Heat conduction equation

Generated by Doxygen 1.8.11

Contents

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

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

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

Chapter 4

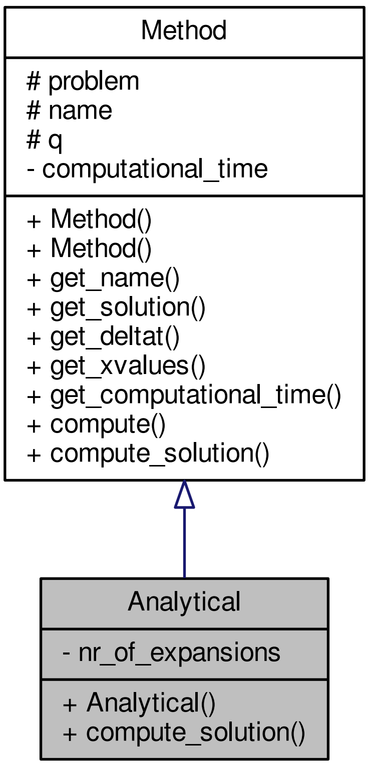
Class Documentation

4.1 Analytical Class Reference

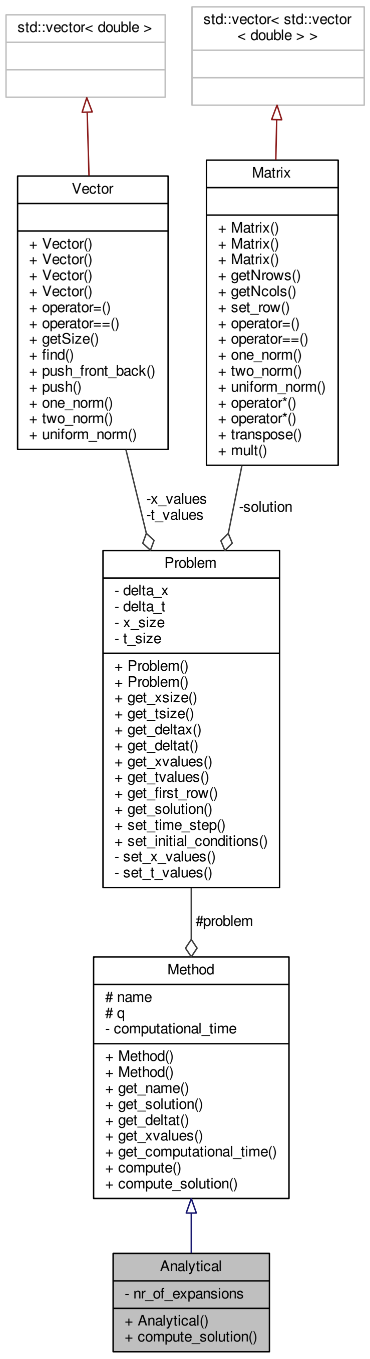
An Analytical class to compute the solution with standard procedures
The implementation is derived from the Method Object.

```
#include <analytical.h>
```

Inheritance diagram for Analytical:



Collaboration diagram for Analytical:



Public Member Functions

- Analytical (Problem problem)

  Default constructor.
- void compute_solution ()

  Normal public method.

Private Attributes

- unsigned int nr_of_expansions

  Private unsigned int nr_of_expansions.

Additional Inherited Members

4.1.1 Detailed Description

An Analytical class to compute the solution with standard procedures
The implementation is derived from the Method Object.

The Analytical class provides:
-a basic constructor for an object,
-a method to compute a solution with the correct procedures

4.1.2 Constructor & Destructor Documentation

4.1.2.1 Analytical::Analytical (**Problem** ***problem***)

Default constructor.

Intialize a Analytical object

4.1.3 Member Function Documentation

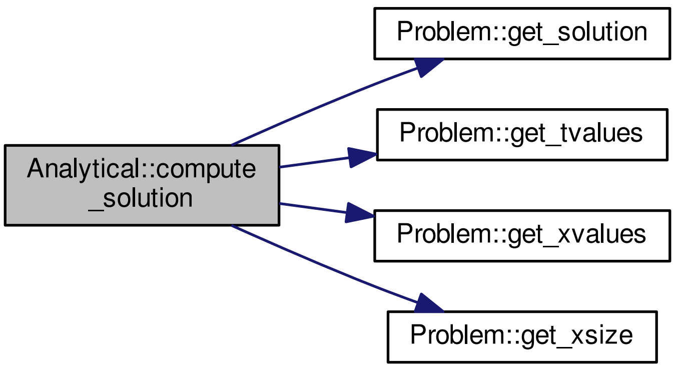
4.1.3.1 void Analytical::compute_solution ( ) `[virtual]`

Normal public method.

compute the solution with specific given rules

Implements Method.

Here is the call graph for this function:



4.1.4 Member Data Documentation

4.1.4.1 unsigned int Analytical::nr_of_expansions `[private]`

Private unsigned int nr_of_expansions.

Limit of expansions to do in the sum used to compute the solution.

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

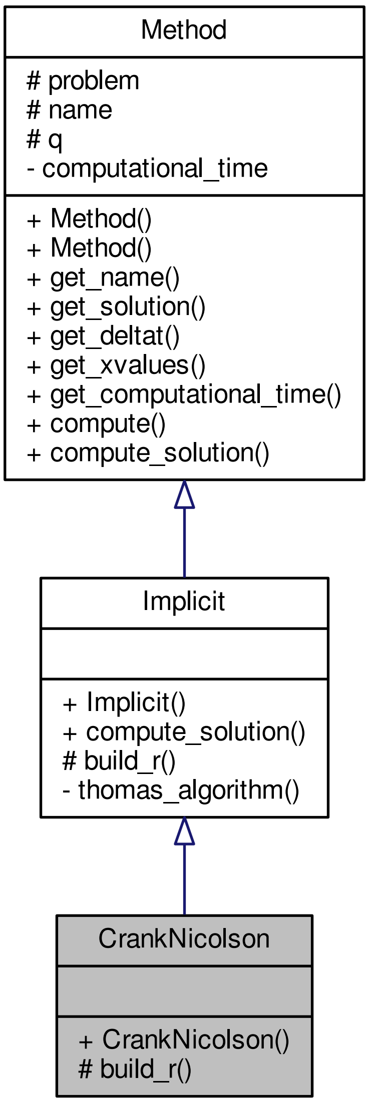
- methods/analytical.h
- methods/analytical.cpp

4.2 CrankNicolson Class Reference

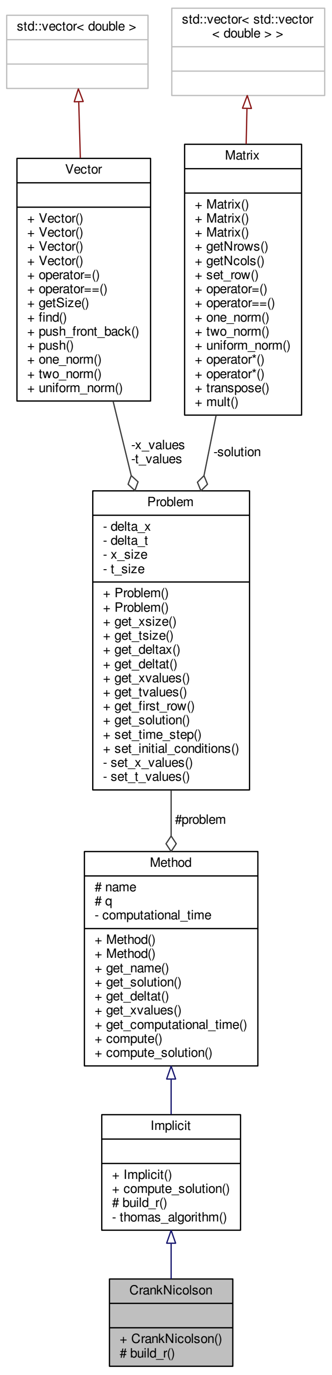
A CrankNicolson method class that contains a r vector builder.

`#include <crank_nicolson.h>`

Inheritance diagram for CrankNicolson:



Collaboration diagram for CrankNicolson:



Public Member Functions

- CrankNicolson (Problem problem)

    Default constructor.

Protected Member Functions

- Vector build_r (Vector previous_step)

  Normal protected method.

Additional Inherited Members

4.2.1 Detailed Description

A CrankNicolson method class that contains a r vector builder.

This builder is used to calculate the r vector in A.x = r linear equation system.

The CrankNicolson class provides:
-a basic constructor for creating a CrankNicolson method object.
-a method to compute the r vector.

4.2.2 Constructor & Destructor Documentation

4.2.2.1 CrankNicolson::CrankNicolson (**Problem** *problem*)

Default constructor.

4.2.3 Member Function Documentation

4.2.3.1 Vector CrankNicolson::build_r (**Vector** *previous_step*) `[protected]`, `[virtual]`

Normal protected method.

get the number of rows

Parameters

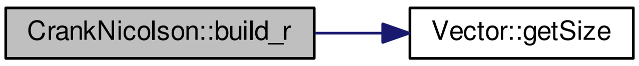
previous_step	Vector representing the solution of the previous time step.

Returns

Vector. r vector to be used in A.x = r

Implements Implicit.

Here is the call graph for this function:



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

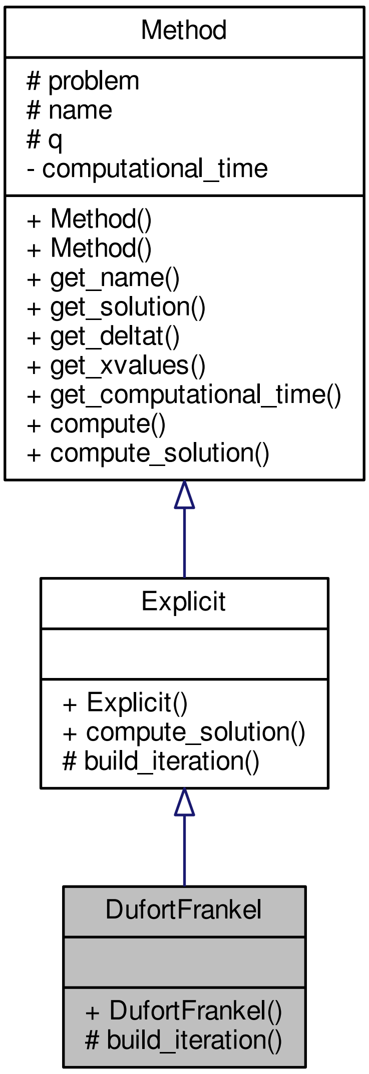
- methods/implicit/crank_nicolson.h
- methods/implicit/crank_nicolson.cpp

4.3 DufortFrankel Class Reference

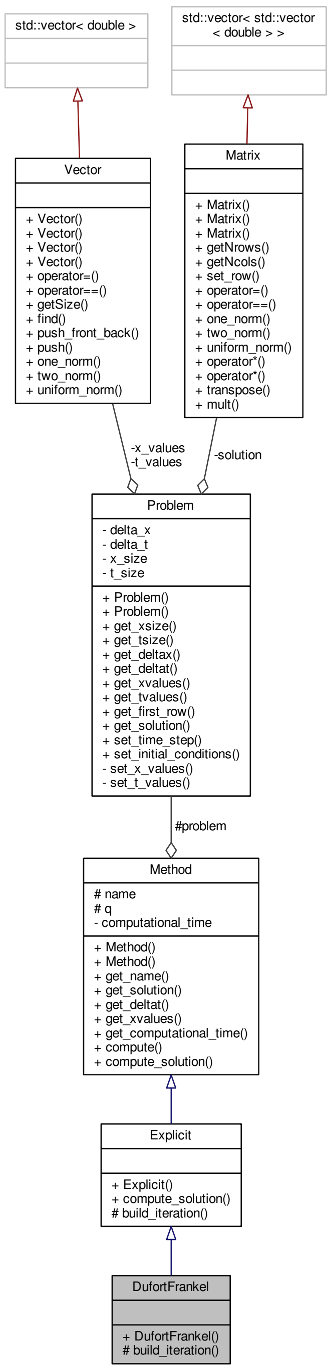
A DufortFrankel method class that contains an iteration builder.

```
#include <dufort_frankel.h>
```

Inheritance diagram for DufortFrankel:



Collaboration diagram for DufortFrankel:



Public Member Functions

- DufortFrankel (Problem problem)

  Default constructor.

Protected Member Functions

- Vector build_iteration (Vector current_step, Vector previous_step)

  Normal protected method.

Additional Inherited Members

4.3.1 Detailed Description

A DufortFrankel method class that contains an iteration builder.

This builder is used to calculate a solution using the Dufort-Frankel mathod.

The DufortFrankel class provides:
-a basic constructor for creating a DufortFrankel method object.
-a method to compute a solution of the current iteration

4.3.2 Constructor & Destructor Documentation

4.3.2.1 DufortFrankel::DufortFrankel (**Problem** *problem*)

Default constructor.

4.3.3 Member Function Documentation

4.3.3.1 Vector DufortFrankel::build_iteration (**Vector** *current_step,* **Vector** *previous_step*) `[protected]`, `[virtual]`

Normal protected method.

Calculate a next time step solution requiring a previous time step and a current time step solution.

Parameters

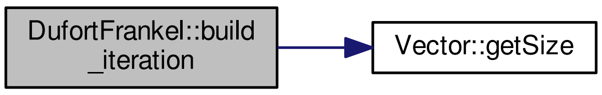
current_step	A vector representing the current time step solution.
previous_step	A vector representing the previous time step solution.

Returns

Vector. The computed solution.

Implements Explicit.

Here is the call graph for this function:



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

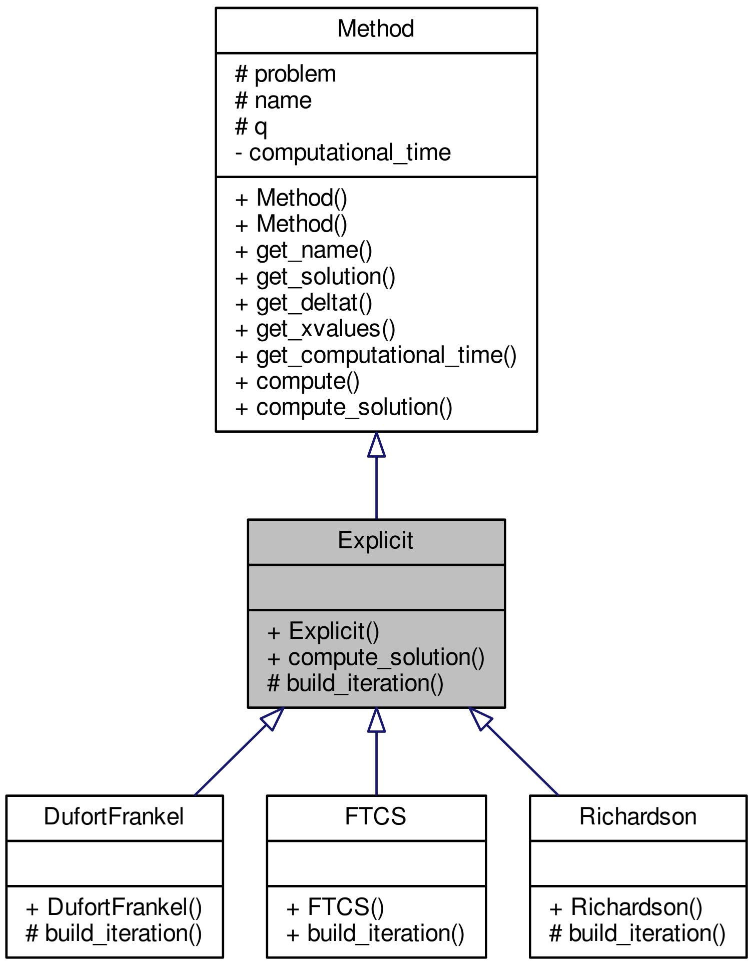
- methods/explicit/dufort_frankel.h
- methods/explicit/dufort_frankel.cpp

4.4 Explicit Class Reference

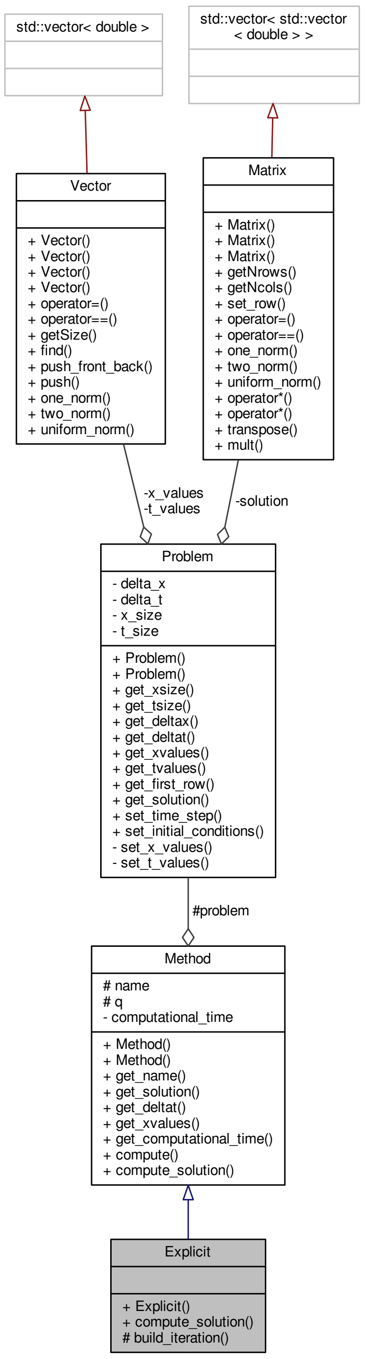
An explicit method class that contains default methods that only explicit methods use
The implementation is derived from the Method class.

```
#include <explicit.h>
```

Inheritance diagram for Explicit:



Collaboration diagram for Explicit:



Public Member Functions

- Explicit (Problem problem)

  Default constructor.
- void compute_solution ()

  Normal public method.

Protected Member Functions

- virtual Vector build_iteration (Vector current_step, Vector previous_step)=0

  A pure virtual member.

Additional Inherited Members

4.4.1 Detailed Description

An explicit method class that contains default methods that only explicit methods use
The implementation is derived from the Method class.

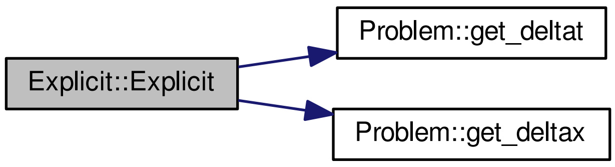
The Explicit class provides:
-a basic constructor for creating an explicit method object.
-a method to compute a solution following explicit methods rules

4.4.2 Constructor & Destructor Documentation

4.4.2.1 Explicit::Explicit (**Problem** ***problem***)

Default constructor.

Here is the call graph for this function:



4.4.3 Member Function Documentation

4.4.3.1 virtual Vector Explicit::build_iteration (**Vector** ***current_step,*** **Vector** ***previous_step***) `[protected]`, `[pure virtual]`

A pure virtual member.

Build the solution of the next time step, using the previous time step and the next time step solutions

Parameters

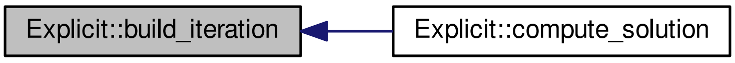
previous_step	A vector containing the previous time step solution.
current_step	A vector containing the current time step solution.

Returns

Vector. A vector representing the next time step solution.

Implemented in FTCS, DufortFrankel, and Richardson.

Here is the caller graph for this function:



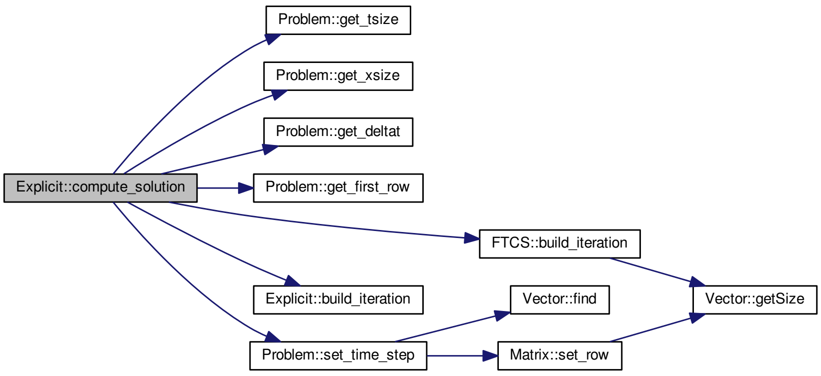
4.4.3.2 void Explicit::compute_solution ( ) [virtual]

Normal public method.

Calculates a solution for the given problem by populating the solution grid with the correct values.

Implements Method.

