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:

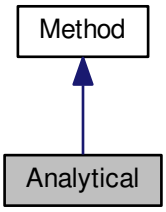
Chapter 3

Class Documentation

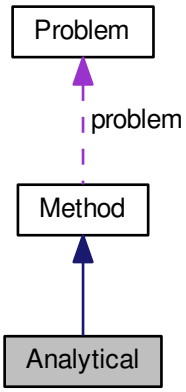
3.1 Analytical Class Reference

`#include <analytical.h>`

Inheritance diagram for Analytical:



Collaboration diagram for Analytical:



Public Member Functions

- Analytical (Problem problem)
- void compute_solution ()

Additional Inherited Members

3.1.1 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

3.1.2 Constructor & Destructor Documentation

3.1.2.1 Analytical::Analytical (Problem *problem*)

Default constructor. Intialize a Analytical object

3.1.3 Member Function Documentation

3.1.3.1 void Analytical::compute_solution ( ) [virtual]

Normal public method. compute the solution with specific given rules

Implements Method.

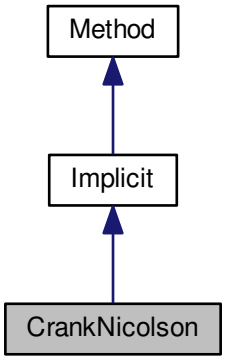
The documentation for this class was generated from the following files:

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

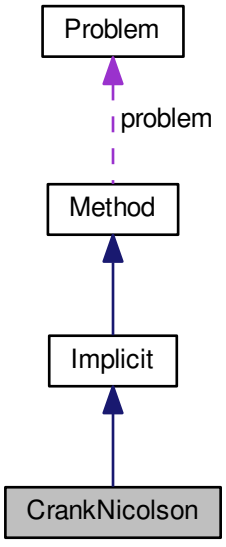
3.2 CrankNicolson Class Reference

`#include <crank_nicolson.h>`

Inheritance diagram for CrankNicolson:



Collaboration diagram for CrankNicolson:



Public Member Functions

- CrankNicolson (Problem problem)

Protected Member Functions

- Vector build_r (Vector previous_step)

Additional Inherited Members

3.2.1 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.

3.2.2 Constructor & Destructor Documentation

3.2.2.1 CrankNicolson::CrankNicolson (Problem *problem*)

Default constructor.

3.2.3 Member Function Documentation

3.2.3.1 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.

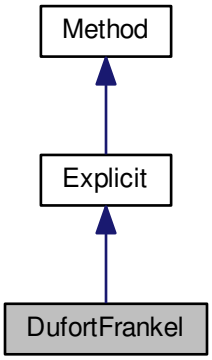
The documentation for this class was generated from the following files:

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

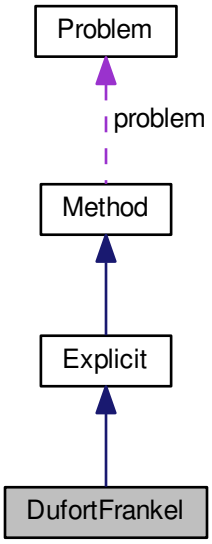
3.3 DufortFrankel Class Reference

`#include <dufort_frankel.h>`

Inheritance diagram for DufortFrankel:



Collaboration diagram for DufortFrankel:



Public Member Functions

- DufortFrankel (Problem problem)

Protected Member Functions

- Vector build_iteration (Vector current_step, Vector previous_step)

Additional Inherited Members

3.3.1 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

3.3.2 Constructor & Destructor Documentation

3.3.2.1 DufortFrankel::DufortFrankel (Problem *problem*)

Default constructor.

3.3.3 Member Function Documentation

3.3.3.1 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.

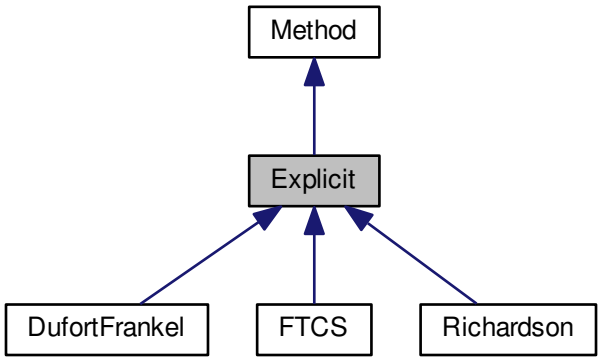
The documentation for this class was generated from the following files:

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

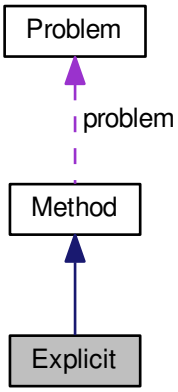
3.4 Explicit Class Reference

`#include <explicit.h>`

Inheritance diagram for Explicit:



Collaboration diagram for Explicit:



Public Member Functions

- Explicit (Problem problem)
- void compute_solution ()

Protected Member Functions

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

Additional Inherited Members

3.4.1 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

3.4.2 Constructor & Destructor Documentation

3.4.2.1 Explicit::Explicit (Problem *problem*)

Default constructor.

