roptions)

  Methods for OptionsList.

Additional Inherited Members

3.123.1 Detailed Description

Class for the iteration summary output for the original NLP.

Definition at line 19 of file IpOrigIterationOutput.hpp.

3.123.2 Member Function Documentation

3.123.2.1 virtual void Ipopt::OrigIterationOutput::WriteOutput ( ) [virtual]

Method to do all the summary output per iteration.

This include the one-line summary output as well as writing the details about the iterates if desired

Implements Ipopt::IterationOutput.

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

- IpOrigIterationOutput.hpp

3.124 Ipopt::PardisoSolverInterface Class Reference

Interface to the linear solver Pardiso, derived from SparseSymLinearSolverInterface.

`#include <IpPardisoSolverInterface.hpp>`

Inheritance diagram for Ipopt::PardisoSolverInterface:

Public Member Functions

- bool InitializeImpl (const OptionsList &options, const std::string &prefix)

  overloaded from AlgorithmStrategyObject

Constructor/Destructor

- PardisoSolverInterface ()

  Constructor.
- virtual ∼PardisoSolverInterface ()

  Destructor.

Methods for requesting solution of the linear system.

- virtual ESymSolverStatus InitializeStructure (Index dim, Index nonzeros, const Index ∗ia, const Index ∗ja)

  Method for initializing internal stuctures.
- virtual double ∗ GetValuesArrayPtr ()

  Method returing an internal array into which the nonzero elements are to be stored.
- virtual ESymSolverStatus MultiSolve (bool new_matrix, const Index ∗ia, const Index ∗ja, Index nrhs, double ∗rhs_vals, bool check_NegEVals, Index numberOfNegEVals)

  Solve operation for multiple right hand sides.
- virtual Index NumberOfNegEVals () const

  Number of negative eigenvalues detected during last factorization.

- virtual bool IncreaseQuality ()

  Request to increase quality of solution for next solve.
- virtual bool ProvidesInertia () const

  Query whether inertia is computed by linear solver.
- EMatrixFormat MatrixFormat () const

  Query of requested matrix type that the linear solver understands.

Static Public Member Functions

- static void RegisterOptions (SmartPtr< RegisteredOptions > roptions)

  Methods for IpoptType.

Additional Inherited Members

3.124.1 Detailed Description

Interface to the linear solver Pardiso, derived from SparseSymLinearSolverInterface.

For details, see description of SparseSymLinearSolverInterface base class.

Definition at line 24 of file IpPardisoSolverInterface.hpp.

3.124.2 Member Function Documentation

3.124.2.1 virtual ESymSolverStatus Ipopt::PardisoSolverInterface::InitializeStructure (Index *dim,* Index *nonzeros,* const Index ∗ *ia,* const Index ∗ *ja*) [virtual]

Method for initializing internal stuctures.

Implements Ipopt::SparseSymLinearSolverInterface.

3.124.2.2 virtual double∗ Ipopt::PardisoSolverInterface::GetValuesArrayPtr ( ) [virtual]

Method returing an internal array into which the nonzero elements are to be stored.

Implements Ipopt::SparseSymLinearSolverInterface.

3.124.2.3 virtual ESymSolverStatus Ipopt::PardisoSolverInterface::MultiSolve (bool *new_matrix*, const Index ∗ *ia*, const Index ∗ *ja*, Index *nrhs*, double ∗ *rhs_vals*, bool *check_NegEVals*, Index *numberOfNegEVals*) [virtual]

Solve operation for multiple right hand sides.

Implements Ipopt::SparseSymLinearSolverInterface.

3.124.2.4 virtual bool Ipopt::PardisoSolverInterface::ProvidesInertia ( ) const [inline], [virtual]

Query whether inertia is computed by linear solver.

Returns true, if linear solver provides inertia.

Implements Ipopt::SparseSymLinearSolverInterface.

Definition at line 76 of file IpPardisoSolverInterface.hpp.

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

- IpPardisoSolverInterface.hpp

3.125 Ipopt::PDFullSpaceSolver Class Reference

This is the implemetation of the Primal-Dual System, using the full space approach with a direct linear solver.

`#include <IpPDFullSpaceSolver.hpp>`

Inheritance diagram for Ipopt::PDFullSpaceSolver:

Public Member Functions

- bool InitializeImpl (const OptionsList &options, const std::string &prefix)

  overloaded from AlgorithmStrategyObject
- virtual bool Solve (Number alpha, Number beta, const IteratesVector &rhs, IteratesVector &res, bool allow_↩ inexact=false, bool improve_solution=false)

  Solve the primal dual system, given one right hand side.

/Destructor

- PDFullSpaceSolver (AugSystemSolver &augSysSolver, PDPerturbationHandler &perturbHandler)

  Constructor that takes in the Augmented System solver that is to be used inside.
- virtual ∼PDFullSpaceSolver ()

  Default destructor.

Static Public Member Functions

- static void RegisterOptions (SmartPtr< RegisteredOptions > roptions)

  Methods for IpoptType.

Additional Inherited Members

3.125.1 Detailed Description

This is the implemetation of the Primal-Dual System, using the full space approach with a direct linear solver.

A note on the iterative refinement: We perform at least min_refinement_steps number of iterative refinement steps. If after one iterative refinement the quality of the solution (defined in ResidualRatio) does not improve or the maximal number of iterative refinement steps is exceeded before the tolerance residual_ratio_max_ is satisfied, we first ask the linear solver to solve the system more accurately (e.g. by increasing the pivot tolerance). If that doesn't help or is not possible, we treat the system, as if it is singular (i.e. increase delta's).

Definition at line 32 of file IpPDFullSpaceSolver.hpp.

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

- IpPDFullSpaceSolver.hpp

3.126 Ipopt::PDPerturbationHandler Class Reference

Class for handling the perturbation factors delta_x, delta_s, delta_c, and delta_d in the primal dual system.

`#include <IpPDPerturbationHandler.hpp>`

Inheritance diagram for Ipopt::PDPerturbationHandler:

Public Member Functions

- virtual bool InitializeImpl (const OptionsList &options, const std::string &prefix)
  Implementation of the initialization method that has to be overloaded by for each derived class.
- virtual bool ConsiderNewSystem (Number &delta_x, Number &delta_s, Number &delta_c, Number &delta_d)
  This method must be called for each new matrix, and before any other method for generating perturbation factors.
- virtual bool PerturbForSingularity (Number &delta_x, Number &delta_s, Number &delta_c, Number &delta_d)
  This method returns pertubation factors for the case when the most recent factorization resulted in a singular matrix.
- virtual bool PerturbForWrongInertia (Number &delta_x, Number &delta_s, Number &delta_c, Number &delta_d)
  This method returns pertubation factors for the case when the most recent factorization resulted in a matrix with an incorrect number of negative eigenvalues.
- virtual void CurrentPerturbation (Number &delta_x, Number &delta_s, Number &delta_c, Number &delta_d)
  Just return the perturbation values that have been determined most recently.

Constructors/Destructors

- PDPerturbationHandler ()
  Default Constructor.
- virtual ∼PDPerturbationHandler ()
  Default destructor.

Static Public Member Functions

- static void RegisterOptions (SmartPtr< RegisteredOptions > roptions)
  Methods for IpoptType.

Protected Member Functions

Default Compiler Generated Methods

(Hidden to avoid implicit creation/calling).

These methods are not implemented and we do not want the compiler to implement them for us, so we declare them private and do not define them. This ensures that they will not be implicitly created/called.

- PDPerturbationHandler (const PDPerturbationHandler &)

  Copy Constructor.
- void operator= (const PDPerturbationHandler &)

  Overloaded Equals Operator.

Auxilliary methods

- bool get_deltas_for_wrong_inertia (Number &delta_x, Number &delta_s, Number &delta_c, Number &delta_d)

  Internal version of PerturbForWrongInertia with the difference, that finalize_test is not called.
- void finalize_test ()

  This method is call whenever a matrix had been factorization and is not singular.
- Number delta_cd ()

  Compute perturbation value for constraints.

Protected Attributes

- bool get_deltas_for_wrong_inertia_called_

  Flag indicating if for the given matrix the perturb for wrong inertia method has already been called.

Size of the most recent non-zero perturbation.

- Number delta_x_last_

  The last nonzero value for delta_x.
- Number delta_s_last_

  The last nonzero value for delta_s.
- Number delta_c_last_

  The last nonzero value for delta_c.
- Number delta_d_last_

  The last nonzero value for delta_d.

Size of the most recently suggested perturbation for the

current matrix.

- Number delta_x_curr_

  The current value for delta_x.
- Number delta_s_curr_

  The current value for delta_s.
- Number delta_c_curr_

  The current value for delta_c.
- Number delta_d_curr_

  The current value for delta_d.

Algorithmic parameters.

- Number delta_xs_max_

Maximal perturbation for x and s.

- Number delta_xs_min_

  Smallest possible perturbation for x and s.

- Number delta_xs_first_inc_fact_

  Increase factor for delta_xs for first required perturbation.

- Number delta_xs_inc_fact_

  Increase factor for delta_xs for later perturbations.

- Number delta_xs_dec_fact_

  Decrease factor for delta_xs for later perturbations.

- Number delta_xs_init_

  Very first trial value for delta_xs perturbation.

- Number delta_cd_val_

  Size of perturbation for c and d blocks.

- Number delta_cd_exp_

  Exponent on mu in formula for of perturbation for c and d blocks.

- bool reset_last_

  Flag indicating whether the new values are based on the perturbations in the last iteration or in the more recent iteration in which a perturbation was done.

- Index degen_iters_max_

  Required number of iterations for degeneracy conclusions.

- bool perturb_always_cd_

  Flag indicating that the delta_c, delta_d perturbation should always be used.

Handling structural degeneracy

- enum DegenType

  Type for degeneracy flags.

- enum TrialStatus

  Status of current trial configuration.

- DegenType hess_degenerate_

  Flag indicating whether the reduced Hessian matrix is thought to be structurally singular.

- DegenType jac_degenerate_

  Flag indicating whether the Jacobian of the constraints is thought to be structurally rank-deficient.

- Index degen_iters_

  Flag counting matrices in which degeneracy was observed in the first successive iterations.

- TrialStatus test_status_

  Current status.

3.126.1 Detailed Description

Class for handling the perturbation factors delta_x, delta_s, delta_c, and delta_d in the primal dual system.

This class is used by the PDFullSpaceSolver to handle the cases where the primal-dual system is singular or has the wrong inertia. The perturbation factors are obtained based on simple heuristics, taking into account the size of previous perturbations.

Definition at line 24 of file IpPDPerturbationHandler.hpp.

3.126.2 Member Function Documentation

3.126.2.1 virtual bool Ipopt::PDPerturbationHandler::InitializeImpl (const OptionsList & *options*, const std::string & *prefix*) [virtual]

Implementation of the initialization method that has to be overloaded by for each derived class.

Implements Ipopt::AlgorithmStrategyObject.

Reimplemented in Ipopt::CGPerturbationHandler.

3.126.2.2 virtual bool Ipopt::PDPerturbationHandler::ConsiderNewSystem (Number & *delta_x*, Number & *delta_s*, Number & *delta_c*, Number & *delta_d*) [virtual]

This method must be called for each new matrix, and before any other method for generating perturbation factors.

Usually, the returned perturbation factors are zero, but if the system is thought to be structurally singular, they might be positive. If the return value is false, no suitable perturbation could be found.

Reimplemented in Ipopt::CGPerturbationHandler.

3.126.2.3 virtual bool Ipopt::PDPerturbationHandler::PerturbForSingularity (Number & *delta_x*, Number & *delta_s*, Number & *delta_c*, Number & *delta_d*) [virtual]

This method returns pertubation factors for the case when the most recent factorization resulted in a singular matrix.

If the return value is false, no suitable perturbation could be found.

Reimplemented in Ipopt::CGPerturbationHandler.

3.126.2.4 virtual bool Ipopt::PDPerturbationHandler::PerturbForWrongInertia (Number & *delta_x*, Number & *delta_s*, Number & *delta_c*, Number & *delta_d*) [virtual]

This method returns pertubation factors for the case when the most recent factorization resulted in a matrix with an incorrect number of negative eigenvalues.

If the return value is false, no suitable perturbation could be found.

Reimplemented in Ipopt::CGPerturbationHandler.

3.126.2.5 bool Ipopt::PDPerturbationHandler::get_deltas_for_wrong_inertia (Number & *delta_x*, Number & *delta_s*, Number & *delta_c*, Number & *delta_d*) [protected]

Internal version of PerturbForWrongInertia with the difference, that finalize_test is not called.

Returns false if the delta_x and delta_s parameters become too large.

3.126.2.6 void Ipopt::PDPerturbationHandler::finalize_test ( ) [protected]

This method is call whenever a matrix had been factorization and is not singular.

In here, we can evaluate the outcome of the deneracy test heuristics.

3.126.3 Member Data Documentation

3.126.3.1 bool Ipopt::PDPerturbationHandler::get_deltas_for_wrong_inertia_called_ [protected]

Flag indicating if for the given matrix the perturb for wrong inertia method has already been called.

Definition at line 116 of file IpPDPerturbationHandler.hpp.

3.126.3.2 DegenType Ipopt::PDPerturbationHandler::hess_degenerate_ [protected]

Flag indicating whether the reduced Hessian matrix is thought to be structurally singular.

Definition at line 130 of file IpPDPerturbationHandler.hpp.

3.126.3.3 DegenType Ipopt::PDPerturbationHandler::jac_degenerate_ [protected]

Flag indicating whether the Jacobian of the constraints is thought to be structurally rank-deficient.

Definition at line 134 of file IpPDPerturbationHandler.hpp.

3.126.3.4 Index Ipopt::PDPerturbationHandler::degen_iters_ [protected]

Flag counting matrices in which degeneracy was observed in the first successive iterations.

-1 means that there was a non-degenerate (unperturbed) matrix at some point.

Definition at line 139 of file IpPDPerturbationHandler.hpp.

3.126.3.5 Number Ipopt::PDPerturbationHandler::delta_xs_max_ [protected]

Maximal perturbation for x and s.

Definition at line 158 of file IpPDPerturbationHandler.hpp.

3.126.3.6 Number Ipopt::PDPerturbationHandler::delta_xs_min_ [protected]

Smallest possible perturbation for x and s.

Definition at line 160 of file IpPDPerturbationHandler.hpp.

3.126.3.7 Number Ipopt::PDPerturbationHandler::delta_xs_first_inc_fact_ [protected]

Increase factor for delta_xs for first required perturbation.

Definition at line 162 of file IpPDPerturbationHandler.hpp.

3.126.3.8 Number Ipopt::PDPerturbationHandler::delta_xs_inc_fact_ [protected]

Increase factor for delta_xs for later perturbations.

Definition at line 164 of file IpPDPerturbationHandler.hpp.

3.126.3.9 Number Ipopt::PDPerturbationHandler::delta_xs_dec_fact_ [protected]

Decrease factor for delta_xs for later perturbations.

Definition at line 166 of file IpPDPerturbationHandler.hpp.

3.126.3.10 Number Ipopt::PDPerturbationHandler::delta_xs_init_ [protected]

Very first trial value for delta_xs perturbation.

Definition at line 168 of file IpPDPerturbationHandler.hpp.

3.126.3.11 Number Ipopt::PDPerturbationHandler::delta_cd_val_ [protected]

Size of perturbation for c and d blocks.

Definition at line 170 of file IpPDPerturbationHandler.hpp.

3.126.3.12 Number Ipopt::PDPerturbationHandler::delta_cd_exp_ [protected]

Exponent on mu in formula for of perturbation for c and d blocks.

Definition at line 172 of file IpPDPerturbationHandler.hpp.

3.126.3.13 bool Ipopt::PDPerturbationHandler::reset_last_ [protected]

Flag indicating whether the new values are based on the perturbations in the last iteration or in the more recent iteration in which a perturbation was done.

Definition at line 176 of file IpPDPerturbationHandler.hpp.

3.126.3.14 Index Ipopt::PDPerturbationHandler::degen_iters_max_ [protected]

Required number of iterations for degeneracy conclusions.

Definition at line 178 of file IpPDPerturbationHandler.hpp.

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

- IpPDPerturbationHandler.hpp

3.127 Ipopt::PDSearchDirCalculator Class Reference

Implementation of the search direction calculator that computes the pure primal dual step for the current barrier parameter.

`#include <IpPDSearchDirCalc.hpp>`

Inheritance diagram for Ipopt::PDSearchDirCalculator:

Public Member Functions

- virtual bool InitializeImpl (const OptionsList &options, const std::string &prefix)

overloaded from AlgorithmStrategyObject

- virtual bool ComputeSearchDirection ()

  Method for computing the search direction.
- SmartPtr< PDSystemSolver > PDSolver ()

  Method to return the pd_solver for additional processing.

Constructors/Destructors

- PDSearchDirCalculator (const SmartPtr< PDSystemSolver > &pd_solver)

  Constructor.
- virtual ∼PDSearchDirCalculator ()

  Default destructor.

Static Public Member Functions

- static void RegisterOptions (const SmartPtr< RegisteredOptions > &roptions)

  Methods for IpoptType.

Additional Inherited Members

3.127.1 Detailed Description

Implementation of the search direction calculator that computes the pure primal dual step for the current barrier parameter.

Definition at line 21 of file IpPDSearchDirCalc.hpp.

3.127.2 Member Function Documentation

3.127.2.1 virtual bool Ipopt::PDSearchDirCalculator::ComputeSearchDirection ( ) [virtual]

Method for computing the search direction.

The computed direction is stored in IpData().delta().

Implements Ipopt::SearchDirectionCalculator.

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

- IpPDSearchDirCalc.hpp

3.128 Ipopt::PDSystemSolver Class Reference

Pure Primal Dual System Solver Base Class.

`#include <IpPDSystemSolver.hpp>`

Inheritance diagram for Ipopt::PDSystemSolver:

Public Member Functions

- virtual bool InitializeImpl (const OptionsList &options, const std::string &prefix)=0

    overloaded from AlgorithmStrategyObject

- virtual bool Solve (Number alpha, Number beta, const IteratesVector &rhs, IteratesVector &res, bool allow_↩ inexact=false, bool improve_solution=false)=0

    Solve the primal dual system, given one right hand side.

/Destructor

- PDSystemSolver ()

    Default Constructor.

- virtual ∼PDSystemSolver ()

    Default destructor.

Additional Inherited Members

3.128.1 Detailed Description

Pure Primal Dual System Solver Base Class.

This is the base class for all derived Primal-Dual System Solver Types.

Here, we understand the primal-dual system as the following linear system:

$$\left[\begin{array}{cccccccc} W & 0 & J_c^T & J_d^T & -P_L^x & P_U^x & 0 & 0 \\ 0 & 0 & 0 & -I & 0 & 0 & -P_L^d & P_U^d \\ J_c & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ J_d & -I & 0 & 0 & 0 & 0 & 0 & 0 \\ Z_L(P_L^x)^T & 0 & 0 & 0 & Sl_L^x & 0 & 0 & 0 \\ -Z_U(P_U^x)^T & 0 & 0 & 0 & 0 & Sl_U^x & 0 & 0 \\ 0 & V_L(P_L^d)^T & 0 & 0 & 0 & 0 & Sl_L^s & 0 \\ 0 & -V_U(P_U^d)^T & 0 & 0 & 0 & 0 & 0 & Sl_U^s \end{array}\right] \left(\begin{array}{c} sol_x \\ sol_s \\ sol_c \\ sol_d \\ sol_L^z \\ sol_U^z \\ sol_L^v \\ sol_U^v \end{array}\right) = \left(\begin{array}{c} rhs_x \\ rhs_s \\ rhs_c \\ rhs_d \\ rhs_L^z \\ rhs_U^z \\ rhs_L^v \\ rhs_U^v \end{array}\right)$$

Here, $Sl_L^x = (P_L^x)^T x - x_L$, $Sl_U^x = x_U - (P_U^x)^T x$, $Sl_L^d = (P_L^d)^T d(x) - d_L$, $Sl_U^d = d_U - (P_U^d)^T d(x)$. The results returned to the caller is $res = \alpha * sol + \beta * res$.

The solution of this linear system (in order to compute the search direction of the algorthim) usually requires a considerable amount of computation time. Therefore, it is important to tailor the solution of this system to the characteristics of the problem. The purpose of this base class is to provide a generic interface to the algorithm that it can use whenever it requires a solution of the above system. Particular implementation can then be written to provide the methods defined here.

It is implicitly assumed here, that the upper left 2 by 2 block is possibly modified (implicitly or explicitly) so that its projection onto the null space of the overall constraint Jacobian $\left[\begin{array}{cc} J_c & 0 \\ J_d & -I \end{array}\right]$ is positive definite. This is necessary to guarantee certain descent properties of the resulting search direction. For example, in the full space implementation, a multiple of the identity might be added to the upper left 2 by 2 block.

Note that the Solve method might be called several times for different right hand sides, but with identical data. Therefore, if possible, an implemetation of PDSystem should check whether the incoming data has changed, and not redo factorization etc. unless necessary.

Definition at line 76 of file IpPDSystemSolver.hpp.

3.128.2 Member Function Documentation

3.128.2.1 virtual bool Ipopt::PDSystemSolver::Solve (Number *alpha*, Number *beta*, const IteratesVector & *rhs*, IteratesVector & *res*, bool *allow_inexact* = false, bool *improve_solution* = false) [pure virtual]

Solve the primal dual system, given one right hand side.

If the flag allow_inexact is set to true, it is not necessary to solve the system to best accuracy; for example, we don't want iterative refinement during the computation of the second order correction. On the other hand, if improve_solution is true, the solution given in res should be improved (here beta has to be zero, and res is assume to be the solution for the system using rhs, without the factor alpha...). THe return value is false, if a solution could not be computed (for example, when the Hessian regularization parameter becomes too large.)

Implemented in Ipopt::PDFullSpaceSolver.

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

- IpPDSystemSolver.hpp

3.129 Ipopt::PenaltyLSAcceptor Class Reference

Penalty function line search.

`#include <IpPenaltyLSAcceptor.hpp>`

Inheritance diagram for Ipopt::PenaltyLSAcceptor:

Public Member Functions

- virtual bool InitializeImpl (const OptionsList &options, const std::string &prefix)

  InitializeImpl - overloaded from AlgorithmStrategyObject.
- virtual void Reset ()

  Reset the acceptor.
- virtual void InitThisLineSearch (bool in_watchdog)

  Initialization for the next line search.
- virtual void PrepareRestoPhaseStart ()

  Method that is called before the restoration phase is called.
- virtual Number CalculateAlphaMin ()

  Method returning the lower bound on the trial step sizes.
- virtual bool CheckAcceptabilityOfTrialPoint (Number alpha_primal)

  Method for checking if current trial point is acceptable.
- virtual bool TrySecondOrderCorrection (Number alpha_primal_test, Number &alpha_primal, SmartPtr< IteratesVector > &actual_delta)

  Try a second order correction for the constraints.
- virtual bool TryCorrector (Number alpha_primal_test, Number &alpha_primal, SmartPtr< IteratesVector > &actual_delta)

  Try higher order corrector (for fast local convergence).
- virtual char UpdateForNextIteration (Number alpha_primal_test)

  Method for ending the current line search.
- virtual void StartWatchDog ()

  Method for setting internal data if the watchdog procedure is started.

- virtual void StopWatchDog ()

  Method for setting internal data if the watchdog procedure is stopped.

Constructors/Destructors

- PenaltyLSAcceptor (const SmartPtr< PDSystemSolver > &pd_solver)

  Constructor.
- virtual ∼PenaltyLSAcceptor ()

  Default destructor.

Trial Point Accepting Methods. Used internally to check certain

acceptability criteria and used externally (by the restoration phase convergence check object, for instance)

- bool IsAcceptableToCurrentIterate (Number trial_barr, Number trial_theta, bool called_from_restoration=false) const

  Checks if a trial point is acceptable to the current iterate.

Static Public Member Functions

- static void RegisterOptions (SmartPtr< RegisteredOptions > roptions)

  Methods for OptionsList.

Additional Inherited Members

3.129.1 Detailed Description

Penalty function line search.

This class implements the penalty function line search procedure as proposed by Waltz, Morales, Nocedal, Orban.

Definition at line 23 of file IpPenaltyLSAcceptor.hpp.

3.129.2 Constructor & Destructor Documentation

3.129.2.1 Ipopt::PenaltyLSAcceptor::PenaltyLSAcceptor (const SmartPtr< PDSystemSolver > & *pd_solver*)

Constructor.

The PDSystemSolver object only needs to be provided (i.e. not NULL) if second order correction or corrector steps are to be used.

3.129.3 Member Function Documentation

3.129.3.1 virtual void Ipopt::PenaltyLSAcceptor::Reset ( ) [virtual]

Reset the acceptor.

This function should be called if all previous information should be discarded when the line search is performed the next time. For example, this method should be called if the barrier parameter is changed.

Implements Ipopt::BacktrackingLSAcceptor.

3.129.3.2 virtual void Ipopt::PenaltyLSAcceptor::InitThisLineSearch (bool *in_watchdog*) [virtual]

Initialization for the next line search.

The flag in_watchdog indicates if we are currently in an active watchdog procedure.

Implements Ipopt::BacktrackingLSAcceptor.

3.129.3.3 virtual void Ipopt::PenaltyLSAcceptor::PrepareRestoPhaseStart ( ) [virtual]

Method that is called before the restoration phase is called.

Here, we can set up things that are required in the termination test for the restoration phase.

Implements Ipopt::BacktrackingLSAcceptor.

3.129.3.4 virtual Number Ipopt::PenaltyLSAcceptor::CalculateAlphaMin ( ) [virtual]

Method returning the lower bound on the trial step sizes.

Implements Ipopt::BacktrackingLSAcceptor.

3.129.3.5 virtual bool Ipopt::PenaltyLSAcceptor::CheckAcceptabilityOfTrialPoint (Number *alpha_primal*) [virtual]

Method for checking if current trial point is acceptable.

It is assumed that the delta information in ip_data is the search direction used in criteria. The primal trial point has to be set before the call.

Implements Ipopt::BacktrackingLSAcceptor.

3.129.3.6 virtual bool Ipopt::PenaltyLSAcceptor::TrySecondOrderCorrection (Number *alpha_primal_test,* Number & *alpha_primal,* SmartPtr< IteratesVector > & *actual_delta*) [virtual]

Try a second order correction for the constraints.

If the first trial step (with incoming alpha_primal) has been reject, this tries up to max_soc_ second order corrections for the constraints. Here, alpha_primal_test is the step size that has to be used in the penalty function acceptance tests. On output actual_delta_ has been set to the step including the second order correction if it has been accepted, otherwise it is unchanged. If the SOC step has been accepted, alpha_primal has the fraction-to-the-boundary value for the SOC step on output. The return value is true, if a SOC step has been accepted.

Implements Ipopt::BacktrackingLSAcceptor.

3.129.3.7 virtual bool Ipopt::PenaltyLSAcceptor::TryCorrector (Number *alpha_primal_test,* Number & *alpha_primal,* SmartPtr< IteratesVector > & *actual_delta*) [virtual]

Try higher order corrector (for fast local convergence).

In contrast to a second order correction step, which tries to make an unacceptable point acceptable by improving constraint violation, this corrector step is tried even if the regular primal-dual step is acceptable.

Implements Ipopt::BacktrackingLSAcceptor.

3.129.3.8 virtual char Ipopt::PenaltyLSAcceptor::UpdateForNextIteration (Number *alpha_primal_test*) [virtual]

Method for ending the current line search.

When it is called, the internal data should be updates. alpha_primal_test is the value of alpha that has been used for in the acceptence test ealier.

Implements Ipopt::BacktrackingLSAcceptor.

3.129.3.9 virtual void Ipopt::PenaltyLSAcceptor::StartWatchDog ( ) [virtual]

Method for setting internal data if the watchdog procedure is started.

Implements Ipopt::BacktrackingLSAcceptor.

3.129.3.10 virtual void Ipopt::PenaltyLSAcceptor::StopWatchDog ( ) [virtual]

Method for setting internal data if the watchdog procedure is stopped.

Implements Ipopt::BacktrackingLSAcceptor.

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

- IpPenaltyLSAcceptor.hpp

3.130 Ipopt::PiecewisePenalty Class Reference

Class for the Piecewise Penalty.

`#include <IpPiecewisePenalty.hpp>`

Public Member Functions

- void Clear ()

  Delete all Piecewise Penalty entries.
- void Print (const Journalist &jnlst)

  Print current Piecewise Penalty entries.

Constructors/Destructors

- PiecewisePenalty (Index dim)

  Default Constructor.
- ∼PiecewisePenalty ()

  Default Destructor.

- bool Acceptable (Number Fzconst, Number Fzlin)

  Check acceptability of given coordinates with respect to the Piecewise Penalty.
- Number BiggestBarr ()

  Get the value of the biggest barrier function so far.
- void UpdateEntry (Number barrier_obj, Number infeasi)

  Update Piecewise Penalty entry for given coordinates.

- void AddEntry (Number pen_r, Number barrier_obj, Number infeasi)

  Add a entry to the list.
- void ResetList (Number pen_r, Number barrier_obj, Number infeasi)

  Clear and reset the piecewise penalty list.

3.130.1 Detailed Description

Class for the Piecewise Penalty.

This class contains all Piecewise Penalty entries. The entries are stored as the corner point, including the margin.

Definition at line 39 of file IpPiecewisePenalty.hpp.

3.130.2 Member Function Documentation

3.130.2.1 bool Ipopt::PiecewisePenalty::Acceptable (Number *Fzconst,* Number *Fzlin*)

Check acceptability of given coordinates with respect to the Piecewise Penalty.

Returns true, if pair is acceptable

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

- IpPiecewisePenalty.hpp

3.131 Ipopt::PiecewisePenEntry Struct Reference

struct for one Piecewise Penalty entry.

```
#include <IpPiecewisePenalty.hpp>
```

3.131.1 Detailed Description

struct for one Piecewise Penalty entry.

Definition at line 25 of file IpPiecewisePenalty.hpp.

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

- IpPiecewisePenalty.hpp

3.132 Ipopt::PointPerturber Class Reference

This class is a simple object for generating randomly perturbed points that are withing the NLP bounds.

```
#include <IpEquilibrationScaling.hpp>
```

Inheritance diagram for Ipopt::PointPerturber:

Public Member Functions

- SmartPtr< Vector > MakeNewPerturbedPoint () const

  Return a new perturbed point.

Constructors/Destructors

- **PointPerturber** (const Vector &reference_point, Number random_pert_radius, const Matrix &Px_L, const Vector &x_L, const Matrix &Px_U, const Vector &x_U)
- virtual ∼PointPerturber ()

  Default destructor.

3.132.1 Detailed Description

This class is a simple object for generating randomly perturbed points that are withing the NLP bounds.

The ramdon_perturb_radius gives the upper bound of the perturbation.

Definition at line 92 of file IpEquilibrationScaling.hpp.

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

- IpEquilibrationScaling.hpp

3.133 Ipopt::AmplOptionsList::PrivatInfo Class Reference

3.133.1 Detailed Description

Definition at line 163 of file AmplTNLP.hpp.

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

- AmplTNLP.hpp

3.134 Ipopt::ProbingMuOracle Class Reference

Implementation of the probing strategy for computing the barrier parameter.

```
#include <IpProbingMuOracle.hpp>
```

Inheritance diagram for Ipopt::ProbingMuOracle:

Public Member Functions

- virtual bool InitializeImpl (const OptionsList &options, const std::string &prefix)

  overloaded from AlgorithmStrategyObject
- virtual bool CalculateMu (Number mu_min, Number mu_max, Number &new_mu)

  Method for computing the value of the barrier parameter that could be used in the current iteration (using Mehrotra's probing heuristic).

Constructors/Destructors

- ProbingMuOracle (const SmartPtr< PDSystemSolver > &pd_solver)

  Constructor.
- virtual ∼ProbingMuOracle ()

  Default destructor.

Static Public Member Functions

- static void RegisterOptions (SmartPtr< RegisteredOptions > roptions)

  Methods for IpoptType.

Additional Inherited Members

3.134.1 Detailed Description

Implementation of the probing strategy for computing the barrier parameter.

Definition at line 21 of file IpProbingMuOracle.hpp.

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

- IpProbingMuOracle.hpp

3.135 Ipopt::QualityFunctionMuOracle Class Reference

Implementation of the probing strategy for computing the barrier parameter.