Here is the call graph for this function:



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

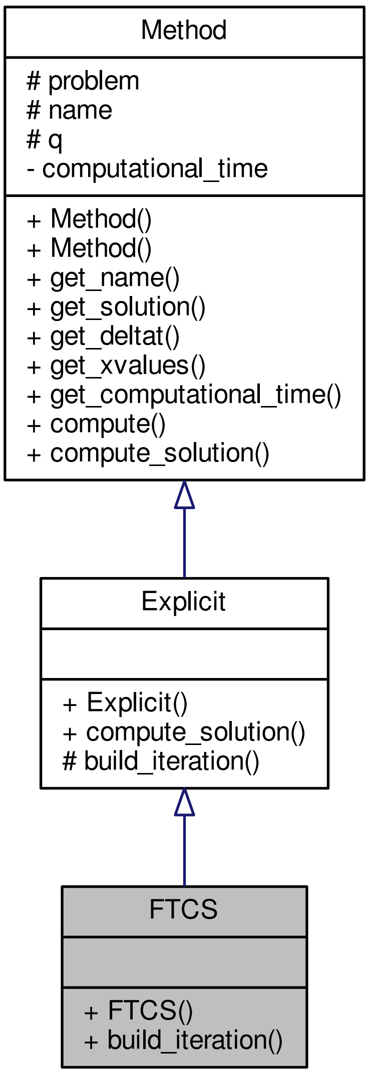
- methods/explicit/explicit.h
- methods/explicit/explicit.cpp

4.5 FTCS Class Reference

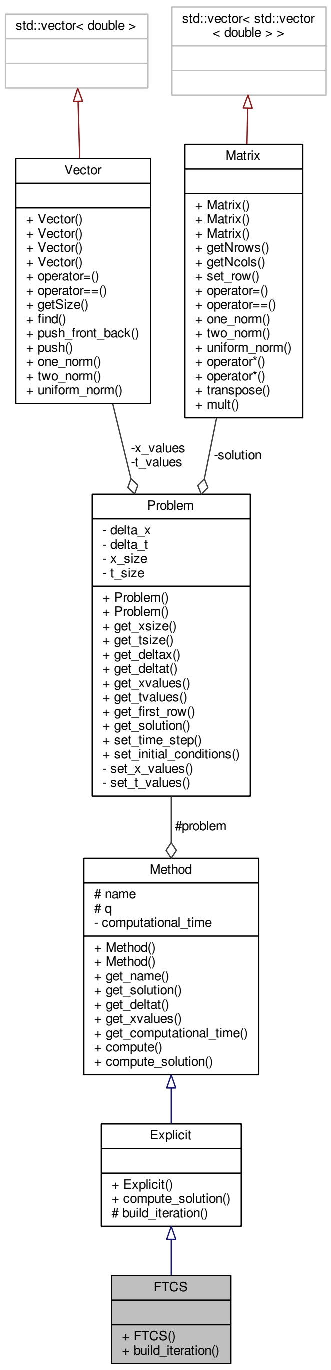
A FTCS method class that contains an iteration builder.

```
#include <forward_t_central_s.h>
```

Inheritance diagram for FTCS:



Collaboration diagram for FTCS:



Public Member Functions

- FTCS (Problem problem)

  Default constructor.
- Vector build_iteration (Vector current_step, Vector previous_step)

  Normal public method.

Additional Inherited Members

4.5.1 Detailed Description

A FTCS method class that contains an iteration builder.

This builder is used to calculate the first iteration of explicit methods, since it only requires the previous step solution to do it.

The FTCS class provides:
-a basic constructor for creating a FTCS method object.
-a method to compute the current iteration

4.5.2 Constructor & Destructor Documentation

4.5.2.1 FTCS::FTCS (**Problem** *problem*)

Default constructor.

4.5.3 Member Function Documentation

4.5.3.1 Vector FTCS::build_iteration (**Vector** *current_step*, **Vector** *previous_step*) `[virtual]`

Normal public method.

Calculate a solution requiring only the previous time step solution.

Parameters

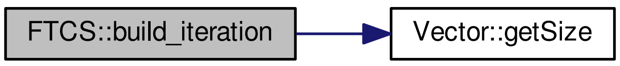
current_step	A vector with size 0, it's not required in this method.
previous_step	A vector representing the previous time step solution

Returns

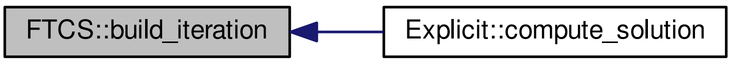
Vector. The computed solution.

Implements Explicit.

Here is the call graph for this function:



Here is the caller graph for this function:



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

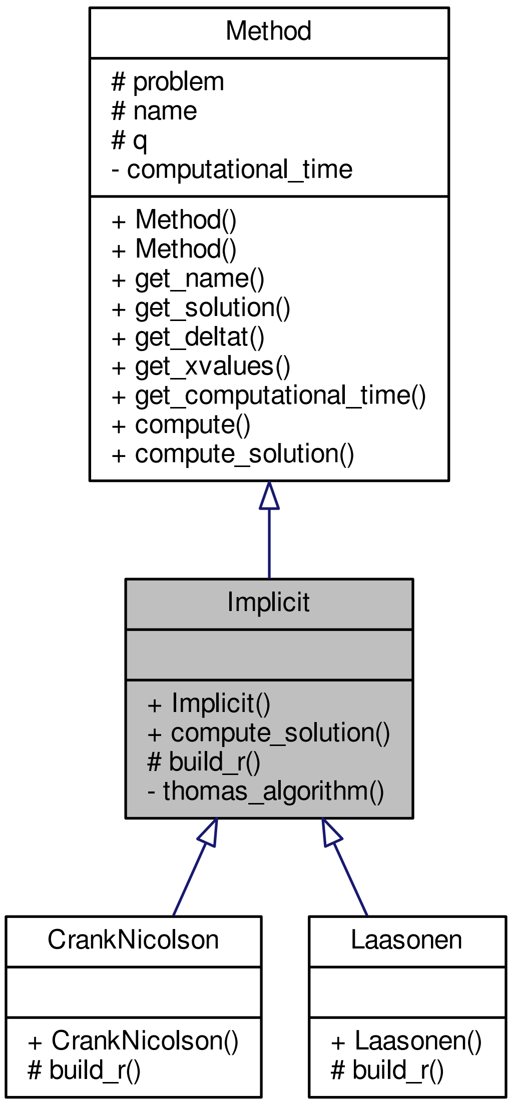
- methods/explicit/forward_t_central_s.h
- methods/explicit/forward_t_central_s.cpp

4.6 Implicit Class Reference

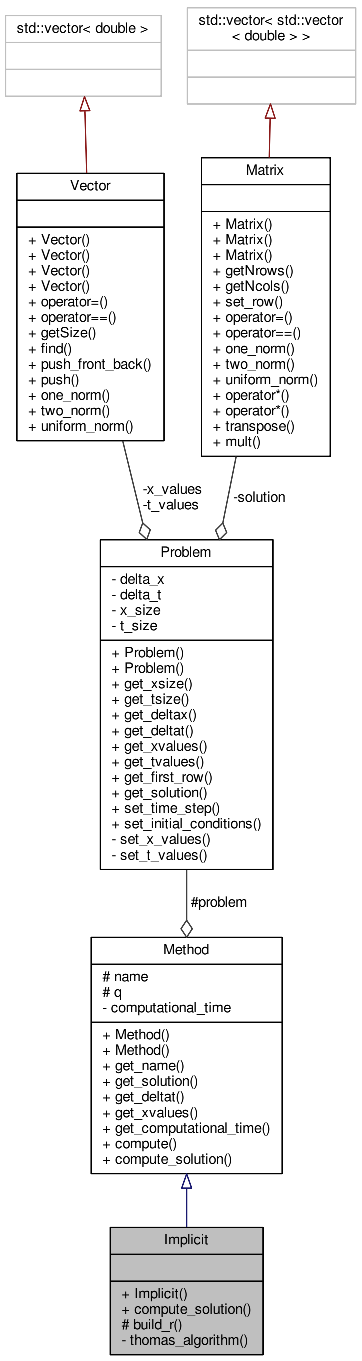
An implicit method class that contains default methods that only implicit methods use
The implementation is derived from the Method class.

```
#include <implicit.h>
```

Inheritance diagram for Implicit:



Collaboration diagram for Implicit:



Public Member Functions

- Implicit (Problem problem)

  Default constructor.
- void compute_solution ()

  Normal public method.

Protected Member Functions

- virtual Vector build_r (Vector previous_step)=0

  A pure virtual member.

Private Member Functions

- Vector thomas_algorithm (Vector r, double a, double b, double c)

  Normal private method.

Additional Inherited Members

4.6.1 Detailed Description

An implicit method class that contains default methods that only implicit methods use
The implementation is derived from the Method class.

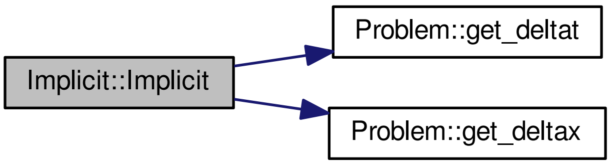
The Implicit class provides:
-a basic constructor for creating an implicit method object.
-a method to compute a solution following implicit methods rules

4.6.2 Constructor & Destructor Documentation

4.6.2.1 Implicit::Implicit (**Problem** *problem*)

Default constructor.

Here is the call graph for this function:



4.6.3 Member Function Documentation

4.6.3.1 virtual Vector Implicit::build_r (**Vector** *previous_step*) [protected], [pure virtual]

A pure virtual member.

Build the r vector in a linear system of A.x = r in which A is a matrix, whereas b and r are vectors.
This method is used to compute a solution using the thomas algorithm, which can be used in a triadiogonal matrix.

Parameters

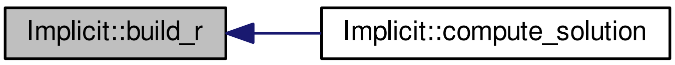
previous_step	A vector containing the previous time step solution.

Returns

Vector. The r vector, which can be used in to calculate the current time step solution with Tomas Algorithm.

Implemented in CrankNicolson, and Laasonen.

Here is the caller graph for this function:



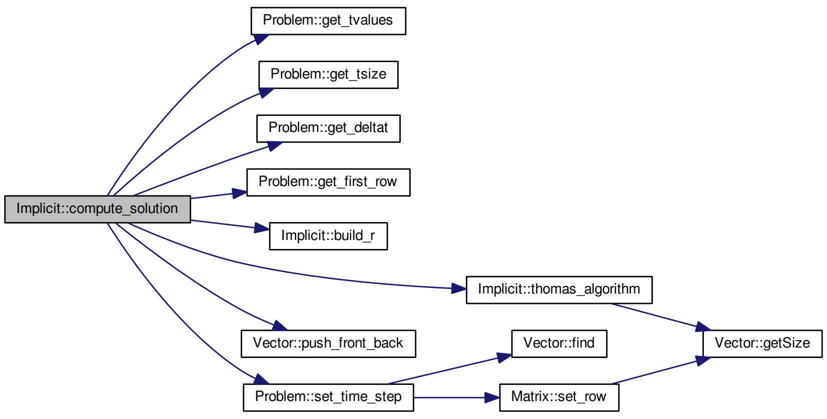
4.6.3.2 void Implicit::compute_solution ( ) [virtual]

Normal public method.

Calculates a solution for the given problem by populating the solution grid with the correct values.

Implements Method.

Here is the call graph for this function:



4.6.3.3 Vector Implicit::thomas_algorithm (Vector *r*, double *a*, double *b*, double *c*) [private]

Normal private method.

Calculates the current time step with Tomas Algorithm. Giving the A.x = r, in which A is a matrix, whereas b and r are vectors, it calculates the b vector, since A and b are known variables.

See also

build_r(Vector previous_step)

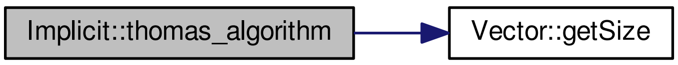
Parameters

r	Vector calculated by the build_r method.
a	Lower diagonal value of the tridiagonal matrix
b	Center diagonal value of the tridiagonal matrix
c	Upper diagonal value of the tridiagonal matrix

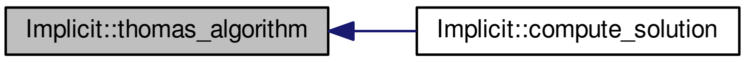
Returns

Vector. Vector that represents the current time step solution.

Here is the call graph for this function:



Here is the caller graph for this function:



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

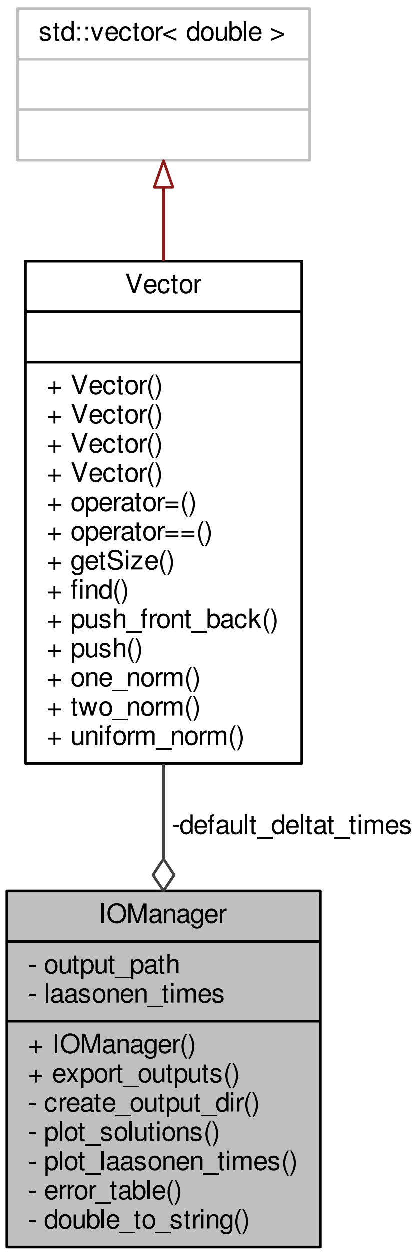
- methods/implicit/implicit.h
- methods/implicit/implicit.cpp

4.7 IOManager Class Reference

An input/output manager class to handle plot exportations and future implementations of input handling.

```
#include <iomanager.h>
```

Collaboration diagram for IOManager:



Public Member Functions

- IOManager ()

  Default constructor.
- void export_outputs (Method ∗analytical, std::vector< Method ∗ > methods)

  Exports outputs regarding plots images and error tables for each computed solution, comparing them to the analytical solution.

Private Member Functions

- bool create_output_dir ()

  Method to create ouput folder if the folder does not exist.
- void plot_solutions (std::string output_name, Method ∗analytical, Method ∗method)

  Exports a plot chart that compares the analytical solution to any other solution using gnuplot.
- void plot_laasonen_times ()

  Exports a plot with Laasonen times computational times.
- void error_table (std::string output_name, Method ∗analytical, Method ∗method)

  Exports an error table to a .lsx file that compares the analytical solution to any other solution.
- std::string double_to_string (int precision, double value)

  Converts a double to a string with a precison of 2 decimal places.

Private Attributes

- std::string output_path

  Private string output_path.
- std::vector< double > laasonen_times

  Private Vector laasonen_times.
- Vector default_deltat_times

  Private Vector default_deltat_times.

4.7.1 Detailed Description

An input/output manager class to handle plot exportations and future implementations of input handling.

The IOManager class provides:
-plot method which compares the analytical solution with a set of given methods, ploting them with a custom configuration using gnuplot

4.7.2 Constructor & Destructor Documentation

4.7.2.1 IOManager::IOManager ( )

Default constructor.

Initialize an IOManager object.

4.7.3 Member Function Documentation

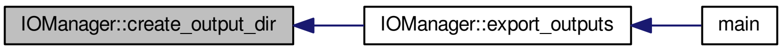
4.7.3.1 bool IOManager::create_output_dir ( ) [private]

Method to create ouput folder if the folder does not exist.

Returns

bool. true if successfull, false if not

Here is the caller graph for this function:



4.7.3.2 std::string IOManager::double_to_string (int *precision*, double *value*) [private]

Converts a double to a string with a precison of 2 decimal places.

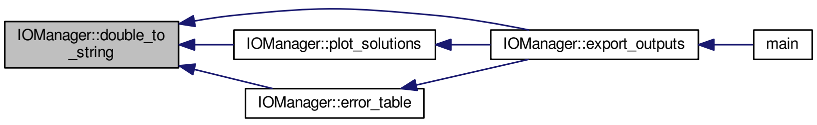
Parameters

double	value Number to be converted
int	precision Precision to have

Returns

string. String containing the converted number

Here is the caller graph for this function:


IOManager::double_to_string
IOManager::plot_solutions
IOManager::export_outputs
main
IOManager::error_table

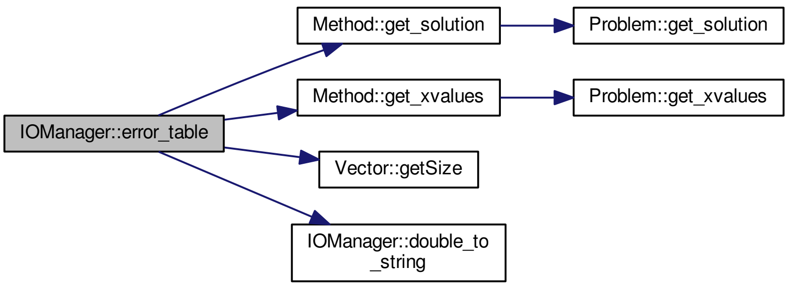
4.7.3.3 void IOManager::error_table (std::string *output_name*, Method ∗ *analytical*, Method ∗ *method*) [private]

Exports an error table to a .lsx file that compares the analytical solution to any other solution.

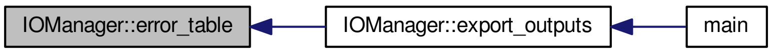
Parameters

string	output_name File name to be exported
Method∗	analytical The analytical solution
Method∗	method Any method solution

Here is the call graph for this function:



Here is the caller graph for this function:



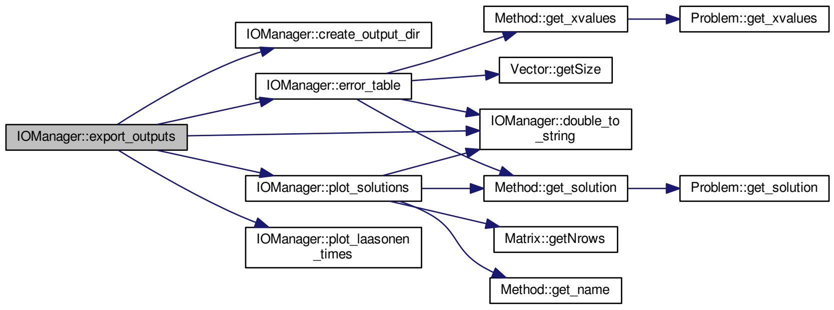
4.7.3.4 void IOManager::export_outputs (**Method** ∗ *analytical,* std::vector< **Method** ∗ > *methods*)

Exports outputs regarding plots images and error tables for each computed solution, comparing them to the analytical solution.

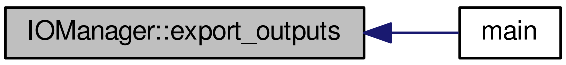
Parameters

Method∗	analytical The analytical solution
vector<Method∗>	methods Vector containing the solutions

Here is the call graph for this function:



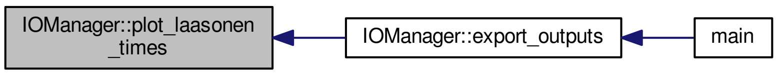
Here is the caller graph for this function:



4.7.3.5 void IOManager::plot_laasonen_times ( ) [private]

Exports a plot with Laasonen times computational times.

Here is the caller graph for this function:



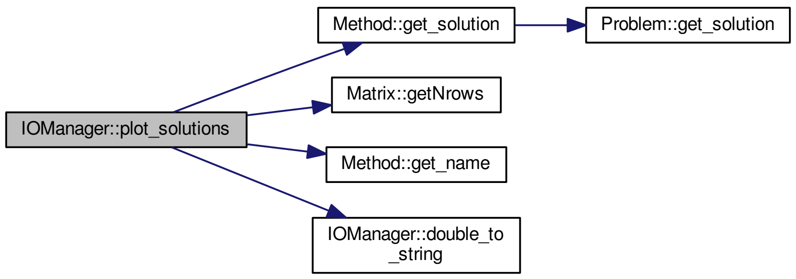
4.7.3.6 void IOManager::plot_solutions (std::string *output_name*, Method ∗ *analytical*, Method ∗ *method*) [private]

Exports a plot chart that compares the analytical solution to any other solution using gnuplot.

