

# My Project

Generated by Doxygen 1.8.6

Tue Aug 14 2018 10:56:33



# Contents

<b>1</b>	<b>Class Index</b>	<b>1</b>
1.1	Class List . . . . .	1
<b>2</b>	<b>File Index</b>	<b>3</b>
2.1	File List . . . . .	3
<b>3</b>	<b>Class Documentation</b>	<b>5</b>
3.1	ConstantGridSolver Class Reference . . . . .	5
3.1.1	Constructor & Destructor Documentation . . . . .	5
3.1.1.1	ConstantGridSolver . . . . .	5
3.1.1.2	~ConstantGridSolver . . . . .	5
3.1.1.3	ConstantGridSolver . . . . .	6
3.1.2	Member Function Documentation . . . . .	6
3.1.2.1	calculateEM . . . . .	6
3.1.2.2	calculateEP . . . . .	6
3.1.2.3	calculateS . . . . .	7
3.1.2.4	calculateT . . . . .	8
3.1.2.5	calculateU . . . . .	9
3.1.2.6	fwdIteration . . . . .	10
3.1.2.7	modifyCCnj . . . . .	11
3.1.2.8	saveS . . . . .	11
3.1.2.9	setParameters . . . . .	12
3.1.2.10	solveForEnergies . . . . .	12
3.2	NonconstantGridSolver Class Reference . . . . .	12
3.2.1	Member Function Documentation . . . . .	13
3.2.1.1	calculateEM . . . . .	13
3.2.1.2	calculateEP . . . . .	13
3.2.1.3	fwdIteration . . . . .	14
3.2.1.4	generateGrid . . . . .	15
3.2.1.5	modifyCCnj . . . . .	15
3.2.1.6	Q . . . . .	16
3.2.1.7	T . . . . .	16

3.2.1.8	U	17
3.3	Parameters Class Reference	18
3.3.1	Constructor & Destructor Documentation	19
3.3.1.1	Parameters	19
3.3.1.2	~Parameters	19
3.3.1.3	Parameters	19
3.3.2	Member Function Documentation	19
3.3.2.1	checkNumberOfRowsInFile	19
3.3.2.2	FRIEND_TEST	19
3.3.2.3	FRIEND_TEST	19
3.3.2.4	FRIEND_TEST	19
3.3.2.5	FRIEND_TEST	20
3.3.2.6	FRIEND_TEST	20
3.3.2.7	FRIEND_TEST	20
3.3.2.8	FRIEND_TEST	20
3.3.2.9	FRIEND_TEST	20
3.3.2.10	FRIEND_TEST	20
3.3.2.11	FRIEND_TEST	20
3.3.2.12	FRIEND_TEST	20
3.3.2.13	getB	20
3.3.2.14	getDx	20
3.3.2.15	getE	20
3.3.2.16	getGrid_points_per_lambda	20
3.3.2.17	getNChannels	20
3.3.2.18	getNE	20
3.3.2.19	getNSymmetries	20
3.3.2.20	getUnit	20
3.3.2.21	getV	20
3.3.2.22	getVMatrix	20
3.3.2.23	getXMax	21
3.3.2.24	getXMin	21
3.3.2.25	getXValues	21
3.3.2.26	ld	21
3.3.2.27	isOpen	21
3.3.2.28	kappa	21
3.3.2.29	kappa	21
3.3.2.30	lambda	22
3.3.2.31	loadB	22
3.3.2.32	loadE	22
3.3.2.33	loadParams	22

3.3.2.34	loadV	22
3.3.2.35	NX	23
3.3.2.36	requiredDx	23
3.3.2.37	setXValues	23
3.3.2.38	x	23
3.3.3	Friends And Related Function Documentation	23
3.3.3.1	Parameters_getV_Test	23
3.3.3.2	Parameters_isOpen_Test	23
3.3.3.3	Parameters_kappaDouble_Test	23
3.3.3.4	Parameters_kappaInt_Test	23
3.3.3.5	Parameters_lambda_Test	23
3.3.3.6	Parameters_loadV_Test	23
3.3.3.7	Parameters_requiredDX_Test	24
<b>4</b>	<b>File Documentation</b>	<b>25</b>
4.1	ConstantGridSolver.cpp File Reference	25
4.1.1	Detailed Description	25
4.2	ConstantGridSolver.h File Reference	25
4.2.1	Detailed Description	26
4.3	main.cpp File Reference	26
4.3.1	Function Documentation	27
4.3.1.1	main	27
4.3.1.2	read_file	27
4.4	NonconstantGridSolver.cpp File Reference	28
4.5	NonconstantGridSolver.h File Reference	28
4.6	Parameters.cpp File Reference	29
4.6.1	Detailed Description	29
4.7	Parameters.h File Reference	29
4.7.1	Detailed Description	30
4.8	Solver_z_liczeniem_psi.cpp File Reference	30
4.8.1	Detailed Description	31
<b>Index</b>		<b>32</b>



# Chapter 1

## Class Index

### 1.1 Class List

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

<a href="#">ConstantGridSolver</a>	5
<a href="#">NonconstantGridSolver</a>	12
<a href="#">Parameters</a>	18





## Chapter 2

# File Index

### 2.1 File List

Here is a list of all files with brief descriptions:

<a href="#">ConstantGridSolver.cpp</a>	
Definitions of <a href="#">ConstantGridSolver</a> class methods	25
<a href="#">ConstantGridSolver.h</a>	
Definition of <a href="#">ConstantGridSolver</a> class	25
<a href="#">main.cpp</a>	26
<a href="#">NonconstantGridSolver.cpp</a>	28
<a href="#">NonconstantGridSolver.h</a>	28
<a href="#">Parameters.cpp</a>	
Definitions of <a href="#">Parameters</a> class methods	29
<a href="#">Parameters.h</a>	
Definition of <a href="#">Parameters</a> class	29
<a href="#">Solver_z_liczeniem_psi.cpp</a>	
Definitions of Solver class methods	30



## Chapter 3

# Class Documentation

### 3.1 ConstantGridSolver Class Reference

```
#include <ConstantGridSolver.h>
```

#### Public Member Functions

- arma::cx\_mat [calculateT](#) (int j, double E) const  
*Calculates  $\mathbf{T}(x_j, E)$  matrix.*
- arma::cx\_mat [calculateU](#) (int j, double E)  
*Calculates  $\mathbf{U}(x_j, E)$  matrix.*
- arma::cx\_mat [calculateEP](#) (int j, double E)  
*Calculates  $\mathbf{E}^+(x_j, E)$  matrix.*
- arma::cx\_mat [calculateEM](#) (int j, double E)  
*Calculates  $\mathbf{E}^-(x_j, E)$  matrix.*
- void [modifyCCnj](#) (arma::cx\_mat &n1, arma::cx\_mat &n0, arma::cx\_mat &j1, arma::cx\_mat &j0, double E)  
*Modifies closed channels elements.*
- arma::cx\_mat [fwdIteration](#) (const arma::cx\_mat &B, double E)  
*Iterates Numerov algorithm forward up to  $N - 1$  and returns  $\mathbf{R}_{N-1}$  matrix for a given energy.*
- arma::cx\_mat [calculateS](#) (const arma::cx\_mat R\_N, double E)  
*Calculates  $\mathbf{S}$  matrix for given  $\mathbf{R}_{N-1}$ .*
- void [saveS](#) (const arma::cx\_mat &S, const int E, const std::string directory)  
*Saves  $\mathbf{S}$  matrix (Im and Re part separately).*
- void [setParameters](#) (const [Parameters](#) &parameters)
- [ConstantGridSolver](#) ()=default
- [~ConstantGridSolver](#) ()=default
- [ConstantGridSolver](#) (const [Parameters](#) &params)  
*Constructor.*
- void [solveForEnergies](#) (std::string directory)  
*Performs Numerov calculations for a given set of parameters for all energies.*

#### 3.1.1 Constructor & Destructor Documentation

3.1.1.1 [ConstantGridSolver::ConstantGridSolver \( \)](#) [default]

3.1.1.2 [ConstantGridSolver::~~ConstantGridSolver \( \)](#) [default]

### 3.1.1.3 ConstantGridSolver::ConstantGridSolver ( const Parameters & params ) [explicit]

Constructor.