`#include <IpQualityFunctionMuOracle.hpp>`

Inheritance diagram for Ipopt::QualityFunctionMuOracle:

Public Types

Public enums. Some of those are also used for the

quality function

- enum NormEnum

  enum for norm type
- enum CentralityEnum

  enum for centrality type
- enum BalancingTermEnum

  enum for the quality function balancing term type

Public Member Functions

- virtual bool InitializeImpl (const OptionsList &options, const std::string &prefix)

  overloaded from AlgorithmStrategyObject
- virtual bool CalculateMu (Number mu_min, Number mu_max, Number &new_mu)

  Method for computing the value of the barrier parameter that could be used in the current iteration (using the LOQO formula).

Constructors/Destructors

- QualityFunctionMuOracle (const SmartPtr< PDSystemSolver > &pd_solver)
  Constructor.
- virtual ~QualityFunctionMuOracle ()
  Default destructor.

Static Public Member Functions

- static void RegisterOptions (SmartPtr< RegisteredOptions > roptions)
  Methods for IpoptType.

Additional Inherited Members

3.135.1 Detailed Description

Implementation of the probing strategy for computing the barrier parameter.

Definition at line 22 of file IpQualityFunctionMuOracle.hpp.

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

- IpQualityFunctionMuOracle.hpp

3.136 Ipopt::ReferencedObject Class Reference

ReferencedObject class.

`#include <IpReferenced.hpp>`

Inheritance diagram for Ipopt::ReferencedObject:

3.136.1 Detailed Description

ReferencedObject class.

This is part of the implementation of an intrusive smart pointer design. This class stores the reference count of all the smart pointers that currently reference it. See the documentation for the SmartPtr class for more details.

A SmartPtr behaves much like a raw pointer, but manages the lifetime of an object, deleting the object automatically. This class implements a reference-counting, intrusive smart pointer design, where all objects pointed to must inherit off of ReferencedObject, which stores the reference count. Although this is intrusive (native types and externally authored classes require wrappers to be referenced by smart pointers), it is a safer design. A more detailed discussion of these issues follows after the usage information.

Usage Example: Note: to use the SmartPtr, all objects to which you point MUST inherit off of ReferencedObject.

```
*
* In MyClass.hpp...
*
* #include "IpReferenced.hpp"

* namespace Ipopt {
*
```

```
*  class MyClass : public ReferencedObject // must derive from ReferencedObject
*    {
*      ...
*    }
* } // namespace Ipopt
*
*
* In my_usage.cpp...
*
* #include "IpSmartPtr.hpp"
* #include "MyClass.hpp"
*
* void func(AnyObject& obj)
*  {
*    SmartPtr<MyClass> ptr_to_myclass = new MyClass(...);
*    // ptr_to_myclass now points to a new MyClass,
*    // and the reference count is 1
*
*    ...
*
*    obj.SetMyClass(ptr_to_myclass);
*    // Here, let's assume that AnyObject uses a
*    // SmartPtr<MyClass> internally here.
*    // Now, both ptr_to_myclass and the internal
*    // SmartPtr in obj point to the same MyClass object
*    // and its reference count is 2.
*
*    ...
*
*    // No need to delete ptr_to_myclass, this
*    // will be done automatically when the
*    // reference count drops to zero.
*
*  }
*
*
```

Other Notes: The SmartPtr implements both dereference operators -> & ∗. The SmartPtr does NOT implement a conversion operator to the raw pointer. Use the GetRawPtr() method when this is necessary. Make sure that the raw pointer is NOT deleted. The SmartPtr implements the comparison operators == & != for a variety of types. Use these instead of

```
*    if (GetRawPtr(smrt_ptr) == ptr) // Don't use this
*
```

SmartPtr's, as currently implemented, do NOT handle circular references. For example: consider a higher level object using SmartPtrs to point to A and B, but A and B also point to each other (i.e. A has a SmartPtr to B and B has a SmartPtr to A). In this scenario, when the higher level object is finished with A and B, their reference counts will never drop to zero (since they reference each other) and they will not be deleted. This can be detected by memory leak tools like valgrind. If the circular reference is necessary, the problem can be overcome by a number of techniques:

1) A and B can have a method that "releases" each other, that is they set their internal SmartPtrs to NULL.

```
*        void AClass::ReleaseCircularReferences()
*          {
*          smart_ptr_to_B = NULL;
*          }
*
```

Then, the higher level class can call these methods before it is done using A & B.

2) Raw pointers can be used in A and B to reference each other. Here, an implicit assumption is made that the lifetime is controlled by the higher level object and that A and B will both exist in a controlled manner. Although this seems dangerous, in many situations, this type of referencing is very controlled and this is reasonably safe.

3) This SmartPtr class could be redesigned with the Weak/Strong design concept. Here, the SmartPtr is identified as being Strong (controls lifetime of the object) or Weak (merely referencing the object). The Strong SmartPtr increments (and decrements) the reference count in ReferencedObject but the Weak SmartPtr does not. In the example above, the higher level object would have Strong SmartPtrs to A and B, but A and B would have Weak SmartPtrs to each other. Then, when the higher level object was done with A and B, they would be deleted. The Weak SmartPtrs in A and B would not decrement the reference count and would, of course, not delete the object. This idea is very similar to item (2), where it is implied that the sequence of events is controlled such that A and B will not call anything using their pointers following the higher level delete (i.e. in their destructors!). This is somehow safer, however, because code can be written (however expensive) to perform run-time detection of this situation. For example, the ReferencedObject could store pointers to all Weak SmartPtrs that are referencing it and, in its destructor, tell these pointers that it is dying. They could then set themselves to NULL, or set an internal flag to detect usage past this point.

For every most derived object only one ReferencedObject may exist, that is multiple inheritance requires virtual inheritance, see also the 2nd point in ticket #162.

Comments on Non-Intrusive Design: In a non-intrusive design, the reference count is stored somewhere other than the object being referenced. This means, unless the reference counting pointer is the first referencer, it must get a pointer to the referenced object from another smart pointer (so it has access to the reference count location). In this non-intrusive design, if we are pointing to an object with a smart pointer (or a number of smart pointers), and we then give another smart pointer the address through a RAW pointer, we will have two independent, AND INCORRECT, reference counts. To avoid this pitfall, we use an intrusive reference counting technique where the reference count is stored in the object being referenced.

Definition at line 174 of file IpReferenced.hpp.

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

- IpReferenced.hpp

3.137 Ipopt::Referencer Class Reference

Psydo-class, from which everything has to inherit that wants to use be registered as a Referencer for a ReferencedObject.

```
#include <IpReferenced.hpp>
```

Inherited by Ipopt::SmartPtr< T >, Ipopt::SmartPtr< const Ipopt::AmplOptionsList::AmplOption >, Ipopt::SmartPtr< const Ipopt::CompoundVectorSpace >, Ipopt::SmartPtr< const Ipopt::ExpansionMatrix >, Ipopt::SmartPtr< const Ipopt::IteratesVector >, Ipopt::SmartPtr< const Ipopt::Journalist >, Ipopt::SmartPtr< const Ipopt::LowRankUpdateSymMatrixSpace >, Ipopt::SmartPtr< const Ipopt::Matrix >, Ipopt::SmartPtr< const Ipopt::MatrixSpace >, Ipopt::SmartPtr< const Ipopt::MultiVectorMatrix >, Ipopt::SmartPtr< const Ipopt::NLP >, Ipopt::SmartPtr< const Ipopt::ScaledMatrixSpace >, Ipopt::SmartPtr< const Ipopt::SymMatrix >, Ipopt::SmartPtr< const Ipopt::SymMatrixSpace >, Ipopt::SmartPtr< const Ipopt::SymScaledMatrixSpace >, Ipopt::SmartPtr< const Ipopt::Vector >, Ipopt::SmartPtr< const Ipopt::VectorSpace >, Ipopt::SmartPtr< Ipopt::AmplSuffixHandler >, Ipopt::SmartPtr< Ipopt::AugSystemSolver >, Ipopt::SmartPtr< Ipopt::BacktrackingLSAcceptor >, Ipopt::SmartPtr< Ipopt::CompoundMatrix >, Ipopt::SmartPtr< Ipopt::CompoundMatrixSpace >, Ipopt::SmartPtr< Ipopt::CompoundSymMatrix >, Ipopt::SmartPtr< Ipopt::CompoundSymMatrixSpace >, Ipopt::SmartPtr< Ipopt::CompoundVector >, Ipopt::SmartPtr< Ipopt::CompoundVectorSpace >, Ipopt::SmartPtr< Ipopt::ConvergenceCheck >, Ipopt::SmartPtr< Ipopt::DenseGenMatrix >, Ipopt::SmartPtr< Ipopt::DenseSymMatrix >, Ipopt::SmartPtr< Ipopt::DenseVector >, Ipopt::SmartPtr< Ipopt::DiagMatrix >, Ipopt::SmartPtr< Ipopt::DiagMatrixSpace >, Ipopt::SmartPtr< Ipopt::EqMultiplierCalculator >, Ipopt::SmartPtr< Ipopt::ExpandedMultiVectorMatrix >, Ipopt::SmartPtr< Ipopt::ExpansionMatrix >, Ipopt::SmartPtr< Ipopt::ExpansionMatrixSpace >, Ipopt::SmartPtr< Ipopt::GenKKTSolverInterface >, Ipopt::SmartPtr< Ipopt::HessianUpdater >, Ipopt::SmartPtr< Ipopt::IdentityMatrixSpace >, Ipopt::SmartPtr< Ipopt::InexactNewtonNormalStep >, Ipopt::SmartPtr< Ipopt::InexactNormalStepCalculator >, Ipopt::SmartPtr< Ipopt::InexactNormalTerminationTester >, Ipopt::SmartPtr< Ipopt::InexactPDSolver >, Ipopt::SmartPtr< Ipopt::IpoptAdditionalCq >, Ipopt::SmartPtr< Ipopt::IpoptAdditionalData >, Ipopt::SmartPtr< Ipopt::IpoptAlgorithm >, Ipopt::SmartPtr< Ipopt::IpoptCalculatedQuantities >, Ipopt::Smart

Ptr< Ipopt::IpoptData >, Ipopt::SmartPtr< Ipopt::IpoptNLP >, Ipopt::SmartPtr< Ipopt::IterateInitializer >, Ipopt::SmartPtr< Ipopt::IteratesVectorSpace >, Ipopt::SmartPtr< Ipopt::IterationOutput >, Ipopt::SmartPtr< Ipopt::IterativeSolverTerminationTester >, Ipopt::SmartPtr< Ipopt::Journal >, Ipopt::SmartPtr< Ipopt::Journalist >, Ipopt::SmartPtr< Ipopt::LineSearch >, Ipopt::SmartPtr< Ipopt::Matrix >, Ipopt::SmartPtr< Ipopt::MultiVectorMatrix >, Ipopt::SmartPtr< Ipopt::MuOracle >, Ipopt::SmartPtr< Ipopt::MuUpdate >, Ipopt::SmartPtr< Ipopt::NLP >, Ipopt::SmartPtr< Ipopt::NLPScalingObject >, Ipopt::SmartPtr< Ipopt::OptionsList >, Ipopt::SmartPtr< Ipopt::OrigIterationOutput >, Ipopt::SmartPtr< Ipopt::PDPerturbationHandler >, Ipopt::SmartPtr< Ipopt::PDSystemSolver >, Ipopt::SmartPtr< Ipopt::RegisteredOption >, Ipopt::SmartPtr< Ipopt::RegisteredOptions >, Ipopt::SmartPtr< Ipopt::RestorationPhase >, Ipopt::SmartPtr< Ipopt::ScaledMatrixSpace >, Ipopt::SmartPtr< Ipopt::SearchDirectionCalculator >, Ipopt::SmartPtr< Ipopt::SolveStatistics >, Ipopt::SmartPtr< Ipopt::SparseSymLinearSolverInterface >, Ipopt::SmartPtr< Ipopt::SumSymMatrixSpace >, Ipopt::SmartPtr< Ipopt::SymLinearSolver >, Ipopt::SmartPtr< Ipopt::SymMatrix >, Ipopt::SmartPtr< Ipopt::SymScaledMatrixSpace >, Ipopt::SmartPtr< Ipopt::TDependencyDetector >, Ipopt::SmartPtr< Ipopt::TNLP >, Ipopt::SmartPtr< Ipopt::TripletToCSRConverter >, Ipopt::SmartPtr< Ipopt::TSymLinearSolver >, Ipopt::SmartPtr< Ipopt::TSymScalingMethod >, and Ipopt::SmartPtr< Ipopt::Vector >.

3.137.1 Detailed Description

Psydo-class, from which everything has to inherit that wants to use be registered as a Referencer for a ReferencedObject.

Definition at line 27 of file IpReferenced.hpp.

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

- IpReferenced.hpp

3.138 Ipopt::RegisteredOption Class Reference

Base class for registered options.

```
#include <IpRegOptions.hpp>
```

Inheritance diagram for Ipopt::RegisteredOption:

Classes

- class string_entry

  class to hold the valid string settings for a string option

Public Member Functions

- virtual void OutputDescription (const Journalist &jnlst) const

  output a description of the option
- virtual void OutputShortDescription (const Journalist &jnlst) const

  output a more concise version
- virtual void OutputLatexDescription (const Journalist &jnlst) const

  output a latex version

- RegisteredOption (Index counter)

  Constructors / Destructors.

- virtual const std::string & Name () const

  Standard Get / Set Methods.
- virtual void SetName (const std::string &name)

  Set the option's name (tag in the input file)
- virtual const std::string & ShortDescription () const

  Get the short description.
- virtual const std::string & LongDescription () const

  Get the long description.
- virtual void SetShortDescription (const std::string &short_description)

  Set the short description.
- virtual void SetLongDescription (const std::string &long_description)

  Set the long description.
- virtual const std::string & RegisteringCategory () const

  Get the registering class.
- virtual void SetRegisteringCategory (const std::string &registering_category)

  Set the registering class.
- virtual const RegisteredOptionType & Type () const

  Get the Option's type.
- virtual void SetType (const RegisteredOptionType &type)

  Get the Option's type.
- virtual Index Counter () const

  Counter.

Get / Set methods valid for specific types - NOTE: the Type

must be set before calling these methods.

- virtual const bool & HasLower () const

  check if the option has a lower bound - can be called for OT_Number & OT_Integer
- virtual const bool & LowerStrict () const

  check if the lower bound is strict - can be called for OT_Number
- virtual Number LowerNumber () const

  get the Number version of the lower bound - can be called for OT_Number
- virtual void SetLowerNumber (const Number &lower, const bool &strict)

  set the Number version of the lower bound - can be called for OT_Number
- virtual Index LowerInteger () const

  get the Integer version of the lower bound can be called for OT_Integer
- virtual void SetLowerInteger (const Index &lower)

  set the Integer version of the lower bound - can be called for OT_Integer
- virtual const bool & HasUpper () const

  check if the option has an upper bound - can be called for OT_Number & OT_Integer
- virtual const bool & UpperStrict () const

  check if the upper bound is strict - can be called for OT_Number
- virtual Number UpperNumber () const

  get the Number version of the upper bound - can be called for OT_Number
- virtual void SetUpperNumber (const Number &upper, const bool &strict)

  set the Number version of the upper bound - can be called for OT_Number
- virtual Index UpperInteger () const

  get the Integer version of the upper bound - can be called for OT_Integer
- virtual void SetUpperInteger (const Index &upper)

set the Integer version of the upper bound - can be called for OT_Integer

- virtual void AddValidStringSetting (const std::string value, const std::string description)

  method to add valid string entries - can be called for OT_String
- virtual Number DefaultNumber () const

  get the default as a Number - can be called for OT_Number
- virtual void SetDefaultNumber (const Number &default_value)

  Set the default as a Number - can be called for OT_Number.
- virtual Index DefaultInteger () const

  get the default as an Integer - can be called for OT_Integer
- virtual void SetDefaultInteger (const Index &default_value)

  Set the default as an Integer - can be called for OT_Integer.
- virtual std::string DefaultString () const

  get the default as a string - can be called for OT_String
- virtual Index DefaultStringAsEnum () const

  get the default as a string, but as the index of the string in the list - helps map from a string to an enum- can be called for OT_String
- virtual void SetDefaultString (const std::string &default_value)

  Set the default as a string - can be called for OT_String.
- virtual std::vector< string_entry > GetValidStrings () const

  get the valid string settings - can be called for OT_String
- virtual bool IsValidNumberSetting (const Number &value) const

  Check if the Number value is a valid setting - can be called for OT_Number.
- virtual bool IsValidIntegerSetting (const Index &value) const

  Check if the Integer value is a valid setting - can be called for OT_Integer.
- virtual bool IsValidStringSetting (const std::string &value) const

  Check if the String value is a valid setting - can be called for OT_String.
- virtual std::string MapStringSetting (const std::string &value) const

  Map a user setting (allowing any case) to the case used when the setting was registered.
- virtual Index MapStringSettingToEnum (const std::string &value) const

  Map a user setting (allowing any case) to the index of the matched setting in the list of string settings.

3.138.1 Detailed Description

Base class for registered options.

The derived types are more specific to a string option or a Number (real) option, etc.

Definition at line 33 of file IpRegOptions.hpp.

3.138.2 Member Function Documentation

3.138.2.1 virtual const std::string& Ipopt::RegisteredOption::Name ( ) const [inline], [virtual]

Standard Get / Set Methods.

Get the option's name (tag in the input file)

Definition at line 97 of file IpRegOptions.hpp.

3.138.2.2 virtual Index Ipopt::RegisteredOption::MapStringSettingToEnum (const std::string & *value*) const [virtual]

Map a user setting (allowing any case) to the index of the matched setting in the list of string settings.

Helps map a string setting to an enumeration.

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

- IpRegOptions.hpp

3.139 Ipopt::RegisteredOptions Class Reference

Class for storing registered options.

`#include <IpRegOptions.hpp>`

Inheritance diagram for Ipopt::RegisteredOptions:

Public Member Functions

- virtual const RegOptionsList & RegisteredOptionsList () const

  Giving access to iteratable representation of the registered options.

- RegisteredOptions ()

  Constructors / Destructors.
- virtual ∼RegisteredOptions ()

  Standard Destructor.

- virtual void SetRegisteringCategory (const std::string &registering_category)

  Methods to interact with registered options.
- virtual std::string RegisteringCategory ()

  retrieve the value of the current registering category
- virtual void AddNumberOption (const std::string &name, const std::string &short_description, Number default_←value, const std::string &long_description="")

  Add a Number option (with no restrictions)
- virtual void AddLowerBoundedNumberOption (const std::string &name, const std::string &short_description, Number lower, bool strict, Number default_value, const std::string &long_description="")

  Add a Number option (with a lower bound)
- virtual void AddUpperBoundedNumberOption (const std::string &name, const std::string &short_description, Number upper, bool strict, Number default_value, const std::string &long_description="")

  Add a Number option (with a upper bound)
- virtual void AddBoundedNumberOption (const std::string &name, const std::string &short_description, Number lower, bool lower_strict, Number upper, bool upper_strict, Number default_value, const std::string &long_←description="")

  Add a Number option (with a both bounds)
- virtual void AddIntegerOption (const std::string &name, const std::string &short_description, Index default_value, const std::string &long_description="")

  Add a Integer option (with no restrictions)
- virtual void AddLowerBoundedIntegerOption (const std::string &name, const std::string &short_description, Index lower, Index default_value, const std::string &long_description="")

Add a Integer option (with a lower bound)

- virtual void AddUpperBoundedIntegerOption (const std::string &name, const std::string &short_description, Index upper, Index default_value, const std::string &long_description="")

  Add a Integer option (with a upper bound)

- virtual void AddBoundedIntegerOption (const std::string &name, const std::string &short_description, Index lower, Index upper, Index default_value, const std::string &long_description="")

  Add a Integer option (with a both bounds)

- virtual void AddStringOption (const std::string &name, const std::string &short_description, const std::string &default_value, const std::vector< std::string > &settings, const std::vector< std::string > &descriptions, const std::string &long_description="")

  Add a String option (with no restrictions)

- virtual void AddStringOption1 (const std::string &name, const std::string &short_description, const std::string &default_value, const std::string &setting1, const std::string &description1, const std::string &long_description="")

  Methods that make adding string options with only a few entries easier.

- virtual SmartPtr< const RegisteredOption > GetOption (const std::string &name)

  Get a registered option - this will return NULL if the option does not exist.

- virtual void OutputOptionDocumentation (const Journalist &jnlst, std::list< std::string > &categories)

  Output documentation for the options - gives a description, etc.

- virtual void OutputLatexOptionDocumentation (const Journalist &jnlst, std::list< std::string > &categories)

  Output documentation in Latex format to include in a latex file.

3.139.1 Detailed Description

Class for storing registered options.

Used for validation and documentation.

Definition at line 390 of file IpRegOptions.hpp.

3.139.2 Constructor & Destructor Documentation

3.139.2.1 Ipopt::RegisteredOptions::RegisteredOptions ( ) [inline]

Constructors / Destructors.

Standard Constructor

Definition at line 396 of file IpRegOptions.hpp.

3.139.3 Member Function Documentation

3.139.3.1 virtual void Ipopt::RegisteredOptions::SetRegisteringCategory (const std::string & *registering_category*) [inline], [virtual]

Methods to interact with registered options.

set the registering class. All subsequent options will be added with the registered class

Definition at line 413 of file IpRegOptions.hpp.

3.139.3.2 virtual void Ipopt::RegisteredOptions::OutputOptionDocumentation (const Journalist & *jnlst*, std::list< std::string > & *categories*) [virtual]

Output documentation for the options - gives a description, etc.

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

- IpRegOptions.hpp

3.140 Ipopt::RestoConvergenceCheck Class Reference

Convergence check for the restoration phase.

`#include <IpRestoConvCheck.hpp>`

Inheritance diagram for Ipopt::RestoConvergenceCheck:

Public Member Functions

- virtual bool InitializeImpl (const OptionsList &options, const std::string &prefix)

  overloaded from AlgorithmStrategyObject
- virtual ConvergenceStatus CheckConvergence (bool call_intermediate_callback=true)

  overloaded from ConvergenceCheck
- virtual void SetOrigLSAcceptor (const BacktrackingLSAcceptor &orig_ls_acceptor)=0

  Method for setting the LS acceptor from the main algorithm.

Constructors/Destructors

- RestoConvergenceCheck ()

  Default Constructor.
- virtual ~RestoConvergenceCheck ()

  Default destructor.

Static Public Member Functions

- static void RegisterOptions (SmartPtr< RegisteredOptions > roptions)

  Methods used by IpoptType.

Additional Inherited Members

3.140.1 Detailed Description

Convergence check for the restoration phase.

This inherits from the OptimalityErrorConvergenceCheck so that the method for the regular optimality error convergence criterion can be checked as well. In addition, this convergence check returns the CONVERGED message, if the current iteration is acceptable to the original globalization scheme.

Definition at line 29 of file IpRestoConvCheck.hpp.

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

- IpRestoConvCheck.hpp

3.141 Ipopt::RestoFilterConvergenceCheck Class Reference

This is the implementation of the restoration convergence check is the original algorithm used the filter globalization mechanism.

`#include <IpRestoFilterConvCheck.hpp>`

Inheritance diagram for Ipopt::RestoFilterConvergenceCheck:

Public Member Functions

- void SetOrigLSAcceptor (const BacktrackingLSAcceptor &orig_ls_acceptor)

  Set the object for the original filter line search.
- virtual bool InitializeImpl (const OptionsList &options, const std::string &prefix)

  overloaded from AlgorithmStrategyObject

Constructors/Destructors

- RestoFilterConvergenceCheck ()

  Default Constructor.
- virtual ∼RestoFilterConvergenceCheck ()

  Default destructor.

Static Public Member Functions

- static void RegisterOptions (SmartPtr< RegisteredOptions > roptions)

  Methods used by IpoptType.

Additional Inherited Members

3.141.1 Detailed Description

This is the implementation of the restoration convergence check is the original algorithm used the filter globalization mechanism.

Definition at line 24 of file IpRestoFilterConvCheck.hpp.

3.141.2 Member Function Documentation

3.141.2.1 void Ipopt::RestoFilterConvergenceCheck::SetOrigLSAcceptor (const BacktrackingLSAcceptor & *orig_ls_acceptor*) [virtual]

Set the object for the original filter line search.

Here, orig_filter_ls_acceptor must be the same strategy object to which the restoration phase object with this object is given. This method must be called to finish the definition of the algorithm, before Initialize is called.

Implements Ipopt::RestoConvergenceCheck.

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

- IpRestoFilterConvCheck.hpp

3.142 Ipopt::RestoIpoptNLP Class Reference

This class maps the traditional NLP into something that is more useful by Ipopt.

`#include <IpRestoIpoptNLP.hpp>`

Inheritance diagram for Ipopt::RestoIpoptNLP:

Public Member Functions

- virtual bool Initialize (const Journalist &jnlst, const OptionsList &options, const std::string &prefix)

    Initialize - overloaded from IpoptNLP.
- virtual bool InitializeStructures (SmartPtr< Vector > &x, bool init_x, SmartPtr< Vector > &y_c, bool init_y_c, SmartPtr< Vector > &y_d, bool init_y_d, SmartPtr< Vector > &z_L, bool init_z_L, SmartPtr< Vector > &z_U, bool init_z_U, SmartPtr< Vector > &v_L, SmartPtr< Vector > &v_U)

    Initialize (create) structures for the iteration data.
- virtual bool GetWarmStartIterate (IteratesVector &warm_start_iterate)

    Method accessing the GetWarmStartIterate of the NLP.
- virtual void GetSpaces (SmartPtr< const VectorSpace > &x_space, SmartPtr< const VectorSpace > &c_space, SmartPtr< const VectorSpace > &d_space, SmartPtr< const VectorSpace > &x_l_space, SmartPtr< const MatrixSpace > &px_l_space, SmartPtr< const VectorSpace > &x_u_space, SmartPtr< const MatrixSpace > &px_u_space, SmartPtr< const VectorSpace > &d_l_space, SmartPtr< const MatrixSpace > &pd_l_space, SmartPtr< const VectorSpace > &d_u_space, SmartPtr< const MatrixSpace > &pd_u_space, SmartPtr< const MatrixSpace > &Jac_c_space, SmartPtr< const MatrixSpace > &Jac_d_space, SmartPtr< const SymMatrix←Space > &Hess_lagrangian_space)

    Accessor method for vector/matrix spaces pointers.
- virtual void AdjustVariableBounds (const Vector &new_x_L, const Vector &new_x_U, const Vector &new_d_L, const Vector &new_d_U)

    Method for adapting the variable bounds.
- bool IntermediateCallBack (AlgorithmMode mode, Index iter, Number obj_value, Number inf_pr, Number inf_du, Number mu, Number d_norm, Number regularization_size, Number alpha_du, Number alpha_pr, Index ls_trials, SmartPtr< const IpoptData > ip_data, SmartPtr< IpoptCalculatedQuantities > ip_cq)

    User callback method.
- Number Rho () const

    Accessor Method for obtaining the Rho penalization factor for the ell_1 norm.
- Number Eta (Number mu) const

    Method to calculate eta, the factor for the regularization term.
- SmartPtr< const Vector > DR_x () const

    Method returning the scaling factors for the 2-norm penalization term.

Constructors/Destructors

- **RestoIpoptNLP** (IpoptNLP &orig_ip_nlp, IpoptData &orig_ip_data, IpoptCalculatedQuantities &orig_ip_cq)
- ∼RestoIpoptNLP ()

    Default destructor.

- void FinalizeSolution (SolverReturn status, const Vector &x, const Vector &z_L, const Vector &z_U, const Vector &c, const Vector &d, const Vector &y_c, const Vector &y_d, Number obj_value, const IpoptData ∗ip_data, Ipopt←CalculatedQuantities ∗ip_cq)

    Solution Routines - overloaded from IpoptNLP.

- virtual bool objective_depends_on_mu () const

  Accessor methods for model data.
- virtual Number f (const Vector &x)

  Objective value (incorrect version for restoration phase)
- virtual Number f (const Vector &x, Number mu)

  Objective value.
- virtual SmartPtr< const Vector > grad_f (const Vector &x)

  Gradient of the objective (incorrect version for restoration phase)
- virtual SmartPtr< const Vector > grad_f (const Vector &x, Number mu)

  Gradient of the objective.
- virtual SmartPtr< const Vector > c (const Vector &x)

  Equality constraint residual.
- virtual SmartPtr< const Matrix > jac_c (const Vector &x)

  Jacobian Matrix for equality constraints.
- virtual SmartPtr< const Vector > d (const Vector &x)

  Inequality constraint residual (reformulated as equalities with slacks.
- virtual SmartPtr< const Matrix > jac_d (const Vector &x)

  Jacobian Matrix for inequality constraints.
- virtual SmartPtr< const SymMatrix > h (const Vector &x, Number obj_factor, const Vector &yc, const Vector &yd)

  Hessian of the Lagrangian (incorrect version for restoration phase)
- virtual SmartPtr< const SymMatrix > h (const Vector &x, Number obj_factor, const Vector &yc, const Vector &yd, Number mu)

  Hessian of the Lagrangian.
- virtual SmartPtr< const SymMatrix > uninitialized_h ()

  Provides a Hessian matrix from the correct matrix space with uninitialized values.
- virtual SmartPtr< const Vector > x_L () const

  Lower bounds on x.
- virtual SmartPtr< const Matrix > Px_L () const

  Permutation matrix (x_L_ -> x)
- virtual SmartPtr< const Vector > x_U () const

  Upper bounds on x.
- virtual SmartPtr< const Matrix > Px_U () const

  Permutation matrix (x_U_ -> x.
- virtual SmartPtr< const Vector > d_L () const

  Lower bounds on d.
- virtual SmartPtr< const Matrix > Pd_L () const

  Permutation matrix (d_L_ -> d)
- virtual SmartPtr< const Vector > d_U () const

  Upper bounds on d.
- virtual SmartPtr< const Matrix > Pd_U () const

  Permutation matrix (d_U_ -> d.
- virtual SmartPtr< const SymMatrixSpace > HessianMatrixSpace () const

  Accessor method to obtain the MatrixSpace for the Hessian matrix (or it's approximation)
- virtual SmartPtr< const VectorSpace > x_space () const

  x_space

Accessor method for the information of the original NLP.

These methods are not overloaded from IpoptNLP

- IpoptNLP & **OrigIpNLP** () const
- IpoptData & **OrigIpData** () const
- IpoptCalculatedQuantities & **OrigIpCq** () const

Counters for the number of function evaluations.

- virtual Index **f_evals** () const
- virtual Index **grad_f_evals** () const
- virtual Index **c_evals** () const
- virtual Index **jac_c_evals** () const
- virtual Index **d_evals** () const
- virtual Index **jac_d_evals** () const
- virtual Index **h_evals** () const

Static Public Member Functions

- static void RegisterOptions (SmartPtr< RegisteredOptions > roptions)

  Methods for IpoptType.

3.142.1 Detailed Description

This class maps the traditional NLP into something that is more useful by Ipopt.

This class takes care of storing the calculated model results, handles cacheing, and (some day) takes care of addition of slacks.

Definition at line 32 of file IpRestoIpoptNLP.hpp.

3.142.2 Member Function Documentation

3.142.2.1 virtual bool Ipopt::RestoIpoptNLP::objective_depends_on_mu ( ) const [inline], [virtual]

Accessor methods for model data.

Method for telling IpoptCalculatedQuantities that the restoration phase objective function depends on the barrier parameter

Reimplemented from Ipopt::IpoptNLP.

Definition at line 89 of file IpRestoIpoptNLP.hpp.

3.142.2.2 virtual SmartPtr<const SymMatrix> Ipopt::RestoIpoptNLP::uninitialized_h ( ) [virtual]

Provides a Hessian matrix from the correct matrix space with uninitialized values.

This can be used in LeastSquareMults to obtain a "zero Hessian".

Implements Ipopt::IpoptNLP.

3.142.2.3 virtual void Ipopt::RestoIpoptNLP::AdjustVariableBounds (const Vector & *new_x_L,* const Vector & *new_x_U,* const Vector & *new_d_L,* const Vector & *new_d_U*) [virtual]

Method for adapting the variable bounds.

This is called if slacks are becoming too small

Implements Ipopt::IpoptNLP.