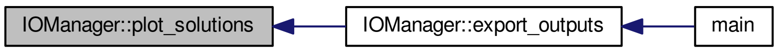
Parameters

string	output_name File name to be exported
Method∗	analytical The analytical solution
Method∗	method Any method solution

Here is the call graph for this function:



Here is the caller graph for this function:



4.7.4 Member Data Documentation

4.7.4.1 Vector IOManager::default_deltat_times [private]

Private Vector default_deltat_times.

Contains the computation time of each method solution, with a time step of 0.01.

4.7.4.2 std::vector<double> IOManager::laasonen_times [private]

Private Vector laasonen_times.

Contains the computation time of each laasonen solution, with a different time step.

4.7.4.3 std::string IOManager::output_path [private]

Private string output_path.

Contains the ouput directory path name.

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

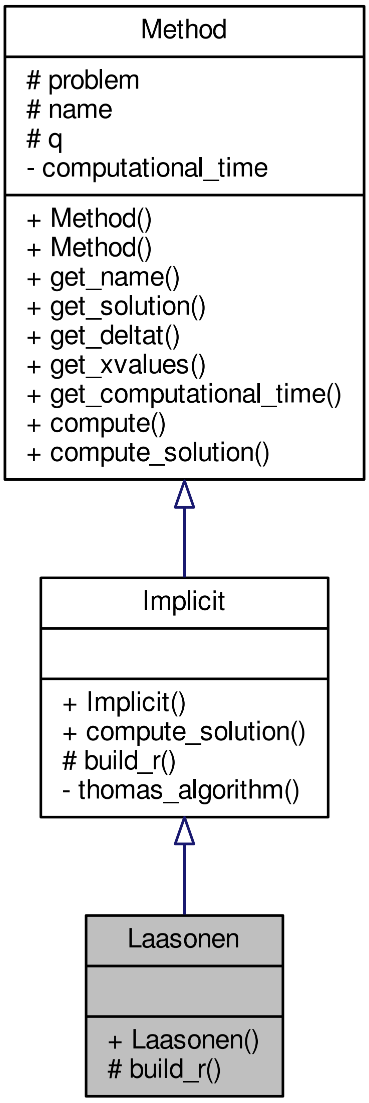
- io/iomanager.h
- io/iomanager.cpp

4.8 Laasonen Class Reference

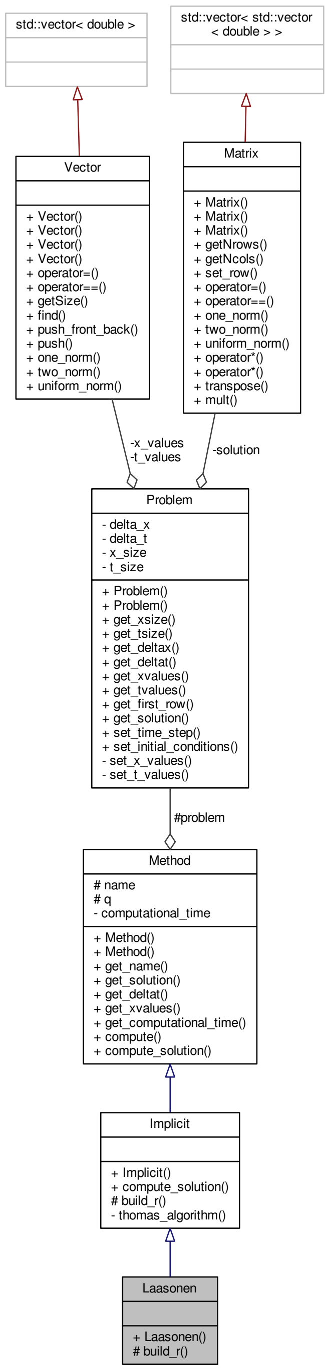
A Laasonen method class that contains a r vector builder.

```
#include <laasonen.h>
```

Inheritance diagram for Laasonen:



Collaboration diagram for Laasonen:



Public Member Functions

- Laasonen (Problem problem)

  Default constructor.

Protected Member Functions

- Vector build_r (Vector previous_step)

  Normal protected method.

Additional Inherited Members

4.8.1 Detailed Description

A Laasonen method class that contains a r vector builder.

This builder is used is used to calculate the r vector in A.x = r linear equation system.

The Laasonen class provides:
-a basic constructor for creating a Laasonen method object.
-a method to compute the r vector.

4.8.2 Constructor & Destructor Documentation

4.8.2.1 Laasonen::Laasonen (**Problem** ***problem***)

Default constructor.

4.8.3 Member Function Documentation

4.8.3.1 Vector Laasonen::build_r (**Vector** ***previous_step***) `[protected]`, `[virtual]`

Normal protected method.

get the number of rows

Parameters

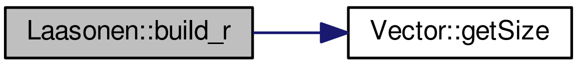
previous_step	Vector representing the solution of the previous time step.

Returns

Vector. r vector to be used in A.x = r

Implements Implicit.

Here is the call graph for this function:



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

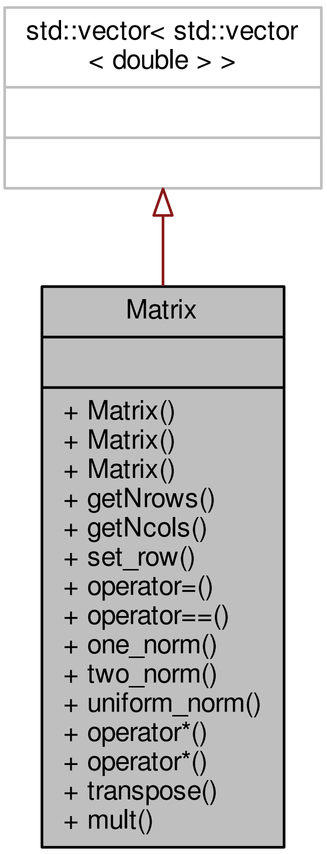
- methods/implicit/laasonen.h
- methods/implicit/laasonen.cpp

4.9 Matrix Class Reference

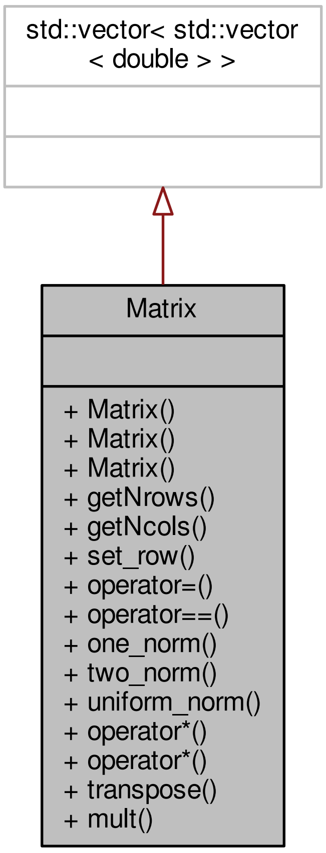
A matrix class for data storage of a 2D array of doubles
The implementation is derived from the standard container vector std::vector
We use private inheritance to base our vector upon the library version whilst usto expose only those base class functions we wish to use - in this the array access operator [].

```
#include <matrix.h>
```

Inheritance diagram for Matrix:



Collaboration diagram for Matrix:



Public Member Functions

- Matrix ()

  Default constructor.
- Matrix (int Nrows, int Ncols)

  Alternate constructor.
- Matrix (const Matrix &m)

  Copy constructor.
- int getNrows () const

  Normal public get method.
- int getNcols () const

  Normal public get method.
- void set_row (int index, Vector v)

  Normal public set method.
- Matrix & operator= (const Matrix &m)

  Overloaded assignment operator.
- bool operator== (const Matrix &m) const

Overloaded comparison operator returns true or false depending on whether the matrices are the same or not.

- double one_norm () const

  Normal public method that returns a double.
- double two_norm () const

  Normal public method that returns a double.
- double uniform_norm () const

  Normal public method that returns a double.
- Matrix operator$*$ (const Matrix &a) const

  Overloaded $*$operator that returns a Matrix.
- Vector operator$*$ (const Vector &v) const

  Overloaded $*$operator that returns a Vector.
- Matrix transpose () const

  public method that returns the transpose of the matrix.
- Matrix mult (const Matrix &a) const

Private Types

- typedef std::vector< std::vector< double > > vec

Friends

- std::istream & operator>> (std::istream &is, Matrix &m)

  Overloaded istream >> operator.
- std::ostream & operator<< (std::ostream &os, const Matrix &m)

  Overloaded ostream << operator.
- std::ifstream & operator>> (std::ifstream &ifs, Matrix &m)

  Overloaded ifstream >> operator.
- std::ofstream & operator<< (std::ofstream &ofs, const Matrix &m)

  Overloaded ofstream << operator.

4.9.1 Detailed Description

A matrix class for data storage of a 2D array of doubles
The implementation is derived from the standard container vector std::vector
We use private inheritance to base our vector upon the library version whilst usto expose only those base class functions we wish to use - in this the array access operator [].

The Matrix class provides:
-basic constructors for creating a matrix object from other matrix object, by creating empty matrix of a given size,
-input and oput operation via >> and << operators using keyboard or file
-basic operations like access via [] operator, assignment and comparision

4.9.2 Member Typedef Documentation

4.9.2.1 typedef std::vector<std::vector<double> > **Matrix::vec** `[private]`

4.9.3 Constructor & Destructor Documentation

4.9.3.1 Matrix::Matrix ( )

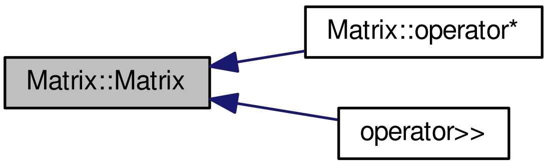
Default constructor.

Intialize an empty Matrix object

See also

Matrix(int Nrows, int Ncols)
Matrix(const Matrix& m)

Here is the caller graph for this function:



4.9.3.2 Matrix::Matrix (int *Nrows*, int *Ncols*)

Alternate constructor.

build a matrix Nrows by Ncols

See also

Matrix()
Matrix(const Matrix& m)

Exceptions

invalid_argument	("matrix size negative or zero")

Parameters

Nrows	int. number of rows in matrix
Ncols	int. number of columns in matrix

4.9.3.3 Matrix::Matrix (const **Matrix** & *m*)

Copy constructor.

build a matrix from another matrix

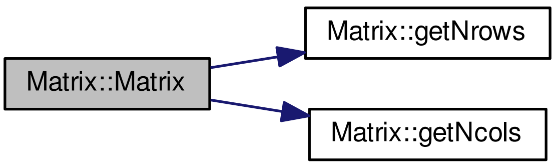
See also

Matrix()
Matrix(int Nrows, int Ncols)

Parameters

m	Matrix&. matrix to copy from

Here is the call graph for this function:



4.9.4 Member Function Documentation

4.9.4.1 int Matrix::getNcols ( ) const

Normal public get method.

get the number of columns

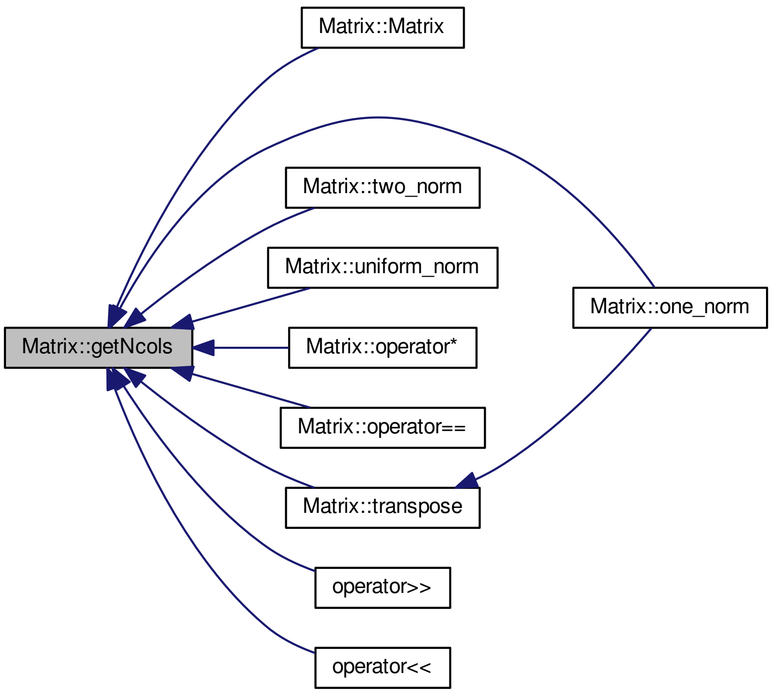
See also

int getNrows()const

Returns

int. number of columns in matrix

Here is the caller graph for this function:



4.9.4.2 int Matrix::getNrows ( ) const

Normal public get method.

get the number of rows

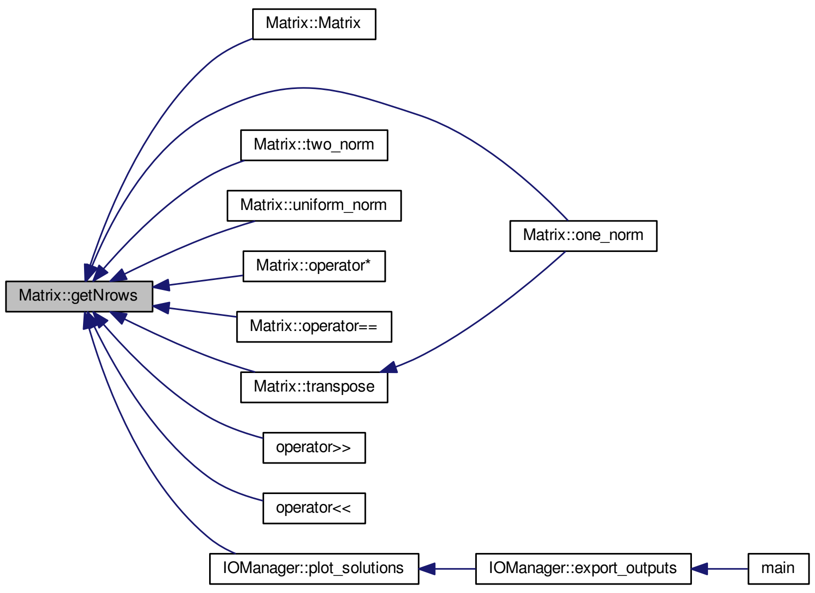
See also

int getNcols()const

Returns

int. number of rows in matrix

Here is the caller graph for this function:



4.9.4.3 Matrix Matrix::mult (const Matrix & *a*) const

4.9.4.4 double Matrix::one_norm ( ) const

Normal public method that returns a double.

It returns L1 norm of matrix

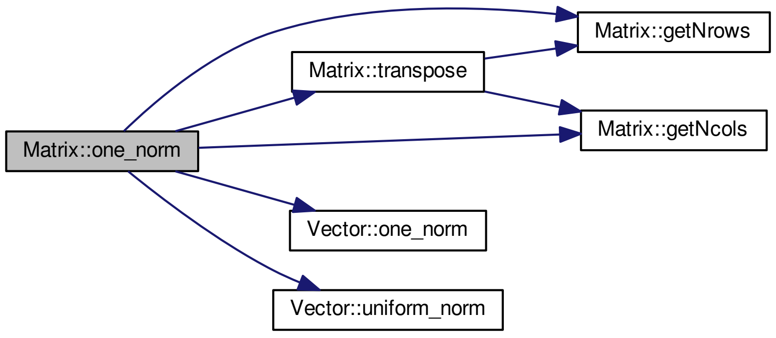
See also

two_norm()const
uniform_norm()const

Returns

double. matrix L1 norm

Here is the call graph for this function:



4.9.4.5 Matrix Matrix::operator∗ (const Matrix & *a*) const

Overloaded ∗operator that returns a Matrix.

It Performs matrix by matrix multiplication.

See also

operator∗(const Matrix & a) const

Exceptions

out_of_range	("Matrix access error") One or more of the matrix have a zero size
std::out_of_range	("uncompatible matrix sizes") Number of columns in first matrix do not match number of columns in second matrix

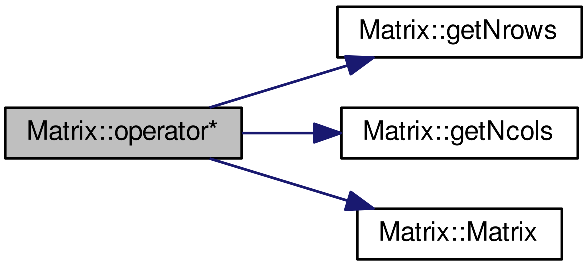
Returns

Matrix. matrix-matrix product

Parameters

a	Matrix. matrix to multiply by

Here is the call graph for this function:



4.9.4.6 Vector Matrix::operator∗ (const Vector & *v*) const

Overloaded ∗operator that returns a Vector.

It Performs matrix by vector multiplication.

See also

operator∗(const Matrix & a)const

Exceptions

std::out_of_range	("Matrix access error") matrix has a zero size
std::out_of_range	("Vector access error") vector has a zero size
std::out_of_range	("uncompatible matrix-vector sizes") Number of columns in matrix do not match the vector size

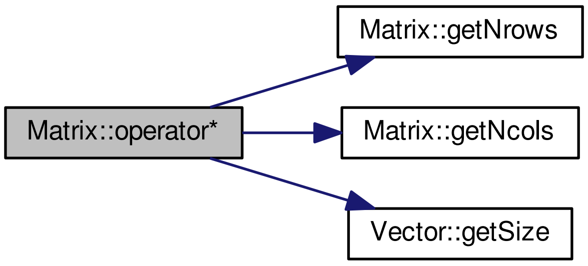
Returns

Vector. matrix-vector product

Parameters

v	Vector. Vector to multiply by

Here is the call graph for this function:



4.9.4.7 **Matrix** & Matrix::operator= (const **Matrix** & *m*)

Overloaded assignment operator.

See also

operator==(const Matrix& m)const

Returns

Matrix&. the matrix on the left of the assignment

Parameters

m	Matrix&. Matrix to assign from

4.9.4.8 bool Matrix::operator== (const **Matrix** & *m*) const

Overloaded comparison operator returns true or false depending on whether the matrices are the same or not.

See also

operator=(const Matrix& m)

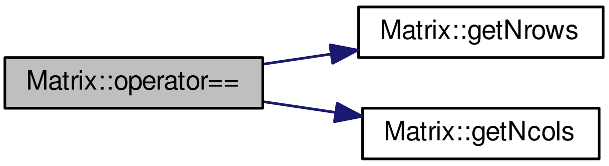
Returns

bool. true or false

Parameters

m	Matrix&. Matrix to compare to

Here is the call graph for this function:



4.9.4.9 void Matrix::set_row (int *index*, Vector *v*)

Normal public set method.

replace a row with a given vector

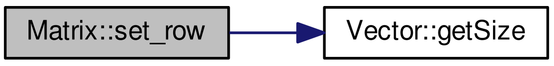
Parameters

index	Index of row to mutate
v	New vector

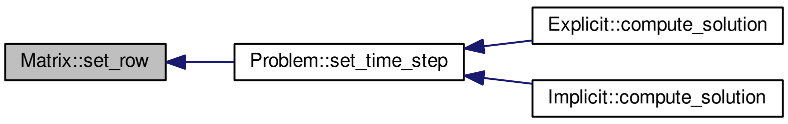
Exceptions

out_of_range	("index out of range.\n")
out_of_range	("vector size is different from matrix columns number.\n")

Here is the call graph for this function:



Here is the caller graph for this function:



4.9.4.10 Matrix Matrix::transpose ( ) const

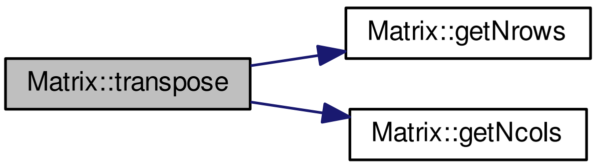
public method that returns the transpose of the matrix.

It returns the transpose of matrix

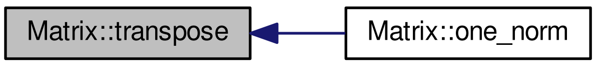
Returns

Matrix. matrix transpose

Here is the call graph for this function:



Here is the caller graph for this function:



4.9.4.11 double Matrix::two_norm ( ) const

Normal public method that returns a double.

It returns L2 norm of matrix

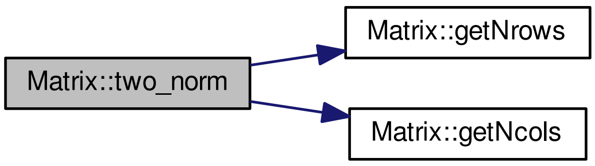
See also

one_norm()const
uniform_norm()const

Returns

double. matrix L2 norm

Here is the call graph for this function:



4.9.4.12 double Matrix::uniform_norm ( ) const

Normal public method that returns a double.