## 3.1.2 Member Function Documentation

### 3.1.2.1 arma::cx\_mat ConstantGridSolver::calculateEM ( int j, double E )

Calculates  $\mathbf{E}^-(x_j, E)$  matrix.

This method calculates  $\mathbf{E}^-$  matrix for for a given point  $x_j$  on the grid and given energy. The matrix is diagonal and its elements are calculated the following way:

- $\mathbf{E}_{n,n}^-(x_j, E) = \exp(-ikx_j)$  if channel  $n$  is open
- $\mathbf{E}_{n,n}^-(x_j, E) = \cosh(kx_j)$  if channel  $n$  is closed
- $\mathbf{E}_{n,m}^-(x_j, E) = 0$  for  $n \neq m$

#### Parameters

$j$	- index of the value on the grid
$E$	- energy

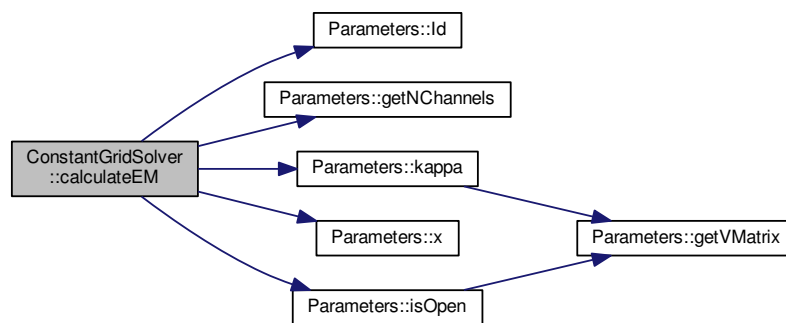
#### Returns

$\mathbf{E}^-(x_j, E)$

#### Exceptions

<code>std::invalid_argument</code>	if j is wrong
------------------------------------	---------------

Here is the call graph for this function:



### 3.1.2.2 arma::cx\_mat ConstantGridSolver::calculateEP ( int j, double E )

Calculates  $\mathbf{E}^+(x_j, E)$  matrix.

This method calculates  $\mathbf{E}^+$  matrix for for a given point  $x_j$  on the grid and given energy. The matrix is diagonal and its elements are calculated the following way:

- $\mathbf{E}_{n,n}^+(x_j, E) = \exp(ikx_j)$  if channel  $n$  is open

- $\mathbf{E}_{n,n}^+(x_j, E) = \sinh(kx_j)$  if channel  $n$  is closed
- $\mathbf{E}_{n,m}^+(x_j, E) = 0$  for  $n \neq m$

**Parameters**

$j$	- index of the value on the grid
$E$	- energy

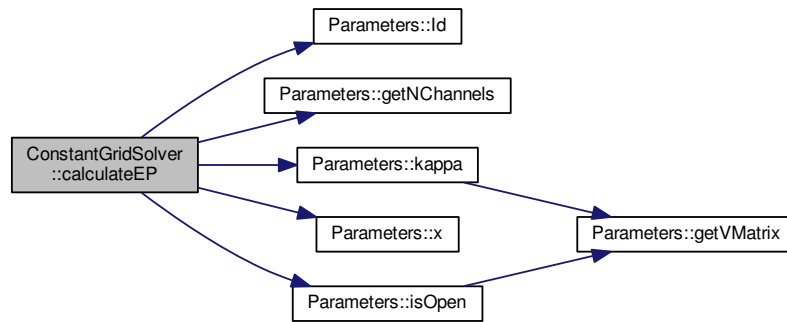
**Returns**

$\mathbf{E}^+(x_j, E)$

**Exceptions**

<code>std::invalid_argument</code>	if $j$ is wrong
------------------------------------	-----------------

Here is the call graph for this function:



### 3.1.2.3 arma::cx\_mat ConstantGridSolver::calculateS ( const arma::cx\_mat $R_N$ , double $E$ )

Calculates  $\mathbf{S}$  matrix for given  $\mathbf{R}_{N-1}$ .

This method calculates the scattering matrix  $\mathbf{S}(E)$ . Its value is given by

$$\mathbf{S} = (\mathbf{R}_{N-1} \mathbf{e}_{N-1}^+ - \mathbf{e}_N^+) - 1 (\mathbf{R}_{N-1} \mathbf{e}_{N-1}^- - \mathbf{e}_N^-) \quad (3.1)$$

where  $\mathbf{e}_i^\pm = (\mathbf{I} - \mathbf{T}_i) \mathbf{E}_i^\pm$ .

**Parameters**

$R_N$	- $\mathbf{R}_{N-1}$
$E$	- energy

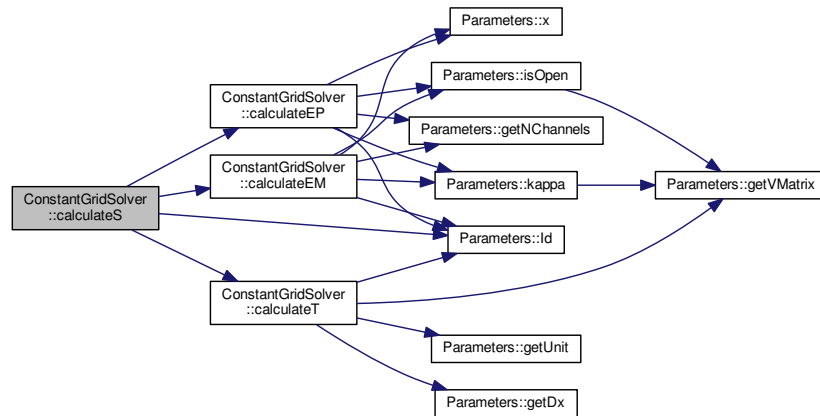
**Returns**

$\mathbf{S}$

## Exceptions

<code>std::runtime_error</code>	if there is a problem with calculating
---------------------------------	--

Here is the call graph for this function:



### 3.1.2.4 arma::cx\_mat ConstantGridSolver::calculateT ( int $j$ , double $E$ ) const

Calculates  $\mathbf{T}(x_j, E)$  matrix.

This method calculates  $\mathbf{T}$  matrix for a given point  $x_j$  on the grid and given energy according to the formula:

$$\mathbf{T}_j = -\frac{dx}{12} \mathbf{Q}_j. \quad (3.2)$$

## Parameters

$j$	- index of the value on the grid
$E$	- energy

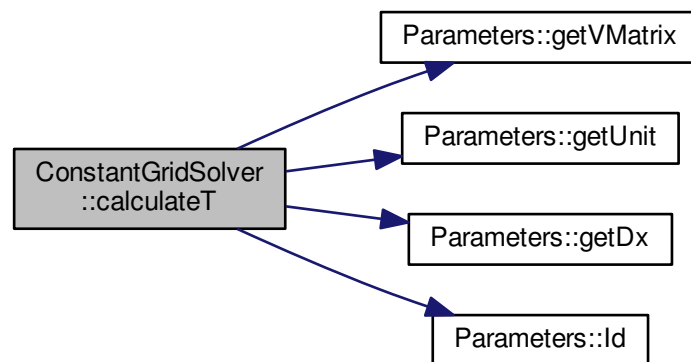
## Returns

$$\mathbf{T}(x_j, E)$$

## Exceptions

<code>std::invalid_argument</code>	if the index $j$ is wrong
------------------------------------	---------------------------

Here is the call graph for this function:



### 3.1.2.5 arma::cx\_mat ConstantGridSolver::calculateU ( int $j$ , double $E$ )

Calculates  $\mathbf{U}(x_j, E)$  matrix.