3.4.3 Member Function Documentation

3.4.3.1 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.

3.4.3.2 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.

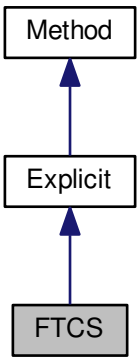
The documentation for this class was generated from the following files:

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

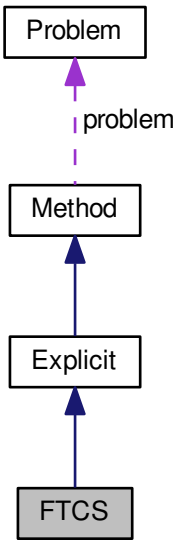
3.5 FTCS Class Reference

`#include <forward_t_central_s.h>`

Inheritance diagram for FTCS:



Collaboration diagram for FTCS:



Public Member Functions

- FTCS (Problem problem)
- Vector build_iteration (Vector current_step, Vector previous_step)

Additional Inherited Members

3.5.1 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

3.5.2 Constructor & Destructor Documentation

3.5.2.1 FTCS::FTCS (Problem *problem*)

Default constructor.

3.5.3 Member Function Documentation

3.5.3.1 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.

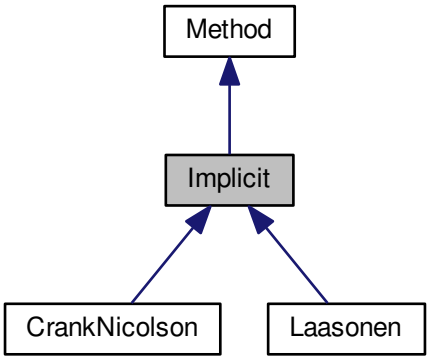
The documentation for this class was generated from the following files:

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

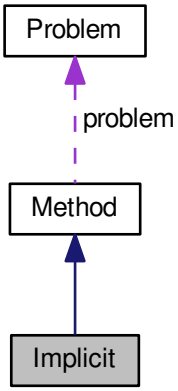
3.6 Implicit Class Reference

`#include <implicit.h>`

Inheritance diagram for Implicit:



Collaboration diagram for Implicit:



Public Member Functions

- Implicit (Problem problem)
- void compute_solution ()

Protected Member Functions

- virtual Vector build_r (Vector previous_step)=0

Additional Inherited Members

3.6.1 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

3.6.2 Constructor & Destructor Documentation

3.6.2.1 Implicit::Implicit (**Problem** *problem*)

Default constructor.

3.6.3 Member Function Documentation

3.6.3.1 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.

3.6.3.2 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.

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

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

3.7 IOManager Class Reference

`#include <iomanager.h>`

Public Member Functions

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

3.7.1 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

3.7.2 Constructor & Destructor Documentation

3.7.2.1 IOManager::IOManager ( )

Default constructor. Initialize an IOManager object.

3.7.3 Member Function Documentation

3.7.3.1 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

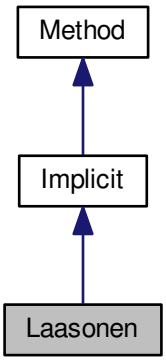
The documentation for this class was generated from the following files:

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

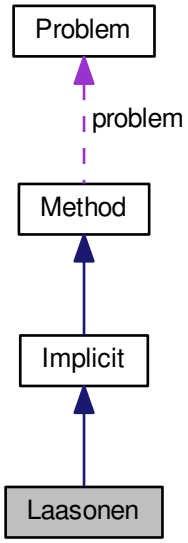
3.8 Laasonen Class Reference

`#include <laasonen.h>`

Inheritance diagram for Laasonen:



Collaboration diagram for Laasonen:



Public Member Functions

- Laasonen (Problem problem)

Protected Member Functions

- Vector build_r (Vector previous_step)

Additional Inherited Members

3.8.1 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.

3.8.2 Constructor & Destructor Documentation

3.8.2.1 Laasonen::Laasonen (Problem *problem*)

Default constructor.