It returns L_max norm of matrix

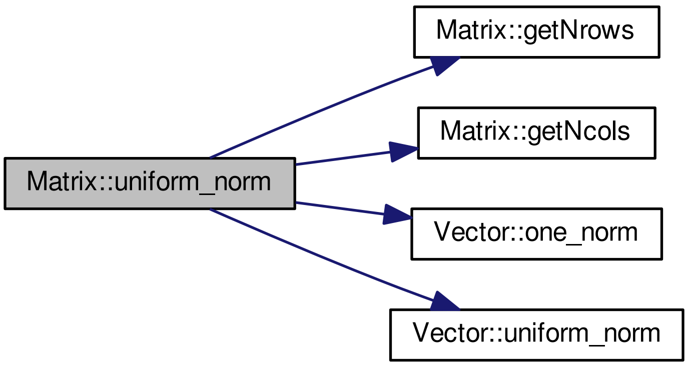
See also

one_norm()const
two_norm()const

Returns

double. matrix L_max norm

Here is the call graph for this function:



4.9.5 Friends And Related Function Documentation

4.9.5.1 std::ostream& operator<< (std::ostream & *os*, const **Matrix** & *m*) [friend]

Overloaded ostream << operator.

Display output if matrix has size user will be asked to input only matrix values if matrix was not initialized user can choose matrix size and input it values

See also

operator>>(std::ifstream& ifs, Matrix& m)
operator>>(std::istream& is, Matrix& m)
operator<<(std::ostream& os, const Matrix& m)

Returns

std::ostream&. The ostream object

Parameters

os	Display output stream
m	Matrix to read from

4.9.5.2 std::ofstream& operator<< (std::ofstream & *ofs*, const **Matrix** & *m*) `[friend]`

Overloaded ofstream << operator.

File output the file output operator is compatible with file input operator, ie. everything written can be read later.

See also

operator>>(std::ifstream& ifs, Matrix& m)
operator<<(std::ofstream& ofs, const Matrix& m)
operator>>(std::istream& is, Matrix& m)

Exceptions

std::invalid_argument	("file read error - negative matrix size");

Returns

std::ofstream&. The ofstream object

Parameters

m	Matrix to read from

4.9.5.3 std::istream& operator>> (std::istream & *is*, **Matrix** & *m*) `[friend]`

Overloaded istream >> operator.

Keyboard input if matrix has size user will be asked to input only matrix values if matrix was not initialized user can choose matrix size and input it values

See also

operator<<(std::ofstream& ofs, const Matrix& m)
operator>>(std::istream& is, Matrix& m)
operator<<(std::ostream& os, const Matrix& m)

Exceptions

std::invalid_argument	("read error - negative matrix size");

Returns

std::istream&. The istream object

Parameters

is	Keyboard input stream
m	Matrix to write into

4.9.5.4 std::ifstream& operator>> (std::ifstream & *ifs*, Matrix & *m*) [friend]

Overloaded ifstream >> operator.

File input the file output operator is compatible with file input operator, ie. everything written can be read later.

See also

operator>>(std::ifstream& ifs, Matrix& m)
operator<<(std::ofstream& ofs, const Matrix& m)
operator<<(std::ostream& os, const Matrix& m)

Returns

std::ifstream&. The ifstream object

Parameters

ifs	Input file stream with opened matrix file
m	Matrix to write into

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

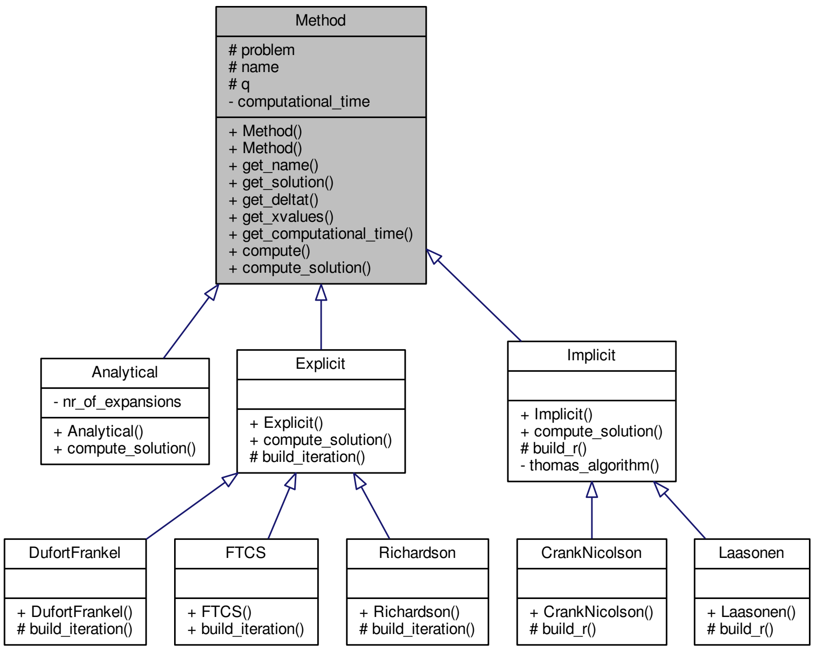
- grid/matrix.h
- grid/matrix.cpp

4.10 Method Class Reference

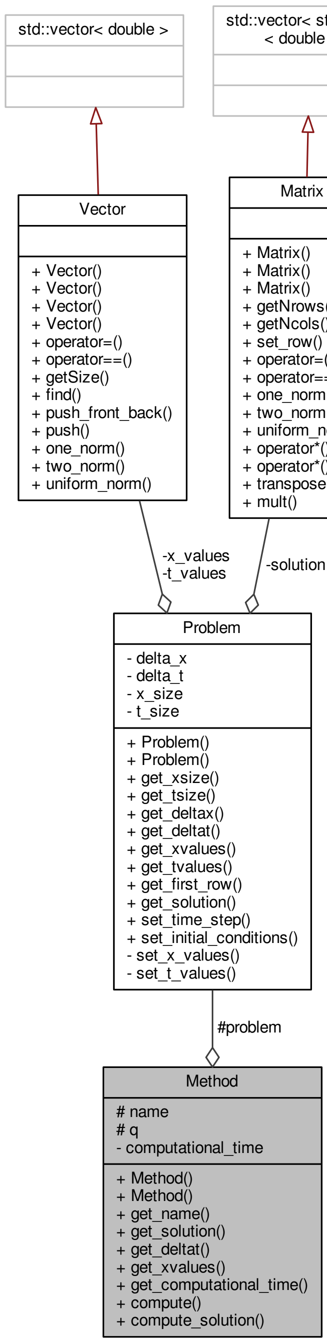
A Method class to structure information used to solve the problem.

```
#include <method.h>
```

Inheritance diagram for Method:



Collaboration diagram for Method:



Public Member Functions

- Method ()

    Default constructor.
- Method (Problem problem)

    Alternate constructor.
- std::string get_name ()

Normal public get method.
- Matrix get_solution ()

  Normal public get method.
- double get_deltat ()

  Normal public get method.
- Vector get_xvalues ()

  Normal public get method.
- double get_computational_time ()

  Normal public get method.
- void compute ()

  Normal public method.
- virtual void compute_solution ()=0

  A pure virtual member.

Protected Attributes

- Problem problem

  Protected Problem problem.
- std::string name

  Protected string name.
- double q

  Protected double q.

Private Attributes

- double computational_time

  Private double computational_time.

4.10.1 Detailed Description

A Method class to structure information used to solve the problem.

The Method class provides:
-basic constructors for creating a Method object.
-acessor methods to retrieve valuable information
-mutator methods to change the problem grid system

4.10.2 Constructor & Destructor Documentation

4.10.2.1 Method::Method ( )

Default constructor.

Intialize a Method object

See also

Method(Problem problem)

4.10.2.2 Method::Method (Problem *problem*)

Alternate constructor.

Initializes a Method with a given parabolic problem.

See also

Method()

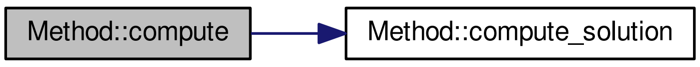
4.10.3 Member Function Documentation

4.10.3.1 void Method::compute ( )

Normal public method.

Keeps track of the time to compute a solution

Here is the call graph for this function:



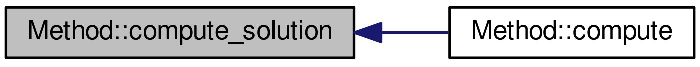
4.10.3.2 virtual void Method::compute_solution ( ) [pure virtual]

A pure virtual member.

compute the solution following the rules of a given method.

Implemented in Implicit, Explicit, and Analytical.

Here is the caller graph for this function:



4.10.3.3 double Method::get_computational_time ( )

Normal public get method.

get the elapsed time value to compute a solution

Returns

double. Elapsed time throughout the computation.

4.10.3.4 double Method::get_deltat ( )

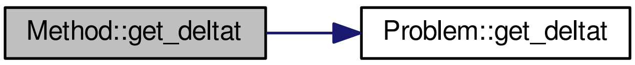
Normal public get method.

get the time step of the solution

Returns

double. Solution time step.

Here is the call graph for this function:



4.10.3.5 std::string Method::get_name ( )

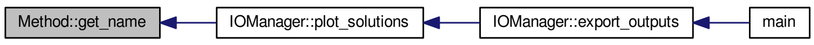
Normal public get method.

get the method name

Returns

string. Method name.

Here is the caller graph for this function:



4.10.3.6 Matrix Method::get_solution ( )

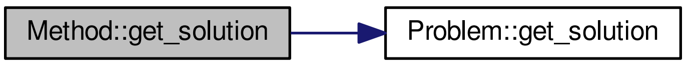
Normal public get method.

get the solution grid

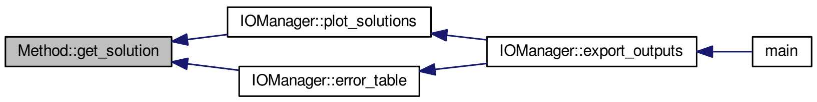
Returns

Matrix. Computed solution grid.

Here is the call graph for this function:



Here is the caller graph for this function:



4.10.3.7 Vector Method::get_xvalues ( )

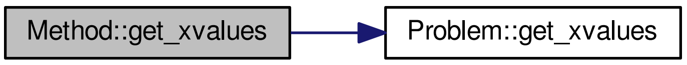
Normal public get method.

get x values vector

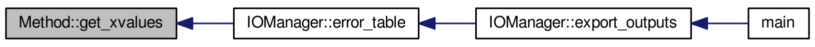
Returns

Vector. x values Vector.

Here is the call graph for this function:



Here is the caller graph for this function:



4.10.4 Member Data Documentation

4.10.4.1 double Method::computational_time [private]

Private double computational_time.

Elapsed time throughout the solution computation.

4.10.4.2 std::string Method::name [protected]

Protected string name.

Name of the method.

4.10.4.3 Problem Method::problem [protected]

Protected Problem problem.

Space step of the solution.

4.10.4.4 double Method::q [protected]

Protected double q.

A coeficient which value depends of way the equation is written, it may vary from method to method.

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

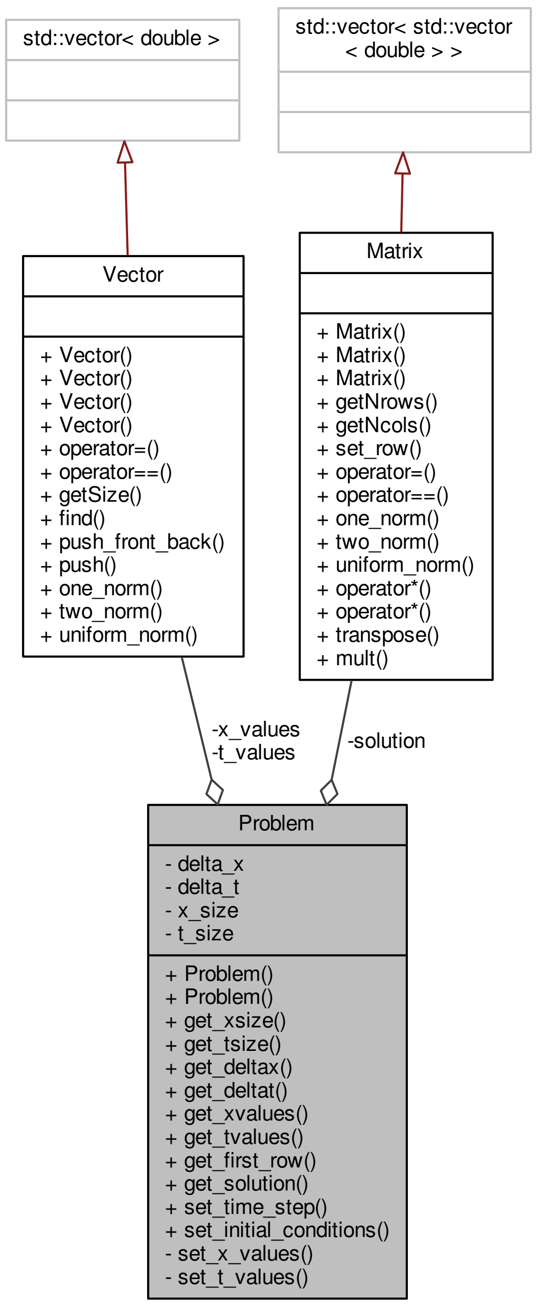
- methods/method.h
- methods/method.cpp

4.11 Problem Class Reference

A Problem class to structure relevant information related with the problem.

```
#include <problem.h>
```

Collaboration diagram for Problem:



Public Member Functions

- Problem ()

  Default constructor.
- Problem (double dt, double dx)

  Intialize Problem object with specific time and space steps.
- unsigned int get_xsize ()

  Normal public get method that returns an unsigned int, the number of columns of the solution.
- unsigned int get_tsize ()

  Normal public get method that returns an unsigned int, the number of rows of the solution.
- double get_deltax ()

  Normal public get method that returns a double, the space step value of the solution.
- double get_deltat ()

  Normal public get method that returns a double, the time step value of the solution.
- Vector get_xvalues ()

  Normal public get method that returns a Vector, containing the space values in each column.
- Vector get_tvalues ()

  Normal public get method that returns a Vector, containing the time values in each row.
- Vector get_first_row ()

  Normal public get method that returns a Vector, containing the initial boundaries in the first row of the solution.
- Matrix $*$ get_solution ()

  Normal public get method that returns a Matrix, containing the solution solution.
- void set_time_step (Vector step, double time)

  Normal public set method.
- void set_initial_conditions ()

  Normal public set method.

Private Member Functions

- void set_x_values ()

  Normal private set method.
- void set_t_values ()

  Normal private set method.

Private Attributes

- double delta_x

  Private double delta_x.
- double delta_t

  Private double delta_t.
- unsigned int x_size

  Private unsigned int x_size.
- unsigned int t_size

  Private unsigned int t_size.
- Vector x_values

  Private Vector x_values.
- Vector t_values

  Private Vector t_values.
- Matrix solution

  Private Matrix solution.

4.11.1 Detailed Description

A Problem class to structure relevant information related with the problem.

The Problem class provides:
-basic constructors for creating a Problem object.
-acessor methods to retrieve valuable information
-mutator methods to change the solution system

4.11.2 Constructor & Destructor Documentation

4.11.2.1 Problem::Problem ( )

Default constructor.

Intialize an empty Problem object

See also

Problem(double dt, double dx)

4.11.2.2 Problem::Problem (double *dt*, double *dx*)

Intialize Problem object with specific time and space steps.

See also

Problem()

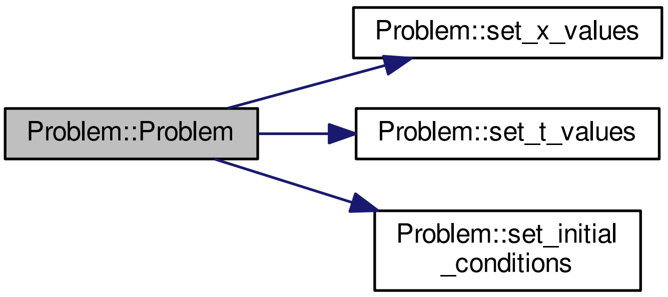
Parameters

dt	Time step to assign
dx	Space step to assign

Exceptions

out_of_range	("space step can't be negative or zero")
out_of_range	("time step can't be negative or zero")

Here is the call graph for this function:



4.11.3 Member Function Documentation

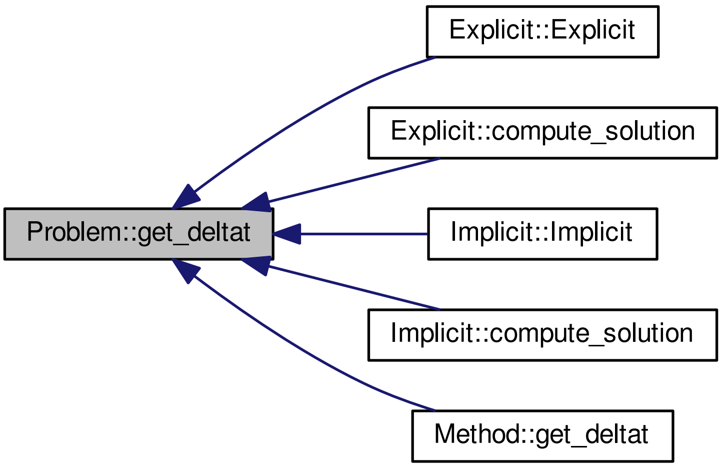
4.11.3.1 double Problem::get_deltat ( )

Normal public get method that returns a double, the time step value of the solution.

Returns

double. The time step value of the solution.

Here is the caller graph for this function:



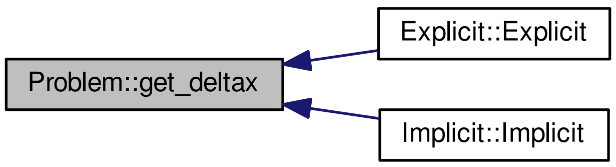
4.11.3.2 double Problem::get_deltax ( )

Normal public get method that returns a double, the space step value of the solution.

Returns

double. The space step value of the solution.

Here is the caller graph for this function:



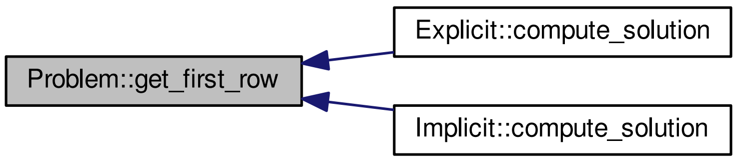
4.11.3.3 Vector Problem::get_first_row ( )

Normal public get method that returns a Vector, containing the initial boundaries in the first row of the solution.

Returns

Vector. The initial boundaries in the first row of the solution.

Here is the caller graph for this function:



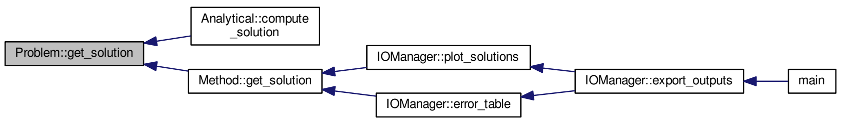
4.11.3.4 Matrix ∗ Problem::get_solution ( )

Normal public get method that returns a Matrix, containing the solution solution.

Returns

Matrix∗. The solution solution.

Here is the caller graph for this function:


Problem::get_solution
Analytical::compute _solution
Method::get_solution
IOManager::plot_solutions
IOManager::error_table
IOManager::export_outputs
main

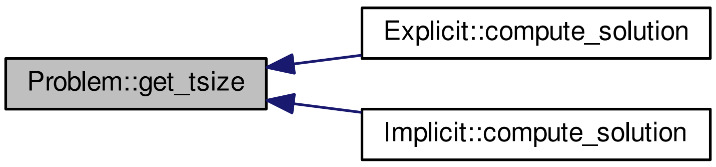
4.11.3.5 unsigned int Problem::get_tsize ( )

Normal public get method that returns an unsigned int, the number of rows of the solution.

Returns

unsigned int. The number of rows of the solution.

Here is the caller graph for this function:



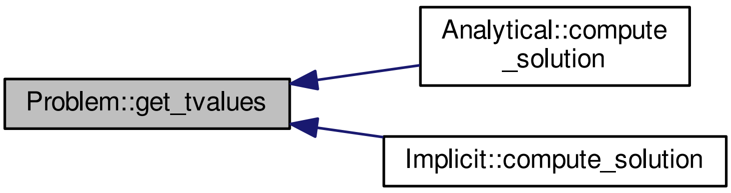
4.11.3.6 Vector Problem::get_tvalues ( )

Normal public get method that returns a Vector, containing the time values in each row.

Returns

Vector. The time values in each row.