3.142.2.4 SmartPtr<const Vector> Ipopt::RestoIpoptNLP::DR_x ( ) const [inline]

Method returning the scaling factors for the 2-norm penalization term.

Definition at line 292 of file IpRestoIpoptNLP.hpp.

3.142.2.5 static void Ipopt::RestoIpoptNLP::RegisterOptions (SmartPtr< RegisteredOptions > *roptions*) [static]

Methods for IpoptType.

Called by IpoptType to register the options

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

- IpRestoIpoptNLP.hpp

3.143 Ipopt::RestoIterateInitializer Class Reference

Class implementing the default initialization procedure (based on user options) for the iterates.

`#include <IpRestoIterateInitializer.hpp>`

Inheritance diagram for Ipopt::RestoIterateInitializer:

Public Member Functions

- virtual bool InitializeImpl (const OptionsList &options, const std::string &prefix)
  overloaded from AlgorithmStrategyObject
- virtual bool SetInitialIterates ()
  Compute the initial iterates and set the into the curr field of the ip_data object.

Constructors/Destructors

- RestoIterateInitializer (const SmartPtr< EqMultiplierCalculator > &eq_mult_calculator)
  Constructor.
- virtual ∼RestoIterateInitializer ()
  Default destructor.

Static Public Member Functions

- static void RegisterOptions (SmartPtr< RegisteredOptions > roptions)
  Methods for IpoptType.

Additional Inherited Members

3.143.1 Detailed Description

Class implementing the default initialization procedure (based on user options) for the iterates.

It is used at the very beginning of the optimization for determine the starting point for all variables.

Definition at line 22 of file IpRestoIterateInitializer.hpp.

3.143.2 Constructor & Destructor Documentation

3.143.2.1 Ipopt::RestoIterateInitializer::RestoIterateInitializer (const SmartPtr< EqMultiplierCalculator > & *eq_mult_calculator*)

Constructor.

If eq_mult_calculator is not NULL, it will be used to compute the initial values for equality constraint multipliers.

3.143.3 Member Function Documentation

3.143.3.1 virtual bool Ipopt::RestoIterateInitializer::SetInitialIterates ( ) [virtual]

Compute the initial iterates and set the into the curr field of the ip_data object.

Implements Ipopt::IterateInitializer.

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

- IpRestoIterateInitializer.hpp

3.144 Ipopt::RestoIterationOutput Class Reference

Class for the iteration summary output for the restoration phase.

`#include <IpRestoIterationOutput.hpp>`

Inheritance diagram for Ipopt::RestoIterationOutput:

Public Member Functions

- virtual bool InitializeImpl (const OptionsList &options, const std::string &prefix)
  overloaded from AlgorithmStrategyObject
- virtual void WriteOutput ()
  Method to do all the summary output per iteration.

Constructors/Destructors

- RestoIterationOutput (const SmartPtr< OrigIterationOutput > &resto_orig_iteration_output)
  Constructor.
- virtual ∼RestoIterationOutput ()
  Default destructor.

Additional Inherited Members

3.144.1 Detailed Description

Class for the iteration summary output for the restoration phase.

This prints information for the ORIGINAL NLP (and possibly for the restoration phase NLP.

Definition at line 22 of file IpRestoIterationOutput.hpp.

3.144.2 Constructor & Destructor Documentation

3.144.2.1 Ipopt::RestoIterationOutput::RestoIterationOutput (const SmartPtr< OrigIterationOutput > & *resto_orig_iteration_output*)

Constructor.

If resto_orig_iteration_output is not NULL, the output will be done twice per iteration, first for the restoration phase problem, and secondly using the functions for the original NLP.

3.144.3 Member Function Documentation

3.144.3.1 virtual void Ipopt::RestoIterationOutput::WriteOutput ( ) [virtual]

Method to do all the summary output per iteration.

This include the one-line summary output as well as writing the details about the iterates if desired

Implements Ipopt::IterationOutput.

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

- IpRestoIterationOutput.hpp

3.145 Ipopt::RestoPenaltyConvergenceCheck Class Reference

This is the implementation of the restoration convergence check is the original algorithm used the filter globalization mechanism.

`#include <IpRestoPenaltyConvCheck.hpp>`

Inheritance diagram for Ipopt::RestoPenaltyConvergenceCheck:

Public Member Functions

- void SetOrigLSAcceptor (const BacktrackingLSAcceptor &orig_ls_acceptor)

  Set the object for the original penalty line search.
- virtual bool InitializeImpl (const OptionsList &options, const std::string &prefix)

  overloaded from AlgorithmStrategyObject

Constructors/Destructors

- RestoPenaltyConvergenceCheck ()

  Default Constructor.
- virtual ~RestoPenaltyConvergenceCheck ()

  Default destructor.

Static Public Member Functions

- static void RegisterOptions (SmartPtr< RegisteredOptions > roptions)

  Methods used by IpoptType.

Additional Inherited Members

3.145.1 Detailed Description

This is the implementation of the restoration convergence check is the original algorithm used the filter globalization mechanism.

Definition at line 23 of file IpRestoPenaltyConvCheck.hpp.

3.145.2 Member Function Documentation

3.145.2.1 void Ipopt::RestoPenaltyConvergenceCheck::SetOrigLSAcceptor (const BacktrackingLSAcceptor & *orig_ls_acceptor*) [virtual]

Set the object for the original penalty line search.

Here, orig_penalty_ls_acceptor must be the same strategy object to which the restoration phase object with this object is given. This method must be called to finish the definition of the algorithm, before Initialize is called.

Implements Ipopt::RestoConvergenceCheck.

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

- IpRestoPenaltyConvCheck.hpp

3.146 Ipopt::RestorationPhase Class Reference

Base class for different restoration phases.

#include <IpRestoPhase.hpp>

Inheritance diagram for Ipopt::RestorationPhase:

Public Member Functions

- virtual bool InitializeImpl (const OptionsList &options, const std::string &prefix)=0

  overloaded from AlgorithmStrategyObject
- virtual bool PerformRestoration ()=0

  Method called to perform restoration for the filter line search method.

Constructors/Destructors

- RestorationPhase ()

  Default Constructor.
- virtual ∼RestorationPhase ()

  Default Destructor.

Additional Inherited Members

3.146.1 Detailed Description

Base class for different restoration phases.

The restoration phase is part of the FilterLineSearch.

Definition at line 34 of file IpRestoPhase.hpp.

3.146.2 Member Function Documentation

3.146.2.1 virtual bool Ipopt::RestorationPhase::PerformRestoration ( ) [pure virtual]

Method called to perform restoration for the filter line search method.

Implemented in Ipopt::MinC_1NrmRestorationPhase, and Ipopt::RestoRestorationPhase.

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

- IpRestoPhase.hpp

3.147 Ipopt::RestoRestorationPhase Class Reference

Recursive Restoration Phase for the.MinC_1NrmRestorationPhase.

`#include <IpRestoRestoPhase.hpp>`

Inheritance diagram for Ipopt::RestoRestorationPhase:

Public Member Functions

- virtual bool InitializeImpl (const OptionsList &options, const std::string &prefix)

    Overloaded from AlgorithmStrategy case class.

Constructors/Destructors

- RestoRestorationPhase ()

    Default Constructor.
- virtual ∼RestoRestorationPhase ()

    Default destructor.

Protected Member Functions

- virtual bool PerformRestoration ()

    Overloaded method from RestorationPhase.

3.147.1 Detailed Description

Recursive Restoration Phase for the.MinC_1NrmRestorationPhase.

This procedure chooses the n and p variables in the MinC_1NrmRestorationPhase problem formulation by treating the problem as separable (assuming that the x and s variables are fixed).

Definition at line 25 of file IpRestoRestoPhase.hpp.

3.147.2 Constructor & Destructor Documentation

3.147.2.1 Ipopt::RestoRestorationPhase::RestoRestorationPhase ( )

Default Constructor.

3.147.3 Member Function Documentation

3.147.3.1 virtual bool Ipopt::RestoRestorationPhase::PerformRestoration ( ) [protected], [virtual]

Overloaded method from RestorationPhase.

Implements Ipopt::RestorationPhase.

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

- IpRestoRestoPhase.hpp

3.148 Ipopt::ScaledMatrix Class Reference

Class for a Matrix in conjunction with its scaling factors for row and column scaling.

`#include <IpScaledMatrix.hpp>`

Inheritance diagram for Ipopt::ScaledMatrix:

Public Member Functions

- void SetUnscaledMatrix (const SmartPtr< const Matrix > unscaled_matrix)
  Set the unscaled matrix.
- void SetUnscaledMatrixNonConst (const SmartPtr< Matrix > &unscaled_matrix)
  Set the unscaled matrix in a non-const version.
- SmartPtr< const Matrix > GetUnscaledMatrix () const
  Return the unscaled matrix in const form.
- SmartPtr< Matrix > GetUnscaledMatrixNonConst ()
  Return the unscaled matrix in non-const form.
- SmartPtr< const Vector > RowScaling () const
  return the vector for the row scaling
- SmartPtr< const Vector > ColumnScaling () const
  return the vector for the column scaling

Constructors / Destructors

- ScaledMatrix (const ScaledMatrixSpace *owner_space)
  Constructor, taking the owner_space.
- ~ScaledMatrix ()
  Destructor.

Protected Member Functions

Methods overloaded from Matrix

- virtual void MultVectorImpl (Number alpha, const Vector &x, Number beta, Vector &y) const
  Matrix-vector multiply.
- virtual void TransMultVectorImpl (Number alpha, const Vector &x, Number beta, Vector &y) const
  Matrix(transpose) vector multiply.
- virtual bool HasValidNumbersImpl () const
  Method for determining if all stored numbers are valid (i.e., no Inf or Nan).
- virtual void ComputeRowAMaxImpl (Vector &rows_norms, bool init) const
  Compute the max-norm of the rows in the matrix.
- virtual void ComputeColAMaxImpl (Vector &cols_norms, bool init) const
  Compute the max-norm of the columns in the matrix.
- virtual void PrintImpl (const Journalist &jnlst, EJournalLevel level, EJournalCategory category, const std::string &name, Index indent, const std::string &prefix) const
  Print detailed information about the matrix.
- virtual void AddMSinvZImpl (Number alpha, const Vector &S, const Vector &Z, Vector &X) const
  X = beta∗X + alpha∗(Matrix S^{-1} Z).
- virtual void SinvBlrmZMTdBrImpl (Number alpha, const Vector &S, const Vector &R, const Vector &Z, const Vector &D, Vector &X) const
  X = S^{-1} (r + alpha∗Z∗M^Td).

Additional Inherited Members

3.148.1 Detailed Description

Class for a Matrix in conjunction with its scaling factors for row and column scaling.

Operations on the matrix are performed using the scaled matrix. You can pull out the pointer to the unscaled matrix for unscaled calculations.

Definition at line 26 of file IpScaledMatrix.hpp.

3.148.2 Member Function Documentation

3.148.2.1 virtual void Ipopt::ScaledMatrix::MultVectorImpl (Number *alpha*, const Vector & *x*, Number *beta*, Vector & *y*) const [protected], [virtual]

Matrix-vector multiply.

Computes y = alpha ∗ Matrix ∗ x + beta ∗ y

Implements Ipopt::Matrix.

3.148.2.2 virtual void Ipopt::ScaledMatrix::TransMultVectorImpl (Number *alpha*, const Vector & *x*, Number *beta*, Vector & *y*) const [protected], [virtual]

Matrix(transpose) vector multiply.

Computes y = alpha ∗ Matrix^T ∗ x + beta ∗ y

Implements Ipopt::Matrix.

3.148.2.3 virtual bool Ipopt::ScaledMatrix::HasValidNumbersImpl ( ) const [protected], [virtual]

Method for determining if all stored numbers are valid (i.e., no Inf or Nan).

It is assumed that the scaling factors are valid.

Reimplemented from Ipopt::Matrix.

3.148.2.4 virtual void Ipopt::ScaledMatrix::ComputeRowAMaxImpl (Vector & *rows_norms,* bool *init*) const [protected], [virtual]

Compute the max-norm of the rows in the matrix.

The result is stored in rows_norms. The vector is assumed to be initialized.

Implements Ipopt::Matrix.

3.148.2.5 virtual void Ipopt::ScaledMatrix::ComputeColAMaxImpl (Vector & *cols_norms,* bool *init*) const [protected], [virtual]

Compute the max-norm of the columns in the matrix.

The result is stored in cols_norms. The vector is assumed to be initialized.

Implements Ipopt::Matrix.

3.148.2.6 virtual void Ipopt::ScaledMatrix::PrintImpl (const Journalist & *jnlst,* EJournalLevel *level,* EJournalCategory *category,* const std::string & *name,* Index *indent,* const std::string & *prefix*) const [protected], [virtual]

Print detailed information about the matrix.

Implements Ipopt::Matrix.

3.148.2.7 virtual void Ipopt::ScaledMatrix::AddMSinvZImpl (Number *alpha,* const Vector & *S,* const Vector & *Z,* Vector & *X*) const [protected], [virtual]

X = beta∗X + alpha∗(Matrix S^{-1} Z).

Specialized implementation missing so far!

Reimplemented from Ipopt::Matrix.

3.148.2.8 virtual void Ipopt::ScaledMatrix::SinvBlrmZMTdBrImpl (Number *alpha,* const Vector & *S,* const Vector & *R,* const Vector & *Z,* const Vector & *D,* Vector & *X*) const [protected], [virtual]

X = S^{-1} (r + alpha∗Z∗M^Td).

Specialized implementation missing so far!

Reimplemented from Ipopt::Matrix.

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

- IpScaledMatrix.hpp

3.149 Ipopt::ScaledMatrixSpace Class Reference

This is the matrix space for ScaledMatrix.

`#include <IpScaledMatrix.hpp>`

Inheritance diagram for Ipopt::ScaledMatrixSpace:

Public Member Functions

- ScaledMatrix * MakeNewScaledMatrix (bool allocate_unscaled_matrix=false) const

  Method for creating a new matrix of this specific type.
- virtual Matrix * MakeNew () const

  Overloaded MakeNew method for the MatrixSpace base class.
- SmartPtr< const Vector > RowScaling () const

  return the vector for the row scaling
- SmartPtr< const MatrixSpace > UnscaledMatrixSpace () const

  return the matrix space for the unscaled matrix
- SmartPtr< const Vector > ColumnScaling () const

  return the vector for the column scaling

Constructors / Destructors

- ScaledMatrixSpace (const SmartPtr< const Vector > &row_scaling, bool row_scaling_reciprocal, const SmartPtr< const MatrixSpace > &unscaled_matrix_space, const SmartPtr< const Vector > &column_scaling, bool column_scaling_reciprocal)

  Constructor, given the number of row and columns blocks, as well as the totel number of rows and columns.
- ~ScaledMatrixSpace ()

  Destructor.

3.149.1 Detailed Description

This is the matrix space for ScaledMatrix.

Definition at line 128 of file IpScaledMatrix.hpp.

3.149.2 Member Function Documentation

3.149.2.1 ScaledMatrix* Ipopt::ScaledMatrixSpace::MakeNewScaledMatrix (bool *allocate_unscaled_matrix* = `false`) const `[inline]`

Method for creating a new matrix of this specific type.

Definition at line 148 of file IpScaledMatrix.hpp.

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

- IpScaledMatrix.hpp

3.150 Ipopt::SearchDirectionCalculator Class Reference

Base class for computing the search direction for the line search.

`#include <IpSearchDirCalculator.hpp>`

Inheritance diagram for Ipopt::SearchDirectionCalculator:

Public Member Functions

- virtual bool InitializeImpl (const OptionsList &options, const std::string &prefix)=0

    overloaded from AlgorithmStrategyObject
- virtual bool ComputeSearchDirection ()=0

    Pure virtual method for computing the search direction.

Constructors/Destructors

- SearchDirectionCalculator ()

    Constructor.
- virtual ∼SearchDirectionCalculator ()

    Default destructor.

Additional Inherited Members

3.150.1 Detailed Description

Base class for computing the search direction for the line search.

Definition at line 20 of file IpSearchDirCalculator.hpp.

3.150.2 Member Function Documentation

3.150.2.1 virtual bool Ipopt::SearchDirectionCalculator::ComputeSearchDirection ( ) [pure virtual]

Pure virtual method for computing the search direction.

The computed direction is stored in IpData().delta().

Implemented in Ipopt::InexactSearchDirCalculator, Ipopt::CGSearchDirCalculator, and Ipopt::PDSearchDirCalculator.

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

- IpSearchDirCalculator.hpp

3.151 Ipopt::SlackBasedTSymScalingMethod Class Reference

Class for the method for computing scaling factors for symmetric matrices in triplet format, specifically for the inexaxct algorithm.

`#include <IpSlackBasedTSymScalingMethod.hpp>`

Inheritance diagram for Ipopt::SlackBasedTSymScalingMethod:

Public Member Functions

- virtual bool InitializeImpl (const OptionsList &options, const std::string &prefix)

  overloaded from AlgorithmStrategyObject
- virtual bool ComputeSymTScalingFactors (Index n, Index nnz, const ipfint ∗airn, const ipfint ∗ajcn, const double ∗a, double ∗scaling_factors)

  Method for computing the symmetric scaling factors, given the symmtric matrix in triplet (MA27) format.

Constructor/Destructor

- **SlackBasedTSymScalingMethod** ()
- virtual ∼**SlackBasedTSymScalingMethod** ()

Additional Inherited Members

3.151.1 Detailed Description

Class for the method for computing scaling factors for symmetric matrices in triplet format, specifically for the inexaxct algorithm.

The scaling is only considering the current slacks.

Definition at line 23 of file IpSlackBasedTSymScalingMethod.hpp.

3.151.2 Member Function Documentation

3.151.2.1 virtual bool Ipopt::SlackBasedTSymScalingMethod::ComputeSymTScalingFactors (Index *n,* Index *nnz,* const ipfint ∗ *airn,* const ipfint ∗ *ajcn,* const double ∗ *a,* double ∗ *scaling_factors*) [virtual]

Method for computing the symmetric scaling factors, given the symmtric matrix in triplet (MA27) format.

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

- IpSlackBasedTSymScalingMethod.hpp

3.152 Ipopt::SmartPtr< T > Class Template Reference

Template class for Smart Pointers.

`#include <IpSmartPtr.hpp>`

Inherits Ipopt::Referencer.

Public Member Functions

Constructors/Destructors

- SmartPtr ()

  Default constructor, initialized to NULL.
- SmartPtr (const SmartPtr< T > &copy)

  Copy constructor, initialized from copy of type T.

- template<class U >
  SmartPtr (const SmartPtr< U > &copy)

  Copy constructor, initialized from copy of type U.
- SmartPtr (T ∗ptr)

  Constructor, initialized from T∗ ptr.
- ∼SmartPtr ()

  Destructor, automatically decrements the reference count, deletes the object if necessary.

Friends

friend method declarations.

- template<class U >
  U ∗ GetRawPtr (const SmartPtr< U > &smart_ptr)

  Returns the raw pointer contained.
- template<class U >
  SmartPtr< const U > ConstPtr (const SmartPtr< U > &smart_ptr)

  Returns a const pointer.
- template<class U >
  bool IsValid (const SmartPtr< U > &smart_ptr)

  Returns true if the SmartPtr is NOT NULL.
- template<class U >
  bool IsNull (const SmartPtr< U > &smart_ptr)

  Returns true if the SmartPtr is NULL.

Overloaded operators.

- T ∗ operator-> () const

  Overloaded arrow operator, allows the user to call methods using the contained pointer.
- T & operator∗ () const

  Overloaded dereference operator, allows the user to dereference the contained pointer.
- SmartPtr< T > & operator= (T ∗rhs)

  Overloaded equals operator, allows the user to set the value of the SmartPtr from a raw pointer.
- SmartPtr< T > & operator= (const SmartPtr< T > &rhs)

  Overloaded equals operator, allows the user to set the value of the SmartPtr from another SmartPtr.
- template<class U >
  SmartPtr< T > & operator= (const SmartPtr< U > &rhs)

  Overloaded equals operator, allows the user to set the value of the SmartPtr from another SmartPtr of a different type.
- template<class U1 , class U2 >
  bool operator== (const SmartPtr< U1 > &lhs, const SmartPtr< U2 > &rhs)

  Overloaded equality comparison operator, allows the user to compare the value of two SmartPtrs.
- template<class U1 , class U2 >
  bool operator== (const SmartPtr< U1 > &lhs, U2 ∗raw_rhs)

  Overloaded equality comparison operator, allows the user to compare the value of a SmartPtr with a raw pointer.
- template<class U1 , class U2 >
  bool operator== (U1 ∗lhs, const SmartPtr< U2 > &raw_rhs)

  Overloaded equality comparison operator, allows the user to compare the value of a raw pointer with a SmartPtr.
- template<class U1 , class U2 >
  bool operator!= (const SmartPtr< U1 > &lhs, const SmartPtr< U2 > &rhs)

  Overloaded in-equality comparison operator, allows the user to compare the value of two SmartPtrs.

- template<class U1 , class U2 >
  bool operator!= (const SmartPtr< U1 > &lhs, U2 *raw_rhs)

  Overloaded in-equality comparison operator, allows the user to compare the value of a SmartPtr with a raw pointer.
- template<class U1 , class U2 >
  bool operator!= (U1 *lhs, const SmartPtr< U2 > &raw_rhs)

  Overloaded in-equality comparison operator, allows the user to compare the value of a SmartPtr with a raw pointer.
- template<class U >
  bool operator< (const SmartPtr< U > &lhs, const SmartPtr< U > &rhs)

  Overloaded less-than comparison operator, allows the user to compare the value of two SmartPtrs.

3.152.1 Detailed Description

template<class T>class Ipopt::SmartPtr< T >

Template class for Smart Pointers.

A SmartPtr behaves much like a raw pointer, but manages the lifetime of an object, deleting the object automatically. This class implements a reference-counting, intrusive smart pointer design, where all objects pointed to must inherit off of ReferencedObject, which stores the reference count. Although this is intrusive (native types and externally authored classes require wrappers to be referenced by smart pointers), it is a safer design. A more detailed discussion of these issues follows after the usage information.

Usage Example: Note: to use the SmartPtr, all objects to which you point MUST inherit off of ReferencedObject.

```
*
* In MyClass.hpp...
*
* #include "IpReferenced.hpp"

* namespace Ipopt {
*
*  class MyClass : public ReferencedObject // must derive from ReferencedObject
*    {
*      ...
*    }
* } // namespace Ipopt
*
*
* In my_usage.cpp...
*
* #include "IpSmartPtr.hpp"
* #include "MyClass.hpp"
*
* void func(AnyObject& obj)
*  {
*    SmartPtr<MyClass> ptr_to_myclass = new MyClass(...);
*    // ptr_to_myclass now points to a new MyClass,
*    // and the reference count is 1
*
*    ...
*
*    obj.SetMyClass(ptr_to_myclass);
*    // Here, let's assume that AnyObject uses a
*    // SmartPtr<MyClass> internally here.
*    // Now, both ptr_to_myclass and the internal
*    // SmartPtr in obj point to the same MyClass object
*    // and its reference count is 2.
*
*    ...
*
*    // No need to delete ptr_to_myclass, this
```

```
*    // will be done automatically when the
*    // reference count drops to zero.
*
*  }
*
*
```

It is not necessary to use SmartPtr's in all cases where an object is used that has been allocated "into" a SmartPtr. It is possible to just pass objects by reference or regular pointers, even if lower down in the stack a SmartPtr is to be held on to. Everything should work fine as long as a pointer created by "new" is immediately passed into a SmartPtr, and if SmartPtr's are used to hold on to objects.

Other Notes: The SmartPtr implements both dereference operators -> & $*$. The SmartPtr does NOT implement a conversion operator to the raw pointer. Use the GetRawPtr() method when this is necessary. Make sure that the raw pointer is NOT deleted. The SmartPtr implements the comparison operators == & != for a variety of types. Use these instead of

```
*    if (GetRawPtr(smrt_ptr) == ptr) // Don't use this
*
```

SmartPtr's, as currently implemented, do NOT handle circular references. For example: consider a higher level object using SmartPtrs to point to A and B, but A and B also point to each other (i.e. A has a SmartPtr to B and B has a SmartPtr to A). In this scenario, when the higher level object is finished with A and B, their reference counts will never drop to zero (since they reference each other) and they will not be deleted. This can be detected by memory leak tools like valgrind. If the circular reference is necessary, the problem can be overcome by a number of techniques:

1) A and B can have a method that "releases" each other, that is they set their internal SmartPtrs to NULL.

```
*        void AClass::ReleaseCircularReferences()
*          {
*          smart_ptr_to_B = NULL;
*          }
*
```

Then, the higher level class can call these methods before it is done using A & B.

2) Raw pointers can be used in A and B to reference each other. Here, an implicit assumption is made that the lifetime is controlled by the higher level object and that A and B will both exist in a controlled manner. Although this seems dangerous, in many situations, this type of referencing is very controlled and this is reasonably safe.

3) This SmartPtr class could be redesigned with the Weak/Strong design concept. Here, the SmartPtr is identified as being Strong (controls lifetime of the object) or Weak (merely referencing the object). The Strong SmartPtr increments (and decrements) the reference count in ReferencedObject but the Weak SmartPtr does not. In the example above, the higher level object would have Strong SmartPtrs to A and B, but A and B would have Weak SmartPtrs to each other. Then, when the higher level object was done with A and B, they would be deleted. The Weak SmartPtrs in A and B would not decrement the reference count and would, of course, not delete the object. This idea is very similar to item (2), where it is implied that the sequence of events is controlled such that A and B will not call anything using their pointers following the higher level delete (i.e. in their destructors!). This is somehow safer, however, because code can be written (however expensive) to perform run-time detection of this situation. For example, the ReferencedObject could store pointers to all Weak SmartPtrs that are referencing it and, in its destructor, tell these pointers that it is dying. They could then set themselves to NULL, or set an internal flag to detect usage past this point.

Comments on Non-Intrusive Design: In a non-intrusive design, the reference count is stored somewhere other than the object being referenced. This means, unless the reference counting pointer is the first referencer, it must get a pointer to the referenced object from another smart pointer (so it has access to the reference count location). In this non-intrusive design, if we are pointing to an object with a smart pointer (or a number of smart pointers), and we then give another smart pointer the address through a RAW pointer, we will have two independent, AND INCORRECT, reference counts. To avoid this pitfall, we use an intrusive reference counting technique where the reference count is stored in the object being referenced.

Definition at line 172 of file IpSmartPtr.hpp.

3.152.2 Constructor & Destructor Documentation

3.152.2.1 template<class T > **Ipopt::SmartPtr< T >::~SmartPtr** ( )

Destructor, automatically decrements the reference count, deletes the object if necessary.

Definition at line 428 of file IpSmartPtr.hpp.

3.152.3 Member Function Documentation

3.152.3.1 template<class T > T * **Ipopt::SmartPtr< T >::operator->** ( ) const

Overloaded arrow operator, allows the user to call methods using the contained pointer.

Definition at line 439 of file IpSmartPtr.hpp.

3.152.3.2 template<class T > T & **Ipopt::SmartPtr< T >::operator*** ( ) const

Overloaded dereference operator, allows the user to dereference the contained pointer.

Definition at line 455 of file IpSmartPtr.hpp.

3.152.4 Friends And Related Function Documentation

3.152.4.1 template<class T> template<class U1 , class U2 > bool operator== (const **SmartPtr**< U1 > & *lhs*, U2 * *raw_rhs*) [friend]

Overloaded equality comparison operator, allows the user to compare the value of a SmartPtr with a raw pointer.

Definition at line 631 of file IpSmartPtr.hpp.

3.152.4.2 template<class T> template<class U1 , class U2 > bool operator== (U1 * *lhs*, const **SmartPtr**< U2 > & *raw_rhs*) [friend]

Overloaded equality comparison operator, allows the user to compare the value of a raw pointer with a SmartPtr.

Definition at line 644 of file IpSmartPtr.hpp.

3.152.4.3 template<class T> template<class U1 , class U2 > bool operator!= (const **SmartPtr**< U1 > & *lhs*, U2 * *raw_rhs*) [friend]

Overloaded in-equality comparison operator, allows the user to compare the value of a SmartPtr with a raw pointer.

Definition at line 670 of file IpSmartPtr.hpp.

3.152.4.4 template<class T> template<class U1 , class U2 > bool operator!= (U1 * *lhs*, const **SmartPtr**< U2 > & *raw_rhs*) [friend]

Overloaded in-equality comparison operator, allows the user to compare the value of a SmartPtr with a raw pointer.

Definition at line 683 of file IpSmartPtr.hpp.

3.152.4.5 template<class T> template<class U > U∗ GetRawPtr (const SmartPtr< U > & *smart_ptr*) [friend]

Returns the raw pointer contained.

Use to get the value of the raw ptr (i.e. to pass to other methods/functions, etc.) Note: This method does NOT copy, therefore, modifications using this value modify the underlying object contained by the SmartPtr, NEVER delete this returned value.

Definition at line 560 of file IpSmartPtr.hpp.

3.152.4.6 template<class T> template<class U > bool IsValid (const SmartPtr< U > & *smart_ptr*) [friend]

Returns true if the SmartPtr is NOT NULL.

Use this to check if the SmartPtr is not null This is preferred to if(GetRawPtr(sp) != NULL)

Definition at line 579 of file IpSmartPtr.hpp.

3.152.4.7 template<class T> template<class U > bool IsNull (const SmartPtr< U > & *smart_ptr*) [friend]

Returns true if the SmartPtr is NULL.

Use this to check if the SmartPtr IsNull. This is preferred to if(GetRawPtr(sp) == NULL)

Definition at line 585 of file IpSmartPtr.hpp.

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

- IpSmartPtr.hpp

3.153 Ipopt::SolveStatistics Class Reference

This class collects statistics about an optimziation run, such as iteration count, final infeasibilities etc.

`#include <IpSolveStatistics.hpp>`

Inheritance diagram for Ipopt::SolveStatistics:

Public Member Functions

Constructors/Destructors

- SolveStatistics (const SmartPtr< IpoptNLP > &ip_nlp, const SmartPtr< IpoptData > &ip_data, const SmartPtr< IpoptCalculatedQuantities > &ip_cq)
  Default constructor.
- virtual ∼SolveStatistics ()
  Default destructor.

Accessor methods for retrieving different kind of solver

statistics information

- virtual Index IterationCount () const
  Iteration counts.
- virtual Number TotalCpuTime () const
  Total CPU time, including function evaluations.

- Number TotalCPUTime () const
  Total CPU time, including function evaluations.
- virtual Number TotalSysTime () const
  Total System time, including function evaluations.
- virtual Number TotalWallclockTime () const
  Total wall clock time, including function evaluations.
- virtual void NumberOfEvaluations (Index &num_obj_evals, Index &num_constr_evals, Index &num_obj_grad↩_evals, Index &num_constr_jac_evals, Index &num_hess_evals) const
  Number of NLP function evaluations.
- virtual void Infeasibilities (Number &dual_inf, Number &constr_viol, Number &complementarity, Number &kkt↩_error) const
  Unscaled solution infeasibilities.
- virtual void ScaledInfeasibilities (Number &scaled_dual_inf, Number &scaled_constr_viol, Number &scaled↩_complementarity, Number &scaled_kkt_error) const
  Scaled solution infeasibilities.
- virtual Number FinalObjective () const
  Final value of objective function.
- virtual Number FinalScaledObjective () const
  Final scaled value of objective function.

3.153.1 Detailed Description

This class collects statistics about an optimziation run, such as iteration count, final infeasibilities etc.

It is meant to provide such information to a user of Ipopt during the finalize_solution call.

Definition at line 27 of file IpSolveStatistics.hpp.

3.153.2 Constructor & Destructor Documentation

3.153.2.1 Ipopt::SolveStatistics::SolveStatistics (const SmartPtr< IpoptNLP > & *ip_nlp*, const SmartPtr< IpoptData > & *ip_data*, const SmartPtr< IpoptCalculatedQuantities > & *ip_cq*)

Default constructor.

It takes in those collecting Ipopt objects that can provide the statistics information. Those statistics are retrieved at the time of the constructor call.

3.153.3 Member Function Documentation

3.153.3.1 virtual Index Ipopt::SolveStatistics::IterationCount ( ) const [virtual]

Iteration counts.