This method calculates  $\mathbf{U}$  matrix for given index  $j$  using the set of parameters provided to the [ConstantGridSolver](#) object according to the following formula:

$$\mathbf{U}_j = 12(\mathbf{I} - \mathbf{T}_j)^{-1} - 10\mathbf{I}. \quad (3.3)$$

## Parameters

in	$j$	- index on the grid of x value
in	$E$	- energy value

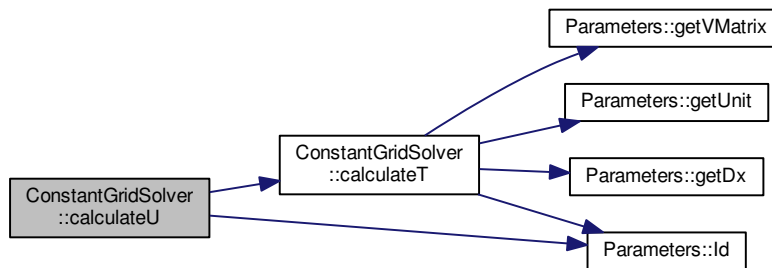
## Returns

 $\mathbf{U}(x_j, E)$ 

## Exceptions

<code>std::invalid_argument</code>	if $x_j$ does not exist
------------------------------------	-------------------------

Here is the call graph for this function:



### 3.1.2.6 arma::cx\_mat ConstantGridSolver::fwdIteration ( const arma::cx\_mat & $B$ , double $E$ )

Iterates Numerov algorithm forward up to  $N - 1$  and returns  $\mathbf{R}_{N-1}$  matrix for a given energy.

This method performs the Numerov iteration for a given energy for a case of some particular symmetry.

The initial value  $\mathbf{R}_0^{-1}$ :

- $\mathbf{R}_0^{-1} = \mathbf{0}$  if no symmetries
- $\mathbf{R}_0^{-1} = \mathbf{U}_0^{-1}(\mathbf{I} + \mathbf{B})$  if the symmetry is described by  $\mathbf{B}$

Every value depends on the previous one:  $\mathbf{R}_j = \mathbf{U}_j - \mathbf{R}_{j-1}^{-1}$ .

## Parameters

in	$B$	- $\mathbf{B}$ matrix to calculate the initial value
in	$E$	- energy



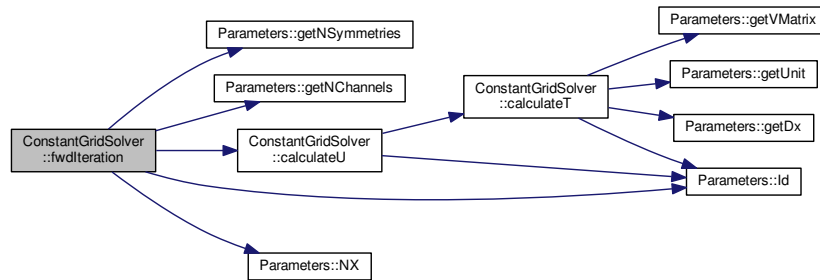
Returns

$$\mathbf{R}_{N-1}$$

Exceptions

<code>std::invalid_argument</code>	if $\mathbf{U}_i$ cannot be calculated for given iteration $i$
<code>std::runtime_error</code>	

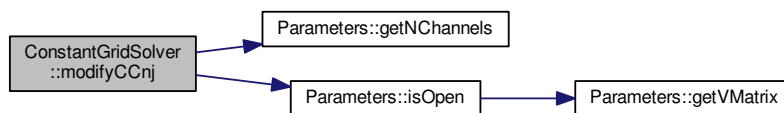
Here is the call graph for this function:



3.1.2.7 `void ConstantGridSolver::modifyCCnj ( arma::cx_mat & n1, arma::cx_mat & n0, arma::cx_mat & j1, arma::cx_mat & j0, double E )`

Modifies closed channels elements.

Here is the call graph for this function:



3.1.2.8 `void ConstantGridSolver::saveS ( const arma::cx_mat & S, const int E, const std::string directory )`

Saves  $\mathbf{S}$  matrix (Im and Re part separately).

This method saves the scattering matrix in a given directory. The real and imaginary part of  $\mathbf{S}$  are saved in separate files.

Paths:

$Re(\mathbf{S})$ : directory/re\_SE.dat (E is the value of the energy)  $Im(\mathbf{S})$ : directory/im\_SE.dat (E is the value of the energy)

Parameters

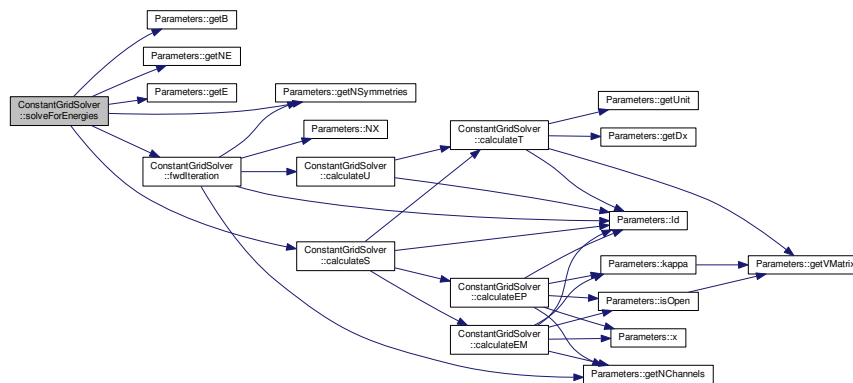
$S$	- scattering matrix to be saved
$E$	- energy
<i>directory</i>	- where to save the files

3.1.2.9 void ConstantGridSolver::setParameters ( const Parameters & *parameters* ) [inline]

3.1.2.10 void ConstantGridSolver::solveForEnergies ( std::string *directory* )

Performs Numerov calculations for a given set of parameters for all energies.

Here is the call graph for this function:



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

- [ConstantGridSolver.h](#)
- [ConstantGridSolver.cpp](#)

## 3.2 NonconstantGridSolver Class Reference

```
#include <NonconstantGridSolver.h>
```

### Public Member Functions

- arma::cx\_mat **Q** (double x, double E)  
calculates **Q** matrix.
- arma::cx\_mat **T** (double x, double E, double dx)  
calculates **T** matrix for given x (uses interpolation from [Parameters](#)).
- arma::cx\_mat **U** (double x, double E, double dx)  
Calculates **U** matrix for given x (uses interpolation from [Parameters](#)).
- void **modifyCCnj** (arma::cx\_mat &n1, arma::cx\_mat &n0, arma::cx\_mat &j1, arma::cx\_mat &j0, double E)  
Modifies closed channels elements.
- arma::cx\_mat **calculateEP** (double x, double E)  
Calculates  $\mathbf{E}^+(x, E)$  matrix.
- arma::cx\_mat **calculateEM** (double x, double E)  
Calculates  $\mathbf{E}^-(x, E)$  matrix.
- std::vector< double > **generateGrid** (double E)

- arma::cx\_mat [fwdIteration](#) (const arma::cx\_mat &B, double E, std::vector< double > grid)  
*Iterates Numerov algorithm forward up to  $N - 1$ .*

### 3.2.1 Member Function Documentation

#### 3.2.1.1 arma::cx\_mat NonconstantGridSolver::calculateEM ( double x, double E )

Calculates  $\mathbf{E}^-(x, E)$  matrix.