Here is the caller graph for this function:



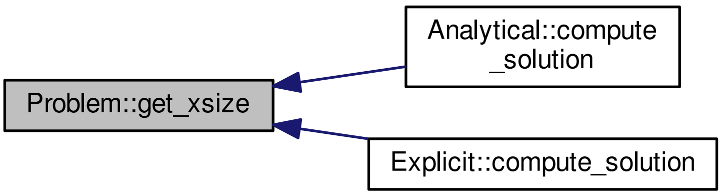
4.11.3.7 unsigned int Problem::get_xsize ( )

Normal public get method that returns an unsigned int, the number of columns of the solution.

Returns

unsigned int. The number of columns of the solution.

Here is the caller graph for this function:



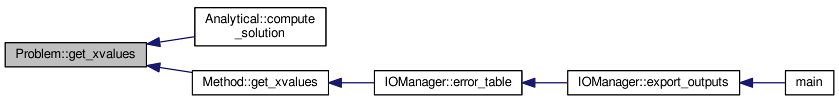
4.11.3.8 Vector Problem::get_xvalues ( )

Normal public get method that returns a Vector, containing the space values in each column.

Returns

Vector. The space values in each column.

Here is the caller graph for this function:

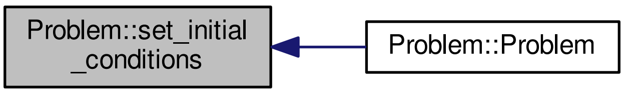


4.11.3.9 void Problem::set_initial_conditions ( )

Normal public set method.

set the problem initial boundaries.

Here is the caller graph for this function:



4.11.3.10 void Problem::set_t_values ( ) [private]

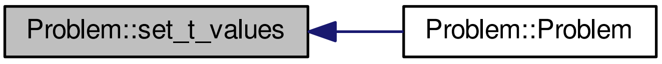
Normal private set method.

Intialize Vector t_values with the correct values.

See also

t_values

Here is the caller graph for this function:



4.11.3.11 void Problem::set_time_step (Vector *step*, double *time*)

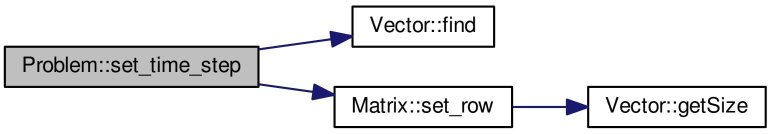
Normal public set method.

replace a row of the solution for a given Vector.

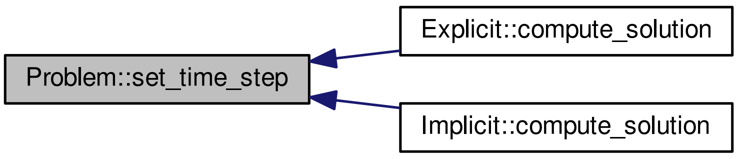
Parameters

step	Vector conatining the new values.
time	Corresponding row to be replaced

Here is the call graph for this function:



Here is the caller graph for this function:



4.11.3.12 void Problem::set_x_values ( ) [private]

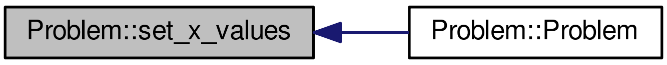
Normal private set method.

Intialize Vector x_values with the correct values.

See also

x_values

Here is the caller graph for this function:



4.11.4 Member Data Documentation

4.11.4.1 double Problem::delta_t [private]

Private double delta_t.

Time step of the solution.

4.11.4.2 double Problem::delta_x [private]

Private double delta_x.

Space step of the solution.

4.11.4.3 Matrix Problem::solution [private]

Private Matrix solution.

Matrix containing the computed solution.

4.11.4.4 unsigned int Problem::t_size [private]

Private unsigned int t_size.

Time size of the solution.

4.11.4.5 Vector Problem::t_values [private]

Private Vector t_values.

Time correspondent value for each row index.

4.11.4.6 unsigned int Problem::x_size [private]

Private unsigned int x_size.

Space size of the solution.

4.11.4.7 Vector Problem::x_values [private]

Private Vector x_values.

Space correspondent value for each column index.

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

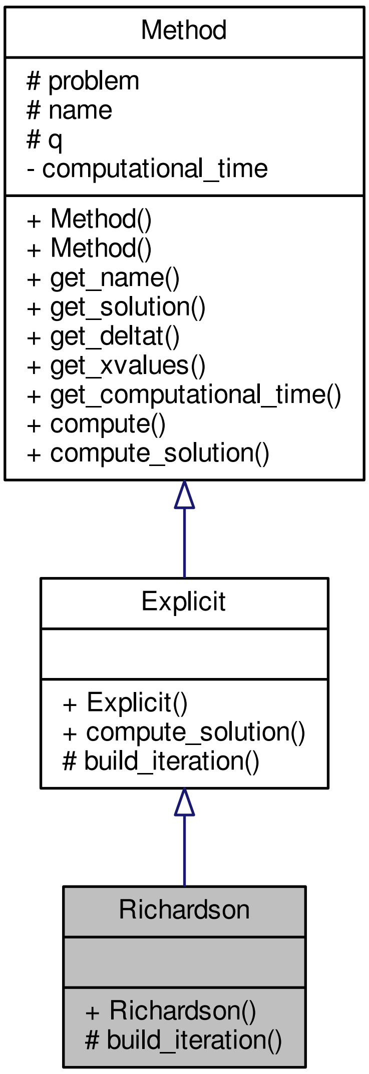
- variants/problem.h
- variants/problem.cpp

4.12 Richardson Class Reference

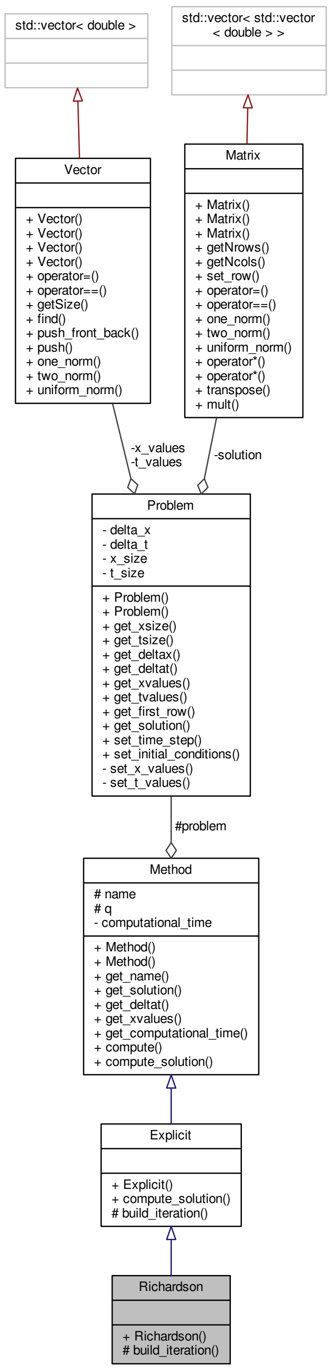
A Richardson method class that contains an iteration builder.

```
#include <richardson.h>
```

Inheritance diagram for Richardson:



Collaboration diagram for Richardson:



Public Member Functions

- Richardson (Problem problem)

    Default constructor.

Protected Member Functions

- Vector build_iteration (Vector current_step, Vector previous_step)

  Normal protected method.

Additional Inherited Members

4.12.1 Detailed Description

A Richardson method class that contains an iteration builder.

This builder is used to calculate a solution using the Richardson method.

The Richardson class provides:
-a basic constructor for creating a Richardson method object.
-a method to compute a solution of the current iteration

4.12.2 Constructor & Destructor Documentation

4.12.2.1 Richardson::Richardson (**Problem** ***problem***)

Default constructor.

4.12.3 Member Function Documentation

4.12.3.1 Vector Richardson::build_iteration (**Vector** ***current_step*,** **Vector** ***previous_step***) `[protected]`, `[virtual]`

Normal protected method.

Calculate a next time step solution requiring a previous time step and a current time step solution.

Parameters

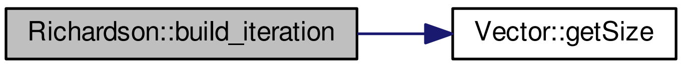
current_step	A vector representing the current time step solution.
previous_step	A vector representing the previous time step solution.

Returns

Vector. The computed solution.

Implements Explicit.

Here is the call graph for this function:



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

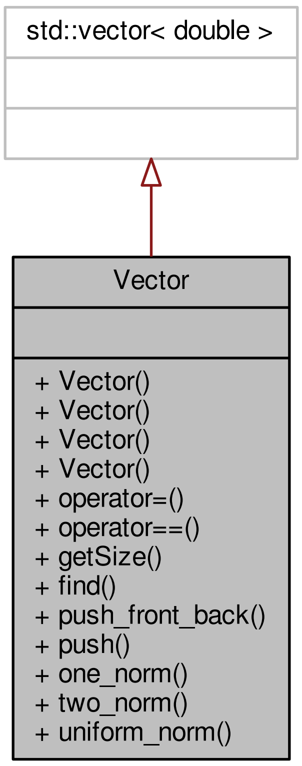
- methods/explicit/richardson.h
- methods/explicit/richardson.cpp

4.13 Vector Class Reference

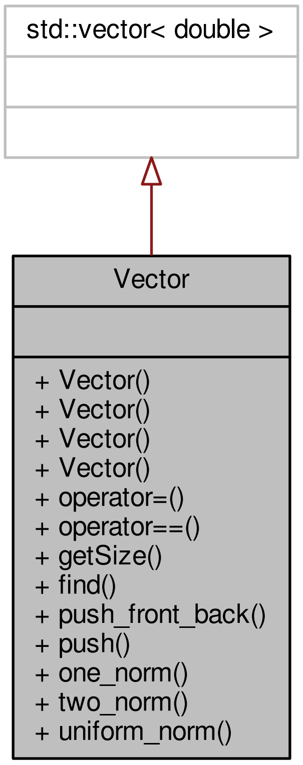
A vector class for data storage of a 1D array of doubles
The implementation is derived from the standard container vector std::vector
We use private inheritance to base our vector upon the library version whilst usto expose only those base class functions we wish to use - in this the array access operator [].

```
#include <vector.h>
```

Inheritance diagram for Vector:



Collaboration diagram for Vector:



Public Member Functions

- Vector ()

    Default constructor.
- Vector (int Num)

    Explicit alterative constructor takes an intiger.
- Vector (const Vector &v)

    Copy constructor takes an Vector object reference.
- Vector (std::vector< double > vec)

    Copy constructor takes an vector<double> object reference.
- Vector & operator= (const Vector &v)

    Overloaded assignment operator.
- bool operator== (const Vector &v) const

    Overloaded comparison operator returns true if vectors are the same within a tolerance (1.e-07)
- int getSize () const

    Normal get method that returns integer, the size of the vector.
- int find (double value)

    Method to find the value index in a vector.
- void push_front_back (double value)

    Method to push a value to the first and last position of a Vector.

- void push (double value)

    Method to push a value to the last position of a Vector.
- double one_norm () const

    Normal public method that returns a double.
- double two_norm () const

    Normal public method that returns a double.
- double uniform_norm () const

    Normal public method that returns a double.

Private Types

- typedef std::vector< double > vec

Friends

- std::istream & operator>> (std::istream &is, Vector &v)

    Overloaded istream >> operator.
- std::ostream & operator<< (std::ostream &os, const Vector &v)

    Overloaded ifstream << operator.
- std::ifstream & operator>> (std::ifstream &ifs, Vector &v)

    Overloaded ifstream >> operator.
- std::ofstream & operator<< (std::ofstream &ofs, const Vector &v)

    Overloaded ofstream << operator.

4.13.1 Detailed Description

A vector class for data storage of a 1D array of doubles
The implementation is derived from the standard container vector std::vector
We use private inheritance to base our vector upon the library version whilst usto expose only those base class functions we wish to use - in this the array access operator [].

The Vector class provides:
-basic constructors for creating vector obcjet from other vector object, or by creating empty vector of a given size,
-input and oput operation via >> and << operators using keyboard or file
-basic operations like access via [] operator, assignment and comparision

4.13.2 Member Typedef Documentation

4.13.2.1 typedef std::vector<double> **Vector::vec** `[private]`

4.13.3 Constructor & Destructor Documentation

4.13.3.1 Vector::Vector ( )

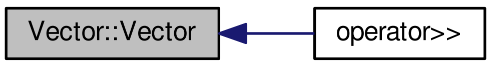
Default constructor.

Intialize an empty Vector object

See also

Vector(int Num)
Vector(const Vector& v)

Here is the caller graph for this function:



4.13.3.2 Vector::Vector (int *Num*) [explicit]

Explicit alterative constructor takes an intiger.

it is explicit since implicit type conversion int -> vector doesn't make sense Intialize Vector object of size Num

See also

Vector()
Vector(const Vector& v)

Exceptions

invalid_argument	("vector size negative")

Parameters

Num	int. Size of a vector

4.13.3.3 Vector::Vector (const Vector & *v*)

Copy constructor takes an Vector object reference.

Intialize Vector object with another Vector object

See also

Vector()
Vector(int Num)

4.13.3.4 Vector::Vector (std::vector< double > *vec*)

Copy constructor takes an vector<double> object reference.

Intialize Vector object with an vector<double> object

See also

Vector()
Vector(int Num)
Vector(const Vector& v)

4.13.4 Member Function Documentation

4.13.4.1 int Vector::find (double *value*)

Method to find the value index in a vector.

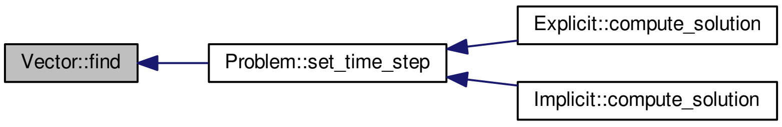
Parameters

value	Value to find

Returns

int. -1 if value was not found or the value index otherwise

Here is the caller graph for this function:



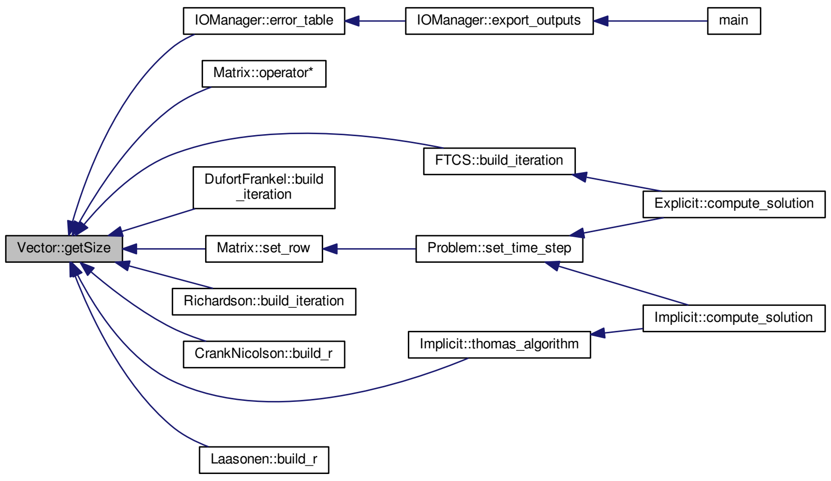
4.13.4.2 int Vector::getSize ( ) const

Normal get method that returns integer, the size of the vector.

Returns

int. the size of the vector

Here is the caller graph for this function:


Vector::getSize
IOManager::error_table
IOManager::export_outputs
main
Matrix::operator*
FTCS::build_iteration
DufortFrankel::build_iteration
Explicit::compute_solution
Matrix::set_row
Problem::set_time_step
Richardson::build_iteration
Implicit::compute_solution
CrankNicolson::build_r
Implicit::thomas_algorithm
Laasonen::build_r

4.13.4.3 double Vector::one_norm ( ) const

Normal public method that returns a double.

It returns L1 norm of vector

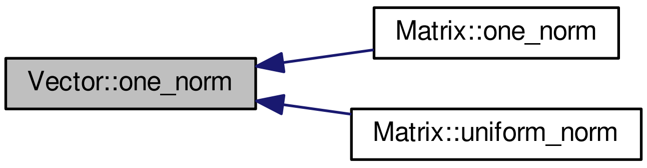
See also

two_norm()const
uniform_norm()const

Returns

double. vectors L1 norm

Here is the caller graph for this function:



4.13.4.4 Vector & Vector::operator= (const Vector & *v*)

Overloaded assignment operator.

See also

operator==(const Vector& v)const

Parameters

v	Vector to assign from

Returns

the object on the left of the assignment

Parameters

v	Vecto&. Vector to assign from

4.13.4.5 bool Vector::operator== (const Vector & *v*) const

Overloaded comparison operator returns true if vectors are the same within a tolerance (1.e-07)

See also

operator=(const Vector& v)
operator[ ](int i)
operator[ ](int i)const

Returns

bool. true or false

Exceptions

invalid_argument	("incompatible vector sizes\n")

Parameters

v	Vector&. vector to compare

4.13.4.6 void Vector::push (double *value*)

Method to push a value to the last position of a Vector.

Parameters

value	Value to be pushed

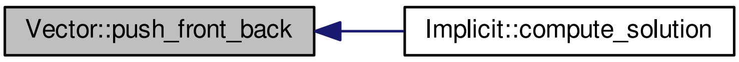
4.13.4.7 void Vector::push_front_back (double *value*)

Method to push a value to the first and last position of a Vector.

Parameters

value	Value to insert

Here is the caller graph for this function:



4.13.4.8 double Vector::two_norm ( ) const

Normal public method that returns a double.

It returns L2 norm of vector

See also

one_norm()const
uniform_norm()const

Returns

double. vectors L2 norm

4.13.4.9 double Vector::uniform_norm ( ) const

Normal public method that returns a double.

It returns L_max norm of vector

See also

one_norm()const
two_norm()const

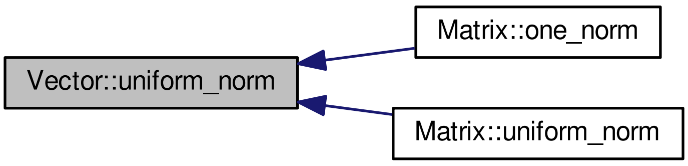
Exceptions

out_of_range	("vector access error") vector has zero size

Returns

double. vectors Lmax norm

Here is the caller graph for this function:



4.13.5 Friends And Related Function Documentation

4.13.5.1 std::ostream& operator<< (std::ostream & *os*, const Vector & *v*) [friend]

Overloaded ifstream << operator.

Display output.

See also

operator>>(std::istream& is, Vector& v)
operator>>(std::ifstream& ifs, Vector& v)
operator<<(std::ofstream& ofs, const Vector& v)

Returns

std::ostream&. the output stream object os

Parameters

os	output file stream
v	vector to read from

4.13.5.2 std::ofstream& operator<< (std::ofstream & *ofs*, const Vector & *v*) [friend]

Overloaded ofstream << operator.

File output. the file output operator is compatible with file input operator, ie. everything written can be read later.

See also

operator>>(std::istream& is, Vector& v)
operator>>(std::ifstream& ifs, Vector& v)
operator<<(std::ostream& os, const Vector& v)

Returns

std::ofstream&. the output ofstream object ofs

Parameters

ofs	outputfile stream. With opened file
v	Vector&. vector to read from

4.13.5.3 std::istream& operator>>(std::istream & *is*, Vector & *v*) [friend]

Overloaded istream >> operator.

Keyboard input if vector has size user will be asked to input only vector values if vector was not initialized user can choose vector size and input it values

See also

operator>>(std::ifstream& ifs, Vector& v)
operator<<(std::ostream& os, const Vector& v)
operator<<(std::ofstream& ofs, const Vector& v)

Returns

std::istream&. the input stream object is

Exceptions

std::invalid_argument	("read error - negative vector size");

Parameters

is	keyboard input straem. For user input
v	Vector&. vector to write to

4.13.5.4 std::ifstream& operator>>(std::ifstream & *ifs*, Vector & *v*) [friend]

Overloaded ifstream >> operator.

File input the file output operator is compatible with file input operator, ie. everything written can be read later.