3.153.3.2 virtual Number Ipopt::SolveStatistics::TotalCpuTime ( ) const [virtual]

Total CPU time, including function evaluations.

3.153.3.3 Number Ipopt::SolveStatistics::TotalCPUTime ( ) const [inline]

Total CPU time, including function evaluations.

Included for backward compatibility.

Definition at line 54 of file IpSolveStatistics.hpp.

3.153.3.4 virtual Number Ipopt::SolveStatistics::TotalSysTime ( ) const [virtual]

Total System time, including function evaluations.

3.153.3.5 virtual Number Ipopt::SolveStatistics::TotalWallclockTime ( ) const [virtual]

Total wall clock time, including function evaluations.

3.153.3.6 virtual void Ipopt::SolveStatistics::NumberOfEvaluations (Index & *num_obj_evals,* Index & *num_constr_evals,* Index & *num_obj_grad_evals,* Index & *num_constr_jac_evals,* Index & *num_hess_evals*) const [virtual]

Number of NLP function evaluations.

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

- IpSolveStatistics.hpp

3.154 Ipopt::SparseSymLinearSolverInterface Class Reference

Base class for interfaces to symmetric indefinite linear solvers for sparse matrices.

`#include <IpSparseSymLinearSolverInterface.hpp>`

Inheritance diagram for Ipopt::SparseSymLinearSolverInterface:

Public Types

- enum EMatrixFormat {
  Triplet_Format, CSR_Format_0_Offset, CSR_Format_1_Offset, CSR_Full_Format_0_Offset,
  CSR_Full_Format_1_Offset }

  Enum to specify sparse matrix format.

Public Member Functions

- virtual bool InitializeImpl (const OptionsList &options, const std::string &prefix)=0

  overloaded from AlgorithmStrategyObject

Constructor/Destructor

- **SparseSymLinearSolverInterface** ()
- virtual ∼**SparseSymLinearSolverInterface** ()

Methods for requesting solution of the linear system.

- virtual ESymSolverStatus InitializeStructure (Index dim, Index nonzeros, const Index ∗ia, const Index ∗ja)=0

  Method for initializing internal stuctures.

- virtual double * GetValuesArrayPtr ()=0

    Method returing an internal array into which the nonzero elements (in the same order as ja) will be stored by the calling routine before a call to MultiSolve with a new_matrix=true (or after a return of MultiSolve with SYMSOLV_CALL_AG←AIN).
- virtual ESymSolverStatus MultiSolve (bool new_matrix, const Index *ia, const Index *ja, Index nrhs, double *rhs_vals, bool check_NegEVals, Index numberOfNegEVals)=0

    Solve operation for multiple right hand sides.
- virtual Index NumberOfNegEVals () const =0

    Number of negative eigenvalues detected during last factorization.

- virtual bool IncreaseQuality ()=0

    Request to increase quality of solution for next solve.
- virtual bool ProvidesInertia () const =0

    Query whether inertia is computed by linear solver.
- virtual EMatrixFormat MatrixFormat () const =0

    Query of requested matrix type that the linear solver understands.

Methods related to the detection of linearly dependent

rows in a matrix

- virtual bool ProvidesDegeneracyDetection () const

    Query whether the indices of linearly dependent rows/columns can be determined by this linear solver.
- virtual ESymSolverStatus DetermineDependentRows (const Index *ia, const Index *ja, std::list< Index > &c←_deps)

    This method determines the list of row indices of the linearly dependent rows.

Additional Inherited Members

3.154.1 Detailed Description

Base class for interfaces to symmetric indefinite linear solvers for sparse matrices.

This defines the general interface to linear solvers for sparse symmetric indefinite matrices. The matrices can be provided either in "triplet format" (like for Harwell's MA27 solver), or in compressed sparse row (CSR) format for the lower triangular part of the symmetric matrix.

The solver should be able to compute the interia of the matrix, or more specifically, the number of negative eigenvalues in the factorized matrix.

This interface is used by the calling objective in the following way:

1. The InitializeImpl method is called at the very beginning (for every optimization run), which allows the linear solver object to retrieve options given in the OptionsList (such as pivot tolerances etc). At this point, some internal data can also be initialized.

2. The calling class calls MatrixFormat to find out which matrix representation the linear solver requires. The possible options are Triplet_Format, as well as CSR_Format_0_Offset and CSR_Format_1_Offset. The difference between the last two is that for CSR_Format_0_Offset the couning of the element position in the ia and ja arrays starts are 0 (C-style numbering), whereas for the other one it starts at 1 (Fortran-style numbering).

3. After this, the InitializeStructure method is called (once). Here, the structure of the matrix is provided. If the linear solver requires a symbolic preprocessing phase that can be done without knowledge of the matrix element values, it can be done here.

4. The calling class will request an array for storing the actual values for a matrix using the GetValuesArrayPtr method. This array must be at least as large as the number of nonzeros in the matrix (as given to this class by the InitializeStructure method call). After a call of this method, the calling class will fill this array with the actual values of the matrix.
5. Every time lateron, when actual solves of a linear system is requested, the calling class will call the MultiSolve to request the solve, possibly for mulitple right-hand sides. The flag new_matrix then indicates if the values of the matrix have changed and if a factorization is required, or if an old factorization can be used to do the solve.

Note that the GetValuesArrayPtr method will be called before every call of MultiSolve with new_matrix=true, or before a renewed call of MultiSolve if the most previous return value was SYMSOLV_CALL_AGAIN.

1. The calling class might request with NumberOfNegEVals the number of the negative eigenvalues for the original matrix that were detected during the most recently performed factorization.
2. The calling class might ask the linear solver to increase the quality of the solution. For example, if the linear solver uses a pivot tolerance, a larger value should be used for the next solve (which might require a refactorization).
3. Finally, when the destructor is called, the internal storage, also in the linear solver, should be released.

Note, if the matrix is given in triplet format, entries might be listed multiple times, in which case the corresponsing elements have to be added.

A note for warm starts: If the option "warm_start_same_structure" is specified with "yes", the algorithm assumes that a problem with the same sparsity structure is solved for a repeated time. In that case, the linear solver might reuse information from the previous optimization. See Ma27TSolverInterface for an example.

Definition at line 98 of file IpSparseSymLinearSolverInterface.hpp.

3.154.2 Member Enumeration Documentation

3.154.2.1 enum **Ipopt::SparseSymLinearSolverInterface::EMatrixFormat**

Enum to specify sparse matrix format.

Enumerator

Triplet_Format Triplet (MA27) format.

CSR_Format_0_Offset Compressed sparse row format for lower triangular part, with 0 offset.

CSR_Format_1_Offset Compressed sparse row format for lower triangular part, with 1 offset.

CSR_Full_Format_0_Offset Compressed sparse row format for both lwr and upr parts, with 0 offset.

CSR_Full_Format_1_Offset Compressed sparse row format for both lwr and upr parts, with 1 offset.

Definition at line 102 of file IpSparseSymLinearSolverInterface.hpp.

3.154.3 Member Function Documentation

3.154.3.1 virtual ESymSolverStatus Ipopt::SparseSymLinearSolverInterface::InitializeStructure (Index *dim*, Index *nonzeros*, const Index ∗ *ia*, const Index ∗ *ja*) [pure virtual]

Method for initializing internal stuctures.

Here, ndim gives the number of rows and columns of the matrix, nonzeros give the number of nonzero elements, and ia and ja give the positions of the nonzero elements, given in the matrix format determined by MatrixFormat.

Implemented in Ipopt::Ma97SolverInterface, Ipopt::Ma86SolverInterface, Ipopt::Ma77SolverInterface, Ipopt::Ma57↩
TSolverInterface, Ipopt::MumpsSolverInterface, Ipopt::IterativePardisoSolverInterface, Ipopt::PardisoSolverInterface, Ipopt::WsmpSolverInterface, Ipopt::IterativeWsmpSolverInterface, and Ipopt::Ma27TSolverInterface.

3.154.3.2 virtual double∗ Ipopt::SparseSymLinearSolverInterface::GetValuesArrayPtr ( ) [pure virtual]

Method returing an internal array into which the nonzero elements (in the same order as ja) will be stored by the calling routine before a call to MultiSolve with a new_matrix=true (or after a return of MultiSolve with SYMSOLV_CALL_AGAIN).

The returned array must have space for at least nonzero elements.

Implemented in Ipopt::Ma97SolverInterface, Ipopt::Ma86SolverInterface, Ipopt::Ma77SolverInterface, Ipopt::Ma57↩
TSolverInterface, Ipopt::MumpsSolverInterface, Ipopt::IterativePardisoSolverInterface, Ipopt::Ma27TSolverInterface, Ipopt::PardisoSolverInterface, Ipopt::WsmpSolverInterface, and Ipopt::IterativeWsmpSolverInterface.

3.154.3.3 virtual ESymSolverStatus Ipopt::SparseSymLinearSolverInterface::MultiSolve (bool *new_matrix,* const Index ∗ *ia,* const Index ∗ *ja,* Index *nrhs,* double ∗ *rhs_vals,* bool *check_NegEVals,* Index *numberOfNegEVals*) [pure virtual]

Solve operation for multiple right hand sides.

Solves the linear system A ∗ x = b with multiple right hand sides, where A is the symmtric indefinite matrix. Here, ia and ja give the positions of the values (in the required matrix data format). The actual values of the matrix will have been given to this object by copying them into the array provided by GetValuesArrayPtr. ia and ja are identical to the ones given to InitializeStructure. The flag new_matrix is set to true, if the values of the matrix has changed, and a refactorzation is required.

The return code is SYMSOLV_SUCCESS if the factorization and solves were successful, SYMSOLV_SINGULAR if the linear system is singular, and SYMSOLV_WRONG_INERTIA if check_NegEVals is true and the number of negative eigenvalues in the matrix does not match numberOfNegEVals. If SYMSOLV_CALL_AGAIN is returned, then the calling function will request the pointer for the array for storing a again (with GetValuesPtr), write the values of the nonzero elements into it, and call this MultiSolve method again with the same right-hand sides. (This can be done, for example, if the linear solver realized it does not have sufficient memory and needs to redo the factorization; e.g., for MA27.)

The number of right-hand sides is given by nrhs, the values of the right-hand sides are given in rhs_vals (one full right-hand side stored immediately after the other), and solutions are to be returned in the same array.

check_NegEVals will not be chosen true, if ProvidesInertia() returns false.

Implemented in Ipopt::Ma97SolverInterface, Ipopt::Ma86SolverInterface, Ipopt::Ma77SolverInterface, Ipopt::Ma57↩
TSolverInterface, Ipopt::MumpsSolverInterface, Ipopt::Ma27TSolverInterface, Ipopt::IterativePardisoSolverInterface, Ipopt::PardisoSolverInterface, Ipopt::WsmpSolverInterface, and Ipopt::IterativeWsmpSolverInterface.

3.154.3.4 virtual Index Ipopt::SparseSymLinearSolverInterface::NumberOfNegEVals ( ) const [pure virtual]

Number of negative eigenvalues detected during last factorization.

Returns the number of negative eigenvalues of the most recent factorized matrix. This must not be called if the linear solver does not compute this quantities (see ProvidesInertia).

Implemented in Ipopt::Ma97SolverInterface, Ipopt::Ma86SolverInterface, Ipopt::Ma77SolverInterface, Ipopt::Ma57↩
TSolverInterface, Ipopt::MumpsSolverInterface, Ipopt::Ma27TSolverInterface, Ipopt::IterativePardisoSolverInterface, Ipopt::PardisoSolverInterface, Ipopt::WsmpSolverInterface, and Ipopt::IterativeWsmpSolverInterface.

3.154.3.5 virtual bool Ipopt::SparseSymLinearSolverInterface::IncreaseQuality ( ) [pure virtual]

Request to increase quality of solution for next solve.

The calling class asks linear solver to increase quality of solution for the next solve (e.g. increase pivot tolerance). Returns false, if this is not possible (e.g. maximal pivot tolerance already used.)

Implemented in Ipopt::Ma97SolverInterface, Ipopt::Ma86SolverInterface, Ipopt::Ma77SolverInterface, Ipopt::Ma57↩TSolverInterface, Ipopt::MumpsSolverInterface, Ipopt::Ma27TSolverInterface, Ipopt::IterativePardisoSolverInterface, Ipopt::PardisoSolverInterface, Ipopt::WsmpSolverInterface, and Ipopt::IterativeWsmpSolverInterface.

3.154.3.6 virtual bool Ipopt::SparseSymLinearSolverInterface::ProvidesInertia ( ) const [pure virtual]

Query whether inertia is computed by linear solver.

Returns true, if linear solver provides inertia.

Implemented in Ipopt::Ma97SolverInterface, Ipopt::Ma86SolverInterface, Ipopt::Ma77SolverInterface, Ipopt::Ma57↩TSolverInterface, Ipopt::MumpsSolverInterface, Ipopt::Ma27TSolverInterface, Ipopt::IterativePardisoSolverInterface, Ipopt::PardisoSolverInterface, Ipopt::WsmpSolverInterface, and Ipopt::IterativeWsmpSolverInterface.

3.154.3.7 virtual bool Ipopt::SparseSymLinearSolverInterface::ProvidesDegeneracyDetection ( ) const [inline], [virtual]

Query whether the indices of linearly dependent rows/columns can be determined by this linear solver.

Reimplemented in Ipopt::Ma97SolverInterface, Ipopt::Ma86SolverInterface, Ipopt::Ma77SolverInterface, Ipopt::↩MumpsSolverInterface, and Ipopt::WsmpSolverInterface.

Definition at line 226 of file IpSparseSymLinearSolverInterface.hpp.

3.154.3.8 virtual ESymSolverStatus Ipopt::SparseSymLinearSolverInterface::DetermineDependentRows (const Index ∗ *ia*, const Index ∗ *ja*, std::list< Index > & *c_deps*) [inline], [virtual]

This method determines the list of row indices of the linearly dependent rows.

Reimplemented in Ipopt::Ma97SolverInterface, Ipopt::Ma86SolverInterface, Ipopt::Ma77SolverInterface, Ipopt::↩MumpsSolverInterface, and Ipopt::WsmpSolverInterface.

Definition at line 232 of file IpSparseSymLinearSolverInterface.hpp.

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

- IpSparseSymLinearSolverInterface.hpp

3.155 Ipopt::StandardScalingBase Class Reference

This is a base class for many standard scaling techniques.

`#include <IpNLPScaling.hpp>`

Inheritance diagram for Ipopt::StandardScalingBase:

Public Member Functions

- virtual void DetermineScaling (const SmartPtr< const VectorSpace > x_space, const SmartPtr< const Vector↩Space > c_space, const SmartPtr< const VectorSpace > d_space, const SmartPtr< const MatrixSpace > jac_c_space, const SmartPtr< const MatrixSpace > jac_d_space, const SmartPtr< const SymMatrixSpace >

h_space, SmartPtr< const MatrixSpace > &new_jac_c_space, SmartPtr< const MatrixSpace > &new_jac_↩
d_space, SmartPtr< const SymMatrixSpace > &new_h_space, const Matrix &Px_L, const Vector &x_L, const Matrix &Px_U, const Vector &x_U)

This method is called by the IpoptNLP's at a convenient time to compute and/or read scaling factors.

Constructors/Destructors

- **StandardScalingBase** ()
- virtual ∼StandardScalingBase ()

  Default destructor.

- virtual Number apply_obj_scaling (const Number &f)

  Methods to map scaled and unscaled matrices.
- virtual Number unapply_obj_scaling (const Number &f)

  Returns an obj-unscaled version of the given scalar.
- virtual SmartPtr< Vector > apply_vector_scaling_x_NonConst (const SmartPtr< const Vector > &v)

  Returns an x-scaled version of the given vector.
- virtual SmartPtr< const Vector > apply_vector_scaling_x (const SmartPtr< const Vector > &v)

  Returns an x-scaled version of the given vector.
- virtual SmartPtr< Vector > unapply_vector_scaling_x_NonConst (const SmartPtr< const Vector > &v)

  Returns an x-unscaled version of the given vector.
- virtual SmartPtr< const Vector > unapply_vector_scaling_x (const SmartPtr< const Vector > &v)

  Returns an x-unscaled version of the given vector.
- virtual SmartPtr< const Vector > apply_vector_scaling_c (const SmartPtr< const Vector > &v)

  Returns an c-scaled version of the given vector.
- virtual SmartPtr< const Vector > unapply_vector_scaling_c (const SmartPtr< const Vector > &v)

  Returns an c-unscaled version of the given vector.
- virtual SmartPtr< Vector > apply_vector_scaling_c_NonConst (const SmartPtr< const Vector > &v)

  Returns an c-scaled version of the given vector.
- virtual SmartPtr< Vector > unapply_vector_scaling_c_NonConst (const SmartPtr< const Vector > &v)

  Returns an c-unscaled version of the given vector.
- virtual SmartPtr< const Vector > apply_vector_scaling_d (const SmartPtr< const Vector > &v)

  Returns an d-scaled version of the given vector.
- virtual SmartPtr< const Vector > unapply_vector_scaling_d (const SmartPtr< const Vector > &v)

  Returns an d-unscaled version of the given vector.
- virtual SmartPtr< Vector > apply_vector_scaling_d_NonConst (const SmartPtr< const Vector > &v)

  Returns an d-scaled version of the given vector.
- virtual SmartPtr< Vector > unapply_vector_scaling_d_NonConst (const SmartPtr< const Vector > &v)

  Returns an d-unscaled version of the given vector.
- virtual SmartPtr< const Matrix > apply_jac_c_scaling (SmartPtr< const Matrix > matrix)

  Returns a scaled version of the jacobian for c.
- virtual SmartPtr< const Matrix > apply_jac_d_scaling (SmartPtr< const Matrix > matrix)

  Returns a scaled version of the jacobian for d If the overloaded method does not create a new matrix, make sure to set the matrix ptr passed in to NULL.
- virtual SmartPtr< const SymMatrix > apply_hessian_scaling (SmartPtr< const SymMatrix > matrix)

  Returns a scaled version of the hessian of the lagrangian If the overloaded method does not create a new matrix, make sure to set the matrix ptr passed in to NULL.

Methods for determining whether scaling for entities is

done

- virtual bool have_x_scaling ()

  Returns true if the primal x variables are scaled.
- virtual bool have_c_scaling ()

  Returns true if the equality constraints are scaled.
- virtual bool have_d_scaling ()

  Returns true if the inequality constraints are scaled.

Static Public Member Functions

- static void RegisterOptions (SmartPtr< RegisteredOptions > roptions)

  Methods for IpoptType.

Protected Member Functions

- virtual bool InitializeImpl (const OptionsList &options, const std::string &prefix)

  Overloaded initialization method.
- virtual void DetermineScalingParametersImpl (const SmartPtr< const VectorSpace > x_space, const SmartPtr< const VectorSpace > c_space, const SmartPtr< const VectorSpace > d_space, const SmartPtr< const MatrixSpace > jac_c_space, const SmartPtr< const MatrixSpace > jac_d_space, const SmartPtr< const SymMatrixSpace > h_space, const Matrix &Px_L, const Vector &x_L, const Matrix &Px_U, const Vector &x_U, Number &df, SmartPtr< Vector > &dx, SmartPtr< Vector > &dc, SmartPtr< Vector > &dd)=0

  This is the method that has to be overloaded by a particular scaling method that somehow computes the scaling vectors dx, dc, and dd.

3.155.1 Detailed Description

This is a base class for many standard scaling techniques.

The overloaded classes only need to provide the scaling parameters

Definition at line 229 of file IpNLPScaling.hpp.

3.155.2 Member Function Documentation

3.155.2.1 virtual Number Ipopt::StandardScalingBase::apply_obj_scaling (const Number & *f*) [virtual]

Methods to map scaled and unscaled matrices.

Returns an obj-scaled version of the given scalar

Implements Ipopt::NLPScalingObject.

3.155.2.2 virtual SmartPtr<const Matrix> Ipopt::StandardScalingBase::apply_jac_c_scaling (SmartPtr< const Matrix > *matrix*) [virtual]

Returns a scaled version of the jacobian for c.

If the overloaded method does not make a new matrix, make sure to set the matrix ptr passed in to NULL.

Implements Ipopt::NLPScalingObject.

3.155.2.3 virtual bool Ipopt::StandardScalingBase::have_x_scaling ( ) [virtual]

Returns true if the primal x variables are scaled.

Implements Ipopt::NLPScalingObject.

3.155.2.4 virtual bool Ipopt::StandardScalingBase::have_c_scaling ( ) [virtual]

Returns true if the equality constraints are scaled.

Implements Ipopt::NLPScalingObject.

3.155.2.5 virtual bool Ipopt::StandardScalingBase::have_d_scaling ( ) [virtual]

Returns true if the inequality constraints are scaled.

Implements Ipopt::NLPScalingObject.

3.155.2.6 virtual void Ipopt::StandardScalingBase::DetermineScalingParametersImpl (const SmartPtr< const VectorSpace > *x_space*, const SmartPtr< const VectorSpace > *c_space*, const SmartPtr< const VectorSpace > *d_space*, const SmartPtr< const MatrixSpace > *jac_c_space*, const SmartPtr< const MatrixSpace > *jac_d_space*, const SmartPtr< const SymMatrixSpace > *h_space*, const Matrix & *Px_L*, const Vector & *x_L*, const Matrix & *Px_U*, const Vector & *x_U*, Number & *df*, SmartPtr< Vector > & *dx*, SmartPtr< Vector > & *dc*, SmartPtr< Vector > & *dd*) [protected], [pure virtual]

This is the method that has to be overloaded by a particular scaling method that somehow computes the scaling vectors dx, dc, and dd.

The pointers to those vectors can be NULL, in which case no scaling for that item will be done later.

Implemented in Ipopt::NoNLPScalingObject, Ipopt::EquilibrationScaling, Ipopt::GradientScaling, and Ipopt::UserScaling.

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

- IpNLPScaling.hpp

3.156 Ipopt::StdAugSystemSolver Class Reference

Solver for the augmented system for triple type matrices.

`#include <IpStdAugSystemSolver.hpp>`

Inheritance diagram for Ipopt::StdAugSystemSolver:

Public Member Functions

- bool InitializeImpl (const OptionsList &options, const std::string &prefix)

  overloaded from AlgorithmStrategyObject
- virtual ESymSolverStatus MultiSolve (const SymMatrix ∗W, double W_factor, const Vector ∗D_x, double delta_x, const Vector ∗D_s, double delta_s, const Matrix ∗J_c, const Vector ∗D_c, double delta_c, const Matrix ∗J_d, const Vector ∗D_d, double delta_d, std::vector< SmartPtr< const Vector > > &rhs_xV, std::vector< SmartPtr< const Vector > > &rhs_sV, std::vector< SmartPtr< const Vector > > &rhs_cV, std::vector< SmartPtr< const Vector

> > &rhs_dV, std::vector< SmartPtr< Vector > > &sol_xV, std::vector< SmartPtr< Vector > > &sol_sV, std::vector< SmartPtr< Vector > > &sol_cV, std::vector< SmartPtr< Vector > > &sol_dV, bool check_NegEVals, Index numberOfNegEVals)

Set up the augmented system and solve it for a set of given right hand side - implementation for GenTMatrices and SymTMatrices.

- virtual Index NumberOfNegEVals () const

  Number of negative eigenvalues detected during last solve.
- virtual bool ProvidesInertia () const

  Query whether inertia is computed by linear solver.
- virtual bool IncreaseQuality ()

  Request to increase quality of solution for next solve.

Constructors/Destructors

- StdAugSystemSolver (SymLinearSolver &LinSolver)

  Constructor using only a linear solver object.
- virtual ∼StdAugSystemSolver ()

  Default destructor.

Additional Inherited Members

3.156.1 Detailed Description

Solver for the augmented system for triple type matrices.

The current implemetation assumes that all matrices are of the type SymTMatrix, and all vectors are of the type DenseVector.

Definition at line 27 of file IpStdAugSystemSolver.hpp.

3.156.2 Member Function Documentation

3.156.2.1 virtual Index Ipopt::StdAugSystemSolver::NumberOfNegEVals ( ) const [virtual]

Number of negative eigenvalues detected during last solve.

Returns the number of negative eigenvalues of the most recent factorized matrix. This must not be called if the linear solver does not compute this quantities (see ProvidesInertia).

Implements Ipopt::AugSystemSolver.

3.156.2.2 virtual bool Ipopt::StdAugSystemSolver::ProvidesInertia ( ) const [virtual]

Query whether inertia is computed by linear solver.

Returns true, if linear solver provides inertia.

Implements Ipopt::AugSystemSolver.

3.156.2.3 virtual bool Ipopt::StdAugSystemSolver::IncreaseQuality ( ) [virtual]

Request to increase quality of solution for next solve.

Ask underlying linear solver to increase quality of solution for the next solve (e.g. increase pivot tolerance). Returns false, if this is not possible (e.g. maximal pivot tolerance already used.)

Implements Ipopt::AugSystemSolver.

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

- IpStdAugSystemSolver.hpp

3.157 Ipopt::StdInterfaceTNLP Class Reference

Implementation of a TNLP for the Standard C interface.

`#include <IpStdInterfaceTNLP.hpp>`

Inheritance diagram for Ipopt::StdInterfaceTNLP:

Public Member Functions

Constructors/Destructors

- StdInterfaceTNLP (Index n_var, const Number ∗x_L, const Number ∗x_U, Index n_con, const Number ∗g←_L, const Number ∗g_U, Index nele_jac, Index nele_hess, Index index_style, const Number ∗start_x, const Number ∗start_lam, const Number ∗start_z_L, const Number ∗start_z_U, Eval_F_CB eval_f, Eval_G_C←B eval_g, Eval_Grad_F_CB eval_grad_f, Eval_Jac_G_CB eval_jac_g, Eval_H_CB eval_h, Intermediate_CB intermediate_cb, Number ∗x_sol, Number ∗z_L_sol, Number ∗z_U_sol, Number ∗g_sol, Number ∗lam_←sol, Number ∗obj_sol, UserDataPtr user_data, Number obj_scaling=1, const Number ∗x_scaling=NULL, const Number ∗g_scaling=NULL)

  Constructor, given dimensions of problem, function pointers for evaluation callback functions, and starting points.
- virtual ∼StdInterfaceTNLP ()

  Default destructor.

methods to gather information about the NLP. These methods are

overloaded from TNLP.

See TNLP for their more detailed documentation.

- virtual bool get_nlp_info (Index &n, Index &m, Index &nnz_jac_g, Index &nnz_h_lag, IndexStyleEnum &index←_style)

  returns dimensions of the nlp.
- virtual bool get_bounds_info (Index n, Number ∗x_l, Number ∗x_u, Index m, Number ∗g_l, Number ∗g_u)

  returns bounds of the nlp.
- virtual bool get_scaling_parameters (Number &obj_scaling, bool &use_x_scaling, Index n, Number ∗x_scaling, bool &use_g_scaling, Index m, Number ∗g_scaling)

  returns scaling parameters (if nlp_scaling_method is selected as user-scaling).
- virtual bool get_starting_point (Index n, bool init_x, Number ∗x, bool init_z, Number ∗z_L, Number ∗z_U, Index m, bool init_lambda, Number ∗lambda)

  provides a starting point for the nlp variables.
- virtual bool eval_f (Index n, const Number ∗x, bool new_x, Number &obj_value)

  evaluates the objective value for the nlp.
- virtual bool eval_grad_f (Index n, const Number ∗x, bool new_x, Number ∗grad_f)

evaluates the gradient of the objective for the nlp.

- virtual bool eval_g (Index n, const Number ∗x, bool new_x, Index m, Number ∗g)

  evaluates the constraint residuals for the nlp.
- virtual bool eval_jac_g (Index n, const Number ∗x, bool new_x, Index m, Index nele_jac, Index ∗iRow, Index ∗jCol, Number ∗values)

  specifies the jacobian structure (if values is NULL) and evaluates the jacobian values (if values is not NULL) for the nlp.
- virtual bool eval_h (Index n, const Number ∗x, bool new_x, Number obj_factor, Index m, const Number ∗lambda, bool new_lambda, Index nele_hess, Index ∗iRow, Index ∗jCol, Number ∗values)

  specifies the structure of the hessian of the lagrangian (if values is NULL) and evaluates the values (if values is not NULL).
- virtual bool intermediate_callback (AlgorithmMode mode, Index iter, Number obj_value, Number inf_pr, Number inf_du, Number mu, Number d_norm, Number regularization_size, Number alpha_du, Number alpha_pr, Index ls_trials, const IpoptData ∗ip_data, IpoptCalculatedQuantities ∗ip_cq)

  Intermediate Callback method for the user.

Solution Methods

- virtual void finalize_solution (SolverReturn status, Index n, const Number ∗x, const Number ∗z_L, const Number ∗z_U, Index m, const Number ∗g, const Number ∗lambda, Number obj_value, const IpoptData ∗ip_data, IpoptCalculatedQuantities ∗ip_cq)

  This method is called when the algorithm is complete so the TNLP can store/write the solution.

Additional Inherited Members

3.157.1 Detailed Description

Implementation of a TNLP for the Standard C interface.

The standard C interface is exposed to the user as a single C function that is given problem dimension, starting points, and pointers for functions that evaluate objective function etc.

Definition at line 30 of file IpStdInterfaceTNLP.hpp.

3.157.2 Constructor & Destructor Documentation

3.157.2.1 Ipopt::StdInterfaceTNLP::StdInterfaceTNLP (Index *n_var*, const Number ∗ *x_L*, const Number ∗ *x_U*, Index *n_con*, const Number ∗ *g_L*, const Number ∗ *g_U*, Index *nele_jac*, Index *nele_hess*, Index *index_style*, const Number ∗ *start_x*, const Number ∗ *start_lam*, const Number ∗ *start_z_L*, const Number ∗ *start_z_U*, Eval_F_CB *eval_f*, Eval_G_CB *eval_g*, Eval_Grad_F_CB *eval_grad_f*, Eval_Jac_G_CB *eval_jac_g*, Eval_H_CB *eval_h*, Intermediate_CB *intermediate_cb*, Number ∗ *x_sol*, Number ∗ *z_L_sol*, Number ∗ *z_U_sol*, Number ∗ *g_sol*, Number ∗ *lam_sol*, Number ∗ *obj_sol*, UserDataPtr *user_data*, Number *obj_scaling* = `1`, const Number ∗ *x_scaling* = `NULL`, const Number ∗ *g_scaling* = `NULL`)

Constructor, given dimensions of problem, function pointers for evaluation callback functions, and starting points.

Note that the constrctor does not make a copy of any of the Number arrays, i.e. it is up to the called to keep them around.

3.157.3 Member Function Documentation

3.157.3.1 virtual bool Ipopt::StdInterfaceTNLP::get_nlp_info (Index & *n*, Index & *m*, Index & *nnz_jac_g*, Index & *nnz_h_lag*, IndexStyleEnum & *index_style*) `[virtual]`

returns dimensions of the nlp.

Overloaded from TNLP

Implements Ipopt::TNLP.

3.157.3.2 virtual bool Ipopt::StdInterfaceTNLP::get_bounds_info (Index *n*, Number * *x_l*, Number * *x_u*, Index *m*, Number * *g_l*, Number * *g_u*) [virtual]

returns bounds of the nlp.

Overloaded from TNLP

Implements Ipopt::TNLP.

3.157.3.3 virtual bool Ipopt::StdInterfaceTNLP::get_scaling_parameters (Number & *obj_scaling*, bool & *use_x_scaling*, Index *n*, Number * *x_scaling*, bool & *use_g_scaling*, Index *m*, Number * *g_scaling*) [virtual]

returns scaling parameters (if nlp_scaling_method is selected as user-scaling).

Overloaded from TNLP

Reimplemented from Ipopt::TNLP.

3.157.3.4 virtual bool Ipopt::StdInterfaceTNLP::get_starting_point (Index *n*, bool *init_x*, Number * *x*, bool *init_z*, Number * *z_L*, Number * *z_U*, Index *m*, bool *init_lambda*, Number * *lambda*) [virtual]

provides a starting point for the nlp variables.

Overloaded from TNLP

Implements Ipopt::TNLP.

3.157.3.5 virtual bool Ipopt::StdInterfaceTNLP::eval_f (Index *n*, const Number * *x*, bool *new_x*, Number & *obj_value*) [virtual]

evaluates the objective value for the nlp.

Overloaded from TNLP

Implements Ipopt::TNLP.