This method calculates  $\mathbf{E}^-$  matrix for a given point  $x$  and given energy. The matrix is diagonal and its elements are calculated the following way:

- $\mathbf{E}_{n,n}^-(x, E) = \exp(-ikx)$  if channel  $n$  is open
- $\mathbf{E}_{n,n}^-(x, E) = \cosh(kx)$  if channel  $n$  is closed
- $\mathbf{E}_{n,m}^-(x, E) = 0$  for  $n \neq m$

##### Parameters

$x$	
$E$	- energy

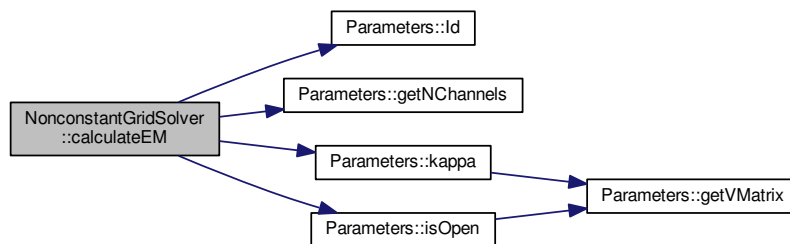
##### Returns

$\mathbf{E}^-(x, E)$

##### Exceptions

<code>std::invalid_argument</code>	if j is wrong
------------------------------------	---------------

Here is the call graph for this function:



#### 3.2.1.2 arma::cx\_mat NonconstantGridSolver::calculateEP ( double x, double E )

Calculates  $\mathbf{E}^+(x, E)$  matrix.

This method calculates  $\mathbf{E}^+$  matrix for a given point  $x$  and given energy. The matrix is diagonal and its elements are calculated the following way:

- $\mathbf{E}_{n,n}^+(x, E) = \exp(ikx)$  if channel  $n$  is open
- $\mathbf{E}_{n,n}^+(x, E) = \sinh(kx)$  if channel  $n$  is closed
- $\mathbf{E}_{n,m}^+(x, E) = 0$  for  $n \neq m$

## Parameters

$x$	
$E$	- energy

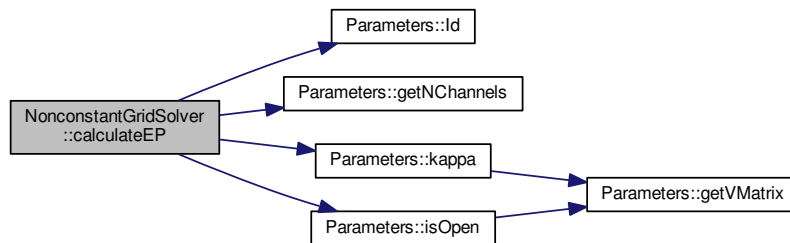
## Returns

$\mathbf{E}^+(x, E)$

## Exceptions

<code>std::invalid_argument</code>	if $x$ is wrong
------------------------------------	-----------------

Here is the call graph for this function:



### 3.2.1.3 arma::cx\_mat NonconstantGridSolver::fwdIteration ( const arma::cx\_mat & $B$ , double $E$ , std::vector< double > $grid$ )

Iterates Numerov algorithm forward up to  $N - 1$ .

This method performs the Numerov iteration for a given energy for a case of some particular symmetry.

The initial value  $\mathbf{R}_0^{-1}$ :

- $\mathbf{R}_0^{-1} = \mathbf{0}$  if no symmetries
- $\mathbf{R}_0^{-1} = \mathbf{U}_0^{-1}(\mathbf{I} + \mathbf{B})$  if the symmetry is described by  $\mathbf{B}$

## Parameters

in	$B$	- $\mathbf{B}$ matrix to calculate the initial value
in	$E$	- energy

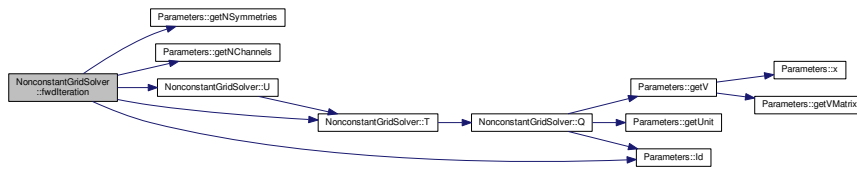
## Returns

$\mathbf{R}_{N-1}$

## Exceptions

<code>std::invalid_argument</code>	if $\mathbf{U}_i$ cannot be calculated for given iteration $i$
<code>std::runtime_error</code>	

Here is the call graph for this function:



### 3.2.1.4 `std::vector< double > NonconstantGridSolver::generateGrid ( double $E$ )`

Calculates the grid points.

This method generates grid points for Numerov calculations based on the energy  $E$  and potential.

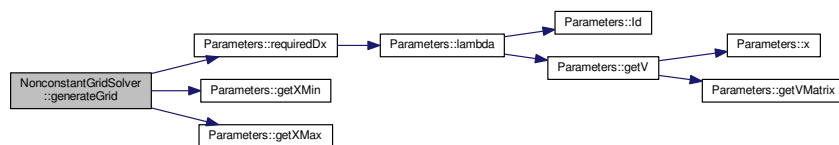
**Parameters**

$E$	
-----	--

**Returns**

$x$

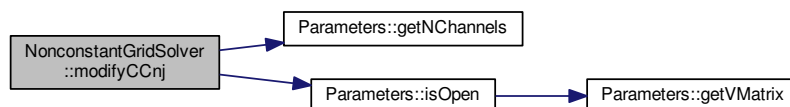
Here is the call graph for this function:



### 3.2.1.5 `void NonconstantGridSolver::modifyCCnj ( arma::cx_mat & $n1$ , arma::cx_mat & $n0$ , arma::cx_mat & $j1$ , arma::cx_mat & $j0$ , double $E$ )`

Modifies closed channels elements.

Here is the call graph for this function:



### 3.2.1.6 arma::cx\_mat NonconstantGridSolver::Q ( double $x$ , double $E$ )

calculates  $\mathbf{Q}$  matrix.

This method calculates  $\mathbf{Q}$  matrix for a given point  $x$  and given energy according to the formula:

$$\mathbf{Q}(x) = -\frac{1}{unit} (\mathbf{V}(x) - E\mathbf{I}). \quad (3.4)$$

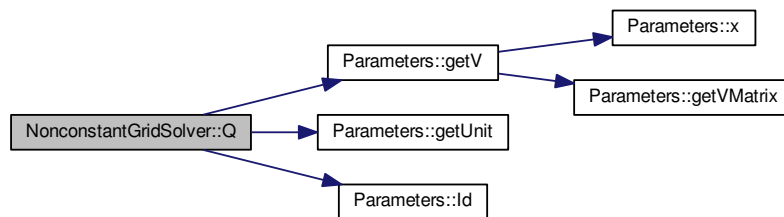
#### Parameters

$x$	
$E$	- energy

#### Returns

$\mathbf{Q}(x, E)$

Here is the call graph for this function:



### 3.2.1.7 arma::cx\_mat NonconstantGridSolver::T ( double $x$ , double $E$ , double $dx$ )

calculates  $\mathbf{T}$  matrix for given  $x$  (uses interpolation from [Parameters](#)).