See also

operator>>(std::istream& is, Vector& v)
operator<<(std::ostream& os, const Vector& v)
operator<<(std::ofstream& ofs, const Vector& v)

Returns

ifstream&. the input ifstream object ifs

Exceptions

std::invalid_argument	("file read error - negative vector size");

Parameters

ifs	input file straem. With opened matrix file
v	Vector&. vector to write to

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

- grid/vector.h
- grid/vector.cpp

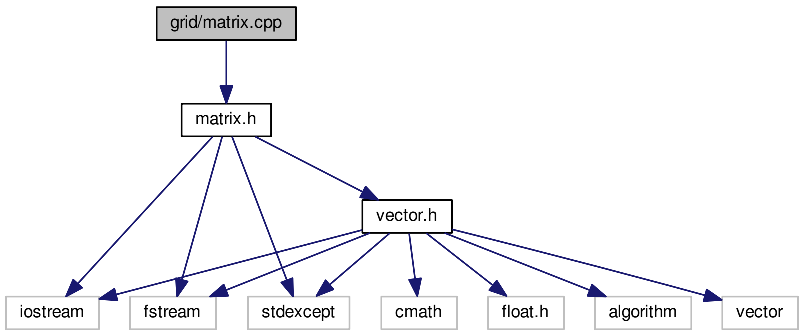
Chapter 5

File Documentation

5.1 grid/matrix.cpp File Reference

```
#include "matrix.h"
```

Include dependency graph for matrix.cpp:



Functions

- std::istream & operator>> (std::istream &is, Matrix &m)
- std::ostream & operator<< (std::ostream &os, const Matrix &m)
- std::ifstream & operator>> (std::ifstream &ifs, Matrix &m)
- std::ofstream & operator<< (std::ofstream &ofs, const Matrix &m)

5.1.1 Function Documentation

5.1.1.1 std::ostream& operator<< (std::ostream & *os*, const Matrix & *m*)

Display output if matrix has size user will be asked to input only matrix values if matrix was not initialized user can choose matrix size and input it values

See also

operator>>(std::ifstream& ifs, Matrix& m)
operator>>(std::istream& is, Matrix& m)
operator<<(std::ostream& os, const Matrix& m)

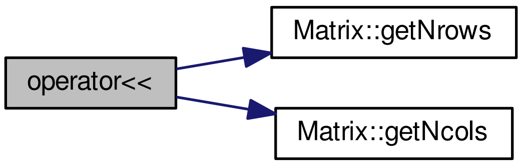
Returns

std::ostream&. The ostream object

Parameters

os	Display output stream
m	Matrix to read from

Here is the call graph for this function:



5.1.1.2 std::ofstream& operator<< (std::ofstream & *ofs*, const **Matrix** & *m*)

File output the file output operator is compatible with file input operator, ie. everything written can be read later.

See also

operator>>(std::ifstream& ifs, Matrix& m)
operator<<(std::ofstream& ofs, const Matrix& m)
operator>>(std::istream& is, Matrix& m)

Exceptions

std::invalid_argument	("file read error - negative matrix size");

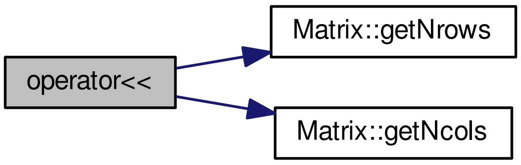
Returns

std::ofstream&. The ofstream object

Parameters

m	Matrix to read from

Here is the call graph for this function:



5.1.1.3 std::istream& operator>> (std::istream & *is*, **Matrix** & *m*)

Keyboard input if matrix has size user will be asked to input only matrix values if matrix was not initialized user can choose matrix size and input it values

See also

operator<<(std::ofstream& ofs, const Matrix& m)
operator>>(std::istream& is, Matrix& m)
operator<<(std::ostream& os, const Matrix& m)

Exceptions

std::invalid_argument	("read error - negative matrix size");

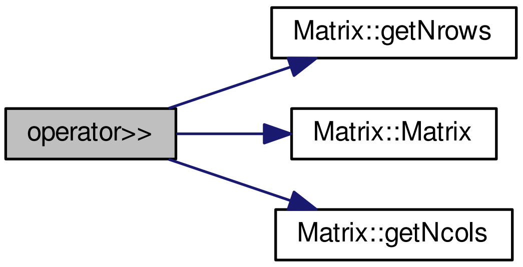
Returns

std::istream&. The istream object

Parameters

is	Keyboard input stream
m	Matrix to write into

Here is the call graph for this function:



5.1.1.4 std::ifstream& operator>> (std::ifstream & *ifs*, Matrix & *m*)

File input the file output operator is compatible with file input operator, ie. everything written can be read later.

See also

operator>>(std::ifstream& ifs, Matrix& m)
operator<<(std::ofstream& ofs, const Matrix& m)
operator<<(std::ostream& os, const Matrix& m)

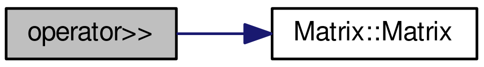
Returns

std::ifstream&. The ifstream object

Parameters

ifs	Input file stream with opened matrix file
m	Matrix to write into

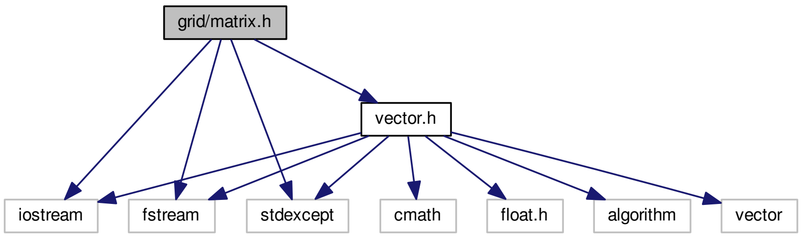
Here is the call graph for this function:



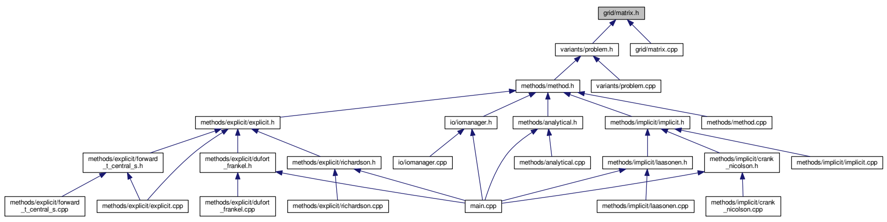
5.2 grid/matrix.h File Reference

```
#include <iostream>
#include <fstream>
#include <stdexcept>
#include "vector.h"
```

Include dependency graph for matrix.h:



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


grid/matrix.h
variants/problem.h
grid/matrix.cpp
methods/method.h
variants/problem.cpp
methods/explicit/explicit.h
io/iomanager.h
methods/analytical.h
methods/implicit/implicit.h
methods/method.cpp
methods/explicit/forward _t_central_s.h
methods/explicit/dufort _frankel.h
methods/explicit/richardson.h
io/iomanager.cpp
methods/analytical.cpp
methods/implicit/laasonen.h
methods/implicit/crank _nicolson.h
methods/implicit/implicit.cpp
methods/explicit/forward _t_central_s.cpp
methods/explicit/explicit.cpp
methods/explicit/dufort _frankel.cpp
methods/explicit/richardson.cpp
main.cpp
methods/implicit/laasonen.cpp
methods/implicit/crank _nicolson.cpp

Classes

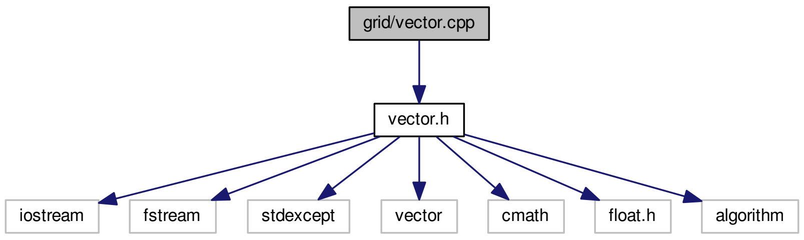
- class Matrix

  A matrix class for data storage of a 2D array of doubles
  The implementation is derived from the standard container vector std::vector
  We use private inheritance to base our vector upon the library version whilst usto expose only those base class functions we wish to use - in this the array access operator [].

5.3 grid/vector.cpp File Reference

```
#include "vector.h"
```

Include dependency graph for vector.cpp:



Functions

- std::istream & operator>> (std::istream &is, Vector &v)
- std::ifstream & operator>> (std::ifstream &ifs, Vector &v)
- std::ostream & operator<< (std::ostream &os, const Vector &v)
- std::ofstream & operator<< (std::ofstream &ofs, const Vector &v)

5.3.1 Function Documentation

5.3.1.1 std::ostream& operator<< (std::ostream & *os*, const Vector & *v*)

Display output.

See also

operator>>(std::istream& is, Vector& v)
operator>>(std::ifstream& ifs, Vector& v)
operator<<(std::ofstream& ofs, const Vector& v)

Returns

std::ostream&. the output stream object os

Parameters

os	output file stream
v	vector to read from

5.3.1.2 std::ofstream& operator<< (std::ofstream & *ofs*, const Vector & *v*)

File output. the file output operator is compatible with file input operator, ie. everything written can be read later.

See also

operator>>(std::istream& is, Vector& v)
operator>>(std::ifstream& ifs, Vector& v)
operator<<(std::ostream& os, const Vector& v)

Returns

std::ofstream&. the output ofstream object ofs

Parameters

ofs	outputfile stream. With opened file
v	Vector&. vector to read from

5.3.1.3 std::istream& operator>> (std::istream & *is,* Vector & *v*)

Keyboard input if vector has size user will be asked to input only vector values if vector was not initialized user can choose vector size and input it values

See also

operator>>(std::ifstream& ifs, Vector& v)
operator<<(std::ostream& os, const Vector& v)
operator<<(std::ofstream& ofs, const Vector& v)

Returns

std::istream&. the input stream object is

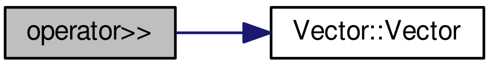
Exceptions

std::invalid_argument	("read error - negative vector size");

Parameters

is	keyboard input straem. For user input
v	Vector&. vector to write to

Here is the call graph for this function:



5.3.1.4 std::ifstream& operator>> (std::ifstream & *ifs*, Vector & *v*)

File input the file output operator is compatible with file input operator, ie. everything written can be read later.

See also

operator>>(std::istream& is, Vector& v)
operator<<(std::ostream& os, const Vector& v)
operator<<(std::ofstream& ofs, const Vector& v)

Returns

ifstream&. the input ifstream object ifs

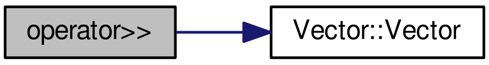
Exceptions

std::invalid_argument	("file read error - negative vector size");

Parameters

ifs	input file straem. With opened matrix file
v	Vector&. vector to write to

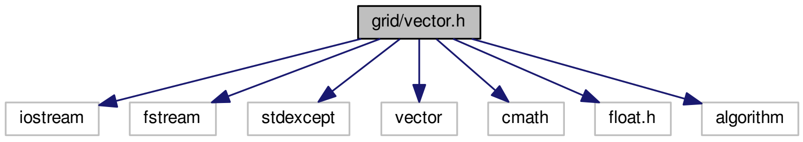
Here is the call graph for this function:



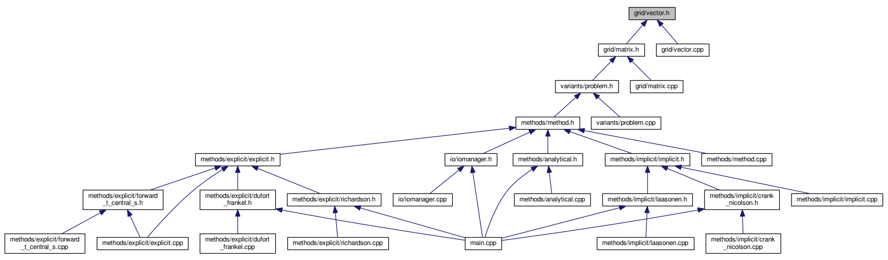
5.4 grid/vector.h File Reference

```
#include <iostream>
#include <fstream>
#include <stdexcept>
#include <vector>
#include <cmath>
#include <float.h>
#include <algorithm>
```

Include dependency graph for vector.h:



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



Classes

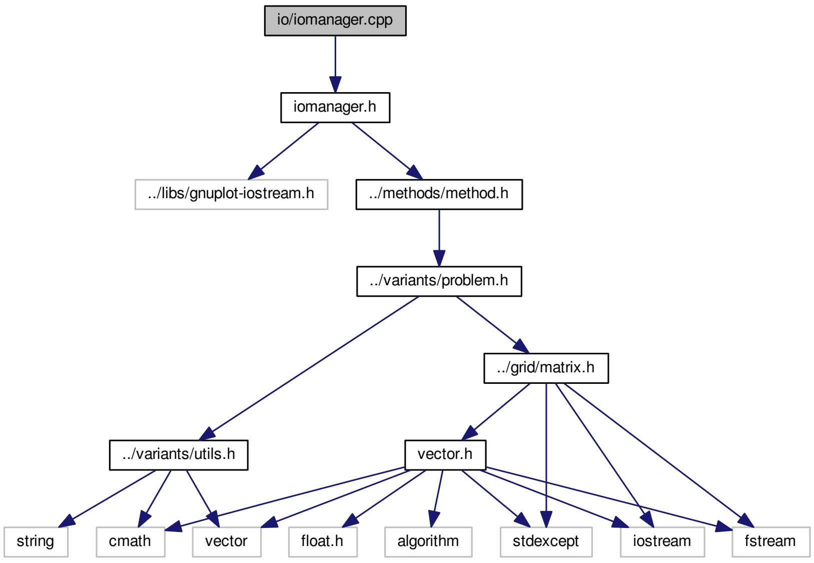
- class Vector

  A vector class for data storage of a 1D array of doubles
  The implementation is derived from the standard container vector std::vector
  We use private inheritance to base our vector upon the library version whilst usto expose only those base class functions we wish to use - in this the array access operator [].

5.5 io/iomanager.cpp File Reference

```
#include "iomanager.h"
```

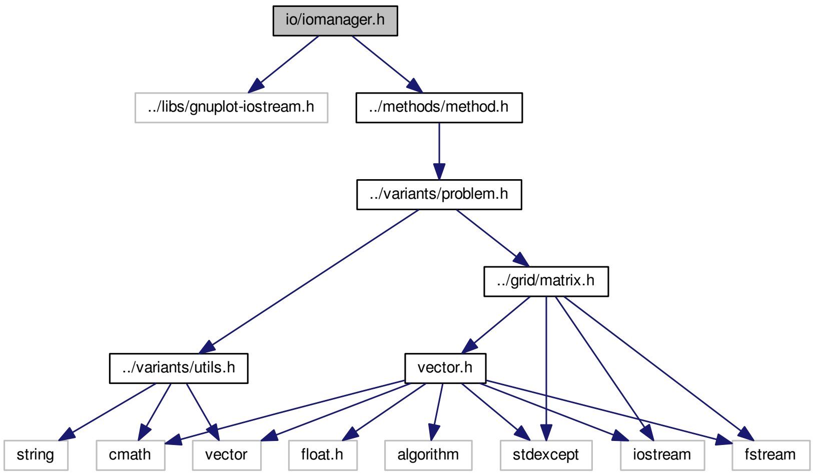
Include dependency graph for iomanager.cpp:



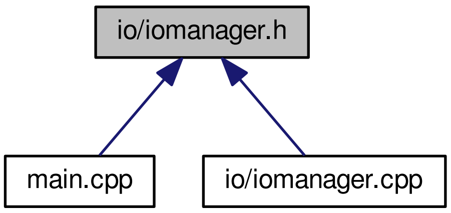
5.6 io/iomanager.h File Reference

```
#include "../libs/gnuplot-iostream.h"
#include "../methods/method.h"
```

Include dependency graph for iomanager.h:



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



Classes

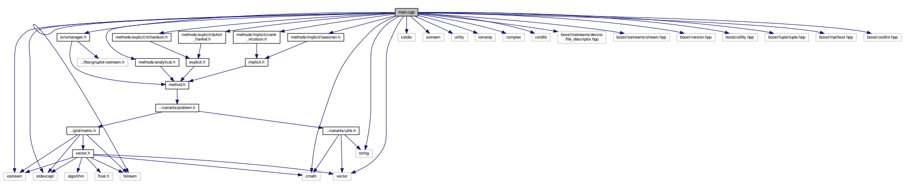
- class IOManager

  An input/output manager class to handle plot exportations and future implementations of input handling.

5.7 main.cpp File Reference

```
#include <iostream>
#include "methods/analytical.h"
#include "methods/explicit/dufort_frankel.h"
#include "methods/explicit/richardson.h"
#include "methods/implicit/laasonen.h"
#include "methods/implicit/crank_nicolson.h"
#include "io/iomanager.h"
```

Include dependency graph for main.cpp:



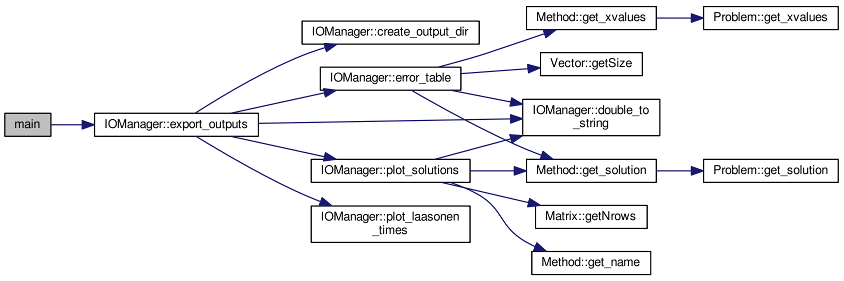
Functions

- int main ()

5.7.1 Function Documentation

5.7.1.1 int main ( )

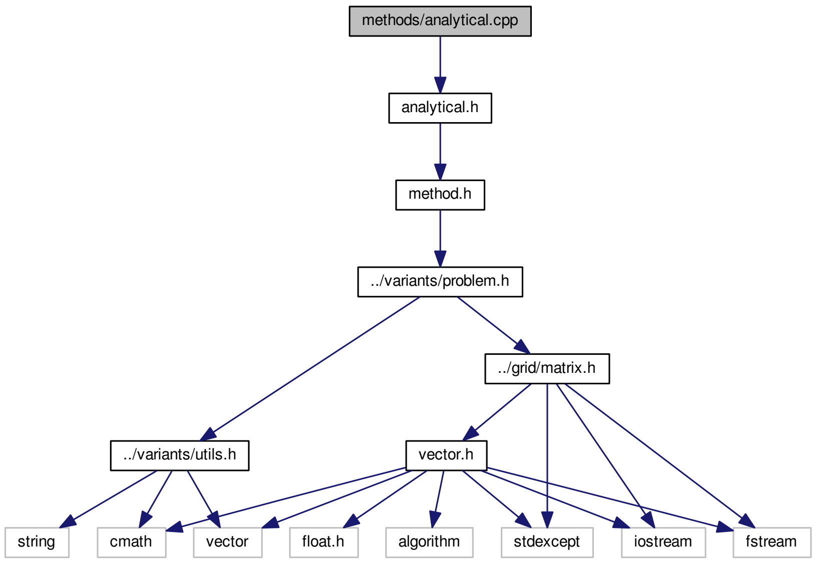
Here is the call graph for this function:



5.8 methods/analytical.cpp File Reference

```
#include "analytical.h"
```

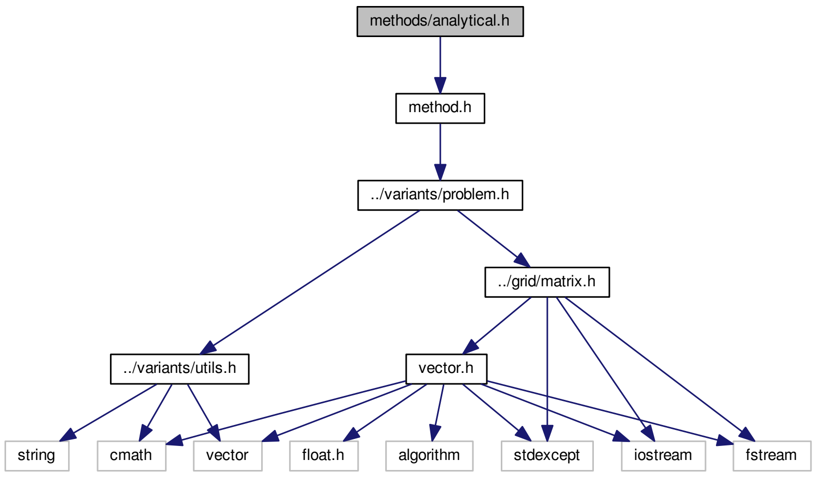
Include dependency graph for analytical.cpp:



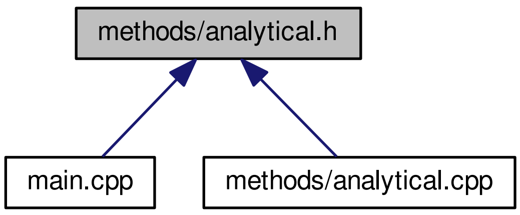
5.9 methods/analytical.h File Reference

```
#include "method.h"
```

Include dependency graph for analytical.h:



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



Classes

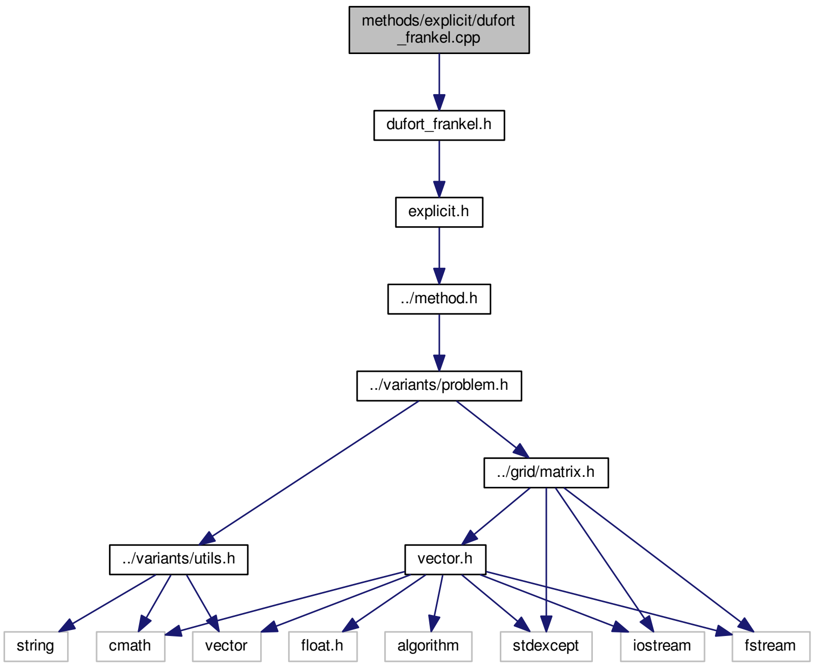
- class Analytical

  An Analytical class to compute the solution with standard procedures The implementation is derived from the Method Object.