3.157.3.6 virtual bool Ipopt::StdInterfaceTNLP::eval_grad_f (Index *n*, const Number * *x*, bool *new_x*, Number * *grad_f*) [virtual]

evaluates the gradient of the objective for the nlp.

Overloaded from TNLP

Implements Ipopt::TNLP.

3.157.3.7 virtual bool Ipopt::StdInterfaceTNLP::eval_g (Index *n*, const Number * *x*, bool *new_x*, Index *m*, Number * *g*) [virtual]

evaluates the constraint residuals for the nlp.

Overloaded from TNLP

Implements Ipopt::TNLP.

3.157.3.8 virtual bool Ipopt::StdInterfaceTNLP::eval_jac_g (Index *n*, const Number ∗ *x*, bool *new_x*, Index *m*, Index *nele_jac*, Index ∗ *iRow*, Index ∗ *jCol*, Number ∗ *values*) [virtual]

specifies the jacobian structure (if values is NULL) and evaluates the jacobian values (if values is not NULL) for the nlp.

Overloaded from TNLP

Implements Ipopt::TNLP.

3.157.3.9 virtual bool Ipopt::StdInterfaceTNLP::eval_h (Index *n*, const Number ∗ *x*, bool *new_x*, Number *obj_factor*, Index *m*, const Number ∗ *lambda*, bool *new_lambda*, Index *nele_hess*, Index ∗ *iRow*, Index ∗ *jCol*, Number ∗ *values*) [virtual]

specifies the structure of the hessian of the lagrangian (if values is NULL) and evaluates the values (if values is not NULL).

Overloaded from TNLP

Reimplemented from Ipopt::TNLP.

3.157.3.10 virtual bool Ipopt::StdInterfaceTNLP::intermediate_callback (AlgorithmMode *mode*, Index *iter*, Number *obj_value*, Number *inf_pr*, Number *inf_du*, Number *mu*, Number *d_norm*, Number *regularization_size*, Number *alpha_du*, Number *alpha_pr*, Index *ls_trials*, const IpoptData ∗ *ip_data*, IpoptCalculatedQuantities ∗ *ip_cq*) [virtual]

Intermediate Callback method for the user.

Overloaded from TNLP

Reimplemented from Ipopt::TNLP.

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

- IpStdInterfaceTNLP.hpp

3.158 Ipopt::StreamJournal Class Reference

StreamJournal class.

`#include <IpJournalist.hpp>`

Inheritance diagram for Ipopt::StreamJournal:

Public Member Functions

- StreamJournal (const std::string &name, EJournalLevel default_level)

  Constructor.
- virtual ∼StreamJournal ()

  Destructor.
- void SetOutputStream (std::ostream ∗os)

  Setting the output stream pointer.

Protected Member Functions

Implementation version of Print methods - Overloaded from

Journal base class.

- virtual void PrintImpl (EJournalCategory category, EJournalLevel level, const char ∗str)
  Print to the designated output location.
- virtual void PrintfImpl (EJournalCategory category, EJournalLevel level, const char ∗pformat, va_list ap)
  Printf to the designated output location.
- virtual void FlushBufferImpl ()
  Flush output buffer.

3.158.1 Detailed Description

StreamJournal class.

This is a particular Journal implementation that writes to a stream for output.

Definition at line 440 of file IpJournalist.hpp.

3.158.2 Constructor & Destructor Documentation

3.158.2.1 Ipopt::StreamJournal::StreamJournal (const std::string & *name,* EJournalLevel *default_level*)

Constructor.

3.158.2.2 virtual Ipopt::StreamJournal::∼StreamJournal ( ) [inline], [virtual]

Destructor.

Definition at line 447 of file IpJournalist.hpp.

3.158.3 Member Function Documentation

3.158.3.1 virtual void Ipopt::StreamJournal::FlushBufferImpl ( ) [protected], [virtual]

Flush output buffer.

Implements Ipopt::Journal.

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

- IpJournalist.hpp

3.159 Ipopt::RegisteredOption::string_entry Class Reference

class to hold the valid string settings for a string option

`#include <IpRegOptions.hpp>`

3.159.1 Detailed Description

class to hold the valid string settings for a string option

Definition at line 37 of file IpRegOptions.hpp.

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

- IpRegOptions.hpp

3.160 Ipopt::Subject Class Reference

Slight Variation of the Observer Design Pattern (Subject part).

#include <IpObserver.hpp>

Inheritance diagram for Ipopt::Subject:

Public Member Functions

Constructors/Destructors

- Subject ()
  Default Constructor.
- virtual ∼Subject ()
  Default destructor.

Methods to Add and Remove Observers.

Currently, the notify_type flags are not used, and Observers are attached in general and will recieve all notifications (of the type requested and possibly of types not requested).

It is up to the observer to ignore the types they are not interested in. The NotifyType in the parameter list is so a more efficient mechanism depending on type could be implemented later if necessary.

- void AttachObserver (Observer::NotifyType notify_type, Observer ∗observer) const
  Attach the specified observer (i.e., begin recieving notifications).
- void DetachObserver (Observer::NotifyType notify_type, Observer ∗observer) const
  Detach the specified observer (i.e., no longer recieve notifications).

3.160.1 Detailed Description

Slight Variation of the Observer Design Pattern (Subject part).

This class implements the Subject class of the Observer Design Pattern. An Observer "Attach"es to a Subject, indicating that it would like to be notified of changes in the Subject. Any derived class that is to be observed has to inherit off the Subject base class. If the subject needs to notify the Observer, it calls the Notify method.

Definition at line 129 of file IpObserver.hpp.

3.160.2 Member Function Documentation

3.160.2.1 void Ipopt::Subject::AttachObserver (Observer::NotifyType *notify_type*, Observer ∗ *observer*) const [inline]

Attach the specified observer (i.e., begin recieving notifications).

Definition at line 309 of file IpObserver.hpp.

3.160.2.2 void Ipopt::Subject::DetachObserver (Observer::NotifyType *notify_type*, Observer ∗ *observer*) const [inline]

Detach the specified observer (i.e., no longer recieve notifications).

Definition at line 329 of file IpObserver.hpp.

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

- IpObserver.hpp

3.161 Ipopt::SumMatrix Class Reference

Class for Matrices which are sum of matrices.

`#include <IpSumMatrix.hpp>`

Inheritance diagram for Ipopt::SumMatrix:

Public Member Functions

- void SetTerm (Index iterm, Number factor, const Matrix &matrix)

  Method for setting term iterm for the sum.
- void GetTerm (Index iterm, Number &factor, SmartPtr< const Matrix > &matrix) const

  Method for getting term iterm for the sum.
- Index NTerms () const

  Return the number of terms.

Constructors / Destructors

- SumMatrix (const SumMatrixSpace ∗owner_space)

  Constructor, taking the owner_space.
- virtual ∼SumMatrix ()

  Destructor.

Protected Member Functions

Methods overloaded from matrix

- virtual void MultVectorImpl (Number alpha, const Vector &x, Number beta, Vector &y) const

  Matrix-vector multiply.
- virtual void TransMultVectorImpl (Number alpha, const Vector &x, Number beta, Vector &y) const

  Matrix(transpose) vector multiply.
- virtual bool HasValidNumbersImpl () const

  Method for determining if all stored numbers are valid (i.e., no Inf or Nan).

- virtual void ComputeRowAMaxImpl (Vector &rows_norms, bool init) const
  Compute the max-norm of the rows in the matrix.
- virtual void ComputeColAMaxImpl (Vector &cols_norms, bool init) const
  Compute the max-norm of the columns in the matrix.
- virtual void PrintImpl (const Journalist &jnlst, EJournalLevel level, EJournalCategory category, const std::string &name, Index indent, const std::string &prefix) const
  Print detailed information about the matrix.

Additional Inherited Members

3.161.1 Detailed Description

Class for Matrices which are sum of matrices.

For each term in the we store the matrix and a factor.

Definition at line 24 of file IpSumMatrix.hpp.

3.161.2 Member Function Documentation

3.161.2.1 void Ipopt::SumMatrix::SetTerm (Index *iterm,* Number *factor,* const Matrix & *matrix*)

Method for setting term iterm for the sum.

3.161.2.2 void Ipopt::SumMatrix::GetTerm (Index *iterm,* Number & *factor,* SmartPtr< const Matrix > & *matrix*) const

Method for getting term iterm for the sum.

Note that counting of terms starts at 0.

3.161.2.3 virtual void Ipopt::SumMatrix::MultVectorImpl (Number *alpha,* const Vector & *x,* Number *beta,* Vector & *y*) const [protected], [virtual]

Matrix-vector multiply.

Computes y = alpha $*$ Matrix $*$ x + beta $*$ y

Implements Ipopt::Matrix.

3.161.2.4 virtual void Ipopt::SumMatrix::TransMultVectorImpl (Number *alpha,* const Vector & *x,* Number *beta,* Vector & *y*) const [protected], [virtual]

Matrix(transpose) vector multiply.

Computes y = alpha $*$ Matrix^T $*$ x + beta $*$ y

Implements Ipopt::Matrix.

3.161.2.5 virtual bool Ipopt::SumMatrix::HasValidNumbersImpl ( ) const [protected], [virtual]

Method for determining if all stored numbers are valid (i.e., no Inf or Nan).

Reimplemented from Ipopt::Matrix.

3.161.2.6 virtual void Ipopt::SumMatrix::ComputeRowAMaxImpl (Vector & *rows_norms,* bool *init*) const [protected], [virtual]

Compute the max-norm of the rows in the matrix.

The result is stored in rows_norms. The vector is assumed to be initialized.

Implements Ipopt::Matrix.

3.161.2.7 virtual void Ipopt::SumMatrix::ComputeColAMaxImpl (Vector & *cols_norms,* bool *init*) const [protected], [virtual]

Compute the max-norm of the columns in the matrix.

The result is stored in cols_norms. The vector is assumed to be initialized.

Implements Ipopt::Matrix.

3.161.2.8 virtual void Ipopt::SumMatrix::PrintImpl (const Journalist & *jnlst,* EJournalLevel *level,* EJournalCategory *category,* const std::string & *name,* Index *indent,* const std::string & *prefix*) const [protected], [virtual]

Print detailed information about the matrix.

Implements Ipopt::Matrix.

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

- IpSumMatrix.hpp

3.162 Ipopt::SumMatrixSpace Class Reference

Class for matrix space for SumMatrix.

`#include <IpSumMatrix.hpp>`

Inheritance diagram for Ipopt::SumMatrixSpace:

Public Member Functions

- Index NTerms () const

  Accessor functions to get the number of terms in the sum.
- void SetTermSpace (Index term_idx, const MatrixSpace &mat_space)

  Set the appropriate matrix space for each term.
- SmartPtr< const MatrixSpace > GetTermSpace (Index term_idx) const

  Get the matrix space for a particular term.
- SumMatrix * MakeNewSumMatrix () const

  Method for creating a new matrix of this specific type.
- virtual Matrix * MakeNew () const

  Overloaded MakeNew method for the MatrixSpace base class.

Constructors / Destructors

- SumMatrixSpace (Index nrows, Index ncols, Index nterms)

Constructor, given the number of row and columns, as well as the number of terms in the sum.

- virtual ~SumMatrixSpace ()

  Destructor.

3.162.1 Detailed Description

Class for matrix space for SumMatrix.

Definition at line 103 of file IpSumMatrix.hpp.

3.162.2 Member Function Documentation

3.162.2.1 Index Ipopt::SumMatrixSpace::NTerms ( ) const [inline]

Accessor functions to get the number of terms in the sum.

Definition at line 123 of file IpSumMatrix.hpp.

3.162.2.2 void Ipopt::SumMatrixSpace::SetTermSpace (Index *term_idx*, const MatrixSpace & *mat_space*)

Set the appropriate matrix space for each term.

This must be called for each term or a runtime error will occur

3.162.2.3 SumMatrix* Ipopt::SumMatrixSpace::MakeNewSumMatrix ( ) const

Method for creating a new matrix of this specific type.

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

- IpSumMatrix.hpp

3.163 Ipopt::SumSymMatrix Class Reference

Class for Matrices which are sum of symmetric matrices.

`#include <IpSumSymMatrix.hpp>`

Inheritance diagram for Ipopt::SumSymMatrix:

Public Member Functions

- void SetTerm (Index iterm, Number factor, const SymMatrix &matrix)

  Method for setting term iterm for the sum.
- void GetTerm (Index iterm, Number &factor, SmartPtr< const SymMatrix > &matrix) const

  Method for getting term iterm for the sum.
- Index NTerms () const

  Return the number of terms.

Constructors / Destructors

- SumSymMatrix (const SumSymMatrixSpace ∗owner_space)

  Constructor, initializing with dimensions of the matrix and the number of terms in the sum.
- ∼SumSymMatrix ()

  Destructor.

Protected Member Functions

Methods overloaded from matrix

- virtual void MultVectorImpl (Number alpha, const Vector &x, Number beta, Vector &y) const

  Matrix-vector multiply.
- virtual bool HasValidNumbersImpl () const

  Method for determining if all stored numbers are valid (i.e., no Inf or Nan).
- virtual void ComputeRowAMaxImpl (Vector &rows_norms, bool init) const

  Compute the max-norm of the rows in the matrix.
- virtual void ComputeColAMaxImpl (Vector &cols_norms, bool init) const

  Since the matrix is symmetric, the row and column max norms are identical.
- virtual void PrintImpl (const Journalist &jnlst, EJournalLevel level, EJournalCategory category, const std::string &name, Index indent, const std::string &prefix) const

  Print detailed information about the matrix.

Additional Inherited Members

3.163.1 Detailed Description

Class for Matrices which are sum of symmetric matrices.

For each term in the we store the matrix and a factor.

Definition at line 24 of file IpSumSymMatrix.hpp.

3.163.2 Member Function Documentation

3.163.2.1 void Ipopt::SumSymMatrix::SetTerm (Index *iterm*, Number *factor*, const SymMatrix & *matrix*)

Method for setting term iterm for the sum.

Note that counting of terms starts at 0.

3.163.2.2 void Ipopt::SumSymMatrix::GetTerm (Index *iterm*, Number & *factor*, SmartPtr< const SymMatrix > & *matrix*) const

Method for getting term iterm for the sum.

Note that counting of terms starts at 0.

3.163.2.3 virtual void Ipopt::SumSymMatrix::MultVectorImpl (Number *alpha*, const Vector & *x*, Number *beta*, Vector & *y*) const [protected], [virtual]

Matrix-vector multiply.

Computes y = alpha ∗ Matrix ∗ x + beta ∗ y

Implements Ipopt::Matrix.

3.163.2.4 virtual bool Ipopt::SumSymMatrix::HasValidNumbersImpl ( ) const [protected], [virtual]

Method for determining if all stored numbers are valid (i.e., no Inf or Nan).

Reimplemented from Ipopt::Matrix.

3.163.2.5 virtual void Ipopt::SumSymMatrix::ComputeRowAMaxImpl (Vector & *rows_norms*, bool *init*) const [protected], [virtual]

Compute the max-norm of the rows in the matrix.

The result is stored in rows_norms. The vector is assumed to be initialized.

Implements Ipopt::Matrix.

3.163.2.6 virtual void Ipopt::SumSymMatrix::PrintImpl (const Journalist & *jnlst*, EJournalLevel *level*, EJournalCategory *category*, const std::string & *name*, Index *indent*, const std::string & *prefix*) const [protected], [virtual]

Print detailed information about the matrix.

Implements Ipopt::Matrix.

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

- IpSumSymMatrix.hpp

3.164 Ipopt::SumSymMatrixSpace Class Reference

Class for matrix space for SumSymMatrix.

`#include <IpSumSymMatrix.hpp>`

Inheritance diagram for Ipopt::SumSymMatrixSpace:

Public Member Functions

- void SetTermSpace (Index term_idx, const SymMatrixSpace &space)
  Use this method to set the matrix spaces for the various terms.
- SmartPtr< const SymMatrixSpace > GetTermSpace (Index term_idx) const
  Get the matix space for a particular term.
- SumSymMatrix * MakeNewSumSymMatrix () const
  Method for creating a new matrix of this specific type.
- virtual SymMatrix * MakeNewSymMatrix () const
  Overloaded MakeNew method for the SymMatrixSpace base class.

Constructors / Destructors

- SumSymMatrixSpace (Index ndim, Index nterms)
  Constructor, given the dimension of the matrix and the number of terms in the sum.
- ~SumSymMatrixSpace ()
  Destructor.

Accessor functions

- Index NTerms () const
  Number of terms in the sum.

3.164.1 Detailed Description

Class for matrix space for SumSymMatrix.

Definition at line 103 of file IpSumSymMatrix.hpp.

3.164.2 Constructor & Destructor Documentation

3.164.2.1 Ipopt::SumSymMatrixSpace::SumSymMatrixSpace (Index *ndim*, Index *nterms*) [inline]

Constructor, given the dimension of the matrix and the number of terms in the sum.

Definition at line 110 of file IpSumSymMatrix.hpp.

3.164.3 Member Function Documentation

3.164.3.1 Index Ipopt::SumSymMatrixSpace::NTerms ( ) const [inline]

Number of terms in the sum.

Definition at line 124 of file IpSumSymMatrix.hpp.

3.164.3.2 void Ipopt::SumSymMatrixSpace::SetTermSpace (Index *term_idx*, const SymMatrixSpace & *space*)

Use this method to set the matrix spaces for the various terms.

You will not be able to create a matrix until all these spaces are set.

3.164.3.3 SumSymMatrix* Ipopt::SumSymMatrixSpace::MakeNewSumSymMatrix ( ) const

Method for creating a new matrix of this specific type.

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

- IpSumSymMatrix.hpp

3.165 Ipopt::SymLinearSolver Class Reference

Base class for all derived symmetric linear solvers.

`#include <IpSymLinearSolver.hpp>`

Inheritance diagram for Ipopt::SymLinearSolver:

Public Member Functions

- virtual bool InitializeImpl (const OptionsList &options, const std::string &prefix)=0

  overloaded from AlgorithmStrategyObject

Constructor/Destructor

- **SymLinearSolver** ()
- virtual ∼**SymLinearSolver** ()

Methods for requesting solution of the linear system.

- virtual ESymSolverStatus MultiSolve (const SymMatrix &A, std::vector< SmartPtr< const Vector > > &rhsV, std::vector< SmartPtr< Vector > > &solV, bool check_NegEVals, Index numberOfNegEVals)=0

  Solve operation for multiple right hand sides.
- ESymSolverStatus Solve (const SymMatrix &A, const Vector &rhs, Vector &sol, bool check_NegEVals, Index numberOfNegEVals)

  Solve operation for a single right hand side.
- virtual Index NumberOfNegEVals () const =0

  Number of negative eigenvalues detected during last factorization.

- virtual bool IncreaseQuality ()=0

  Request to increase quality of solution for next solve.
- virtual bool ProvidesInertia () const =0

  Query whether inertia is computed by linear solver.

Additional Inherited Members

3.165.1 Detailed Description

Base class for all derived symmetric linear solvers.

In the full space version of Ipopt a large linear system has to be solved for the augmented system. This case is meant to be the base class for all derived linear solvers for symmetric matrices (of type SymMatrix).

A linear solver can be used repeatedly for matrices with identical structure of nonzero elements. The nonzero structure of those matrices must not be changed between calls.

The called might ask the solver to only solve the linear system if the system is nonsingular, and if the number of negative eigenvalues matches a given number.

Definition at line 50 of file IpSymLinearSolver.hpp.

3.165.2 Member Function Documentation

3.165.2.1 virtual ESymSolverStatus Ipopt::SymLinearSolver::MultiSolve (const SymMatrix & *A*, std::vector< SmartPtr< const Vector > > & *rhsV*, std::vector< SmartPtr< Vector > > & *solV*, bool *check_NegEVals*, Index *numberOfNegEVals*) [pure virtual]

Solve operation for multiple right hand sides.

Solves the linear system $A * Sol = Rhs$ with multiple right hand sides. If necessary, A is factorized. Correct solutions are only guaranteed if the return values is SYMSOLVER_SUCCESS. The solver will return SYMSOLVER_SINGULAR if the linear system is singular, and it will return SYMSOLVER_WRONG_INERTIA if check_NegEVals is true and the number of negative eigenvalues in the matrix does not match numberOfNegEVals.

check_NegEVals cannot be chosen true, if ProvidesInertia() returns false.

Implemented in Ipopt::TSymLinearSolver.

3.165.2.2 ESymSolverStatus Ipopt::SymLinearSolver::Solve (const SymMatrix & *A*, const Vector & *rhs*, Vector & *sol*, bool *check_NegEVals*, Index *numberOfNegEVals*) [inline]

Solve operation for a single right hand side.

Solves the linear system A ∗ Sol = Rhs. See MultiSolve for more details.

Definition at line 89 of file IpSymLinearSolver.hpp.

3.165.2.3 virtual Index Ipopt::SymLinearSolver::NumberOfNegEVals ( ) const [pure virtual]

Number of negative eigenvalues detected during last factorization.

Returns the number of negative eigenvalues of the most recent factorized matrix. This must not be called if the linear solver does not compute this quantities (see ProvidesInertia).

Implemented in Ipopt::TSymLinearSolver.

3.165.2.4 virtual bool Ipopt::SymLinearSolver::IncreaseQuality ( ) [pure virtual]

Request to increase quality of solution for next solve.

Ask linear solver to increase quality of solution for the next solve (e.g. increase pivot tolerance). Returns false, if this is not possible (e.g. maximal pivot tolerance already used.)

Implemented in Ipopt::TSymLinearSolver.

3.165.2.5 virtual bool Ipopt::SymLinearSolver::ProvidesInertia ( ) const [pure virtual]

Query whether inertia is computed by linear solver.

Returns true, if linear solver provides inertia.

Implemented in Ipopt::TSymLinearSolver.

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

- IpSymLinearSolver.hpp

3.166 Ipopt::SymMatrix Class Reference

This is the base class for all derived symmetric matrix types.

`#include <IpSymMatrix.hpp>`

Inheritance diagram for Ipopt::SymMatrix:

Public Member Functions

Constructor/Destructor

- SymMatrix (const SymMatrixSpace ∗owner_space)
  Constructor, taking the owner_space.
- virtual ∼SymMatrix ()
  Destructor.

Information about the size of the matrix

- Index Dim () const
  Dimension of the matrix (number of rows and columns)

Protected Member Functions

Overloaded methods from Matrix.

- virtual void TransMultVectorImpl (Number alpha, const Vector &x, Number beta, Vector &y) const
  Since the matrix is symmetric, it is only necessary to implement the MultVectorImpl method in a class that inherits from this base class.
- virtual void ComputeColAMaxImpl (Vector &cols_norms, bool init) const
  Since the matrix is symmetric, the row and column max norms are identical.

Additional Inherited Members

3.166.1 Detailed Description

This is the base class for all derived symmetric matrix types.

Definition at line 23 of file IpSymMatrix.hpp.

3.166.2 Member Function Documentation

3.166.2.1 virtual void Ipopt::SymMatrix::TransMultVectorImpl (Number *alpha,* const Vector & *x,* Number *beta,* Vector & *y*) const [inline], [protected], [virtual]

Since the matrix is symmetric, it is only necessary to implement the MultVectorImpl method in a class that inherits from this base class.

If the TransMultVectorImpl is called, this base class automatically calls MultVectorImpl instead.

Implements Ipopt::Matrix.

Reimplemented in Ipopt::ZeroSymMatrix.

Definition at line 56 of file IpSymMatrix.hpp.

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

- IpSymMatrix.hpp

3.167 Ipopt::SymMatrixSpace Class Reference

SymMatrixSpace base class, corresponding to the SymMatrix base class.

`#include <IpSymMatrix.hpp>`

Inheritance diagram for Ipopt::SymMatrixSpace:

Public Member Functions

- virtual SymMatrix ∗ MakeNewSymMatrix () const =0

  Pure virtual method for creating a new matrix of this specific type.
- virtual Matrix ∗ MakeNew () const

  Overloaded MakeNew method for the MatrixSpace base class.
- Index Dim () const

  Accessor method for the dimension of the matrices in this matrix space.

Constructors/Destructors

- SymMatrixSpace (Index dim)

  Constructor, given the dimension (identical to the number of rows and columns).
- virtual ∼SymMatrixSpace ()

  Destructor.

3.167.1 Detailed Description

SymMatrixSpace base class, corresponding to the SymMatrix base class.

Definition at line 81 of file IpSymMatrix.hpp.

3.167.2 Member Function Documentation

3.167.2.1 virtual SymMatrix∗ Ipopt::SymMatrixSpace::MakeNewSymMatrix ( ) const [pure virtual]

Pure virtual method for creating a new matrix of this specific type.

Implemented in Ipopt::CompoundSymMatrixSpace, Ipopt::SymTMatrixSpace, Ipopt::LowRankUpdateSymMatrixSpace, Ipopt::DenseSymMatrixSpace, Ipopt::SymScaledMatrixSpace, Ipopt::SumSymMatrixSpace, Ipopt::IdentityMatrixSpace, Ipopt::DiagMatrixSpace, and Ipopt::ZeroSymMatrixSpace.

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

- IpSymMatrix.hpp

3.168 Ipopt::SymScaledMatrix Class Reference

Class for a Matrix in conjunction with its scaling factors for row and column scaling.

`#include <IpSymScaledMatrix.hpp>`

Inheritance diagram for Ipopt::SymScaledMatrix:

Public Member Functions

- void SetUnscaledMatrix (const SmartPtr< const SymMatrix > unscaled_matrix)

  Set the unscaled matrix.
- void SetUnscaledMatrixNonConst (const SmartPtr< SymMatrix > &unscaled_matrix)

  Set the unscaled matrix in a non-const version.
- SmartPtr< const SymMatrix > GetUnscaledMatrix () const

  Return the unscaled matrix in const form.
- SmartPtr< SymMatrix > GetUnscaledMatrixNonConst ()

  Return the unscaled matrix in non-const form.
- SmartPtr< const Vector > RowColScaling () const

  return the vector for the row and column scaling

Constructors / Destructors

- SymScaledMatrix (const SymScaledMatrixSpace *owner_space)

  Constructor, taking the owner_space.
- ~SymScaledMatrix ()

  Destructor.

Protected Member Functions

Methods overloaded from Matrix

- virtual void MultVectorImpl (Number alpha, const Vector &x, Number beta, Vector &y) const

  Matrix-vector multiply.
- virtual bool HasValidNumbersImpl () const

  Method for determining if all stored numbers are valid (i.e., no Inf or Nan).
- virtual void ComputeRowAMaxImpl (Vector &rows_norms, bool init) const

  Compute the max-norm of the rows in the matrix.
- virtual void PrintImpl (const Journalist &jnlst, EJournalLevel level, EJournalCategory category, const std::string &name, Index indent, const std::string &prefix) const

  Print detailed information about the matrix.

Additional Inherited Members

3.168.1 Detailed Description

Class for a Matrix in conjunction with its scaling factors for row and column scaling.

Operations on the matrix are performed using the scaled matrix. You can pull out the pointer to the unscaled matrix for unscaled calculations.

Definition at line 26 of file IpSymScaledMatrix.hpp.

3.168.2 Member Function Documentation

3.168.2.1 virtual void Ipopt::SymScaledMatrix::MultVectorImpl (Number *alpha*, const Vector & *x*, Number *beta*, Vector & *y*) const [protected], [virtual]

Matrix-vector multiply.

Computes y = alpha ∗ Matrix ∗ x + beta ∗ y

Implements Ipopt::Matrix.

3.168.2.2 virtual bool Ipopt::SymScaledMatrix::HasValidNumbersImpl ( ) const [protected], [virtual]

Method for determining if all stored numbers are valid (i.e., no Inf or Nan).

It is assumed here that the scaling factors are always valid numbers.

Reimplemented from Ipopt::Matrix.

3.168.2.3 virtual void Ipopt::SymScaledMatrix::ComputeRowAMaxImpl (Vector & *rows_norms*, bool *init*) const [protected], [virtual]

Compute the max-norm of the rows in the matrix.

The result is stored in rows_norms. The vector is assumed to be initialized.

Implements Ipopt::Matrix.

3.168.2.4 virtual void Ipopt::SymScaledMatrix::PrintImpl (const Journalist & *jnlst*, EJournalLevel *level*, EJournalCategory *category*, const std::string & *name*, Index *indent*, const std::string & *prefix*) const [protected], [virtual]

Print detailed information about the matrix.

Implements Ipopt::Matrix.

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

- IpSymScaledMatrix.hpp

3.169 Ipopt::SymScaledMatrixSpace Class Reference

This is the matrix space for SymScaledMatrix.

`#include <IpSymScaledMatrix.hpp>`

Inheritance diagram for Ipopt::SymScaledMatrixSpace:

Public Member Functions

- SymScaledMatrix ∗ MakeNewSymScaledMatrix (bool allocate_unscaled_matrix=false) const

  Method for creating a new matrix of this specific type.
- virtual SymMatrix ∗ MakeNewSymMatrix () const

  Overloaded method from SymMatrixSpace.
- virtual Matrix ∗ MakeNew () const

  Overloaded MakeNew method for the MatrixSpace base class.
- SmartPtr< const Vector > RowColScaling () const

  return the vector for the row and column scaling
- SmartPtr< const SymMatrixSpace > UnscaledMatrixSpace () const

  return the matrix space for the unscaled matrix

Constructors / Destructors

- SymScaledMatrixSpace (const SmartPtr< const Vector > &row_col_scaling, bool row_col_scaling_reciprocal, const SmartPtr< const SymMatrixSpace > &unscaled_matrix_space)

  Constructor, given the number of row and columns blocks, as well as the totel number of rows and columns.
- ∼SymScaledMatrixSpace ()

  Destructor.

3.169.1 Detailed Description

This is the matrix space for SymScaledMatrix.

Definition at line 107 of file IpSymScaledMatrix.hpp.

3.169.2 Member Function Documentation

3.169.2.1 SymScaledMatrix∗ Ipopt::SymScaledMatrixSpace::MakeNewSymScaledMatrix (bool *allocate_unscaled_matrix* = false) const [inline]

Method for creating a new matrix of this specific type.

Definition at line 134 of file IpSymScaledMatrix.hpp.

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

- IpSymScaledMatrix.hpp

3.170 Ipopt::SymTMatrix Class Reference

Class for symmetric matrices stored in triplet format.

#include <IpSymTMatrix.hpp>

Inheritance diagram for Ipopt::SymTMatrix:

Public Member Functions

Constructors / Destructors

- SymTMatrix (const SymTMatrixSpace ∗owner_space)

  Constructor, taking the corresponding matrix space.
- ∼SymTMatrix ()

  Destructor.

Changing the Values.

- void SetValues (const Number ∗Values)

  Set values of nonzero elements.

Accessor Methods

- Index Nonzeros () const

Number of nonzero entries.
- const Index ∗ Irows () const

  Obtain pointer to the internal Index array irn_ without the intention to change the matrix data (USE WITH CARE!).
- const Index ∗ Jcols () const

  Obtain pointer to the internal Index array jcn_ without the intention to change the matrix data (USE WITH CARE!).
- Number ∗ Values ()

  Obtain pointer to the internal Number array values_ with the intention to change the matrix data (USE WITH CARE!).
- const Number ∗ Values () const

  Obtain pointer to the internal Number array values_ without the intention to change the matrix data (USE WITH CARE!).

Methods for providing copy of the matrix data

- void FillStruct (ipfint ∗Irn, ipfint ∗Jcn) const

  Copy the nonzero structure into provided space.
- void FillValues (Number ∗Values) const

  Copy the value data into provided space.

Protected Member Functions

Methods overloaded from matrix

- virtual void MultVectorImpl (Number alpha, const Vector &x, Number beta, Vector &y) const

  Matrix-vector multiply.
- virtual bool HasValidNumbersImpl () const

  Method for determining if all stored numbers are valid (i.e., no Inf or Nan).
- virtual void ComputeRowAMaxImpl (Vector &rows_norms, bool init) const

  Compute the max-norm of the rows in the matrix.
- virtual void PrintImpl (const Journalist &jnlst, EJournalLevel level, EJournalCategory category, const std::string &name, Index indent, const std::string &prefix) const

  Print detailed information about the matrix.

Additional Inherited Members

3.170.1 Detailed Description

Class for symmetric matrices stored in triplet format.