This method calculates  $\mathbf{T}$  matrix for a given point  $x$  and given energy according to the formula:

$$\mathbf{T}(x) = -\frac{dx}{12} \mathbf{Q}(x). \quad (3.5)$$

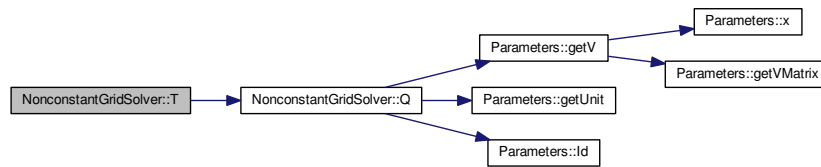
#### Parameters

$x$	
$E$	- energy
$dx$	- the distance from the next point on the grid

## Returns

$$\mathbf{T}(x, E)$$

Here is the call graph for this function:



### 3.2.1.8 arma::cx\_mat NonconstantGridSolver::U ( double $x$ , double $E$ , double $dx$ )

Calculates  $\mathbf{U}$  matrix for given  $x$  (uses interpolation from [Parameters](#)).

This method calculates  $\mathbf{U}$  matrix for given  $x$  using the set of parameters provided to the [NonconstantGridSolver](#) object according to the following formula:

$$\mathbf{U}(x) = 12(\mathbf{I} - \mathbf{T}(x))^{-1} - 10\mathbf{I}. \quad (3.6)$$

## Parameters

in	$x$	
in	$E$	- energy value
		.
	$dx$	- the distance from the next point on the grid

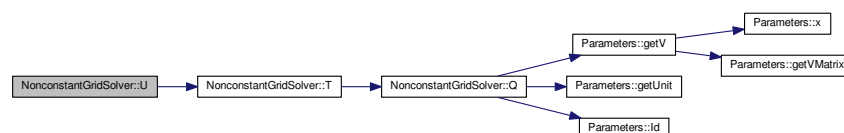
## Returns

$$\mathbf{U}(x, E)$$

## Exceptions

<code>std::invalid_argument</code>	if $x$ is out of range
------------------------------------	------------------------

Here is the call graph for this function:



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

- [NonconstantGridSolver.h](#)
- [NonconstantGridSolver.cpp](#)

### 3.3 Parameters Class Reference

```
#include <Parameters.h>
```

#### Public Member Functions

- void [loadParams](#) (std::string="Params.txt")  
*Reading the values of parameters from the file generated in Mathematica.*
- void [setXValues](#) ()  
*Setting xValues.*
- [FRIEND\\_TEST](#) (ParametersInputTest, setXValues\_failsIfXMaxLessOrEqualXMin)
- [FRIEND\\_TEST](#) (ParametersInputTest, setXValues\_failsIfInvalidDX)
- [FRIEND\\_TEST](#) (ParametersInputTest, setXValues\_failsIfInvalidCombinationOfXMinXMaxDx)
- [FRIEND\\_TEST](#) (ParametersInputTest, setXValues\_worksGoodForCorrectValues)
- void [loadE](#) (std::string filename="E.dat")  
*Reading the values of energies from the file generated in Mathematica.*
- void [loadV](#) (std::string filename="V.dat")  
*Reading the values of V from the file generated in Mathematica.*
- [FRIEND\\_TEST](#) (Parameters\_loadV\_Test, failsForIncorrectNumberOfRows)
- [FRIEND\\_TEST](#) (Parameters\_loadV\_Test, worksGoodForGoodFileOneChannel)
- [FRIEND\\_TEST](#) (Parameters\_loadV\_Test, worksGoodForGoodFileTwoChannels)
- void [loadB](#) (std::string filename="B")  
*Reading the values of B from the file generated in Mathematica.*
- [FRIEND\\_TEST](#) (ParametersInputTest, loadB\_failsIfAnyFileDoesNotExistAndPositiveNSymmetries)
- [FRIEND\\_TEST](#) (ParametersInputTest, loadB\_worksGoodForGoodFilesOneChannel)
- [FRIEND\\_TEST](#) (ParametersInputTest, loadB\_failsIfAnyFileIsIncorrect)
- bool [checkNumberOfRowsInFile](#) (std::string filename, const int required\_number\_of\_columns)
- [Parameters](#) ()=default
- [~Parameters](#) ()=default
- [Parameters](#) (std::vector< std::string > filenames)  
*From a given directory takes all the needed values and creates [Parameters](#) object.*
- arma::cx\_mat [getVMatrix](#) (int) const  
*V matrix for a given x<sub>i</sub>.*
- double [getE](#) (int) const
- int [NX](#) () const
- double [getXMin](#) () const
- double [getXMax](#) () const
- double [getDx](#) () const
- double [x](#) (int i) const
- [FRIEND\\_TEST](#) (ParametersOutputTest, x\_worksCorrectForNegativeIndices)
- double [getUnit](#) () const
- int [getNChannels](#) () const
- int [getNE](#) () const
- arma::cx\_mat [getB](#) (int i) const
- int [getNSymmetries](#) () const
- int [getGrid\\_points\\_per\\_lambda](#) () const
- arma::cx\_mat [ld](#) () const
- bool [isOpen](#) (int nChannel, double energy) const  
*Check if the channel is open.*
- double [kappa](#) (int n1, int n2, int i, double E) const
- double [kappa](#) (int n1, int n2, double x, double E) const
- arma::cx\_mat [getV](#) (double x) const



*Linear interpolation of  $V$  (works also for  $V$  given on non-constant grid if needed)*

- double [lambda](#) (double  $x$ , double  $E$ ) const  
*de Broglie length for a given potential and  $x$*
- double [requiredDx](#) (double  $x$ , double  $E$ ) const
- const std::vector< double > & [getXValues](#) () const

## Friends

- class [Parameters\\_loadV\\_Test](#)
- class [Parameters\\_isOpen\\_Test](#)
- class [Parameters\\_kappaInt\\_Test](#)
- class [Parameters\\_kappaDouble\\_Test](#)
- class [Parameters\\_getV\\_Test](#)
- class [Parameters\\_lambda\\_Test](#)
- class [Parameters\\_requiredDX\\_Test](#)

## 3.3.1 Constructor & Destructor Documentation

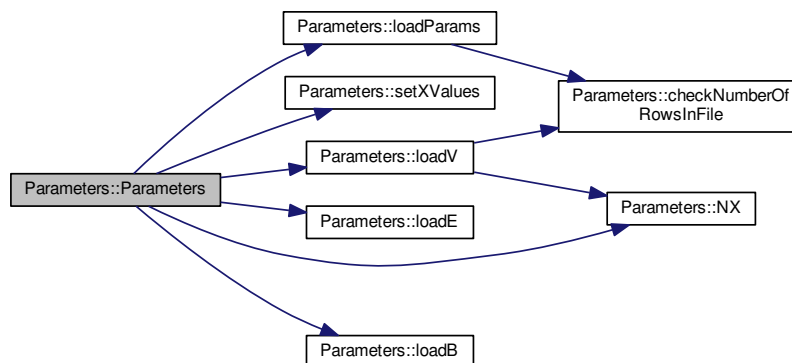
3.3.1.1 `Parameters::Parameters ( )` [default]

3.3.1.2 `Parameters::~~Parameters ( )` [default]

3.3.1.3 `Parameters::Parameters ( std::vector< std::string > filenames )` [explicit]

From a given directory takes all the needed values and creates [Parameters](#) object.