3.8.3 Member Function Documentation

3.8.3.1 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.

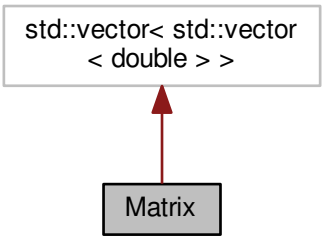
The documentation for this class was generated from the following files:

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

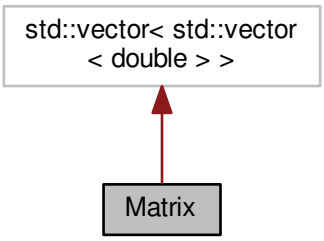
3.9 Matrix Class Reference

`#include <matrix.h>`

Inheritance diagram for Matrix:



Collaboration diagram for Matrix:



Public Member Functions

- Matrix ()
- Matrix (int Nrows, int Ncols)
- Matrix (const Matrix &m)
- int getNrows () const
- int getNcols () const
- void set_row (int index, Vector v)
- Matrix & operator= (const Matrix &m)
- bool operator== (const Matrix &m) const
- double one_norm () const
- double two_norm () const
- double uniform_norm () const
- Matrix operator∗ (const Matrix &a) const
- Vector operator∗ (const Vector &v) const
- Matrix transpose () const
- Matrix **mult** (const Matrix &a) const

Friends

- 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)

3.9.1 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

3.9.2 Constructor & Destructor Documentation

3.9.2.1 Matrix::Matrix ( )

Default constructor. Intialize an empty Matrix object

See also

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

3.9.2.2 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

3.9.2.3 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

3.9.3 Member Function Documentation

3.9.3.1 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

3.9.3.2 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

3.9.3.3 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

3.9.3.4 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

3.9.3.5 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

3.9.3.6 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

3.9.3.7 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

3.9.3.8 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")

3.9.3.9 Matrix Matrix::transpose ( ) const

public method that returns the transpose of the matrix. It returns the transpose of matrix

Returns

Matrix. matrix transpose

3.9.3.10 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

3.9.3.11 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

3.9.4 Friends And Related Function Documentation

3.9.4.1 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

3.9.4.2 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

3.9.4.3 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

3.9.4.4 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

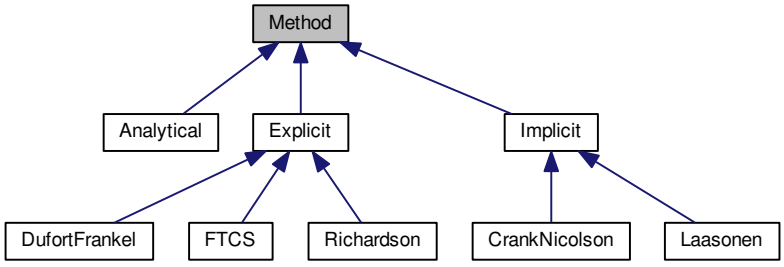
The documentation for this class was generated from the following files:

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

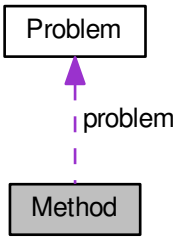
3.10 Method Class Reference

`#include <method.h>`

Inheritance diagram for Method:



Collaboration diagram for Method:



Public Member Functions

- Method ()
- Method (Problem problem)
- std::string get_name ()
- Matrix get_solution ()
- double get_deltat ()
- Vector get_xvalues ()
- double get_computational_time ()
- double get_two_norm ()
- void compute ()
- void **compute_norms** (Matrix analytical_matrix)
- virtual void compute_solution ()=0

Protected Attributes

- Problem problem
- std::string name
- double q

3.10.1 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

3.10.2 Constructor & Destructor Documentation

3.10.2.1 Method::Method ( )

Default constructor. Intialize a Method object

See also

Method(Problem problem)

3.10.2.2 Method::Method (Problem *problem*)

Alternate constructor. Initializes a Method with a given parabolic problem.

See also

Method()

3.10.3 Member Function Documentation

3.10.3.1 void Method::compute ( )

Normal public method. Keeps track of the time to compute a solution

3.10.3.2 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.

3.10.3.3 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.

3.10.3.4 double Method::get_deltat ( )

Normal public get method. get the time step of the solution

Returns

double. Solution time step.

3.10.3.5 std::string Method::get_name ( )

Normal public get method. get the method name

Returns

string. Method name.

3.10.3.6 Matrix Method::get_solution ( )

Normal public get method. get the solution grid

Returns

Matrix. Computed solution grid.