In the triplet format, the nonzeros elements of a symmetric matrix is stored in three arrays, Irn, Jcn, and Values, all of length Nonzeros. The first two arrays indicate the location of a non-zero element (as the row and column indices), and the last array stores the value at that location. Off-diagonal elements need to be stored only once since the matrix is symmetric. For example, the element $a_{1,2} = a_{2,1}$ would be stored only once, either with Irn[i]=1 and Jcn[i]=2, or with Irn[i]=2 and Jcn[i]=1. Both representations are identical. If nonzero elements (or their symmetric counter part) are listed more than once, their values are added.

The structure of the nonzeros (i.e. the arrays Irn and Jcn) cannot be changed after the matrix can been initialized. Only the values of the nonzero elements can be modified.

Note that the first row and column of a matrix has index 1, not 0.

Definition at line 42 of file IpSymTMatrix.hpp.

3.170.2 Member Function Documentation

3.170.2.1 void Ipopt::SymTMatrix::SetValues (const Number * *Values*)

Set values of nonzero elements.

The values of the nonzero elements is copied from the incoming Number array. Important: It is assume that the order of the values in Values corresponds to the one of Irn and Jcn given to the matrix space.

3.170.2.2 const Index * Ipopt::SymTMatrix::Irows ( ) const [inline]

Obtain pointer to the internal Index array irn_ without the intention to change the matrix data (USE WITH CARE!).

This does not produce a copy, and lifetime is not guaranteed!

Definition at line 240 of file IpSymTMatrix.hpp.

3.170.2.3 const Index * Ipopt::SymTMatrix::Jcols ( ) const [inline]

Obtain pointer to the internal Index array jcn_ without the intention to change the matrix data (USE WITH CARE!).

This does not produce a copy, and lifetime is not guaranteed!

Definition at line 246 of file IpSymTMatrix.hpp.

3.170.2.4 Number* Ipopt::SymTMatrix::Values ( )

Obtain pointer to the internal Number array values_ with the intention to change the matrix data (USE WITH CARE!).

This does not produce a copy, and lifetime is not guaranteed!

3.170.2.5 const Number* Ipopt::SymTMatrix::Values ( ) const

Obtain pointer to the internal Number array values_ without the intention to change the matrix data (USE WITH CARE!).

This does not produce a copy, and lifetime is not guaranteed!

3.170.2.6 virtual void Ipopt::SymTMatrix::MultVectorImpl (Number *alpha*, const Vector & *x*, Number *beta*, Vector & *y*) const [protected], [virtual]

Matrix-vector multiply.

Computes y = alpha * Matrix * x + beta * y

Implements Ipopt::Matrix.

3.170.2.7 virtual bool Ipopt::SymTMatrix::HasValidNumbersImpl ( ) const [protected], [virtual]

Method for determining if all stored numbers are valid (i.e., no Inf or Nan).

Reimplemented from Ipopt::Matrix.

3.170.2.8 virtual void Ipopt::SymTMatrix::ComputeRowAMaxImpl (Vector & *rows_norms,* bool *init*) const [protected], [virtual]

Compute the max-norm of the rows in the matrix.

The result is stored in rows_norms. The vector is assumed to be initialized.

Implements Ipopt::Matrix.

3.170.2.9 virtual void Ipopt::SymTMatrix::PrintImpl (const Journalist & *jnlst,* EJournalLevel *level,* EJournalCategory *category,* const std::string & *name,* Index *indent,* const std::string & *prefix*) const [protected], [virtual]

Print detailed information about the matrix.

Implements Ipopt::Matrix.

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

- IpSymTMatrix.hpp

3.171 Ipopt::SymTMatrixSpace Class Reference

This is the matrix space for a SymTMatrix with fixed sparsity structure.

`#include <IpSymTMatrix.hpp>`

Inheritance diagram for Ipopt::SymTMatrixSpace:

Public Member Functions

- virtual SymMatrix ∗ MakeNewSymMatrix () const

    Overloaded MakeNew method for the sYMMatrixSpace base class.
- SymTMatrix ∗ MakeNewSymTMatrix () const

    Method for creating a new matrix of this specific type.

Constructors / Destructors

- SymTMatrixSpace (Index dim, Index nonZeros, const Index ∗iRows, const Index ∗jCols)

    Constructor, given the number of rows and columns (both as dim), as well as the number of nonzeros and the position of the nonzero elements.
- ∼SymTMatrixSpace ()

    Destructor.

Methods describing Matrix structure

- Index Nonzeros () const

    Number of non-zeros in the sparse matrix.
- const Index ∗ Irows () const

    Row index of each non-zero element.
- const Index ∗ Jcols () const

    Column index of each non-zero element.

3.171.1 Detailed Description

This is the matrix space for a SymTMatrix with fixed sparsity structure.

The sparsity structure is stored here in the matrix space.

Definition at line 161 of file IpSymTMatrix.hpp.

3.171.2 Constructor & Destructor Documentation

3.171.2.1 Ipopt::SymTMatrixSpace::SymTMatrixSpace (Index *dim*, Index *nonZeros*, const Index * *iRows*, const Index * *jCols*)

Constructor, given the number of rows and columns (both as dim), as well as the number of nonzeros and the position of the nonzero elements.

Note that the counting of the nonzeros starts a 1, i.e., iRows[i]==1 and jCols[i]==1 refers to the first element in the first row. This is in accordance with the HSL data structure. Off-diagonal elements are stored only once.

3.171.3 Member Function Documentation

3.171.3.1 SymTMatrix* Ipopt::SymTMatrixSpace::MakeNewSymTMatrix ( ) const [inline]

Method for creating a new matrix of this specific type.

Definition at line 189 of file IpSymTMatrix.hpp.

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

- IpSymTMatrix.hpp

3.172 Ipopt::TaggedObject Class Reference

TaggedObject class.

`#include <IpTaggedObject.hpp>`

Inheritance diagram for Ipopt::TaggedObject:

Public Types

- typedef unsigned int Tag

    Type for the Tag values.

Public Member Functions

- TaggedObject ()

    Constructor.

- virtual ~TaggedObject ()

    Destructor.

- Tag GetTag () const

    Users of TaggedObjects call this to update their own internal tags every time they perform the expensive operation.

- bool HasChanged (const Tag comparison_tag) const

  Users of TaggedObjects call this to check if the object HasChanged since they last updated their own internal tag.

Protected Member Functions

- void ObjectChanged ()

  Objects derived from TaggedObject MUST call this method every time their internal state changes to update the internal tag for comparison.

3.172.1 Detailed Description

TaggedObject class.

Often, certain calculations or operations are expensive, and it can be very inefficient to perform these calculations again if the input to the calculation has not changed since the result was last stored. This base class provides an efficient mechanism to update a tag, indicating that the object has changed. Users of a TaggedObject class, need their own Tag data member to keep track of the state of the TaggedObject, the last time they performed a calculation. A basic use case for users of a class inheriting from TaggedObject follows like this:

1. Initialize your own Tag to zero in constructor.
2. Before an expensive calculation, check if the TaggedObject has changed, passing in your own Tag, indicating the last time you used the object for the calculation. If it has changed, perform the calculation again, and store the result. If it has not changed, simply return the stored result.
   Here is a simple example:

```
if (vector.HasChanged(my_vector_tag_)) {
  my_vector_tag_ = vector.GetTag();
  result = PerformExpensiveCalculation(vector);
  return result;
}
else {
  return result;
}
```

Objects derived from TaggedObject must indicate that they have changed to the base class using the protected member function ObjectChanged(). For example, a Vector class, inside its own set method, MUST call ObjectChanged() to update the internally stored tag for comparison.

Definition at line 75 of file IpTaggedObject.hpp.

3.172.2 Constructor & Destructor Documentation

3.172.2.1 Ipopt::TaggedObject::TaggedObject ( ) [inline]

Constructor.

Definition at line 82 of file IpTaggedObject.hpp.

3.172.2.2 virtual Ipopt::TaggedObject::∼TaggedObject ( ) [inline], [virtual]

Destructor.

Definition at line 90 of file IpTaggedObject.hpp.

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

- IpTaggedObject.hpp

3.173 Ipopt::TDependencyDetector Class Reference

Base class for all derived algorithms for detecting linearly dependent rows in the constraint Jacobian.

`#include <IpTDependencyDetector.hpp>`

Inheritance diagram for Ipopt::TDependencyDetector:

Public Member Functions

- virtual bool InitializeImpl (const OptionsList &options, const std::string &prefix)=0
  Has to be called to initialize and reset these objects.
- virtual bool DetermineDependentRows (Index n_rows, Index n_cols, Index n_jac_nz, Number ∗jac_c_vals, Index ∗jac_c_iRow, Index ∗jac_c_jCol, std::list< Index > &c_deps)=0
  Method determining the number of linearly dependent rows in the matrix and the indices of those rows.

Constructor/Destructor

- **TDependencyDetector** ()
- virtual ∼**TDependencyDetector** ()

Additional Inherited Members

3.173.1 Detailed Description

Base class for all derived algorithms for detecting linearly dependent rows in the constraint Jacobian.

Definition at line 20 of file IpTDependencyDetector.hpp.

3.173.2 Member Function Documentation

3.173.2.1 virtual bool Ipopt::TDependencyDetector::InitializeImpl (const OptionsList & *options,* const std::string & *prefix*) `[pure virtual]`

Has to be called to initialize and reset these objects.

Implements Ipopt::AlgorithmStrategyObject.

Implemented in Ipopt::TSymDependencyDetector, and Ipopt::Ma28TDependencyDetector.

3.173.2.2 virtual bool Ipopt::TDependencyDetector::DetermineDependentRows (Index *n_rows,* Index *n_cols,* Index *n_jac_nz,* Number ∗ *jac_c_vals,* Index ∗ *jac_c_iRow,* Index ∗ *jac_c_jCol,* std::list< Index > & *c_deps*) `[pure virtual]`

Method determining the number of linearly dependent rows in the matrix and the indices of those rows.

We assume that the matrix is available in "Triplet" format (MA28 format), and that the arrays given to this method can be modified internally, i.e., they are not used by the calling program anymore after this call. This method returns false if there was a problem with the underlying linear solver.

Implemented in Ipopt::TSymDependencyDetector, and Ipopt::Ma28TDependencyDetector.

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

- IpTDependencyDetector.hpp

3.174 Ipopt::TimedTask Class Reference

This class is used to collect timing information for a particular task.

`#include <IpTimedTask.hpp>`

Public Member Functions

- void Reset ()
  Method for resetting time to zero.
- void Start ()
  Method that is called before execution of the task.
- void End ()
  Method that is called after execution of the task.
- void EndIfStarted ()
  Method that is called after execution of the task for which timing might have been started.
- Number TotalCpuTime () const
  Method returning total CPU time spend for task so far.
- Number TotalSysTime () const
  Method returning total system time spend for task so far.
- Number TotalWallclockTime () const
  Method returning total wall clock time spend for task so far.

Constructors/Destructors

- TimedTask ()
  Default constructor.
- ∼TimedTask ()
  Default destructor.

3.174.1 Detailed Description

This class is used to collect timing information for a particular task.

Definition at line 18 of file IpTimedTask.hpp.

3.174.2 Constructor & Destructor Documentation

3.174.2.1 Ipopt::TimedTask::TimedTask ( ) [inline]

Default constructor.

Definition at line 24 of file IpTimedTask.hpp.

3.174.3 Member Function Documentation

3.174.3.1 void Ipopt::TimedTask::Reset ( ) [inline]

Method for resetting time to zero.

Definition at line 39 of file IpTimedTask.hpp.

3.174.3.2 void Ipopt::TimedTask::Start ( ) [inline]

Method that is called before execution of the task.

Definition at line 49 of file IpTimedTask.hpp.

3.174.3.3 void Ipopt::TimedTask::End ( ) [inline]

Method that is called after execution of the task.

Definition at line 61 of file IpTimedTask.hpp.

3.174.3.4 void Ipopt::TimedTask::EndIfStarted ( ) [inline]

Method that is called after execution of the task for which timing might have been started.

This only updates the timing if the timing has indeed been conducted. This is useful to stop timing after catching exceptions.

Definition at line 76 of file IpTimedTask.hpp.

3.174.3.5 Number Ipopt::TimedTask::TotalCpuTime ( ) const [inline]

Method returning total CPU time spend for task so far.

Definition at line 89 of file IpTimedTask.hpp.

3.174.3.6 Number Ipopt::TimedTask::TotalSysTime ( ) const [inline]

Method returning total system time spend for task so far.

Definition at line 96 of file IpTimedTask.hpp.

3.174.3.7 Number Ipopt::TimedTask::TotalWallclockTime ( ) const [inline]

Method returning total wall clock time spend for task so far.

Definition at line 103 of file IpTimedTask.hpp.

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

- IpTimedTask.hpp

3.175 Ipopt::TimingStatistics Class Reference

This class collects all timing statistics for Ipopt.

```
#include <IpTimingStatistics.hpp>
```

Inheritance diagram for Ipopt::TimingStatistics:

Public Member Functions

- void ResetTimes ()
  Method for resetting all times.
- void PrintAllTimingStatistics (Journalist &jnlst, EJournalLevel level, EJournalCategory category) const
  Method for printing all timing information.

Constructors/Destructors

- TimingStatistics ()
  Default constructor.
- virtual ∼TimingStatistics ()
  Default destructor.

Accessor methods to all timed tasks.

- TimedTask & **OverallAlgorithm** ()
- TimedTask & **PrintProblemStatistics** ()
- TimedTask & **InitializeIterates** ()
- TimedTask & **UpdateHessian** ()
- TimedTask & **OutputIteration** ()
- TimedTask & **UpdateBarrierParameter** ()
- TimedTask & **ComputeSearchDirection** ()
- TimedTask & **ComputeAcceptableTrialPoint** ()
- TimedTask & **AcceptTrialPoint** ()
- TimedTask & **CheckConvergence** ()
- TimedTask & **PDSystemSolverTotal** ()
- TimedTask & **PDSystemSolverSolveOnce** ()
- TimedTask & **ComputeResiduals** ()
- TimedTask & **StdAugSystemSolverMultiSolve** ()
- TimedTask & **LinearSystemScaling** ()
- TimedTask & **LinearSystemSymbolicFactorization** ()
- TimedTask & **LinearSystemFactorization** ()
- TimedTask & **LinearSystemBackSolve** ()
- TimedTask & **LinearSystemStructureConverter** ()
- TimedTask & **LinearSystemStructureConverterInit** ()
- TimedTask & **QualityFunctionSearch** ()
- TimedTask & **TryCorrector** ()
- TimedTask & **Task1** ()
- TimedTask & **Task2** ()
- TimedTask & **Task3** ()
- TimedTask & **Task4** ()
- TimedTask & **Task5** ()
- TimedTask & **Task6** ()

3.175.1 Detailed Description

This class collects all timing statistics for Ipopt.

Definition at line 20 of file IpTimingStatistics.hpp.

3.175.2 Constructor & Destructor Documentation

3.175.2.1 Ipopt::TimingStatistics::TimingStatistics ( ) [inline]

Default constructor.

Definition at line 26 of file IpTimingStatistics.hpp.

3.175.3 Member Function Documentation

3.175.3.1 void Ipopt::TimingStatistics::ResetTimes ( )

Method for resetting all times.

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

- IpTimingStatistics.hpp

3.176 Ipopt::TNLP Class Reference

Base class for all NLP's that use standard triplet matrix form and dense vectors.

`#include <IpTNLP.hpp>`

Inheritance diagram for Ipopt::TNLP:

Public Types

- enum LinearityType { LINEAR, NON_LINEAR }

  Type of the constraints.

Public Member Functions

Constructors/Destructors

- **TNLP** ()
- virtual ∼TNLP ()

  Default destructor.

Solution Methods

- virtual void finalize_solution (SolverReturn status, Index n, const Number ∗x, const Number ∗z_L, const Number ∗z_U, Index m, const Number ∗g, const Number ∗lambda, Number obj_value, const IpoptData ∗ip_data, IpoptCalculatedQuantities ∗ip_cq)=0

  This method is called when the algorithm is complete so the TNLP can store/write the solution.
- virtual void finalize_metadata (Index n, const StringMetaDataMapType &var_string_md, const IntegerMetaDataMapType &var_integer_md, const NumericMetaDataMapType &var_numeric_md, Index m, const StringMetaDataMapType &con_string_md, const IntegerMetaDataMapType &con_integer_md, const NumericMetaDataMapType &con_numeric_md)

  This method is called just before finalize_solution.

- virtual bool intermediate_callback (AlgorithmMode mode, Index iter, Number obj_value, Number inf_pr, Number inf_du, Number mu, Number d_norm, Number regularization_size, Number alpha_du, Number alpha_pr, Index ls_trials, const IpoptData ∗ip_data, IpoptCalculatedQuantities ∗ip_cq)

    Intermediate Callback method for the user.

Methods for quasi-Newton approximation. If the second

derivatives are approximated by Ipopt, it is better to do this only in the space of nonlinear variables.

The following methods are call by Ipopt if the quasi-Newton approximation is selected. If -1 is returned as number of nonlinear variables, Ipopt assumes that all variables are nonlinear. Otherwise, it calls get_list_of_nonlinear_variables with an array into which the indices of the nonlinear variables should be written - the array has the lengths num_nonlin_vars, which is identical with the return value of get_number_of_nonlinear_variables(). It is assumed that the indices are counted starting with 1 in the FORTRAN_STYLE, and 0 for the C_STYLE.

- virtual Index **get_number_of_nonlinear_variables** ()
- virtual bool **get_list_of_nonlinear_variables** (Index num_nonlin_vars, Index ∗pos_nonlin_vars)

methods to gather information about the NLP

- enum IndexStyleEnum

    overload this method to return the number of variables and constraints, and the number of non-zeros in the jacobian and the hessian.
- typedef std::map< std::string, std::vector< std::string > > **StringMetaDataMapType**
- typedef std::map< std::string, std::vector< Index > > **IntegerMetaDataMapType**
- typedef std::map< std::string, std::vector< Number > > **NumericMetaDataMapType**
- virtual bool **get_nlp_info** (Index &n, Index &m, Index &nnz_jac_g, Index &nnz_h_lag, IndexStyleEnum &index_style)=0
- virtual bool get_var_con_metadata (Index n, StringMetaDataMapType &var_string_md, IntegerMetaDataMapType &var_integer_md, NumericMetaDataMapType &var_numeric_md, Index m, StringMetaDataMapType &con_string_md, IntegerMetaDataMapType &con_integer_md, NumericMetaDataMapType &con_numeric_md)

    overload this method to return any meta data for the variables and the constraints
- virtual bool get_bounds_info (Index n, Number ∗x_l, Number ∗x_u, Index m, Number ∗g_l, Number ∗g_u)=0

    overload this method to return the information about the bound on the variables and constraints.
- virtual bool get_scaling_parameters (Number &obj_scaling, bool &use_x_scaling, Index n, Number ∗x_scaling, bool &use_g_scaling, Index m, Number ∗g_scaling)

    overload this method to return scaling parameters.
- virtual bool get_variables_linearity (Index n, LinearityType ∗var_types)

    overload this method to return the variables linearity (TNLP::LINEAR or TNLP::NON_LINEAR).
- virtual bool get_constraints_linearity (Index m, LinearityType ∗const_types)

    overload this method to return the constraint linearity.
- virtual bool get_starting_point (Index n, bool init_x, Number ∗x, bool init_z, Number ∗z_L, Number ∗z_U, Index m, bool init_lambda, Number ∗lambda)=0

    overload this method to return the starting point.
- virtual bool get_warm_start_iterate (IteratesVector &warm_start_iterate)

    overload this method to provide an Ipopt iterate (already in the form Ipopt requires it internally) for a warm start.
- virtual bool eval_f (Index n, const Number ∗x, bool new_x, Number &obj_value)=0

    overload this method to return the value of the objective function
- virtual bool eval_grad_f (Index n, const Number ∗x, bool new_x, Number ∗grad_f)=0

    overload this method to return the vector of the gradient of the objective w.r.t.
- virtual bool eval_g (Index n, const Number ∗x, bool new_x, Index m, Number ∗g)=0

overload this method to return the vector of constraint values

- virtual bool eval_jac_g (Index n, const Number *x, bool new_x, Index m, Index nele_jac, Index *iRow, Index *jCol, Number *values)=0

  overload this method to return the jacobian of the constraints.

- virtual bool eval_h (Index n, const Number *x, bool new_x, Number obj_factor, Index m, const Number *lambda, bool new_lambda, Index nele_hess, Index *iRow, Index *jCol, Number *values)

  overload this method to return the hessian of the lagrangian.

3.176.1 Detailed Description

Base class for all NLP's that use standard triplet matrix form and dense vectors.

This is the standard base class for all NLP's that use the standard triplet matrix form (as for Harwell routines) and dense vectors. The class TNLPAdapter then converts this interface to an interface that can be used directly by ipopt.

This interface presents the problem form:

min f(x)

s.t. gL <= g(x) <= gU

```
xL <=  x   <= xU
```

In order to specify an equality constraint, set gL_i = gU_i = rhs. The value that indicates "infinity" for the bounds (i.e. the variable or constraint has no lower bound (-infinity) or upper bound (+infinity)) is set through the option nlp_lower_bound_inf and nlp_upper_bound_inf. To indicate that a variable has no upper or lower bound, set the bound to -ipopt_inf or +ipopt_inf respectively

Definition at line 50 of file IpTNLP.hpp.

3.176.2 Member Enumeration Documentation

3.176.2.1 enum Ipopt::TNLP::LinearityType

Type of the constraints.

Enumerator

LINEAR Constraint/Variable is linear.

NON_LINEAR Constraint/Varaible is non-linear.

Definition at line 54 of file IpTNLP.hpp.

3.176.2.2 enum Ipopt::TNLP::IndexStyleEnum

overload this method to return the number of variables and constraints, and the number of non-zeros in the jacobian and the hessian.

The index_style parameter lets you specify C or Fortran style indexing for the sparse matrix iRow and jCol parameters. C_STYLE is 0-based, and FORTRAN_STYLE is 1-based.

Definition at line 80 of file IpTNLP.hpp.

3.176.3 Member Function Documentation

3.176.3.1 virtual bool Ipopt::TNLP::get_bounds_info (Index *n*, Number ∗ *x_l*, Number ∗ *x_u*, Index *m*, Number ∗ *g_l*, Number ∗ *g_u*) [pure virtual]

overload this method to return the information about the bound on the variables and constraints.

The value that indicates that a bound does not exist is specified in the parameters nlp_lower_bound_inf and nlp_←upper_bound_inf. By default, nlp_lower_bound_inf is -1e19 and nlp_upper_bound_inf is 1e19. (see TNLPAdapter)

Implemented in Ipopt::AmplTNLP, Ipopt::StdInterfaceTNLP, and Ipopt::TNLPReducer.

3.176.3.2 virtual bool Ipopt::TNLP::get_scaling_parameters (Number & *obj_scaling*, bool & *use_x_scaling*, Index *n*, Number ∗ *x_scaling*, bool & *use_g_scaling*, Index *m*, Number ∗ *g_scaling*) [inline], [virtual]

overload this method to return scaling parameters.

This is only called if the options are set to retrieve user scaling. There, use_x_scaling (or use_g_scaling) should get set to true only if the variables (or constraints) are to be scaled. This method should return true only if the scaling parameters could be provided.

Reimplemented in Ipopt::AmplTNLP, Ipopt::StdInterfaceTNLP, and Ipopt::TNLPReducer.

Definition at line 119 of file IpTNLP.hpp.

3.176.3.3 virtual bool Ipopt::TNLP::get_variables_linearity (Index *n*, LinearityType ∗ *var_types*) [inline], [virtual]

overload this method to return the variables linearity (TNLP::LINEAR or TNLP::NON_LINEAR).

The var_types array has been allocated with length at least n. (default implementation just return false and does not fill the array).

Reimplemented in Ipopt::TNLPReducer.

Definition at line 132 of file IpTNLP.hpp.

3.176.3.4 virtual bool Ipopt::TNLP::get_constraints_linearity (Index *m*, LinearityType ∗ *const_types*) [inline], [virtual]

overload this method to return the constraint linearity.

array has been allocated with length at least n. (default implementation just return false and does not fill the array).

Reimplemented in Ipopt::AmplTNLP, and Ipopt::TNLPReducer.

Definition at line 140 of file IpTNLP.hpp.

3.176.3.5 virtual bool Ipopt::TNLP::get_starting_point (Index *n*, bool *init_x*, Number ∗ *x*, bool *init_z*, Number ∗ *z_L*, Number ∗ *z_U*, Index *m*, bool *init_lambda*, Number ∗ *lambda*) [pure virtual]

overload this method to return the starting point.

The bool variables indicate whether the algorithm wants you to initialize x, z_L/z_u, and lambda, respectively. If, for some reason, the algorithm wants you to initialize these and you cannot, return false, which will cause Ipopt to stop. You will have to run Ipopt with different options then.

Implemented in Ipopt::AmplTNLP, Ipopt::StdInterfaceTNLP, and Ipopt::TNLPReducer.

3.176.3.6 virtual bool Ipopt::TNLP::get_warm_start_iterate (IteratesVector & *warm_start_iterate*) [inline], [virtual]

overload this method to provide an Ipopt iterate (already in the form Ipopt requires it internally) for a warm start.

Since this is only for expert users, a default dummy implementation is provided and returns false.

Reimplemented in Ipopt::TNLPReducer.

Definition at line 161 of file IpTNLP.hpp.

3.176.3.7 virtual bool Ipopt::TNLP::eval_grad_f (Index *n*, const Number ∗ *x*, bool *new_x*, Number ∗ *grad_f*) [pure virtual]

overload this method to return the vector of the gradient of the objective w.r.t.

x

Implemented in Ipopt::AmplTNLP, Ipopt::StdInterfaceTNLP, and Ipopt::TNLPReducer.

3.176.3.8 virtual bool Ipopt::TNLP::eval_jac_g (Index *n*, const Number ∗ *x*, bool *new_x*, Index *m*, Index *nele_jac*, Index ∗ *iRow*, Index ∗ *jCol*, Number ∗ *values*) [pure virtual]

overload this method to return the jacobian of the constraints.

The vectors iRow and jCol only need to be set once. The first call is used to set the structure only (iRow and jCol will be non-NULL, and values will be NULL) For subsequent calls, iRow and jCol will be NULL.

Implemented in Ipopt::AmplTNLP, Ipopt::StdInterfaceTNLP, and Ipopt::TNLPReducer.

3.176.3.9 virtual bool Ipopt::TNLP::eval_h (Index *n*, const Number ∗ *x*, bool *new_x*, Number *obj_factor*, Index *m*, const Number ∗ *lambda*, bool *new_lambda*, Index *nele_hess*, Index ∗ *iRow*, Index ∗ *jCol*, Number ∗ *values*) [inline], [virtual]

overload this method to return the hessian of the lagrangian.

The vectors iRow and jCol only need to be set once (during the first call). The first call is used to set the structure only (iRow and jCol will be non-NULL, and values will be NULL) For subsequent calls, iRow and jCol will be NULL. This matrix is symmetric - specify the lower diagonal only. A default implementation is provided, in case the user wants to se quasi-Newton approximations to estimate the second derivatives and doesn't not neet to implement this method.

Reimplemented in Ipopt::AmplTNLP, Ipopt::StdInterfaceTNLP, and Ipopt::TNLPReducer.

Definition at line 196 of file IpTNLP.hpp.

3.176.3.10 virtual void Ipopt::TNLP::finalize_metadata (Index *n*, const StringMetaDataMapType & *var_string_md*, const IntegerMetaDataMapType & *var_integer_md*, const NumericMetaDataMapType & *var_numeric_md*, Index *m*, const StringMetaDataMapType & *con_string_md*, const IntegerMetaDataMapType & *con_integer_md*, const NumericMetaDataMapType & *con_numeric_md*) [inline], [virtual]

This method is called just before finalize_solution.

With this method, the algorithm returns any metadata collected during its run, including the metadata provided by the user with the above get_var_con_metadata. Each metadata can be of type string, integer, and numeric. It can be associated to either the variables or the constraints. The metadata that was associated with the primal variable vector is stored in var_..._md. The metadata associated with the constraint multipliers is stored in con_..._md. The metadata associated with the bound multipliers is stored in var_..._md, with the suffixes "_z_L", and "_z_U", denoting lower and upper bounds.

Definition at line 226 of file IpTNLP.hpp.

3.176.3.11 virtual bool Ipopt::TNLP::intermediate_callback (AlgorithmMode *mode,* Index *iter,* Number *obj_value,* Number *inf_pr,* Number *inf_du,* Number *mu,* Number *d_norm,* Number *regularization_size,* Number *alpha_du,* Number *alpha_pr,* Index *ls_trials,* const IpoptData ∗ *ip_data,* IpoptCalculatedQuantities ∗ *ip_cq*) [inline], [virtual]

Intermediate Callback method for the user.

Providing dummy default implementation. For details see IntermediateCallBack in IpNLP.hpp.

Reimplemented in Ipopt::StdInterfaceTNLP, and Ipopt::TNLPReducer.

Definition at line 240 of file IpTNLP.hpp.

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

- IpTNLP.hpp

3.177 Ipopt::TNLPAdapter Class Reference

This class Adapts the TNLP interface so it looks like an NLP interface.

`#include <IpTNLPAdapter.hpp>`

Inheritance diagram for Ipopt::TNLPAdapter:

Public Types

- enum FixedVariableTreatmentEnum

    Enum for treatment of fixed variables option.
- enum DerivativeTestEnum

    Enum for specifying which derivative test is to be performed.
- enum JacobianApproxEnum

    Enum for specifying technique for computing Jacobian.

Public Member Functions

- virtual void GetQuasiNewtonApproximationSpaces (SmartPtr< VectorSpace > &approx_space, SmartPtr< Matrix > &P_approx)

    Method returning information on quasi-Newton approximation.
- bool CheckDerivatives (DerivativeTestEnum deriv_test, Index deriv_test_start_index)

    Method for performing the derivative test.
- SmartPtr< TNLP > tnlp () const

    Accessor method for the underlying TNLP.

Constructors/Destructors

- TNLPAdapter (const SmartPtr< TNLP > tnlp, const SmartPtr< const Journalist > jnlst=NULL)

    Default constructor.

- virtual ~TNLPAdapter ()

    Default destructor.

Exceptions

- **DECLARE_STD_EXCEPTION** (INVALID_TNLP)
- **DECLARE_STD_EXCEPTION** (ERROR_IN_TNLP_DERIVATIVE_TEST)

TNLPAdapter Initialization.

- virtual bool ProcessOptions (const OptionsList &options, const std::string &prefix)

    Overload if you want the chance to process options or parameters that may be specific to the NLP.
- virtual bool GetSpaces (SmartPtr< const VectorSpace > &x_space, SmartPtr< const VectorSpace > &c_space, SmartPtr< const VectorSpace > &d_space, SmartPtr< const VectorSpace > &x_l_space, SmartPtr< const MatrixSpace > &px_l_space, SmartPtr< const VectorSpace > &x_u_space, SmartPtr< const MatrixSpace > &px_u_space, SmartPtr< const VectorSpace > &d_l_space, SmartPtr< const MatrixSpace > &pd_l_space, SmartPtr< const VectorSpace > &d_u_space, SmartPtr< const MatrixSpace > &pd_u_space, SmartPtr< const MatrixSpace > &Jac_c_space, SmartPtr< const MatrixSpace > &Jac_d_space, SmartPtr< const SymMatrixSpace > &Hess_lagrangian_space)

    Method for creating the derived vector / matrix types (Do not delete these, the).
- virtual bool GetBoundsInformation (const Matrix &Px_L, Vector &x_L, const Matrix &Px_U, Vector &x_U, const Matrix &Pd_L, Vector &d_L, const Matrix &Pd_U, Vector &d_U)

    Method for obtaining the bounds information.
- virtual bool GetStartingPoint (SmartPtr< Vector > x, bool need_x, SmartPtr< Vector > y_c, bool need_y_c, SmartPtr< Vector > y_d, bool need_y_d, SmartPtr< Vector > z_L, bool need_z_L, SmartPtr< Vector > z_U, bool need_z_U)

    Method for obtaining the starting point for all the iterates.
- virtual bool GetWarmStartIterate (IteratesVector &warm_start_iterate)

    Method for obtaining an entire iterate as a warmstart point.

TNLPAdapter evaluation routines.