Here is the call graph for this function:



## 3.3.2 Member Function Documentation

3.3.2.1 `bool Parameters::checkNumberOfRowsInFile ( std::string filename, const int required_number_of_columns )`

3.3.2.2 `Parameters::FRIEND_TEST ( ParametersInputTest, setXValues_failsIfXMaxLessOrEqualXMin )`

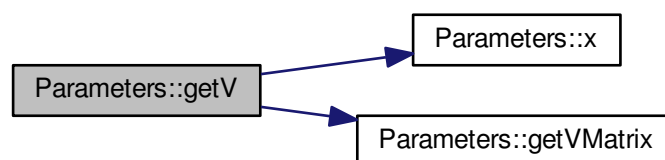
3.3.2.3 `Parameters::FRIEND_TEST ( ParametersInputTest, setXValues_failsIfInvalidDX )`

3.3.2.4 `Parameters::FRIEND_TEST ( ParametersInputTest, setXValues_failsIfInvalidCombinationOfXMinXMaxDx )`

- 3.3.2.5 `Parameters::FRIEND_TEST ( ParametersInputTest , setXValues_worksGoodForCorrectValues )`
- 3.3.2.6 `Parameters::FRIEND_TEST ( Parameters_loadV_Test , failsForIncorrectNumberOfRows )`
- 3.3.2.7 `Parameters::FRIEND_TEST ( Parameters_loadV_Test , worksGoodForGoodFileOneChannel )`
- 3.3.2.8 `Parameters::FRIEND_TEST ( Parameters_loadV_Test , worksGoodForGoodFileTwoChannels )`
- 3.3.2.9 `Parameters::FRIEND_TEST ( ParametersInputTest , loadB_failsIfAnyFileDoesNotExistAndPositiveNSymmetries )`
- 3.3.2.10 `Parameters::FRIEND_TEST ( ParametersInputTest , loadB_worksGoodForGoodFilesOneChannel )`
- 3.3.2.11 `Parameters::FRIEND_TEST ( ParametersInputTest , loadB_failsIfAnyFilesIncorrect )`
- 3.3.2.12 `Parameters::FRIEND_TEST ( ParametersOutputTest , x_worksCorrectForNegativeIndices )`
- 3.3.2.13 `arma::cx_mat Parameters::getB ( int i ) const [inline]`
- 3.3.2.14 `double Parameters::getDx ( ) const [inline]`
- 3.3.2.15 `double Parameters::getE ( int i ) const`
- 3.3.2.16 `int Parameters::getGrid_points_per_lambda ( ) const [inline]`
- 3.3.2.17 `int Parameters::getNChannels ( ) const [inline]`
- 3.3.2.18 `int Parameters::getNE ( ) const [inline]`
- 3.3.2.19 `int Parameters::getNSymmetries ( ) const [inline]`
- 3.3.2.20 `double Parameters::getUnit ( ) const [inline]`
- 3.3.2.21 `arma::cx_mat Parameters::getV ( double x ) const`

Linear interpolation of V (works also for V given on non-constant grid if needed)

Here is the call graph for this function:



- 3.3.2.22 `arma::cx_mat Parameters::getVMatrix ( int i ) const`

V matrix for a given `x_i`.

3.3.2.23 `double Parameters::getXMax ( ) const` `[inline]`

3.3.2.24 `double Parameters::getXMin ( ) const` `[inline]`

3.3.2.25 `const std::vector<double>& Parameters::getXValues ( ) const` `[inline]`

3.3.2.26 `arma::cx_mat Parameters::ld ( ) const` `[inline]`

3.3.2.27 `bool Parameters::isOpen ( int nChannel, double energy ) const`

Check if the channel is open.

Here is the call graph for this function:



3.3.2.28 `double Parameters::kappa ( int n1, int n2, int i, double E ) const`

Here is the call graph for this function:



3.3.2.29 `double Parameters::kappa ( int n1, int n2, double x, double E ) const`

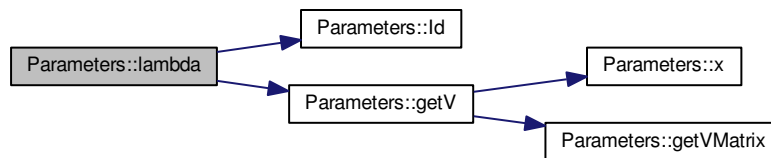
Here is the call graph for this function:



### 3.3.2.30 `double Parameters::lambda ( double $x$ , double $E$ ) const`

de Broglie length for a given potential and  $x$

Here is the call graph for this function:



### 3.3.2.31 `void Parameters::loadB ( std::string filename = "B" )`

Reading the values of  $B$  from the file generated in Mathematica.

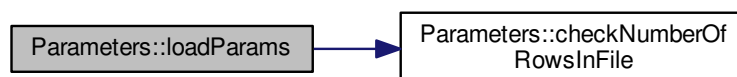
### 3.3.2.32 `void Parameters::loadE ( std::string filename = "E.dat" )`

Reading the values of energies from the file generated in Mathematica.

### 3.3.2.33 `void Parameters::loadParams ( std::string filename = "Params.txt" )`

Reading the values of parameters from the file generated in Mathematica.

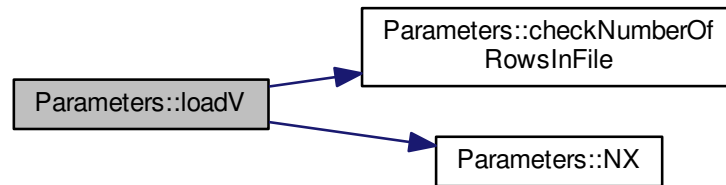
Here is the call graph for this function:



### 3.3.2.34 `void Parameters::loadV ( std::string filename = "V.dat" )`

Reading the values of  $V$  from the file generated in Mathematica.

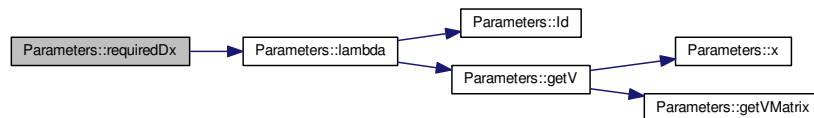
Here is the call graph for this function:



3.3.2.35 `int Parameters::NX ( ) const`

3.3.2.36 `double Parameters::requiredDx ( double x, double E ) const`

Here is the call graph for this function:



3.3.2.37 `void Parameters::setXValues ( )`

Setting xValues.