5.10 methods/explicit/dufort_frankel.cpp File Reference

```
#include "dufort_frankel.h"
```

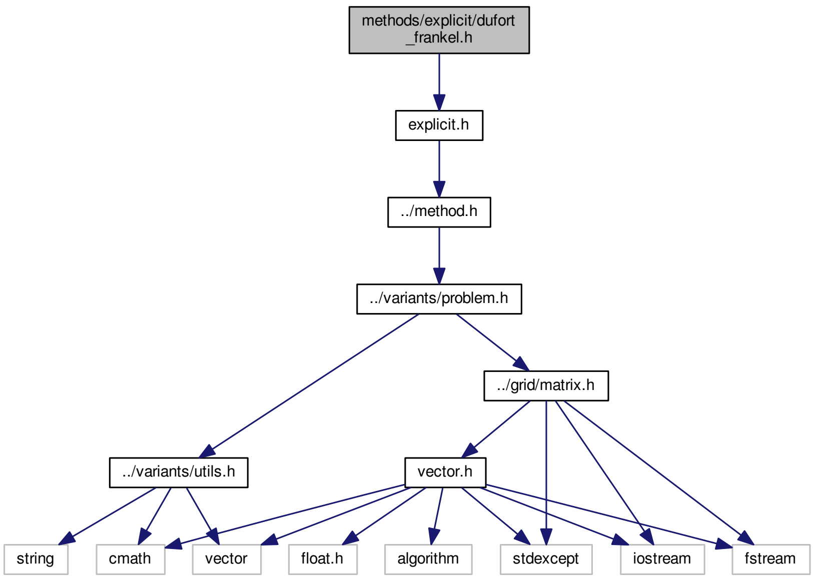
Include dependency graph for dufort_frankel.cpp:



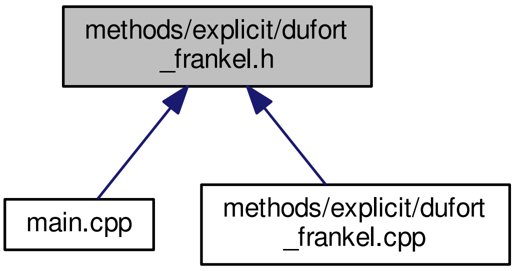
5.11 methods/explicit/dufort_frankel.h File Reference

```
#include "explicit.h"
```

Include dependency graph for dufort_frankel.h:



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



Classes

- class DufortFrankel

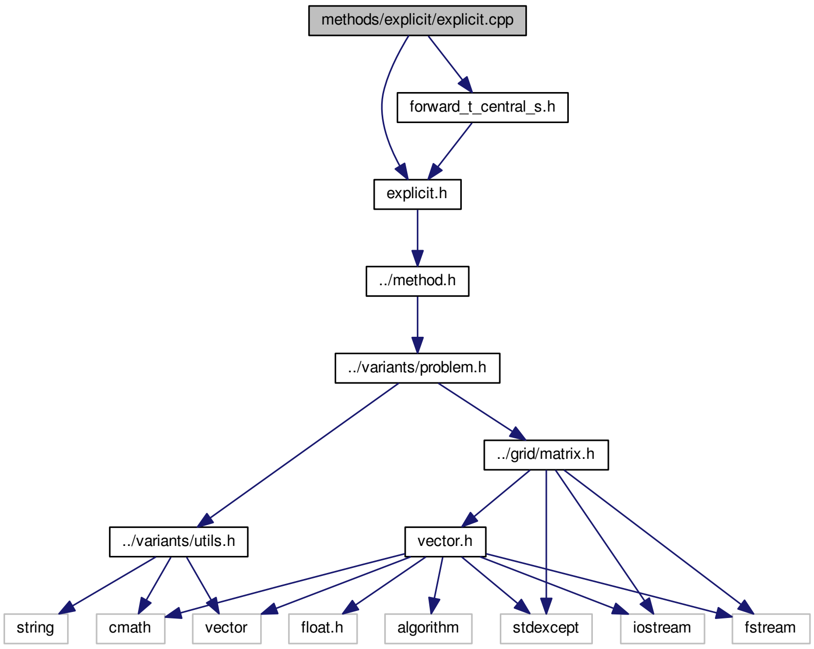
  A DufortFrankel method class that contains an iteration builder.

5.12 methods/explicit/explicit.cpp File Reference

```
#include "explicit.h"
```

```
#include "forward_t_central_s.h"
```

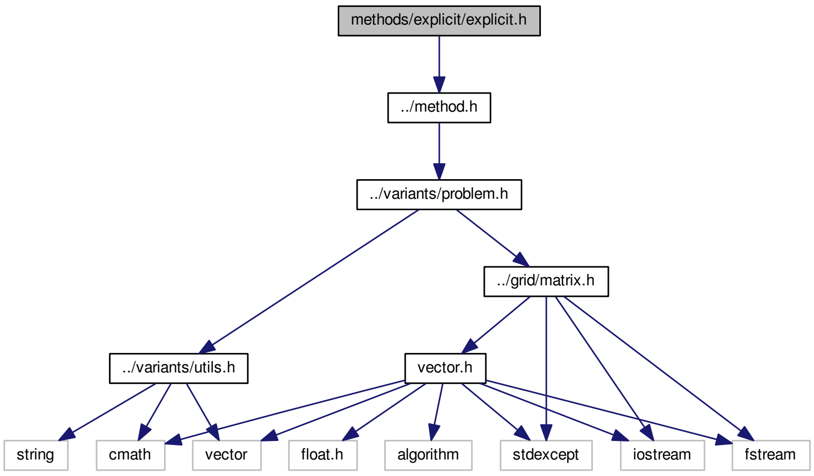
Include dependency graph for explicit.cpp:



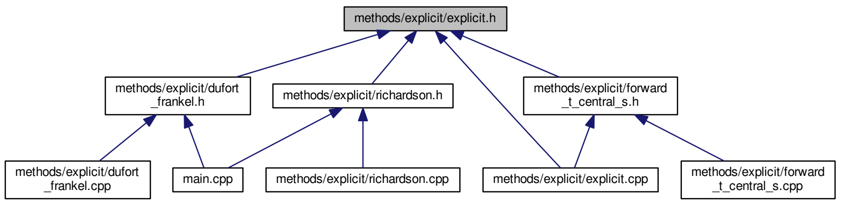
5.13 methods/explicit/explicit.h File Reference

```
#include "../method.h"
```

Include dependency graph for explicit.h:



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


methods/explicit/explicit.h
methods/explicit/dufort_frankel.h
methods/explicit/richardson.h
methods/explicit/forward_t_central_s.h
methods/explicit/dufort_frankel.cpp
main.cpp
methods/explicit/richardson.cpp
methods/explicit/explicit.cpp
methods/explicit/forward_t_central_s.cpp

Classes

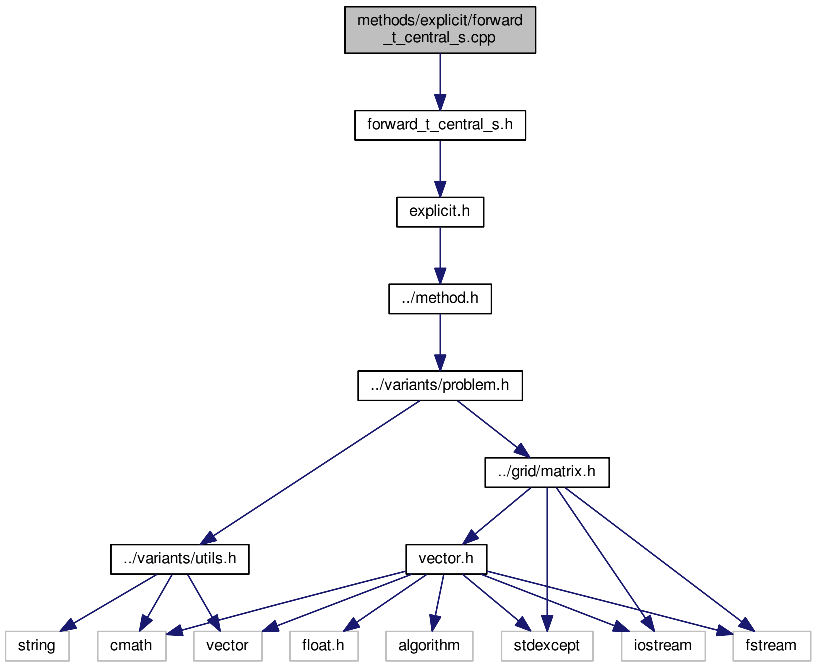
- class Explicit

  An explicit method class that contains default methods that only explicit methods use The implementation is derived from the Method class.

5.14 methods/explicit/forward_t_central_s.cpp File Reference

```
#include "forward_t_central_s.h"
```

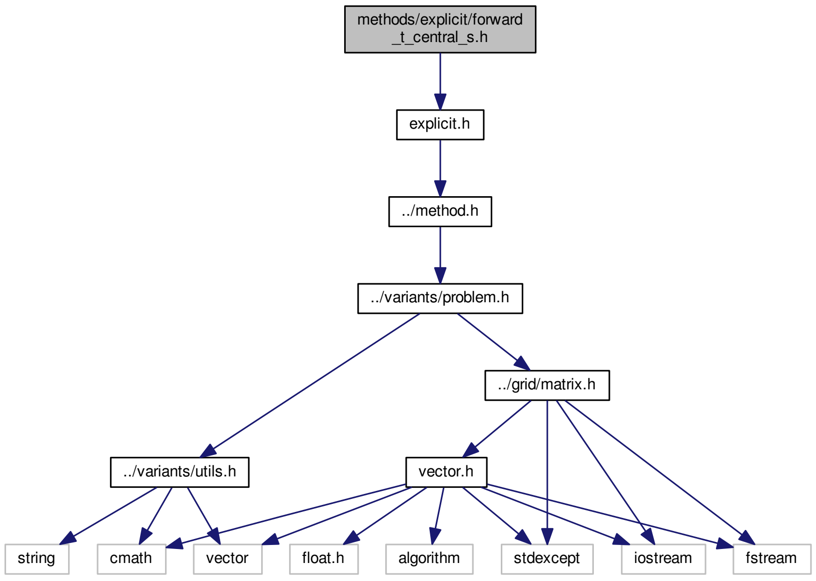
Include dependency graph for forward_t_central_s.cpp:



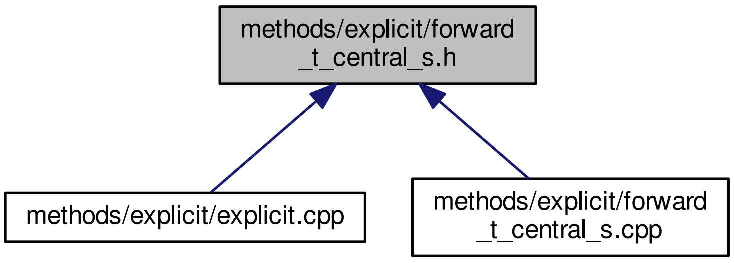
5.15 methods/explicit/forward_t_central_s.h File Reference

```
#include "explicit.h"
```

Include dependency graph for forward_t_central_s.h:



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


methods/explicit/forward _t_central_s.h
methods/explicit/explicit.cpp
methods/explicit/forward _t_central_s.cpp

Classes

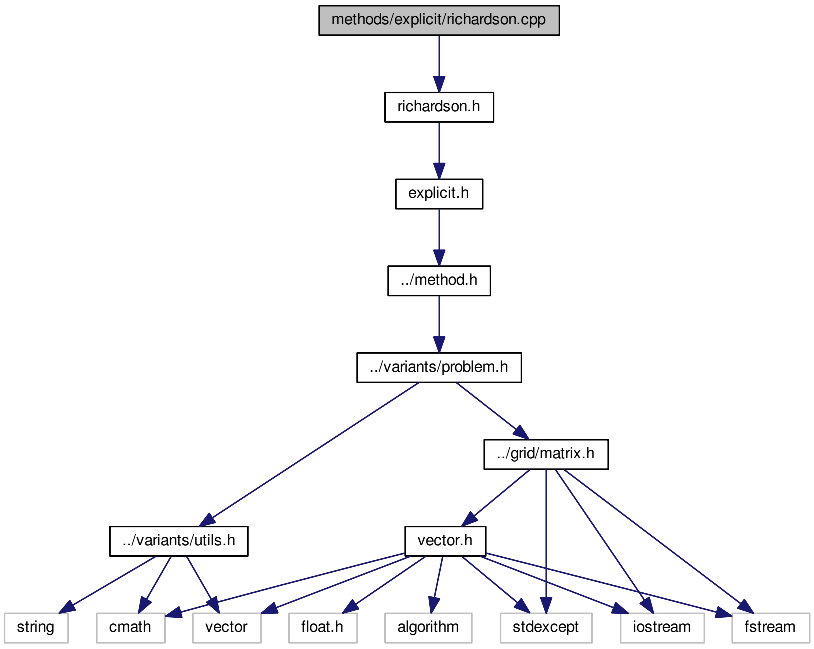
- class FTCS

  A FTCS method class that contains an iteration builder.

5.16 methods/explicit/richardson.cpp File Reference

`#include "richardson.h"`

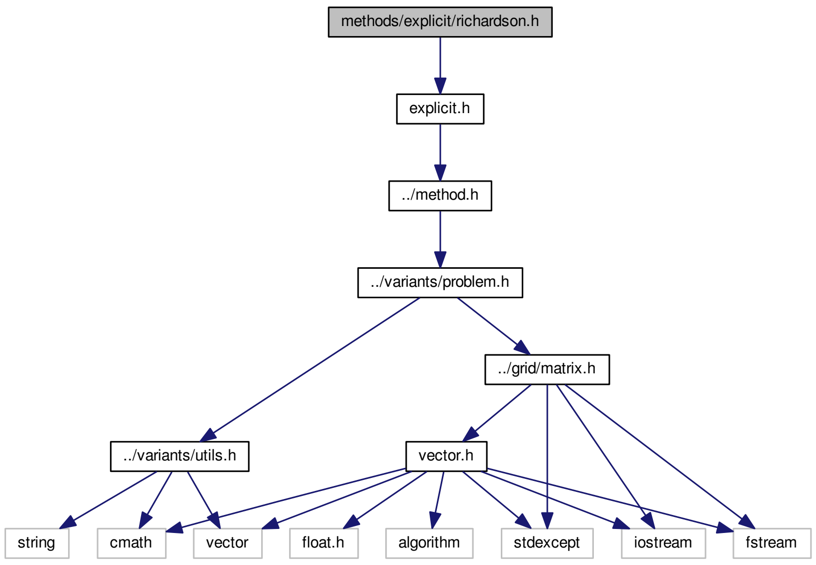
Include dependency graph for richardson.cpp:



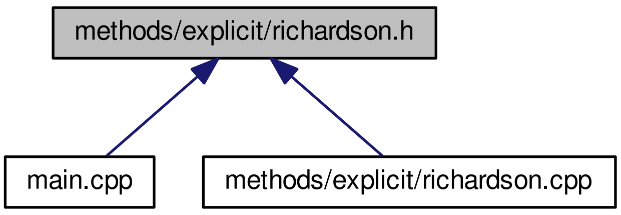
5.17 methods/explicit/richardson.h File Reference

`#include "explicit.h"`

Include dependency graph for richardson.h:



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


methods/explicit/richardson.h
main.cpp
methods/explicit/richardson.cpp

Classes

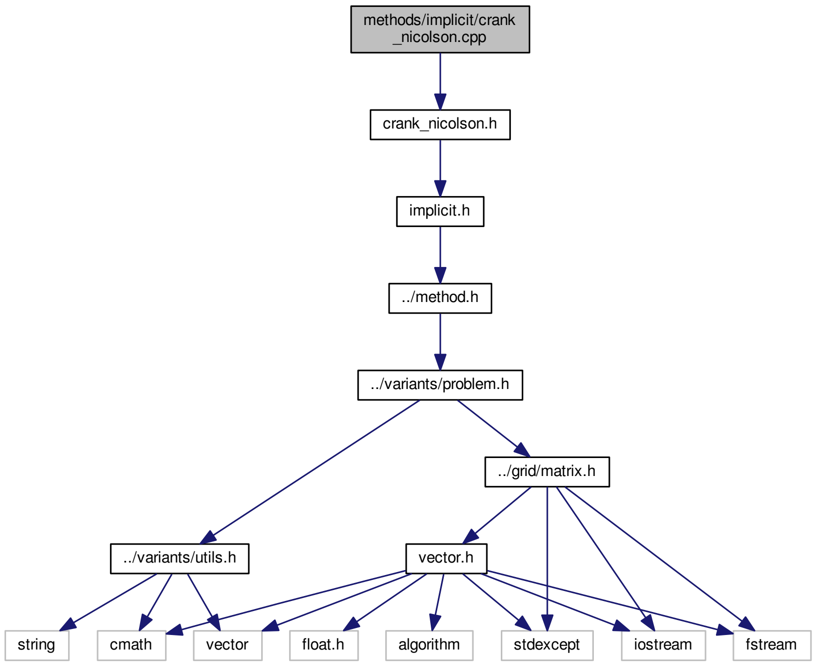
- class Richardson

    A Richardson method class that contains an iteration builder.

5.18 methods/implicit/crank_nicolson.cpp File Reference

```
#include "crank_nicolson.h"
```

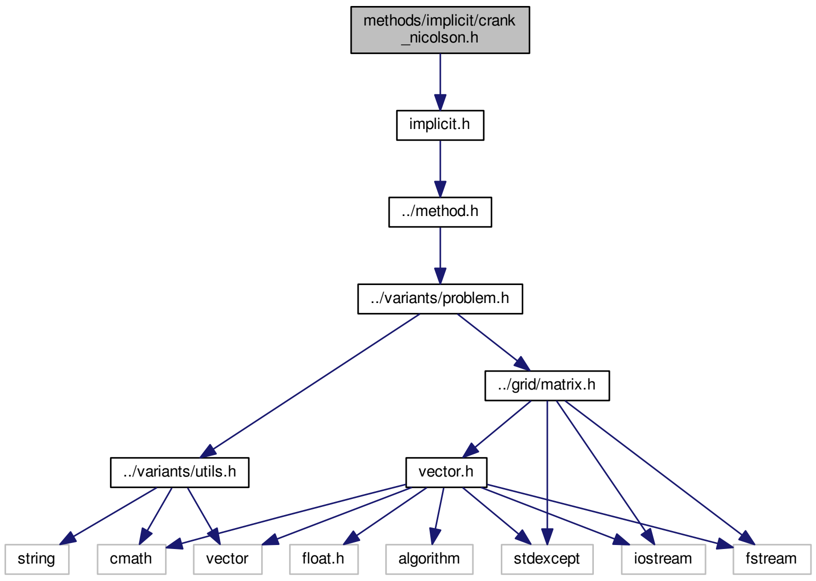
Include dependency graph for crank_nicolson.cpp:



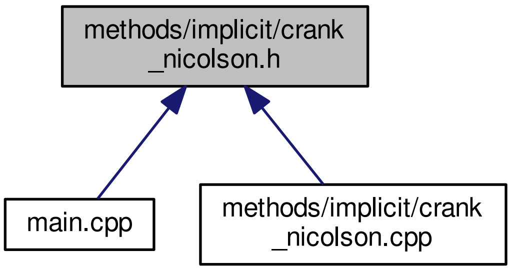
5.19 methods/implicit/crank_nicolson.h File Reference

```
#include "implicit.h"
```

Include dependency graph for crank_nicolson.h:



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



Classes

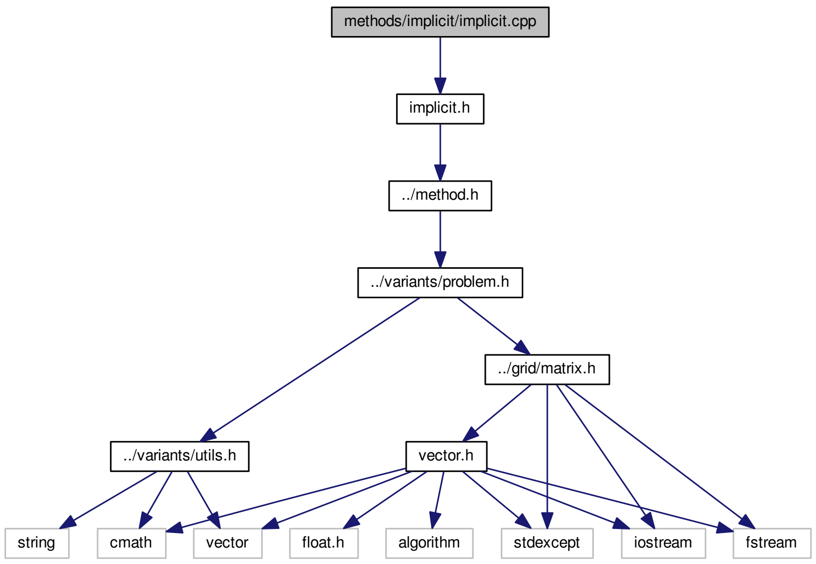
- class CrankNicolson

    A CrankNicolson method class that contains a r vector builder.