- virtual bool **Eval_f** (const Vector &x, Number &f)
- virtual bool **Eval_grad_f** (const Vector &x, Vector &g_f)
- virtual bool **Eval_c** (const Vector &x, Vector &c)
- virtual bool **Eval_jac_c** (const Vector &x, Matrix &jac_c)
- virtual bool **Eval_d** (const Vector &x, Vector &d)
- virtual bool **Eval_jac_d** (const Vector &x, Matrix &jac_d)
- virtual bool **Eval_h** (const Vector &x, Number obj_factor, const Vector &yc, const Vector &yd, SymMatrix &h)
- virtual void GetScalingParameters (const SmartPtr< const VectorSpace > x_space, const SmartPtr< const VectorSpace > c_space, const SmartPtr< const VectorSpace > d_space, Number &obj_scaling, SmartPtr< Vector > &x_scaling, SmartPtr< Vector > &c_scaling, SmartPtr< Vector > &d_scaling) const

    Routines to get the scaling parameters.

Solution Reporting Methods

- virtual void FinalizeSolution (SolverReturn status, const Vector &x, const Vector &z_L, const Vector &z_U, const Vector &c, const Vector &d, const Vector &y_c, const Vector &y_d, Number obj_value, const IpoptData ∗ip_data, IpoptCalculatedQuantities ∗ip_cq)

    This method is called at the very end of the optimization.
- virtual bool IntermediateCallBack (AlgorithmMode mode, Index iter, Number obj_value, Number inf_pr, Number inf_du, Number mu, Number d_norm, Number regularization_size, Number alpha_du, Number alpha_pr, Index ls_trials, const IpoptData ∗ip_data, IpoptCalculatedQuantities ∗ip_cq)

    This method is called once per iteration, after the iteration summary output has been printed.

Methods for translating data for IpoptNLP into the TNLP

data.

These methods are used to obtain the current (or final) data for the TNLP formulation from the IpoptNLP structure.

- void ResortX (const Vector &x, Number ∗x_orig)
  Sort the primal variables, and add the fixed values in x.
- void **ResortG** (const Vector &c, const Vector &d, Number ∗g_orig)
- void **ResortBnds** (const Vector &x_L, Number ∗x_L_orig, const Vector &x_U, Number ∗x_U_orig)

Static Public Member Functions

Methods for IpoptType

- static void **RegisterOptions** (SmartPtr< RegisteredOptions > roptions)

3.177.1 Detailed Description

This class Adapts the TNLP interface so it looks like an NLP interface.

This is an Adapter class (Design Patterns) that converts a TNLP to an NLP. This allows users to write to the "more convenient" TNLP interface.

Definition at line 30 of file IpTNLPAdapter.hpp.

3.177.2 Member Enumeration Documentation

3.177.2.1 enum Ipopt::TNLPAdapter::DerivativeTestEnum

Enum for specifying which derivative test is to be performed.

Definition at line 167 of file IpTNLPAdapter.hpp.

3.177.3 Member Function Documentation

3.177.3.1 virtual bool Ipopt::TNLPAdapter::GetSpaces (SmartPtr< const VectorSpace > & *x_space*, SmartPtr< const VectorSpace > & *c_space*, SmartPtr< const VectorSpace > & *d_space*, SmartPtr< const VectorSpace > & *x_l_space*, SmartPtr< const MatrixSpace > & *px_l_space*, SmartPtr< const VectorSpace > & *x_u_space*, SmartPtr< const MatrixSpace > & *px_u_space*, SmartPtr< const VectorSpace > & *d_l_space*, SmartPtr< const MatrixSpace > & *pd_l_space*, SmartPtr< const VectorSpace > & *d_u_space*, SmartPtr< const MatrixSpace > & *pd_u_space*, SmartPtr< const MatrixSpace > & *Jac_c_space*, SmartPtr< const MatrixSpace > & *Jac_d_space*, SmartPtr< const SymMatrixSpace > & *Hess_lagrangian_space*) [virtual]

Method for creating the derived vector / matrix types (Do not delete these, the).

Implements Ipopt::NLP.

3.177.3.2 virtual bool Ipopt::TNLPAdapter::GetStartingPoint (SmartPtr< Vector > *x*, bool *need_x*, SmartPtr< Vector > *y_c*, bool *need_y_c*, SmartPtr< Vector > *y_d*, bool *need_y_d*, SmartPtr< Vector > *z_L*, bool *need_z_L*, SmartPtr< Vector > *z_U*, bool *need_z_U*) [virtual]

Method for obtaining the starting point for all the iterates.

Implements Ipopt::NLP.

3.177.3.3 virtual bool Ipopt::TNLPAdapter::GetWarmStartIterate (IteratesVector & *warm_start_iterate*) [virtual]

Method for obtaining an entire iterate as a warmstart point.

The incoming IteratesVector has to be filled.

Reimplemented from Ipopt::NLP.

3.177.3.4 virtual void Ipopt::TNLPAdapter::GetScalingParameters (const SmartPtr< const VectorSpace > *x_space*, const SmartPtr< const VectorSpace > *c_space*, const SmartPtr< const VectorSpace > *d_space*, Number & *obj_scaling*, SmartPtr< Vector > & *x_scaling*, SmartPtr< Vector > & *c_scaling*, SmartPtr< Vector > & *d_scaling*) const [virtual]

Routines to get the scaling parameters.

These do not need to be overloaded unless the options are set for User scaling

Reimplemented from Ipopt::NLP.

3.177.3.5 virtual void Ipopt::TNLPAdapter::FinalizeSolution (SolverReturn *status*, const Vector & *x*, const Vector & *z_L*, const Vector & *z_U*, const Vector & *c*, const Vector & *d*, const Vector & *y_c*, const Vector & *y_d*, Number *obj_value*, const IpoptData ∗ *ip_data*, IpoptCalculatedQuantities ∗ *ip_cq*) [virtual]

This method is called at the very end of the optimization.

It provides the final iterate to the user, so that it can be stored as the solution. The status flag indicates the outcome of the optimization, where SolverReturn is defined in IpAlgTypes.hpp.

Reimplemented from Ipopt::NLP.

3.177.3.6 virtual bool Ipopt::TNLPAdapter::IntermediateCallBack (AlgorithmMode *mode*, Index *iter*, Number *obj_value*, Number *inf_pr*, Number *inf_du*, Number *mu*, Number *d_norm*, Number *regularization_size*, Number *alpha_du*, Number *alpha_pr*, Index *ls_trials*, const IpoptData ∗ *ip_data*, IpoptCalculatedQuantities ∗ *ip_cq*) [virtual]

This method is called once per iteration, after the iteration summary output has been printed.

It provides the current information to the user to do with it anything she wants. It also allows the user to ask for a premature termination of the optimization by returning false, in which case Ipopt will terminate with a corresponding return status. The basic information provided in the argument list has the quantities values printed in the iteration summary line. If more information is required, a user can obtain it from the IpData and IpCalculatedQuantities objects. However, note that the provided quantities are all for the problem that Ipopt sees, i.e., the quantities might be scaled, fixed variables might be sorted out, etc. The status indicates things like whether the algorithm is in the restoration phase... In the restoration phase, the dual variables are probably not not changing.

Reimplemented from Ipopt::NLP.

3.177.3.7 virtual void Ipopt::TNLPAdapter::GetQuasiNewtonApproximationSpaces (SmartPtr< VectorSpace > & *approx_space*, SmartPtr< Matrix > & *P_approx*) [virtual]

Method returning information on quasi-Newton approximation.

Reimplemented from Ipopt::NLP.

3.177.3.8 SmartPtr<TNLP> Ipopt::TNLPAdapter::tnlp ( ) const [inline]

Accessor method for the underlying TNLP.

Definition at line 192 of file IpTNLPAdapter.hpp.

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

- IpTNLPAdapter.hpp

3.178 Ipopt::TNLPReducer Class Reference

This is a wrapper around a given TNLP class that takes out a list of constraints that are given to the constructor.

`#include <IpTNLPReducer.hpp>`

Inheritance diagram for Ipopt::TNLPReducer:

Public Member Functions

Constructors/Destructors

- TNLPReducer (TNLP &tnlp, Index n_g_skip, const Index ∗index_g_skip, Index n_xL_skip, const Index ∗index↵ _xL_skip, Index n_xU_skip, const Index ∗index_xU_skip, Index n_x_fix, const Index ∗index_f_fix)

  Constructor is given the indices of the constraints that should be taken out of the problem statement, as well as the original TNLP.
- virtual ∼TNLPReducer ()

  Default destructor.

Overloaded methods from TNLP

- virtual bool **get_nlp_info** (Index &n, Index &m, Index &nnz_jac_g, Index &nnz_h_lag, IndexStyleEnum &index_style)
- virtual bool get_bounds_info (Index n, Number ∗x_l, Number ∗x_u, Index m, Number ∗g_l, Number ∗g_u)

  overload this method to return the information about the bound on the variables and constraints.
- virtual bool get_scaling_parameters (Number &obj_scaling, bool &use_x_scaling, Index n, Number ∗x_scaling, bool &use_g_scaling, Index m, Number ∗g_scaling)

  overload this method to return scaling parameters.
- virtual bool get_variables_linearity (Index n, LinearityType ∗var_types)

  overload this method to return the variables linearity (TNLP::LINEAR or TNLP::NON_LINEAR).
- virtual bool get_constraints_linearity (Index m, LinearityType ∗const_types)

  overload this method to return the constraint linearity.
- virtual bool get_starting_point (Index n, bool init_x, Number ∗x, bool init_z, Number ∗z_L, Number ∗z_U, Index m, bool init_lambda, Number ∗lambda)

  overload this method to return the starting point.
- virtual bool get_warm_start_iterate (IteratesVector &warm_start_iterate)

  overload this method to provide an Ipopt iterate (already in the form Ipopt requires it internally) for a warm start.
- virtual bool eval_f (Index n, const Number ∗x, bool new_x, Number &obj_value)

  overload this method to return the value of the objective function
- virtual bool eval_grad_f (Index n, const Number ∗x, bool new_x, Number ∗grad_f)

  overload this method to return the vector of the gradient of the objective w.r.t.
- virtual bool eval_g (Index n, const Number ∗x, bool new_x, Index m, Number ∗g)

  overload this method to return the vector of constraint values

- virtual bool eval_jac_g (Index n, const Number ∗x, bool new_x, Index m, Index nele_jac, Index ∗iRow, Index ∗jCol, Number ∗values)
  overload this method to return the jacobian of the constraints.
- virtual bool eval_h (Index n, const Number ∗x, bool new_x, Number obj_factor, Index m, const Number ∗lambda, bool new_lambda, Index nele_hess, Index ∗iRow, Index ∗jCol, Number ∗values)
  overload this method to return the hessian of the lagrangian.
- virtual void finalize_solution (SolverReturn status, Index n, const Number ∗x, const Number ∗z_L, const Number ∗z_U, Index m, const Number ∗g, const Number ∗lambda, Number obj_value, const IpoptData ∗ip_data, IpoptCalculatedQuantities ∗ip_cq)
  This method is called when the algorithm is complete so the TNLP can store/write the solution.
- virtual bool intermediate_callback (AlgorithmMode mode, Index iter, Number obj_value, Number inf_pr, Number inf_du, Number mu, Number d_norm, Number regularization_size, Number alpha_du, Number alpha_pr, Index ls_trials, const IpoptData ∗ip_data, IpoptCalculatedQuantities ∗ip_cq)
  Intermediate Callback method for the user.
- virtual Index **get_number_of_nonlinear_variables** ()
- virtual bool **get_list_of_nonlinear_variables** (Index num_nonlin_vars, Index ∗pos_nonlin_vars)

Additional Inherited Members

3.178.1 Detailed Description

This is a wrapper around a given TNLP class that takes out a list of constraints that are given to the constructor.

It is provided for convenience, if one wants to experiment with problems that consist of only a subset of the constraints. But keep in mind that this is not efficient, since behind the scenes we are still evaluation all functions and derivatives, and are making copies of the original data.

Definition at line 23 of file IpTNLPReducer.hpp.

3.178.2 Constructor & Destructor Documentation

3.178.2.1 Ipopt::TNLPReducer::TNLPReducer (TNLP & *tnlp*, Index *n_g_skip*, const Index ∗ *index_g_skip*, Index *n_xL_skip*, const Index ∗ *index_xL_skip*, Index *n_xU_skip*, const Index ∗ *index_xU_skip*, Index *n_x_fix*, const Index ∗ *index_f_fix*)

Constructor is given the indices of the constraints that should be taken out of the problem statement, as well as the original TNLP.

3.178.3 Member Function Documentation

3.178.3.1 virtual bool Ipopt::TNLPReducer::get_bounds_info (Index *n*, Number ∗ *x_l*, Number ∗ *x_u*, Index *m*, Number ∗ *g_l*, Number ∗ *g_u*) [virtual]

overload this method to return the information about the bound on the variables and constraints.

The value that indicates that a bound does not exist is specified in the parameters nlp_lower_bound_inf and nlp_↩ upper_bound_inf. By default, nlp_lower_bound_inf is -1e19 and nlp_upper_bound_inf is 1e19. (see TNLPAdapter)

Implements Ipopt::TNLP.

3.178.3.2 virtual bool Ipopt::TNLPReducer::get_scaling_parameters (Number & *obj_scaling*, bool & *use_x_scaling*, Index *n*, Number ∗ *x_scaling*, bool & *use_g_scaling*, Index *m*, Number ∗ *g_scaling*) [virtual]

overload this method to return scaling parameters.

This is only called if the options are set to retrieve user scaling. There, use_x_scaling (or use_g_scaling) should get set to true only if the variables (or constraints) are to be scaled. This method should return true only if the scaling parameters could be provided.

Reimplemented from Ipopt::TNLP.

3.178.3.3 virtual bool Ipopt::TNLPReducer::get_variables_linearity (Index *n*, LinearityType ∗ *var_types*) [virtual]

overload this method to return the variables linearity (TNLP::LINEAR or TNLP::NON_LINEAR).

The var_types array has been allocated with length at least n. (default implementation just return false and does not fill the array).

Reimplemented from Ipopt::TNLP.

3.178.3.4 virtual bool Ipopt::TNLPReducer::get_constraints_linearity (Index *m*, LinearityType ∗ *const_types*) [virtual]

overload this method to return the constraint linearity.

array has been allocated with length at least n. (default implementation just return false and does not fill the array).

Reimplemented from Ipopt::TNLP.

3.178.3.5 virtual bool Ipopt::TNLPReducer::get_starting_point (Index *n*, bool *init_x*, Number ∗ *x*, bool *init_z*, Number ∗ *z_L*, Number ∗ *z_U*, Index *m*, bool *init_lambda*, Number ∗ *lambda*) [virtual]

overload this method to return the starting point.

The bool variables indicate whether the algorithm wants you to initialize x, z_L/z_u, and lambda, respectively. If, for some reason, the algorithm wants you to initialize these and you cannot, return false, which will cause Ipopt to stop. You will have to run Ipopt with different options then.

Implements Ipopt::TNLP.

3.178.3.6 virtual bool Ipopt::TNLPReducer::get_warm_start_iterate (IteratesVector & *warm_start_iterate*) [virtual]

overload this method to provide an Ipopt iterate (already in the form Ipopt requires it internally) for a warm start.

Since this is only for expert users, a default dummy implementation is provided and returns false.

Reimplemented from Ipopt::TNLP.

3.178.3.7 virtual bool Ipopt::TNLPReducer::eval_grad_f (Index *n*, const Number ∗ *x*, bool *new_x*, Number ∗ *grad_f*) [virtual]

overload this method to return the vector of the gradient of the objective w.r.t.

x

Implements Ipopt::TNLP.

3.178.3.8 virtual bool Ipopt::TNLPReducer::eval_jac_g (Index *n*, const Number ∗ *x*, bool *new_x*, Index *m*, Index *nele_jac*, Index ∗ *iRow*, Index ∗ *jCol*, Number ∗ *values*) [virtual]

overload this method to return the jacobian of the constraints.

The vectors iRow and jCol only need to be set once. The first call is used to set the structure only (iRow and jCol will be non-NULL, and values will be NULL) For subsequent calls, iRow and jCol will be NULL.

Implements Ipopt::TNLP.

3.178.3.9 virtual bool Ipopt::TNLPReducer::eval_h (Index *n*, const Number ∗ *x*, bool *new_x*, Number *obj_factor*, Index *m*, const Number ∗ *lambda*, bool *new_lambda*, Index *nele_hess*, Index ∗ *iRow*, Index ∗ *jCol*, Number ∗ *values*) [virtual]

overload this method to return the hessian of the lagrangian.

The vectors iRow and jCol only need to be set once (during the first call). The first call is used to set the structure only (iRow and jCol will be non-NULL, and values will be NULL) For subsequent calls, iRow and jCol will be NULL. This matrix is symmetric - specify the lower diagonal only. A default implementation is provided, in case the user wants to se quasi-Newton approximations to estimate the second derivatives and doesn't not neet to implement this method.

Reimplemented from Ipopt::TNLP.

3.178.3.10 virtual bool Ipopt::TNLPReducer::intermediate_callback (AlgorithmMode *mode*, Index *iter*, Number *obj_value*, Number *inf_pr*, Number *inf_du*, Number *mu*, Number *d_norm*, Number *regularization_size*, Number *alpha_du*, Number *alpha_pr*, Index *ls_trials*, const IpoptData ∗ *ip_data*, IpoptCalculatedQuantities ∗ *ip_cq*) [virtual]

Intermediate Callback method for the user.

Providing dummy default implementation. For details see IntermediateCallBack in IpNLP.hpp.

Reimplemented from Ipopt::TNLP.

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

- IpTNLPReducer.hpp

3.179 Ipopt::TransposeMatrix Class Reference

Class for Matrices which are the transpose of another matrix.

`#include <IpTransposeMatrix.hpp>`

Inheritance diagram for Ipopt::TransposeMatrix:

Public Member Functions

Constructors / Destructors

- TransposeMatrix (const TransposeMatrixSpace ∗owner_space)
    Constructor, initializing with dimensions of the matrix.
- ∼TransposeMatrix ()
    Destructor.
- SmartPtr< const Matrix > **OrigMatrix** () const

Protected Member Functions

Methods overloaded from matrix

- virtual void MultVectorImpl (Number alpha, const Vector &x, Number beta, Vector &y) const

Matrix-vector multiply.
- virtual void TransMultVectorImpl (Number alpha, const Vector &x, Number beta, Vector &y) const
  Matrix(transpose) vector multiply.
- virtual bool HasValidNumbersImpl () const
  Method for determining if all stored numbers are valid (i.e., no Inf or Nan).
- virtual void ComputeRowAMaxImpl (Vector &rows_norms, bool init) const
  Compute the max-norm of the rows in the matrix.
- virtual void ComputeColAMaxImpl (Vector &rows_norms, bool init) const
  Compute the max-norm of the columns in the matrix.
- virtual void PrintImpl (const Journalist &jnlst, EJournalLevel level, EJournalCategory category, const std::string &name, Index indent, const std::string &prefix) const
  Print detailed information about the matrix.

Additional Inherited Members

3.179.1 Detailed Description

Class for Matrices which are the transpose of another matrix.

Definition at line 23 of file IpTransposeMatrix.hpp.

3.179.2 Member Function Documentation

3.179.2.1 virtual void Ipopt::TransposeMatrix::MultVectorImpl (Number *alpha*, const Vector & *x*, Number *beta*, Vector & *y*) const [inline], [protected], [virtual]

Matrix-vector multiply.

Computes y = alpha ∗ Matrix ∗ x + beta ∗ y

Implements Ipopt::Matrix.

Definition at line 47 of file IpTransposeMatrix.hpp.

3.179.2.2 virtual void Ipopt::TransposeMatrix::TransMultVectorImpl (Number *alpha*, const Vector & *x*, Number *beta*, Vector & *y*) const [inline], [protected], [virtual]

Matrix(transpose) vector multiply.

Computes y = alpha ∗ Matrix^T ∗ x + beta ∗ y

Implements Ipopt::Matrix.

Definition at line 54 of file IpTransposeMatrix.hpp.

3.179.2.3 virtual bool Ipopt::TransposeMatrix::HasValidNumbersImpl ( ) const [inline], [protected], [virtual]

Method for determining if all stored numbers are valid (i.e., no Inf or Nan).

Reimplemented from Ipopt::Matrix.

Definition at line 63 of file IpTransposeMatrix.hpp.

3.179.2.4 virtual void Ipopt::TransposeMatrix::ComputeRowAMaxImpl (Vector & *rows_norms*, bool *init*) const [inline], [protected], [virtual]

Compute the max-norm of the rows in the matrix.

The result is stored in rows_norms. The vector is assumed to be initialized.

Implements Ipopt::Matrix.

Definition at line 69 of file IpTransposeMatrix.hpp.

3.179.2.5 virtual void Ipopt::TransposeMatrix::ComputeColAMaxImpl (Vector & *cols_norms*, bool *init*) const [inline], [protected], [virtual]

Compute the max-norm of the columns in the matrix.

The result is stored in cols_norms. The vector is assumed to be initialized.

Implements Ipopt::Matrix.

Definition at line 75 of file IpTransposeMatrix.hpp.

3.179.2.6 virtual void Ipopt::TransposeMatrix::PrintImpl (const Journalist & *jnlst*, EJournalLevel *level*, EJournalCategory *category*, const std::string & *name*, Index *indent*, const std::string & *prefix*) const [protected], [virtual]

Print detailed information about the matrix.

Implements Ipopt::Matrix.

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

- IpTransposeMatrix.hpp

3.180 Ipopt::TransposeMatrixSpace Class Reference

This is the matrix space for TransposeMatrix.

`#include <IpTransposeMatrix.hpp>`

Inheritance diagram for Ipopt::TransposeMatrixSpace:

Public Member Functions

- virtual Matrix * MakeNew () const

  Overloaded MakeNew method for the MatrixSpace base class.
- TransposeMatrix * MakeNewTransposeMatrix () const

  Method for creating a new matrix of this specific type.

Constructors / Destructors

- TransposeMatrixSpace (const MatrixSpace *orig_matrix_space)

  Constructor, given the dimension of the matrix.
- virtual ~TransposeMatrixSpace ()

  Destructor.

3.180.1 Detailed Description

This is the matrix space for TransposeMatrix.

Definition at line 113 of file IpTransposeMatrix.hpp.

3.180.2 Constructor & Destructor Documentation

3.180.2.1 Ipopt::TransposeMatrixSpace::TransposeMatrixSpace (const MatrixSpace ∗ *orig_matrix_space*) [inline]

Constructor, given the dimension of the matrix.

Definition at line 119 of file IpTransposeMatrix.hpp.

3.180.3 Member Function Documentation

3.180.3.1 TransposeMatrix∗ Ipopt::TransposeMatrixSpace::MakeNewTransposeMatrix ( ) const [inline]

Method for creating a new matrix of this specific type.

Definition at line 138 of file IpTransposeMatrix.hpp.

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

- IpTransposeMatrix.hpp

3.181 Ipopt::TripletHelper Class Reference

Static Public Member Functions

A set of recursive routines that help with the Triplet format.

- static Index GetNumberEntries (const Matrix &matrix)
    find the total number of triplet entries of a Matrix
- static void FillRowCol (Index n_entries, const Matrix &matrix, Index ∗iRow, Index ∗jCol, Index row_offset=0, Index col_offset=0)
    fill the irows, jcols structure for the triplet format from the matrix
- static void FillValues (Index n_entries, const Matrix &matrix, Number ∗values)
    fill the values for the triplet format from the matrix
- static void FillValuesFromVector (Index dim, const Vector &vector, Number ∗values)
    fill the values from the vector into a dense double∗ structure
- static void PutValuesInVector (Index dim, const double ∗values, Vector &vector)
    put the values from the double∗ back into the vector

3.181.1 Detailed Description

Definition at line 40 of file IpTripletHelper.hpp.

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

- IpTripletHelper.hpp

3.182 Ipopt::TripletToCSRConverter Class Reference

Class for converting symmetric matrices given in triplet format to matrices in compressed sparse row (CSR) format of the upper triangular part (or, equivalently, compressed sparse column (CSC) format for the lower triangular part).

`#include <IpTripletToCSRConverter.hpp>`

Inheritance diagram for Ipopt::TripletToCSRConverter:

Public Types

- enum ETriFull { Triangular_Format, Full_Format }

  Enum to specifiy half or full matrix storage.

Public Member Functions

- Index InitializeConverter (Index dim, Index nonzeros, const Index ∗airn, const Index ∗ajcn)

  Initialize the converter, given the fixed structure of the matrix.
- void ConvertValues (Index nonzeros_triplet, const Number ∗a_triplet, Index nonzeros_compressed, Number ∗a↩_compressed)

  Convert the values of the nonzero elements.

Constructor/Destructor

- **TripletToCSRConverter** (Index offset, ETriFull hf=Triangular_Format)
- virtual ∼TripletToCSRConverter ()

  Destructor.

Accessor methods

- const Index ∗ IA () const

  Return the IA array for the condensed format.
- const Index ∗ JA () const

  Return the JA array for the condensed format.
- const Index ∗ **iPosFirst** () const

3.182.1 Detailed Description

Class for converting symmetric matrices given in triplet format to matrices in compressed sparse row (CSR) format of the upper triangular part (or, equivalently, compressed sparse column (CSC) format for the lower triangular part).

In the description for this class, we assume that we discuss the CSR format.

Definition at line 23 of file IpTripletToCSRConverter.hpp.

3.182.2 Member Enumeration Documentation

3.182.2.1 enum Ipopt::TripletToCSRConverter::ETriFull

Enum to specifiy half or full matrix storage.

Enumerator

Triangular_Format Lower (or Upper) triangular stored only.

Full_Format Store both lower and upper parts.

Definition at line 82 of file IpTripletToCSRConverter.hpp.

3.182.3 Member Function Documentation

3.182.3.1 Index Ipopt::TripletToCSRConverter::InitializeConverter (Index *dim,* Index *nonzeros,* const Index ∗ *airn,* const Index ∗ *ajcn*)

Initialize the converter, given the fixed structure of the matrix.

There, ndim gives the number of rows and columns of the matrix, nonzeros give the number of nonzero elements, and airn and acjn give the positions of the nonzero elements. The return value is the number of nonzeros in the condensed matrix. (Since nonzero elements can be listed several times in the triplet format, it is possible that this value is different from the input value nonzeros.) This method must be called before the GetIA, GetJA, Convert Values methods are called.

3.182.3.2 const Index∗ Ipopt::TripletToCSRConverter::IA ( ) const [inline]

Return the IA array for the condensed format.

Definition at line 118 of file IpTripletToCSRConverter.hpp.

3.182.3.3 const Index∗ Ipopt::TripletToCSRConverter::JA ( ) const [inline]

Return the JA array for the condensed format.

Definition at line 125 of file IpTripletToCSRConverter.hpp.

3.182.3.4 void Ipopt::TripletToCSRConverter::ConvertValues (Index *nonzeros_triplet,* const Number ∗ *a_triplet,* Index *nonzeros_compressed,* Number ∗ *a_compressed*)

Convert the values of the nonzero elements.

Given the values a_triplet for the triplet format, return the array of values for the condensed format in a_condensed. nonzeros_condensed is the length of the array a_condensed and must be identical to the return value of Initialize←Converter.

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

- IpTripletToCSRConverter.hpp

3.183 Ipopt::TSymDependencyDetector Class Reference

Base class for all derived algorithms for detecting linearly dependent rows in the constraint Jacobian.

`#include <IpTSymDependencyDetector.hpp>`

Inheritance diagram for Ipopt::TSymDependencyDetector:

Public Member Functions

- virtual bool InitializeImpl (const OptionsList &options, const std::string &prefix)

  Has to be called to initialize and reset these objects.
- virtual bool DetermineDependentRows (Index n_rows, Index n_cols, Index n_jac_nz, Number ∗jac_c_vals, Index ∗jac_c_iRow, Index ∗jac_c_jCol, std::list< Index > &c_deps)

  Method determining the number of linearly dependent rows in the matrix and the indices of those rows.

Constructor/Destructor

- **TSymDependencyDetector** (TSymLinearSolver &tsym_linear_solver)
- virtual ∼**TSymDependencyDetector** ()

Static Public Member Functions

- static void RegisterOptions (SmartPtr< RegisteredOptions > roptions)

  This must be called to make the options for this class known.

Additional Inherited Members

3.183.1 Detailed Description

Base class for all derived algorithms for detecting linearly dependent rows in the constraint Jacobian.

Definition at line 20 of file IpTSymDependencyDetector.hpp.

3.183.2 Member Function Documentation

3.183.2.1 virtual bool Ipopt::TSymDependencyDetector::InitializeImpl (const OptionsList & *options,* const std::string & *prefix*) [virtual]

Has to be called to initialize and reset these objects.

Implements Ipopt::TDependencyDetector.

3.183.2.2 virtual bool Ipopt::TSymDependencyDetector::DetermineDependentRows (Index *n_rows,* Index *n_cols,* Index *n_jac_nz,* Number ∗ *jac_c_vals,* Index ∗ *jac_c_iRow,* Index ∗ *jac_c_jCol,* std::list< Index > & *c_deps*) [virtual]

Method determining the number of linearly dependent rows in the matrix and the indices of those rows.

We assume that the matrix is available in "Triplet" format (MA28 format), and that the arrays given to this method can be modified internally, i.e., they are not used by the calling program anymore after this call. This method returns false if there was a problem with the underlying linear solver.

Implements Ipopt::TDependencyDetector.

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

- IpTSymDependencyDetector.hpp

3.184 Ipopt::TSymLinearSolver Class Reference

General driver for linear solvers for sparse indefinite symmetric matrices.

```
#include <IpTSymLinearSolver.hpp>
```

Inheritance diagram for Ipopt::TSymLinearSolver:

Public Member Functions

- bool InitializeImpl (const OptionsList &options, const std::string &prefix)

  overloaded from AlgorithmStrategyObject

Constructor/Destructor

- TSymLinearSolver (SmartPtr< SparseSymLinearSolverInterface > solver_interface, SmartPtr< TSym↩ ScalingMethod > scaling_method)

  Constructor.
- virtual ∼TSymLinearSolver ()

  Destructor.

Methods for requesting solution of the linear system.

- virtual ESymSolverStatus MultiSolve (const SymMatrix &A, std::vector< SmartPtr< const Vector > > &rhsV, std::vector< SmartPtr< Vector > > &solV, bool check_NegEVals, Index numberOfNegEVals)

  Solve operation for multiple right hand sides.
- virtual Index NumberOfNegEVals () const

  Number of negative eigenvalues detected during last factorization.

- virtual bool IncreaseQuality ()

  Request to increase quality of solution for next solve.
- virtual bool ProvidesInertia () const

  Query whether inertia is computed by linear solver.

Methods related to the detection of linearly dependent

rows in a matrix

- bool ProvidesDegeneracyDetection () const

  Returns true if the underlying linear solver can detect the linearly dependent rows in a matrix.
- ESymSolverStatus DetermineDependentRows (Index n_rows, Index n_cols, Index n_jac_nz, Number ∗jac_↩ c_vals, Index ∗jac_c_iRow, Index ∗jac_c_jCol, std::list< Index > &c_deps)

  Given the entries of a matrix in Triplet format, this method determines the list of row indices of the linearly dependent rows.

Static Public Member Functions

- static void RegisterOptions (SmartPtr< RegisteredOptions > roptions)

  Methods for OptionsList.

Additional Inherited Members

3.184.1 Detailed Description

General driver for linear solvers for sparse indefinite symmetric matrices.

This interface includes a call to a method for scaling of the matrix (if given). This class takes in the contructor a pointer to the interface to an actual linear solver, and possibly a pointer to a method for computing scaling factors. It translates the SymMatrix into the format required by the linear solver and calls the solver via the TSymLinearSolverInterface. If a scaling method has been given, the matrix, the right hand side, and the solution are scaled.

Definition at line 33 of file IpTSymLinearSolver.hpp.

3.184.2 Constructor & Destructor Documentation

3.184.2.1 Ipopt::TSymLinearSolver::TSymLinearSolver (**SmartPtr**< **SparseSymLinearSolverInterface** > *solver_interface*, **SmartPtr**< **TSymScalingMethod** > *scaling_method*)

Constructor.

The solver_interface is a pointer to a linear solver for symmetric matrices in triplet format. If scaling_method not NULL, it must be a pointer to a class for computing scaling factors for the matrix.