3.10.3.7 double Method::get_two_norm ( )

Normal public get method. get the second norm

Returns

double. Second norm value.

3.10.3.8 Vector Method::get_xvalues ( )

Normal public get method. get x values vector

Returns

Vector. x values Vector.

3.10.4 Member Data Documentation

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

Protected string name. Name of the method.

3.10.4.2 Problem Method::problem [protected]

Protected Problem problem. Space step of the solution.

3.10.4.3 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

3.11 Problem Class Reference

`#include <problem.h>`

Public Member Functions

- Problem ()
- Problem (double dt, double dx)
- unsigned int get_xsize ()
- unsigned int get_tsize ()
- double get_deltax ()
- double get_deltat ()
- Vector get_xvalues ()
- Vector get_tvalues ()
- Vector get_first_row ()
- Matrix * get_solution ()
- void set_time_step (Vector step, double time)
- void set_initial_conditions ()

3.11.1 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

3.11.2 Constructor & Destructor Documentation

3.11.2.1 Problem::Problem ( )

Default constructor. Intialize an empty Problem object

See also

Problem(double dt, double dx)

3.11.2.2 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")

3.11.3 Member Function Documentation

3.11.3.1 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.

3.11.3.2 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.

3.11.3.3 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.

3.11.3.4 Matrix * Problem::get_solution ( )

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

Returns

Matrix*. The solution solution.

3.11.3.5 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.

3.11.3.6 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.

3.11.3.7 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.

3.11.3.8 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.

3.11.3.9 void Problem::set_initial_conditions ( )

Normal public set method. set the problem initial boundaries.

3.11.3.10 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

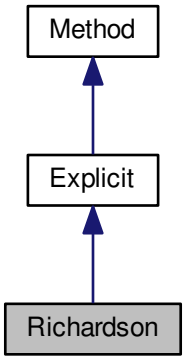
The documentation for this class was generated from the following files:

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

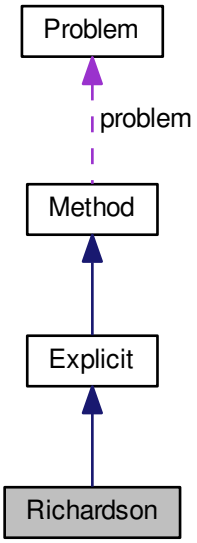
3.12 Richardson Class Reference

`#include <richardson.h>`

Inheritance diagram for Richardson:



Collaboration diagram for Richardson:



Public Member Functions

- Richardson (Problem problem)

Protected Member Functions

- Vector build_iteration (Vector current_step, Vector previous_step)

Additional Inherited Members

3.12.1 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

3.12.2 Constructor & Destructor Documentation

3.12.2.1 Richardson::Richardson (Problem *problem*)

Default constructor.

3.12.3 Member Function Documentation

3.12.3.1 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.

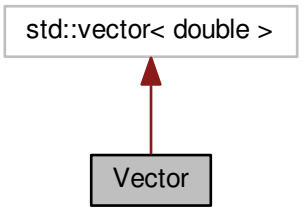
The documentation for this class was generated from the following files:

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

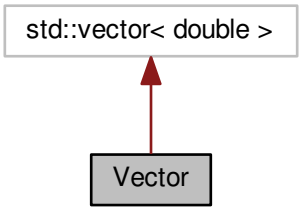
3.13 Vector Class Reference

`#include <vector.h>`

Inheritance diagram for Vector:



Collaboration diagram for Vector:



Public Member Functions

- Vector ()
- Vector (int Num)
- Vector (const Vector &v)
- Vector (std::vector< double > vec)
- Vector & operator= (const Vector &v)
- bool operator== (const Vector &v) const
- int getSize () const
- int find (double value)
- void push_front_back (double value)
- void push (double value)
- double one_norm () const
- double two_norm () const
- double uniform_norm () const

Friends

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

3.13.1 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

3.13.2 Constructor & Destructor Documentation

3.13.2.1 Vector::Vector ( )

Default constructor. Intialize an empty Vector object

See also

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

3.13.2.2 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

3.13.2.3 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)

3.13.2.4 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)

3.13.3 Member Function Documentation

3.13.3.1 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

3.13.3.2 int Vector::getSize ( ) const

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

Returns

int. the size of the vector

3.13.3.3 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

3.13.3.4 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

3.13.3.5 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

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

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

Parameters

value	Value to be pushed

3.13.3.7 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

3.13.3.8 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

3.13.3.9 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

3.13.4 Friends And Related Function Documentation

3.13.4.1 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

3.13.4.2 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

3.13.4.3 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

3.13.4.4 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

Index