3.3.2.38 `double Parameters::x ( int i ) const` `[inline]`

### 3.3.3 Friends And Related Function Documentation

3.3.3.1 `friend class Parameters_getV_Test` `[friend]`

3.3.3.2 `friend class Parameters_isOpen_Test` `[friend]`

3.3.3.3 `friend class Parameters_kappaDouble_Test` `[friend]`

3.3.3.4 `friend class Parameters_kappaInt_Test` `[friend]`

3.3.3.5 `friend class Parameters_lambda_Test` `[friend]`

3.3.3.6 `friend class Parameters_loadV_Test` `[friend]`

### 3.3.3.7 friend class Parameters\_requiredDX\_Test [friend]

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

- [Parameters.h](#)
- [Parameters.cpp](#)

## Chapter 4

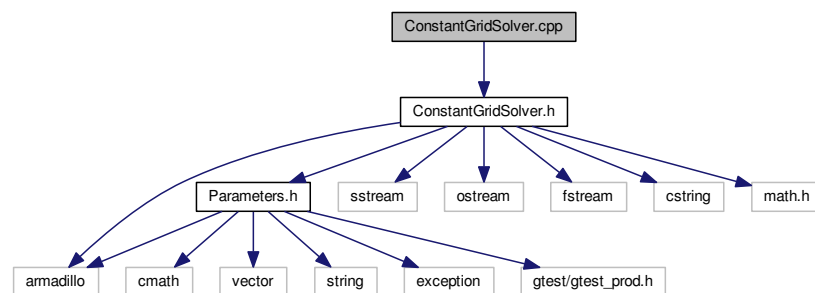
# File Documentation

### 4.1 ConstantGridSolver.cpp File Reference

Definitions of [ConstantGridSolver](#) class methods.

```
#include "ConstantGridSolver.h"
```

Include dependency graph for ConstantGridSolver.cpp:



#### 4.1.1 Detailed Description

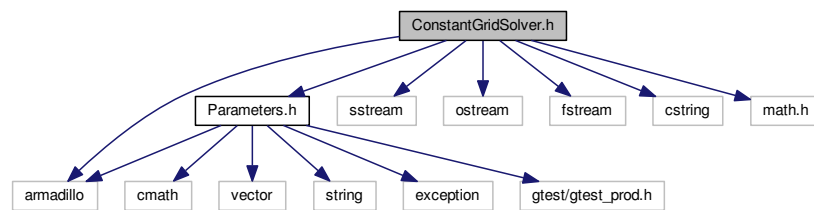
Definitions of [ConstantGridSolver](#) class methods.

### 4.2 ConstantGridSolver.h File Reference

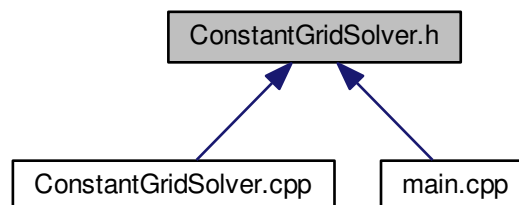
Definition of [ConstantGridSolver](#) class.

```
#include "armadillo"  
#include "Parameters.h"  
#include <sstream>  
#include <ostream>  
#include <fstream>  
#include <cstring>  
#include <math.h>
```

Include dependency graph for ConstantGridSolver.h:



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



## Classes

- class [ConstantGridSolver](#)

### 4.2.1 Detailed Description

Definition of [ConstantGridSolver](#) class. This file contains a definition of [ConstantGridSolver](#) class, performing the calculations for a given set of parameters ([Parameters](#) object).

## 4.3 main.cpp File Reference

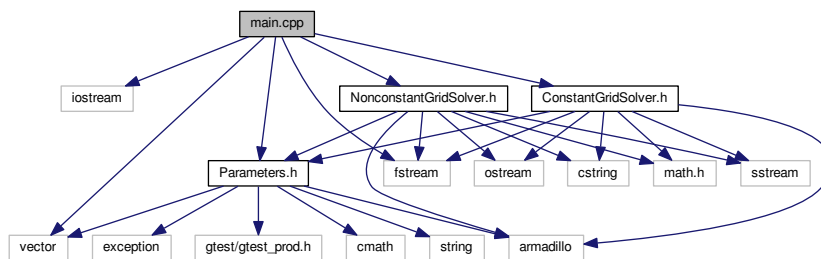
```

#include <iostream>
#include <vector>
#include <fstream>
#include "Parameters.h"
#include "ConstantGridSolver.h"
#include "NonconstantGridSolver.h"

```



Include dependency graph for main.cpp:



## Functions

- `std::vector< std::string > read_file (std::string filename)`
- `int main ()`

### 4.3.1 Function Documentation

#### 4.3.1.1 `int main ( )`

Here is the call graph for this function:

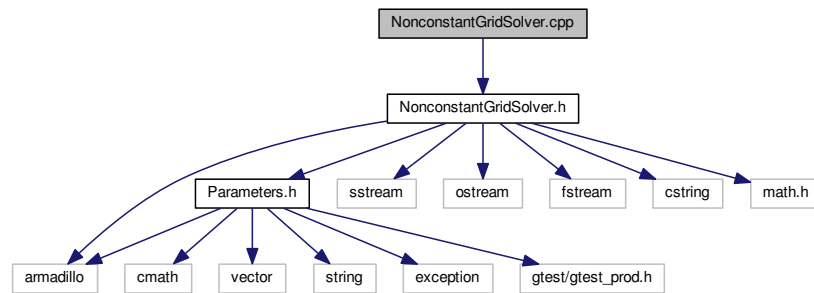


#### 4.3.1.2 `std::vector<std::string> read_file ( std::string filename )`

## 4.4 NonconstantGridSolver.cpp File Reference

```
#include "NonconstantGridSolver.h"
```

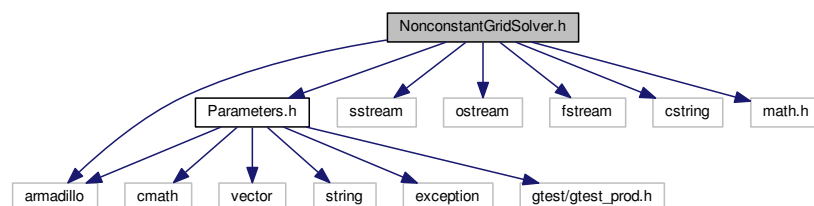
Include dependency graph for NonconstantGridSolver.cpp:



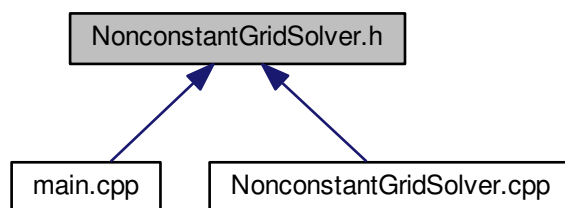
## 4.5 NonconstantGridSolver.h File Reference

```
#include "armadillo"
#include "Parameters.h"
#include <sstream>
#include <ostream>
#include <fstream>
#include <cstring>
#include <math.h>
```

Include dependency graph for NonconstantGridSolver.h:



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



## Classes

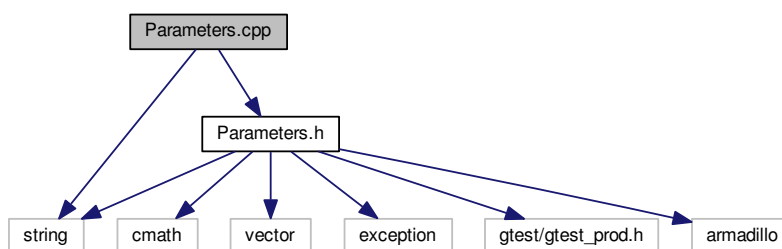
- class [NonconstantGridSolver](#)

## 4.6 Parameters.cpp File Reference

Definitions of [Parameters](#) class methods.

```
#include <string>
#include "Parameters.h"
```

Include dependency graph for Parameters.cpp:



### 4.6.1 Detailed Description

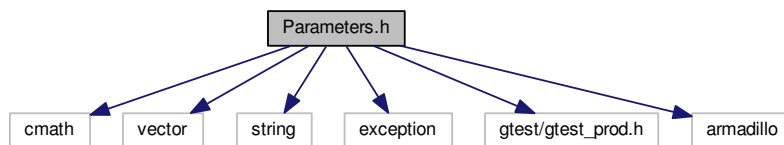
Definitions of [Parameters](#) class methods.