3.184.3 Member Function Documentation

3.184.3.1 virtual ESymSolverStatus Ipopt::TSymLinearSolver::MultiSolve (const **SymMatrix** & *A*, std::vector< **SmartPtr**< const **Vector** > > & *rhsV*, std::vector< **SmartPtr**< **Vector** > > & *solV*, bool *check_NegEVals*, Index *numberOfNegEVals*) [virtual]

Solve operation for multiple right hand sides.

For details see the description in the base class SymLinearSolver.

Implements Ipopt::SymLinearSolver.

3.184.3.2 virtual Index Ipopt::TSymLinearSolver::NumberOfNegEVals ( ) const [virtual]

Number of negative eigenvalues detected during last factorization.

Returns the number of negative eigenvalues of the most recent factorized matrix.

Implements Ipopt::SymLinearSolver.

3.184.3.3 virtual bool Ipopt::TSymLinearSolver::IncreaseQuality ( ) [virtual]

Request to increase quality of solution for next solve.

Ask linear solver to increase quality of solution for the next solve (e.g. increase pivot tolerance). Returns false, if this is not possible (e.g. maximal pivot tolerance already used.)

Implements Ipopt::SymLinearSolver.

3.184.3.4 virtual bool Ipopt::TSymLinearSolver::ProvidesInertia ( ) const [virtual]

Query whether inertia is computed by linear solver.

Returns true, if linear solver provides inertia.

Implements Ipopt::SymLinearSolver.

3.184.3.5 ESymSolverStatus Ipopt::TSymLinearSolver::DetermineDependentRows (Index *n_rows*, Index *n_cols*, Index *n_jac_nz*, Number ∗ *jac_c_vals*, Index ∗ *jac_c_iRow*, Index ∗ *jac_c_jCol*, std::list< Index > & *c_deps*)

Given the entries of a matrix in Triplet format, this method determines the list of row indices of the linearly dependent rows.

This is a specific implementation for Triplet matrices.

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

- IpTSymLinearSolver.hpp

3.185 Ipopt::TSymScalingMethod Class Reference

Base class for the method for computing scaling factors for symmetric matrices in triplet format.

`#include <IpTSymScalingMethod.hpp>`

Inheritance diagram for Ipopt::TSymScalingMethod:

Public Member Functions

- virtual bool InitializeImpl (const OptionsList &options, const std::string &prefix)=0

  overloaded from AlgorithmStrategyObject
- virtual bool ComputeSymTScalingFactors (Index n, Index nnz, const Index ∗airn, const Index ∗ajcn, const double ∗a, double ∗scaling_factors)=0

  Method for computing the symmetric scaling factors, given the symmtric matrix in triplet (MA27) format.

Constructor/Destructor

- **TSymScalingMethod** ()
- ∼**TSymScalingMethod** ()

Additional Inherited Members

3.185.1 Detailed Description

Base class for the method for computing scaling factors for symmetric matrices in triplet format.

Definition at line 23 of file IpTSymScalingMethod.hpp.

3.185.2 Member Function Documentation

3.185.2.1 virtual bool Ipopt::TSymScalingMethod::ComputeSymTScalingFactors (Index *n*, Index *nnz*, const Index ∗ *airn*, const Index ∗ *ajcn*, const double ∗ *a*, double ∗ *scaling_factors*) [pure virtual]

Method for computing the symmetric scaling factors, given the symmtric matrix in triplet (MA27) format.

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

- IpTSymScalingMethod.hpp

3.186 Ipopt::UserScaling Class Reference

This class does problem scaling by getting scaling parameters from the user (through the NLP interface).

`#include <IpUserScaling.hpp>`

Inheritance diagram for Ipopt::UserScaling:

Public Member Functions

Constructors/Destructors

- **UserScaling** (const SmartPtr< const NLP > &nlp)
- virtual ∼UserScaling ()

  Default destructor.

Protected Member Functions

- virtual void DeterminScalingParametersImpl (const SmartPtr< const VectorSpace > x_space, const SmartPtr< const VectorSpace > c_space, const SmartPtr< const VectorSpace > d_space, const SmartPtr< const MatrixSpace > jac_c_space, const SmartPtr< const MatrixSpace > jac_d_space, const SmartPtr< const SymMatrixSpace > h_space, const Matrix &Px_L, const Vector &x_L, const Matrix &Px_U, const Vector &x_U, Number &df, SmartPtr< Vector > &dx, SmartPtr< Vector > &dc, SmartPtr< Vector > &dd)

  This is the method that has to be overloaded by a particular scaling method that somehow computes the scaling vectors dx, dc, and dd.

Additional Inherited Members

3.186.1 Detailed Description

This class does problem scaling by getting scaling parameters from the user (through the NLP interface).

Definition at line 20 of file IpUserScaling.hpp.

3.186.2 Member Function Documentation

3.186.2.1 virtual void Ipopt::UserScaling::DetermineScalingParametersImpl (const SmartPtr< const VectorSpace > *x_space*, const SmartPtr< const VectorSpace > *c_space*, const SmartPtr< const VectorSpace > *d_space*, const SmartPtr< const MatrixSpace > *jac_c_space*, const SmartPtr< const MatrixSpace > *jac_d_space*, const SmartPtr< const SymMatrixSpace > *h_space*, const Matrix & *Px_L*, const Vector & *x_L*, const Matrix & *Px_U*, const Vector & *x_U*, Number & *df*, SmartPtr< Vector > & *dx*, SmartPtr< Vector > & *dc*, SmartPtr< Vector > & *dd*) [protected], [virtual]

This is the method that has to be overloaded by a particular scaling method that somehow computes the scaling vectors dx, dc, and dd.

The pointers to those vectors can be NULL, in which case no scaling for that item will be done later.

Implements Ipopt::StandardScalingBase.

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

- IpUserScaling.hpp

3.187 Ipopt::Vector Class Reference

Vector Base Class.

`#include <IpVector.hpp>`

Inheritance diagram for Ipopt::Vector:

Public Member Functions

- Vector ∗ MakeNew () const

  Create new Vector of the same type with uninitialized data.
- Vector ∗ MakeNewCopy () const

  Create new Vector of the same type and copy the data over.
- bool HasValidNumbers () const

  Method for determining if all stored numbers are valid (i.e., no Inf or Nan).

Constructor/Destructor

- Vector (const VectorSpace ∗owner_space)

  Constructor.
- virtual ∼Vector ()

  Destructor.

Standard BLAS-1 Operations

(derived classes do NOT overload these methods, instead, overload the protected versions of these methods).

- void Copy (const Vector &x)

  Copy the data of the vector x into this vector (DCOPY).
- void Scal (Number alpha)

  Scales the vector by scalar alpha (DSCAL)
- void Axpy (Number alpha, const Vector &x)

  Add the multiple alpha of vector x to this vector (DAXPY)
- Number Dot (const Vector &x) const

  Computes inner product of vector x with this (DDOT)

- Number Nrm2 () const

    Computes the 2-norm of this vector (DNRM2)
- Number Asum () const

    Computes the 1-norm of this vector (DASUM)
- Number Amax () const

    Computes the max-norm of this vector (based on IDAMAX)

Additional (Non-BLAS) Vector Methods

(derived classes do NOT overload these methods, instead, overload the protected versions of these methods).

- void Set (Number alpha)

    Set each element in the vector to the scalar alpha.
- void ElementWiseDivide (const Vector &x)

    Element-wise division $y_i \leftarrow y_i/x_i$.
- void ElementWiseMultiply (const Vector &x)

    Element-wise multiplication $y_i \leftarrow y_i * x_i$.
- void ElementWiseMax (const Vector &x)

    Element-wise max against entries in x.
- void ElementWiseMin (const Vector &x)

    Element-wise min against entries in x.
- void ElementWiseReciprocal ()

    Reciprocates the entries in the vector.
- void ElementWiseAbs ()

    Absolute values of the entries in the vector.
- void ElementWiseSqrt ()

    Element-wise square root of the entries in the vector.
- void ElementWiseSgn ()

    Replaces the vector values with their sgn values (-1 if x_i $<$ 0, 0 if x_i == 0, and 1 if x_i $>$ 0)
- void AddScalar (Number scalar)

    Add scalar to every vector component.
- Number Max () const

    Returns the maximum value in the vector.
- Number Min () const

    Returns the minimum value in the vector.
- Number Sum () const

    Returns the sum of the vector entries.
- Number SumLogs () const

    Returns the sum of the logs of each vector entry.

Methods for specialized operations. A prototype

implementation is provided, but for efficient implementation those should be specially implemented.

- void AddOneVector (Number a, const Vector &v1, Number c)

    Add one vector, y = a $*$ v1 + c $*$ y.
- void AddTwoVectors (Number a, const Vector &v1, Number b, const Vector &v2, Number c)

    Add two vectors, y = a $*$ v1 + b $*$ v2 + c $*$ y.
- Number FracToBound (const Vector &delta, Number tau) const

    Fraction to the boundary parameter.
- void AddVectorQuotient (Number a, const Vector &z, const Vector &s, Number c)

    Add the quotient of two vectors, y = a $*$ z/s + c $*$ y.

Accessor methods

- Index Dim () const

  Dimension of the Vector.
- SmartPtr< const VectorSpace > OwnerSpace () const

  Return the owner VectorSpace.

Output methods

(derived classes do NOT overload these methods, instead, overload the protected versions of these methods).

- void Print (SmartPtr< const Journalist > jnlst, EJournalLevel level, EJournalCategory category, const std↩ ::string &name, Index indent=0, const std::string &prefix="") const

  Print the entire vector.
- void **Print** (const Journalist &jnlst, EJournalLevel level, EJournalCategory category, const std::string &name, Index indent=0, const std::string &prefix="") const

Protected Member Functions

implementation methods (derived classes MUST

overload these pure virtual protected methods.)

- virtual void CopyImpl (const Vector &x)=0

  Copy the data of the vector x into this vector (DCOPY).
- virtual void ScalImpl (Number alpha)=0

  Scales the vector by scalar alpha (DSCAL)
- virtual void AxpyImpl (Number alpha, const Vector &x)=0

  Add the multiple alpha of vector x to this vector (DAXPY)
- virtual Number DotImpl (const Vector &x) const =0

  Computes inner product of vector x with this (DDOT)
- virtual Number Nrm2Impl () const =0

  Computes the 2-norm of this vector (DNRM2)
- virtual Number AsumImpl () const =0

  Computes the 1-norm of this vector (DASUM)
- virtual Number AmaxImpl () const =0

  Computes the max-norm of this vector (based on IDAMAX)
- virtual void SetImpl (Number alpha)=0

  Set each element in the vector to the scalar alpha.
- virtual void ElementWiseDivideImpl (const Vector &x)=0

  Element-wise division $y_i \leftarrow y_i/x_i$.
- virtual void ElementWiseMultiplyImpl (const Vector &x)=0

  Element-wise multiplication $y_i \leftarrow y_i * x_i$.
- virtual void ElementWiseMaxImpl (const Vector &x)=0

  Element-wise max against entries in x.
- virtual void ElementWiseMinImpl (const Vector &x)=0

  Element-wise min against entries in x.
- virtual void ElementWiseReciprocalImpl ()=0

  Reciprocates the elements of the vector.
- virtual void ElementWiseAbsImpl ()=0

  Take elementwise absolute values of the elements of the vector.
- virtual void ElementWiseSqrtImpl ()=0

  Take elementwise square-root of the elements of the vector.
- virtual void ElementWiseSgnImpl ()=0

  Replaces entries with sgn of the entry.

- virtual void AddScalarImpl (Number scalar)=0

  Add scalar to every component of vector.
- virtual Number MaxImpl () const =0

  Max value in the vector.
- virtual Number MinImpl () const =0

  Min number in the vector.
- virtual Number SumImpl () const =0

  Sum of entries in the vector.
- virtual Number SumLogsImpl () const =0

  Sum of logs of entries in the vector.
- virtual void AddTwoVectorsImpl (Number a, const Vector &v1, Number b, const Vector &v2, Number c)

  Add two vectors ($a * v1 + b * v2$).
- virtual Number FracToBoundImpl (const Vector &delta, Number tau) const

  Fraction to boundary parameter.
- virtual void AddVectorQuotientImpl (Number a, const Vector &z, const Vector &s, Number c)

  Add the quotient of two vectors.
- virtual bool HasValidNumbersImpl () const

  Method for determining if all stored numbers are valid (i.e., no Inf or Nan).
- virtual void PrintImpl (const Journalist &jnlst, EJournalLevel level, EJournalCategory category, const std::string &name, Index indent, const std::string &prefix) const =0

  Print the entire vector.

Additional Inherited Members

3.187.1 Detailed Description

Vector Base Class.

This is the base class for all derived vector types. Those vectors are meant to store entities like iterates, Lagrangian multipliers, constraint values etc. The implementation of a vector type depends on the computational environment (e.g. just a double array on a shared memory machine, or distributed double arrays for a distributed memory machine.)

Deriving from Vector: This class inherits from tagged object to implement an advanced caching scheme. Because of this, the TaggedObject method ObjectChanged() must be called each time the Vector changes. If you overload the X↩XXX_Impl protected methods, this taken care of (along with caching if possible) for you. If you have additional methods in your derived class that change the underlying data (vector values), you MUST remember to call ObjectChanged() AFTER making the change!

Definition at line 47 of file IpVector.hpp.

3.187.2 Constructor & Destructor Documentation

3.187.2.1 Ipopt::Vector::Vector (const VectorSpace * *owner_space*) [inline]

Constructor.

It has to be given a pointer to the corresponding VectorSpace.

Definition at line 445 of file IpVector.hpp.

3.187.3 Member Function Documentation

3.187.3.1 void Ipopt::Vector::Copy (const Vector & *x*) [inline]

Copy the data of the vector x into this vector (DCOPY).

Definition at line 478 of file IpVector.hpp.

3.187.3.2 void Ipopt::Vector::Set (Number *alpha*) [inline]

Set each element in the vector to the scalar alpha.

Definition at line 599 of file IpVector.hpp.

3.187.3.3 void Ipopt::Vector::AddOneVector (Number *a*, const Vector & *v1*, Number *c*) [inline]

Add one vector, y = a $*$ v1 + c $*$ y.

This is automatically reduced to call AddTwoVectors.

Definition at line 687 of file IpVector.hpp.

3.187.3.4 void Ipopt::Vector::AddTwoVectors (Number *a*, const Vector & *v1*, Number *b*, const Vector & *v2*, Number *c*) [inline]

Add two vectors, y = a $*$ v1 + b $*$ v2 + c $*$ y.

Here, this vector is y

Definition at line 693 of file IpVector.hpp.

3.187.3.5 Number Ipopt::Vector::FracToBound (const Vector & *delta*, Number *tau*) const [inline]

Fraction to the boundary parameter.

Computes $\alpha = \max\{\bar{\alpha} \in (0,1] : x + \bar{\alpha}\Delta \geq (1-\tau)x\}$

Definition at line 701 of file IpVector.hpp.

3.187.3.6 void Ipopt::Vector::AddVectorQuotient (Number *a*, const Vector & *z*, const Vector & *s*, Number *c*) [inline]

Add the quotient of two vectors, y = a $*$ z/s + c $*$ y.

Definition at line 721 of file IpVector.hpp.

3.187.3.7 bool Ipopt::Vector::HasValidNumbers ( ) const [inline]

Method for determining if all stored numbers are valid (i.e., no Inf or Nan).

Definition at line 729 of file IpVector.hpp.

3.187.3.8 virtual void Ipopt::Vector::CopyImpl (const Vector & *x*) [protected], [pure virtual]

Copy the data of the vector x into this vector (DCOPY).

Implemented in Ipopt::DenseVector, and Ipopt::CompoundVector.

3.187.3.9 virtual void Ipopt::Vector::SetImpl (Number *alpha*) [protected], [pure virtual]

Set each element in the vector to the scalar alpha.

Implemented in Ipopt::DenseVector, and Ipopt::CompoundVector.

3.187.3.10 virtual void Ipopt::Vector::AddTwoVectorsImpl (Number *a,* const Vector & *v1,* Number *b,* const Vector & *v2,* Number *c*) [protected], [virtual]

Add two vectors (a $*$ v1 + b $*$ v2).

Result is stored in this vector.

Reimplemented in Ipopt::DenseVector, and Ipopt::CompoundVector.

3.187.3.11 virtual Number Ipopt::Vector::FracToBoundImpl (const Vector & *delta,* Number *tau*) const [protected], [virtual]

Fraction to boundary parameter.

Reimplemented in Ipopt::DenseVector, and Ipopt::CompoundVector.

3.187.3.12 virtual bool Ipopt::Vector::HasValidNumbersImpl ( ) const [protected], [virtual]

Method for determining if all stored numbers are valid (i.e., no Inf or Nan).

A default implementation using Asum is provided.

Reimplemented in Ipopt::CompoundVector.

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

- IpVector.hpp

3.188 Ipopt::VectorSpace Class Reference

VectorSpace base class, corresponding to the Vector base class.

```
#include <IpVector.hpp>
```

Inheritance diagram for Ipopt::VectorSpace:

Public Member Functions

- virtual Vector $*$ MakeNew () const =0

  Pure virtual method for creating a new Vector of the corresponding type.
- Index Dim () const

  Accessor function for the dimension of the vectors of this type.

Constructors/Destructors

- VectorSpace (Index dim)

  Constructor, given the dimension of all vectors generated by this VectorSpace.
- virtual ∼VectorSpace ()

  Destructor.

3.188.1 Detailed Description

VectorSpace base class, corresponding to the Vector base class.

For each Vector implementation, a corresponding VectorSpace has to be implemented. A VectorSpace is able to create new Vectors of a specific type. The VectorSpace should also store information that is common to all Vectors of that type. For example, the dimension of a Vector is stored in the VectorSpace base class.

Definition at line 390 of file IpVector.hpp.

3.188.2 Member Function Documentation

3.188.2.1 Index Ipopt::VectorSpace::Dim ( ) const [inline]

Accessor function for the dimension of the vectors of this type.

Definition at line 411 of file IpVector.hpp.

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

- IpVector.hpp

3.189 Ipopt::WarmStartIterateInitializer Class Reference

Class implementing an initialization procedure for warm starts.

`#include <IpWarmStartIterateInitializer.hpp>`

Inheritance diagram for Ipopt::WarmStartIterateInitializer:

Public Member Functions

- virtual bool InitializeImpl (const OptionsList &options, const std::string &prefix)

  overloaded from AlgorithmStrategyObject
- virtual bool SetInitialIterates ()

  Compute the initial iterates and set the into the curr field of the ip_data object.

Constructors/Destructors

- WarmStartIterateInitializer ()

  Constructor.
- virtual ∼WarmStartIterateInitializer ()

  Default destructor.

Static Public Member Functions

- static void RegisterOptions (SmartPtr< RegisteredOptions > roptions)

  Methods used by IpoptType.

Additional Inherited Members

3.189.1 Detailed Description

Class implementing an initialization procedure for warm starts.

Definition at line 20 of file IpWarmStartIterateInitializer.hpp.

3.189.2 Constructor & Destructor Documentation

3.189.2.1 Ipopt::WarmStartIterateInitializer::WarmStartIterateInitializer ( )

Constructor.

3.189.3 Member Function Documentation

3.189.3.1 virtual bool Ipopt::WarmStartIterateInitializer::SetInitialIterates ( ) [virtual]

Compute the initial iterates and set the into the curr field of the ip_data object.

Implements Ipopt::IterateInitializer.

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

- IpWarmStartIterateInitializer.hpp

3.190 Ipopt::WsmpSolverInterface Class Reference

Interface to the linear solver Wsmp, derived from SparseSymLinearSolverInterface.

`#include <IpWsmpSolverInterface.hpp>`

Inheritance diagram for Ipopt::WsmpSolverInterface:

Public Member Functions

- bool InitializeImpl (const OptionsList &options, const std::string &prefix)

    overloaded from AlgorithmStrategyObject
- virtual bool ProvidesDegeneracyDetection () const

    Query whether the indices of linearly dependent rows/columns can be determined by this linear solver.
- virtual ESymSolverStatus DetermineDependentRows (const Index ∗ia, const Index ∗ja, std::list< Index > &c_↩ deps)

    This method determines the list of row indices of the linearly dependent rows.

Constructor/Destructor

- WsmpSolverInterface ()

    Constructor.
- virtual ∼WsmpSolverInterface ()

    Destructor.

Methods for requesting solution of the linear system.

- virtual ESymSolverStatus InitializeStructure (Index dim, Index nonzeros, const Index ∗ia, const Index ∗ja)

  Method for initializing internal stuctures.
- virtual double ∗ GetValuesArrayPtr ()

  Method returing an internal array into which the nonzero elements are to be stored.
- virtual ESymSolverStatus MultiSolve (bool new_matrix, const Index ∗ia, const Index ∗ja, Index nrhs, double ∗rhs_vals, bool check_NegEVals, Index numberOfNegEVals)

  Solve operation for multiple right hand sides.
- virtual Index NumberOfNegEVals () const

  Number of negative eigenvalues detected during last factorization.

- virtual bool IncreaseQuality ()

  Request to increase quality of solution for next solve.
- virtual bool ProvidesInertia () const

  Query whether inertia is computed by linear solver.
- EMatrixFormat MatrixFormat () const

  Query of requested matrix type that the linear solver understands.

Static Public Member Functions

- static void RegisterOptions (SmartPtr< RegisteredOptions > roptions)

  Methods for IpoptType.

Additional Inherited Members

3.190.1 Detailed Description

Interface to the linear solver Wsmp, derived from SparseSymLinearSolverInterface.

For details, see description of SparseSymLinearSolverInterface base class.

Definition at line 24 of file IpWsmpSolverInterface.hpp.

3.190.2 Member Function Documentation

3.190.2.1 virtual ESymSolverStatus Ipopt::WsmpSolverInterface::InitializeStructure (Index *dim,* Index *nonzeros,* const Index ∗ *ia,* const Index ∗ *ja*) [virtual]

Method for initializing internal stuctures.

Implements Ipopt::SparseSymLinearSolverInterface.

3.190.2.2 virtual double∗ Ipopt::WsmpSolverInterface::GetValuesArrayPtr ( ) [virtual]

Method returing an internal array into which the nonzero elements are to be stored.

Implements Ipopt::SparseSymLinearSolverInterface.

3.190.2.3 virtual ESymSolverStatus Ipopt::WsmpSolverInterface::MultiSolve (bool *new_matrix,* const Index ∗ *ia,* const Index ∗ *ja,* Index *nrhs,* double ∗ *rhs_vals,* bool *check_NegEVals,* Index *numberOfNegEVals*) [virtual]

Solve operation for multiple right hand sides.

Implements Ipopt::SparseSymLinearSolverInterface.

3.190.2.4 virtual bool Ipopt::WsmpSolverInterface::ProvidesInertia ( ) const [inline], [virtual]

Query whether inertia is computed by linear solver.

Returns true, if linear solver provides inertia.

Implements Ipopt::SparseSymLinearSolverInterface.

Definition at line 76 of file IpWsmpSolverInterface.hpp.

3.190.2.5 virtual bool Ipopt::WsmpSolverInterface::ProvidesDegeneracyDetection ( ) const [virtual]

Query whether the indices of linearly dependent rows/columns can be determined by this linear solver.

Reimplemented from Ipopt::SparseSymLinearSolverInterface.

3.190.2.6 virtual ESymSolverStatus Ipopt::WsmpSolverInterface::DetermineDependentRows (const Index ∗ *ia,* const Index ∗ *ja,* std::list< Index > & *c_deps*) [virtual]

This method determines the list of row indices of the linearly dependent rows.

Reimplemented from Ipopt::SparseSymLinearSolverInterface.

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

- IpWsmpSolverInterface.hpp

3.191 Ipopt::ZeroMatrix Class Reference

Class for Matrices with only zero entries.

`#include <IpZeroMatrix.hpp>`

Inheritance diagram for Ipopt::ZeroMatrix:

Public Member Functions

Constructors / Destructors

- ZeroMatrix (const MatrixSpace ∗owner_space)

  Constructor, taking the corresponding matrix space.
- ∼ZeroMatrix ()

  Destructor.

Protected Member Functions

Methods overloaded from matrix

- virtual void MultVectorImpl (Number alpha, const Vector &x, Number beta, Vector &y) const
    Matrix-vector multiply.
- virtual void TransMultVectorImpl (Number alpha, const Vector &x, Number beta, Vector &y) const
    Matrix(transpose) vector multiply.
- virtual void ComputeRowAMaxImpl (Vector &rows_norms, bool init) const
    Compute the max-norm of the rows in the matrix.
- virtual void ComputeColAMaxImpl (Vector &cols_norms, bool init) const
    Compute the max-norm of the columns in the matrix.
- virtual void PrintImpl (const Journalist &jnlst, EJournalLevel level, EJournalCategory category, const std::string &name, Index indent, const std::string &prefix) const
    Print detailed information about the matrix.

Additional Inherited Members

3.191.1 Detailed Description

Class for Matrices with only zero entries.

Definition at line 20 of file IpZeroMatrix.hpp.

3.191.2 Member Function Documentation

3.191.2.1 virtual void Ipopt::ZeroMatrix::MultVectorImpl (Number *alpha*, const Vector & *x*, Number *beta*, Vector & *y*) const [protected], [virtual]

Matrix-vector multiply.

Computes y = alpha ∗ Matrix ∗ x + beta ∗ y

Implements Ipopt::Matrix.

3.191.2.2 virtual void Ipopt::ZeroMatrix::TransMultVectorImpl (Number *alpha*, const Vector & *x*, Number *beta*, Vector & *y*) const [protected], [virtual]

Matrix(transpose) vector multiply.

Computes y = alpha ∗ Matrix^T ∗ x + beta ∗ y

Implements Ipopt::Matrix.

3.191.2.3 virtual void Ipopt::ZeroMatrix::ComputeRowAMaxImpl (Vector & *rows_norms*, bool *init*) const [inline], [protected], [virtual]

Compute the max-norm of the rows in the matrix.

The result is stored in rows_norms. The vector is assumed to be initialized.

Implements Ipopt::Matrix.

Definition at line 44 of file IpZeroMatrix.hpp.

3.191.2.4 virtual void Ipopt::ZeroMatrix::ComputeColAMaxImpl (Vector & *cols_norms*, bool *init*) const [inline], [protected], [virtual]

Compute the max-norm of the columns in the matrix.

The result is stored in cols_norms. The vector is assumed to be initialized.

Implements Ipopt::Matrix.

Definition at line 47 of file IpZeroMatrix.hpp.

3.191.2.5 virtual void Ipopt::ZeroMatrix::PrintImpl (const Journalist & *jnlst*, EJournalLevel *level*, EJournalCategory *category*, const std::string & *name*, Index *indent*, const std::string & *prefix*) const [protected], [virtual]

Print detailed information about the matrix.

Implements Ipopt::Matrix.

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

- IpZeroMatrix.hpp

3.192 Ipopt::ZeroMatrixSpace Class Reference

Class for matrix space for ZeroMatrix.

`#include <IpZeroMatrix.hpp>`

Inheritance diagram for Ipopt::ZeroMatrixSpace:

Public Member Functions

- virtual Matrix ∗ MakeNew () const

  Overloaded MakeNew method for the MatrixSpace base class.
- ZeroMatrix ∗ MakeNewZeroMatrix () const

  Method for creating a new matrix of this specific type.

Constructors / Destructors

- ZeroMatrixSpace (Index nrows, Index ncols)

  Constructor, given the number of row and columns.
- virtual ∼ZeroMatrixSpace ()

  Destructor.

3.192.1 Detailed Description

Class for matrix space for ZeroMatrix.

Definition at line 79 of file IpZeroMatrix.hpp.

3.192.2 Member Function Documentation

3.192.2.1 ZeroMatrix∗ Ipopt::ZeroMatrixSpace::MakeNewZeroMatrix ( ) const [inline]

Method for creating a new matrix of this specific type.

Definition at line 104 of file IpZeroMatrix.hpp.

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

- IpZeroMatrix.hpp

3.193 Ipopt::ZeroSymMatrix Class Reference

Class for Symmetric Matrices with only zero entries.

`#include <IpZeroSymMatrix.hpp>`

Inheritance diagram for Ipopt::ZeroSymMatrix:

Public Member Functions

Constructors / Destructors

- ZeroSymMatrix (const SymMatrixSpace ∗owner_space)

  Constructor, taking the corresponding matrix space.
- ∼ZeroSymMatrix ()

  Destructor.

Protected Member Functions

Methods overloaded from matrix

- virtual void MultVectorImpl (Number alpha, const Vector &x, Number beta, Vector &y) const

  Matrix-vector multiply.
- virtual void TransMultVectorImpl (Number alpha, const Vector &x, Number beta, Vector &y) const

  Since the matrix is symmetric, it is only necessary to implement the MultVectorImpl method in a class that inherits from this base class.
- virtual void ComputeRowAMaxImpl (Vector &rows_norms, bool init) const

  Compute the max-norm of the rows in the matrix.
- virtual void ComputeColAMaxImpl (Vector &cols_norms, bool init) const

  Since the matrix is symmetric, the row and column max norms are identical.
- virtual void PrintImpl (const Journalist &jnlst, EJournalLevel level, EJournalCategory category, const std::string &name, Index indent, const std::string &prefix) const

  Print detailed information about the matrix.

Additional Inherited Members

3.193.1 Detailed Description

Class for Symmetric Matrices with only zero entries.

Definition at line 20 of file IpZeroSymMatrix.hpp.

3.193.2 Member Function Documentation

3.193.2.1 virtual void Ipopt::ZeroSymMatrix::MultVectorImpl (Number *alpha*, const Vector & *x*, Number *beta*, Vector & *y*) const [protected], [virtual]

Matrix-vector multiply.

Computes y = alpha ∗ Matrix ∗ x + beta ∗ y

Implements Ipopt::Matrix.

3.193.2.2 virtual void Ipopt::ZeroSymMatrix::TransMultVectorImpl (Number *alpha*, const Vector & *x*, Number *beta*, Vector & *y*) const [protected], [virtual]

Since the matrix is symmetric, it is only necessary to implement the MultVectorImpl method in a class that inherits from this base class.

If the TransMultVectorImpl is called, this base class automatically calls MultVectorImpl instead.

Reimplemented from Ipopt::SymMatrix.

3.193.2.3 virtual void Ipopt::ZeroSymMatrix::ComputeRowAMaxImpl (Vector & *rows_norms*, bool *init*) const [inline], [protected], [virtual]

Compute the max-norm of the rows in the matrix.

The result is stored in rows_norms. The vector is assumed to be initialized.

Implements Ipopt::Matrix.

Definition at line 44 of file IpZeroSymMatrix.hpp.

3.193.2.4 virtual void Ipopt::ZeroSymMatrix::PrintImpl (const Journalist & *jnlst*, EJournalLevel *level*, EJournalCategory *category*, const std::string & *name*, Index *indent*, const std::string & *prefix*) const [protected], [virtual]

Print detailed information about the matrix.

Implements Ipopt::Matrix.

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

- IpZeroSymMatrix.hpp

3.194 Ipopt::ZeroSymMatrixSpace Class Reference

Class for matrix space for ZeroSymMatrix.

```
#include <IpZeroSymMatrix.hpp>
```

Inheritance diagram for Ipopt::ZeroSymMatrixSpace:

Public Member Functions

- virtual Matrix ∗ MakeNew () const

    Overloaded MakeNew method for the MatrixSpace base class.

- virtual SymMatrix * MakeNewSymMatrix () const

  Overloaded method from SymMatrixSpace base class.
- ZeroSymMatrix * MakeNewZeroSymMatrix () const

  Method for creating a new matrix of this specific type.

Constructors / Destructors

- ZeroSymMatrixSpace (Index dim)

  Constructor, given the number of row and columns.
- virtual ~ZeroSymMatrixSpace ()

  Destructor.

3.194.1 Detailed Description

Class for matrix space for ZeroSymMatrix.

Definition at line 79 of file IpZeroSymMatrix.hpp.

3.194.2 Member Function Documentation

3.194.2.1 ZeroSymMatrix* Ipopt::ZeroSymMatrixSpace::MakeNewZeroSymMatrix ( ) const [inline]

Method for creating a new matrix of this specific type.

Definition at line 111 of file IpZeroSymMatrix.hpp.

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

- IpZeroSymMatrix.hpp

Index