5.20 methods/implicit/implicit.cpp File Reference

```
#include "implicit.h"
```

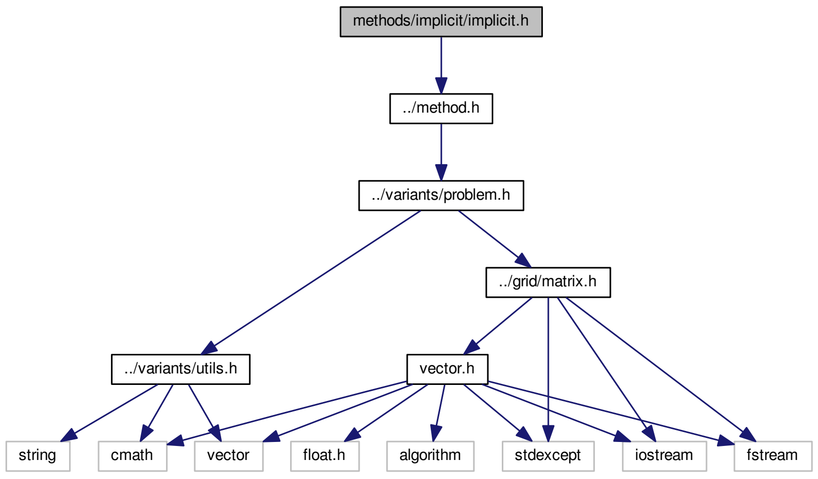
Include dependency graph for implicit.cpp:



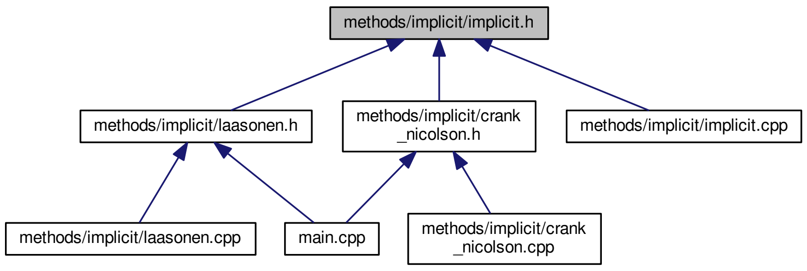
5.21 methods/implicit/implicit.h File Reference

```
#include "../method.h"
```

Include dependency graph for implicit.h:



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


methods/implicit/implicit.h
methods/implicit/laasonen.h
methods/implicit/crank_nicolson.h
methods/implicit/implicit.cpp
methods/implicit/laasonen.cpp
main.cpp
methods/implicit/crank_nicolson.cpp

Classes

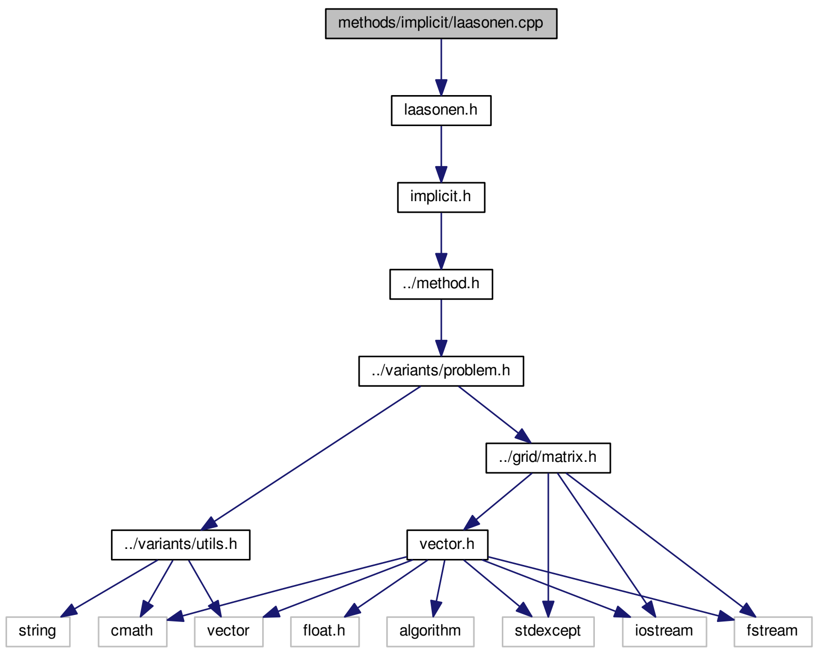
- class Implicit

  An implicit method class that contains default methods that only implicit methods use The implementation is derived from the Method class.

5.22 methods/implicit/laasonen.cpp File Reference

```
#include "laasonen.h"
```

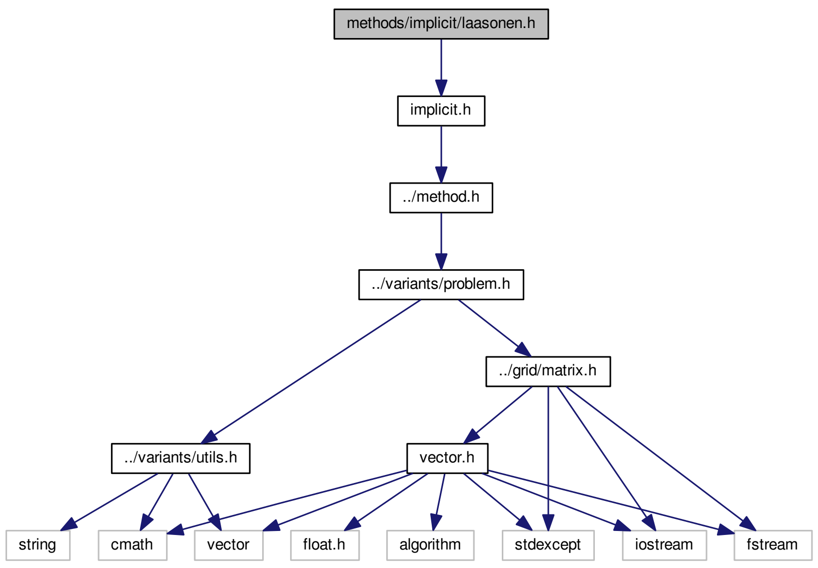
Include dependency graph for laasonen.cpp:



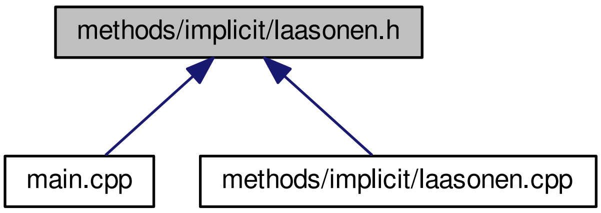
5.23 methods/implicit/laasonen.h File Reference

```
#include "implicit.h"
```

Include dependency graph for laasonen.h:



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



Classes

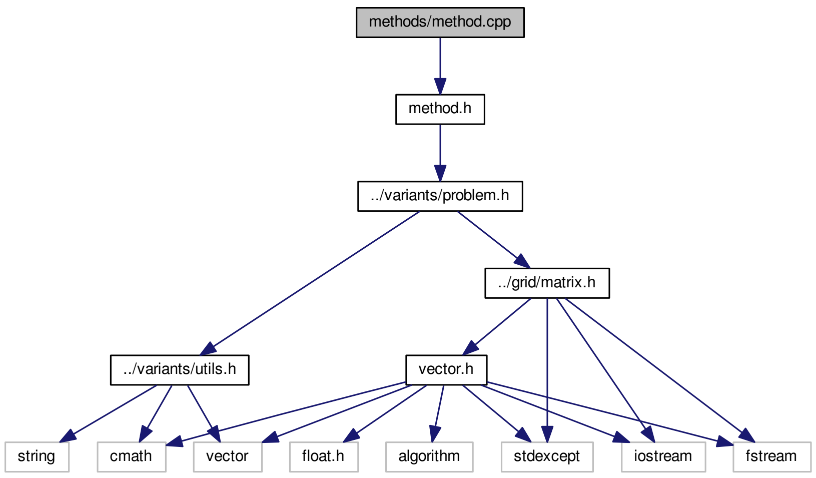
- class Laasonen

  A Laasonen method class that contains a r vector builder.

5.24 methods/method.cpp File Reference

```
#include "method.h"
```

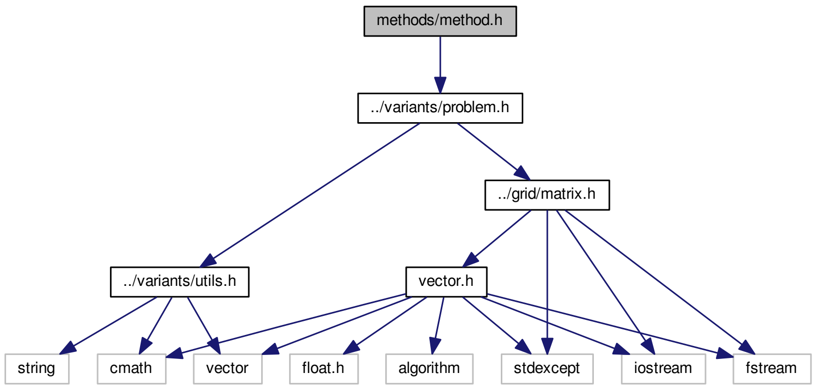
Include dependency graph for method.cpp:



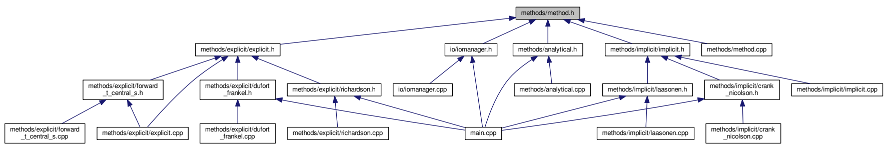
5.25 methods/method.h File Reference

```
#include "../variants/problem.h"
```

Include dependency graph for method.h:



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


methods/method.h
methods/explicit/explicit.h
io/iomanager.h
methods/analytical.h
methods/implicit/implicit.h
methods/method.cpp
methods/explicit/forward _t_central_s.h
methods/explicit/dufort _frankel.h
methods/explicit/richardson.h
io/iomanager.cpp
methods/analytical.cpp
methods/implicit/laasonen.h
methods/implicit/crank _nicolson.h
methods/implicit/implicit.cpp
methods/explicit/forward _t_central_s.cpp
methods/explicit/explicit.cpp
methods/explicit/dufort _frankel.cpp
methods/explicit/richardson.cpp
main.cpp
methods/implicit/laasonen.cpp
methods/implicit/crank _nicolson.cpp

Classes

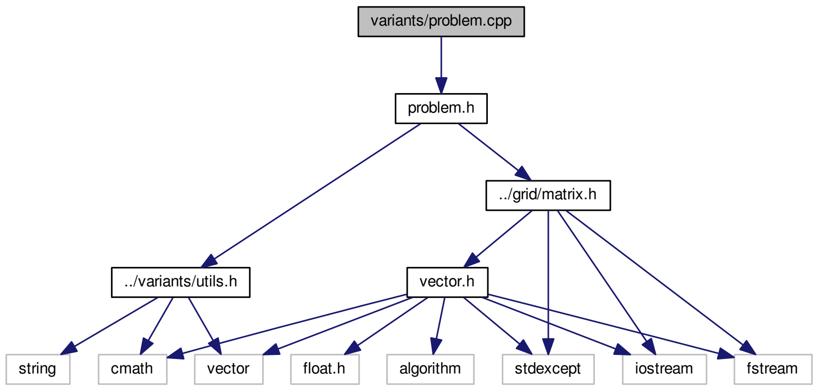
- class Method

  A Method class to structure information used to solve the problem.

5.26 variants/problem.cpp File Reference

```
#include "problem.h"
```

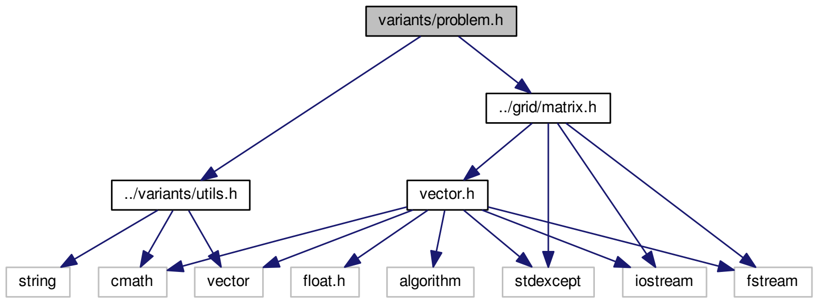
Include dependency graph for problem.cpp:



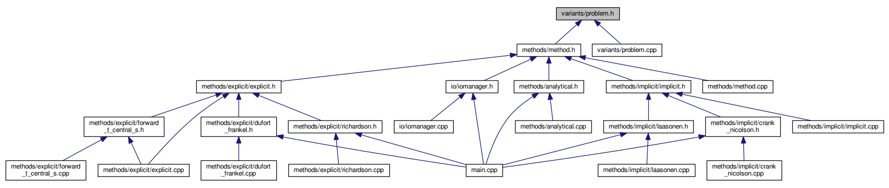
5.27 variants/problem.h File Reference

```
#include "../variants/utils.h"
#include "../grid/matrix.h"
```

Include dependency graph for problem.h:



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


variants/problem.h
methods/method.h
variants/problem.cpp
methods/explicit/explicit.h
io/iomanager.h
methods/analytical.h
methods/implicit/implicit.h
methods/method.cpp
methods/explicit/forward_t_central_s.h
methods/explicit/dufort_frankel.h
methods/explicit/richardson.h
io/iomanager.cpp
methods/analytical.cpp
methods/implicit/laasonen.h
methods/implicit/crank_nicolson.h
methods/implicit/implicit.cpp
methods/explicit/forward_t_central_s.cpp
methods/explicit/explicit.cpp
methods/explicit/dufort_frankel.cpp
methods/explicit/richardson.cpp
main.cpp
methods/implicit/laasonen.cpp
methods/implicit/crank_nicolson.cpp

Classes

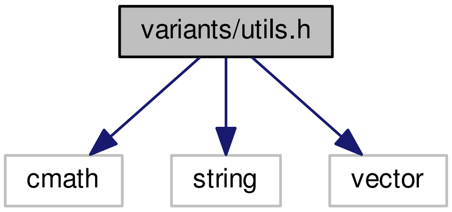
- class Problem

    A Problem class to structure relevant information related with the problem.

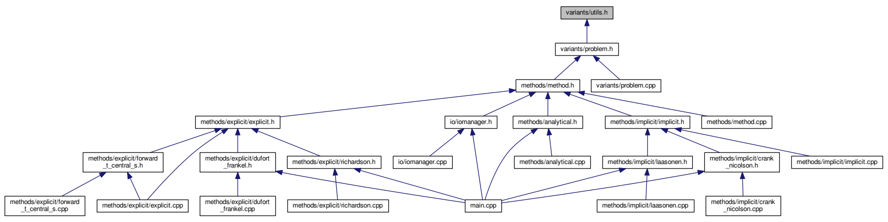
5.28 variants/utils.h File Reference

```
#include <cmath>
#include <string>
#include <vector>
```

Include dependency graph for utils.h:



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


variants/utils.h
variants/problem.h
methods/method.h
variants/problem.cpp
methods/explicit/explicit.h
io/iomanager.h
methods/analytical.h
methods/implicit/implicit.h
methods/method.cpp
methods/explicit/forward _t_central_s.h
methods/explicit/dufort _frankel.h
methods/explicit/richardson.h
io/iomanager.cpp
methods/analytical.cpp
methods/implicit/laasonen.h
methods/implicit/crank _nicolson.h
methods/implicit/implicit.cpp
methods/explicit/forward _t_central_s.cpp
methods/explicit/explicit.cpp
methods/explicit/dufort _frankel.cpp
methods/explicit/richardson.cpp
main.cpp
methods/implicit/laasonen.cpp
methods/implicit/crank _nicolson.cpp

Variables

- const double DELTA_T = 0.01

  Macro double.
- const double DELTA_X = 0.05

  Macro double.
- const std::vector< double > DELTA_T_LASSONEN = {0.01, 0.025, 0.05, 0.1}

  Macro double.
- const double DIFUSIVITY = 0.1

  Macro double.
- const double THICKNESS = 1.0

  Macro double.
- const double TIMELIMIT = 0.5

  Macro double.
- const double SURFACE_TEMPERATURE = 300.0

  Macro double.
- const double INITIAL_TEMPERATURE = 100.0

  Macro double.

- const double NUMBER_TIME_STEPS = 6.0

  Macro double.
- const unsigned int NUMBER_OF_EXPANSIONS = 20

  Macro unsigned int.
- const double PI = std::atan(1) $*$ 4

  Macro double.
- const std::string OUTPUT_PATH = "../outputs"

  Macro string.
- const std::string FORWARD_TIME_CENTRAL_SPACE = "Forward Time Central Space"

  Macro string.
- const std::string RICHARDSON = "Richardson"

  Macro string.
- const std::string DUFORT_FRANKEL = "DuFort-Frankel"

  Macro string.
- const std::string LAASONEN = "Laasonen"

  Macro string.
- const std::string CRANK_NICHOLSON = "Crank-Nicholson"

  Macro string.

5.28.1 Variable Documentation

5.28.1.1 const std::string CRANK_NICHOLSON = "Crank-Nicholson"

Macro string.

Crank-Nicholson method name.

5.28.1.2 const double DELTA_T = 0.01

Macro double.

The default time step.

5.28.1.3 const std::vector<double> DELTA_T_LASSONEN = {0.01, 0.025, 0.05, 0.1}

Macro double.

Time steps to study in Laasonen Implicit Scheme.

5.28.1.4 const double DELTA_X = 0.05

Macro double.

The default space step.

5.28.1.5 const double DIFUSIVITY = 0.1

Macro double.

The default value of difusivity.

5.28.1.6 const std::string DUFORT_FRANKEL = "DuFort-Frankel"

Macro string.

DuFort-Frankel method name.

5.28.1.7 const std::string FORWARD_TIME_CENTRAL_SPACE = "Forward Time Central Space"

Macro string.

Forward in Time and Central in Space method name.

5.28.1.8 const double INITIAL_TEMPERATURE = 100.0

Macro double.

The default initial temperature.

5.28.1.9 const std::string LAASONEN = "Laasonen"

Macro string.

Laasonen method name.

5.28.1.10 const unsigned int NUMBER_OF_EXPANSIONS = 20

Macro unsigned int.

Number of expansions to calculate the analytical solution sum expansion.

5.28.1.11 const double NUMBER_TIME_STEPS = 6.0

Macro double.

The default limit of time steps. 0, 0.1, 0.2, 0.3, 0.4, 0.5

5.28.1.12 const std::string OUTPUT_PATH = "../outputs"

Macro string.

Default outputs path.

5.28.1.13 const double PI = std::atan(1) ∗ 4

Macro double.

Approximated value of PI.

5.28.1.14 const std::string RICHARDSON = "Richardson"

Macro string.

Richardson method name.

5.28.1.15 const double SURFACE_TEMPERATURE = 300.0

Macro double.

The default surface temperature.

5.28.1.16 const double THICKNESS = 1.0

Macro double.

The default value of thickness.

5.28.1.17 const double TIMELIMIT = 0.5

Macro double.

The default value of time limit.

Index