## 4.7 Parameters.h File Reference

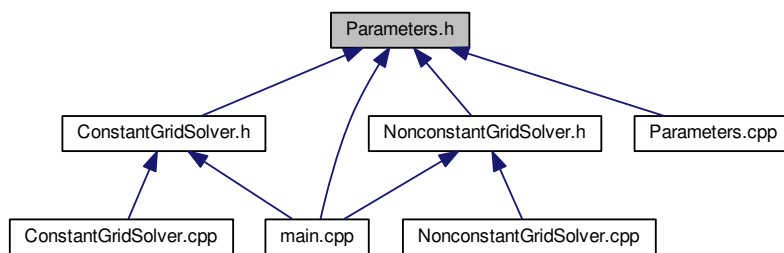
Definition of [Parameters](#) class.

```
#include <cmath>
#include <vector>
#include <string>
#include <exception>
#include <gtest/gtest_prod.h>
#include "armadillo"
```

Include dependency graph for Parameters.h:



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



## Classes

- class [Parameters](#)

### 4.7.1 Detailed Description

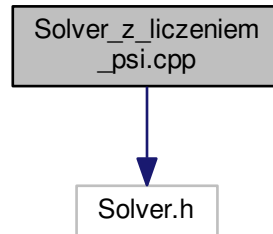
Definition of [Parameters](#) class. This file contains the definition of [Parameters](#) class.

## 4.8 Solver\_z\_liczeniem\_psi.cpp File Reference

Definitions of Solver class methods.

```
#include "Solver.h"
```

Include dependency graph for Solver\_z\_liczeniem\_psi.cpp:



#### 4.8.1 Detailed Description

Definitions of Solver class methods.

# Index

- ~ConstantGridSolver
  - ConstantGridSolver, [5](#)
- ~Parameters
  - Parameters, [19](#)
- calculateEM
  - ConstantGridSolver, [6](#)
  - NonconstantGridSolver, [13](#)
- calculateEP
  - ConstantGridSolver, [6](#)
  - NonconstantGridSolver, [13](#)
- calculateS
  - ConstantGridSolver, [7](#)
- calculateT
  - ConstantGridSolver, [8](#)
- calculateU
  - ConstantGridSolver, [9](#)
- checkNumberOfRowsInFile
  - Parameters, [19](#)
- ConstantGridSolver, [5](#)
  - ~ConstantGridSolver, [5](#)
  - calculateEM, [6](#)
  - calculateEP, [6](#)
  - calculateS, [7](#)
  - calculateT, [8](#)
  - calculateU, [9](#)
  - ConstantGridSolver, [5](#)
  - ConstantGridSolver, [5](#)
  - fwdIteration, [10](#)
  - modifyCCnj, [11](#)
  - saveS, [11](#)
  - setParameters, [12](#)
  - solveForEnergies, [12](#)
- ConstantGridSolver.cpp, [25](#)
- ConstantGridSolver.h, [25](#)
- FRIEND\_TEST
  - Parameters, [19](#), [20](#)
- fwdIteration
  - ConstantGridSolver, [10](#)
  - NonconstantGridSolver, [14](#)
- generateGrid
  - NonconstantGridSolver, [15](#)
- getB
  - Parameters, [20](#)
- getDx
  - Parameters, [20](#)
- getE
  - Parameters, [20](#)
- getGrid\_points\_per\_lambda
  - Parameters, [20](#)
- getNChannels
  - Parameters, [20](#)
- getNE
  - Parameters, [20](#)
- getNSymmetries
  - Parameters, [20](#)
- getUnit
  - Parameters, [20](#)
- getV
  - Parameters, [20](#)
- getVMatrix
  - Parameters, [20](#)
- getXMax
  - Parameters, [20](#)
- getXMin
  - Parameters, [21](#)
- getXValues
  - Parameters, [21](#)
- Id
  - Parameters, [21](#)
- isOpen
  - Parameters, [21](#)
- kappa
  - Parameters, [21](#)
- lambda
  - Parameters, [21](#)
- loadB
  - Parameters, [22](#)
- loadE
  - Parameters, [22](#)
- loadParams
  - Parameters, [22](#)
- loadV
  - Parameters, [22](#)
- main
  - main.cpp, [27](#)
- main.cpp, [26](#)
  - main, [27](#)
  - read\_file, [27](#)
- modifyCCnj
  - ConstantGridSolver, [11](#)
  - NonconstantGridSolver, [15](#)
- NX
  - Parameters, [23](#)

NonconstantGridSolver, [12](#)  
     calculateEM, [13](#)  
     calculateEP, [13](#)  
     fwdIteration, [14](#)  
     generateGrid, [15](#)  
     modifyCCnj, [15](#)  
     Q, [15](#)  
     T, [16](#)  
     U, [17](#)  
 NonconstantGridSolver.cpp, [28](#)  
 NonconstantGridSolver.h, [28](#)  
 Parameters, [18](#)  
     ~Parameters, [19](#)  
     checkNumberOfRowsInFile, [19](#)  
     FRIEND\_TEST, [19](#), [20](#)  
     getB, [20](#)  
     getDx, [20](#)  
     getE, [20](#)  
     getGrid\_points\_per\_lambda, [20](#)  
     getNChannels, [20](#)  
     getNE, [20](#)  
     getNSymmetries, [20](#)  
     getUnit, [20](#)  
     getV, [20](#)  
     getVMatrix, [20](#)  
     getXMax, [20](#)  
     getXMin, [21](#)  
     getXValues, [21](#)  
     Id, [21](#)  
     isOpen, [21](#)  
     kappa, [21](#)  
     lambda, [21](#)  
     loadB, [22](#)  
     loadE, [22](#)  
     loadParams, [22](#)  
     loadV, [22](#)  
     NX, [23](#)  
     Parameters, [19](#)  
     Parameters\_getV\_Test, [23](#)  
     Parameters\_isOpen\_Test, [23](#)  
     Parameters\_kappaDouble\_Test, [23](#)  
     Parameters\_kappaInt\_Test, [23](#)  
     Parameters\_lambda\_Test, [23](#)  
     Parameters\_loadV\_Test, [23](#)  
     Parameters\_requiredDX\_Test, [23](#)  
     requiredDx, [23](#)  
     setXValues, [23](#)  
     x, [23](#)  
 Parameters.cpp, [29](#)  
 Parameters.h, [29](#)  
 Parameters\_getV\_Test  
     Parameters, [23](#)  
 Parameters\_isOpen\_Test  
     Parameters, [23](#)  
 Parameters\_kappaDouble\_Test  
     Parameters, [23](#)  
 Parameters\_kappaInt\_Test  
     Parameters, [23](#)  
 Parameters\_lambda\_Test  
     Parameters, [23](#)  
 Parameters\_loadV\_Test  
     Parameters, [23](#)  
 Parameters\_requiredDX\_Test  
     Parameters, [23](#)  
 Parameters\_lambda\_Test  
     Parameters, [23](#)  
 Parameters\_loadV\_Test  
     Parameters, [23](#)  
 Parameters\_requiredDX\_Test  
     Parameters, [23](#)  
 Q  
     NonconstantGridSolver, [15](#)  
 read\_file  
     main.cpp, [27](#)  
 requiredDx  
     Parameters, [23](#)  
 saveS  
     ConstantGridSolver, [11](#)  
 setParameters  
     ConstantGridSolver, [12](#)  
 setXValues  
     Parameters, [23](#)  
 solveForEnergies  
     ConstantGridSolver, [12](#)  
 Solver\_z\_liczeniem\_psi.cpp, [30](#)  
 T  
     NonconstantGridSolver, [16](#)  
 U  
     NonconstantGridSolver, [17](#)  
 x  
     Parameters, [23](#)