

DD2D - Matryoshka approach

0

Generated by Doxygen 1.8.3.1

Tue May 28 2013 16:26:54

Contents

1	Main Page	1
2	Hierarchical Index	3
2.1	Class Hierarchy	3
3	Data Structure Index	5
3.1	Data Structures	5
4	File Index	7
4.1	File List	7
5	Data Structure Documentation	9
5.1	Defect Class Reference	9
5.1.1	Detailed Description	10
5.1.2	Constructor & Destructor Documentation	11
5.1.2.1	Defect	11
5.1.2.2	Defect	11
5.1.2.3	Defect	11
5.1.3	Member Function Documentation	11
5.1.3.1	getPosition	11
5.1.3.2	getPosition	12
5.1.3.3	getX	12
5.1.3.4	getY	12
5.1.3.5	getZ	13
5.1.3.6	setPosition	13
5.1.3.7	setPosition	13
5.1.3.8	setPosition	13
5.1.3.9	setX	14
5.1.3.10	setY	14
5.1.3.11	setZ	14
5.1.3.12	stressField	14
5.1.4	Field Documentation	15
5.1.4.1	pos	15

5.2	Dislocation Class Reference	15
5.2.1	Detailed Description	17
5.2.2	Constructor & Destructor Documentation	17
5.2.2.1	Dislocation	17
5.2.2.2	Dislocation	18
5.2.3	Member Function Documentation	18
5.2.3.1	calculateRotationMatrix	18
5.2.3.2	getBurgers	19
5.2.3.3	getLineVector	19
5.2.3.4	getPosition	19
5.2.3.5	getPosition	19
5.2.3.6	getX	20
5.2.3.7	getY	20
5.2.3.8	getZ	20
5.2.3.9	setBurgers	20
5.2.3.10	setLineVector	21
5.2.3.11	setMobile	21
5.2.3.12	setPinned	21
5.2.3.13	setPosition	21
5.2.3.14	setPosition	22
5.2.3.15	setPosition	22
5.2.3.16	setX	22
5.2.3.17	setY	22
5.2.3.18	setZ	23
5.2.3.19	stressField	23
5.2.3.20	stressFieldLocal	24
5.2.4	Field Documentation	24
5.2.4.1	bmag	24
5.2.4.2	bvec	24
5.2.4.3	lvec	24
5.2.4.4	mobile	25
5.2.4.5	pos	25
5.2.4.6	rotationMatrix	25
5.3	DislocationSource Class Reference	25
5.3.1	Detailed Description	28
5.3.2	Constructor & Destructor Documentation	28
5.3.2.1	DislocationSource	28
5.3.2.2	DislocationSource	28
5.3.3	Member Function Documentation	29
5.3.3.1	dipoleNucleationLength	29

5.3.3.2	getBurgers	29
5.3.3.3	getBurgersMag	30
5.3.3.4	getIterationCount	30
5.3.3.5	getLineVector	30
5.3.3.6	getNumIterations	30
5.3.3.7	getPosition	31
5.3.3.8	getPosition	31
5.3.3.9	getTauCritical	31
5.3.3.10	getX	31
5.3.3.11	getY	32
5.3.3.12	getZ	32
5.3.3.13	ifEmitDipole	32
5.3.3.14	incrementIterationCount	32
5.3.3.15	resetIterationCounter	33
5.3.3.16	setBurgers	33
5.3.3.17	setBurgersMagnitude	33
5.3.3.18	setLineVector	33
5.3.3.19	setNumIterations	33
5.3.3.20	setPosition	34
5.3.3.21	setPosition	34
5.3.3.22	setPosition	34
5.3.3.23	setTauCritical	35
5.3.3.24	setX	35
5.3.3.25	setY	35
5.3.3.26	setZ	35
5.3.3.27	stressField	36
5.3.4	Field Documentation	36
5.3.4.1	bmag	36
5.3.4.2	bvec	36
5.3.4.3	countIterations	36
5.3.4.4	lvec	37
5.3.4.5	mobile	37
5.3.4.6	nIterations	37
5.3.4.7	pos	37
5.3.4.8	rotationMatrix	37
5.3.4.9	tauCritical	37
5.4	Matrix33 Class Reference	37
5.4.1	Detailed Description	39
5.4.2	Constructor & Destructor Documentation	39
5.4.2.1	Matrix33	39

5.4.2.2	Matrix33	39
5.4.2.3	Matrix33	40
5.4.2.4	Matrix33	40
5.4.3	Member Function Documentation	40
5.4.3.1	adjugate	40
5.4.3.2	getValue	41
5.4.3.3	operator!	41
5.4.3.4	operator*	42
5.4.3.5	operator*	42
5.4.3.6	operator*	43
5.4.3.7	operator*=	43
5.4.3.8	operator*=	43
5.4.3.9	operator+	44
5.4.3.10	operator+=	44
5.4.3.11	operator-	44
5.4.3.12	operator-=	45
5.4.3.13	operator^	45
5.4.3.14	operator~	46
5.4.3.15	setValue	46
5.4.4	Field Documentation	46
5.4.4.1	x	46
5.5	RotationMatrix Class Reference	47
5.5.1	Detailed Description	48
5.5.2	Constructor & Destructor Documentation	48
5.5.2.1	RotationMatrix	48
5.5.2.2	RotationMatrix	49
5.5.3	Member Function Documentation	49
5.5.3.1	adjugate	49
5.5.3.2	getValue	49
5.5.3.3	operator!	50
5.5.3.4	operator*	50
5.5.3.5	operator*	51
5.5.3.6	operator*	51
5.5.3.7	operator*=	52
5.5.3.8	operator*=	52
5.5.3.9	operator+	52
5.5.3.10	operator+=	53
5.5.3.11	operator-	53
5.5.3.12	operator-=	53
5.5.3.13	operator^	54

5.5.3.14	operator~	54
5.5.3.15	setValue	54
5.5.4	Field Documentation	55
5.5.4.1	x	55
5.6	SlipPlane Class Reference	55
5.6.1	Detailed Description	57
5.6.2	Constructor & Destructor Documentation	57
5.6.2.1	SlipPlane	57
5.6.2.2	SlipPlane	58
5.6.3	Member Function Documentation	58
5.6.3.1	calculateRotationMatrix	58
5.6.3.2	getAxis	59
5.6.3.3	getDislocation	59
5.6.3.4	getDislocationList	60
5.6.3.5	getDislocationSource	60
5.6.3.6	getDislocationSourceList	60
5.6.3.7	getExtremities	61
5.6.3.8	getExtremity	61
5.6.3.9	getNormal	61
5.6.3.10	getPosition	62
5.6.3.11	getRotationMatrix	62
5.6.3.12	setDislocationList	62
5.6.3.13	setDislocationSourceList	62
5.6.3.14	setExtremities	63
5.6.3.15	setNormal	63
5.6.3.16	setPosition	63
5.6.4	Field Documentation	63
5.6.4.1	dislocations	64
5.6.4.2	dislocationSources	64
5.6.4.3	extremities	64
5.6.4.4	normalVector	64
5.6.4.5	position	64
5.6.4.6	rotationMatrix	64
5.7	Strain Class Reference	65
5.7.1	Detailed Description	66
5.7.2	Constructor & Destructor Documentation	66
5.7.2.1	Strain	66
5.7.2.2	Strain	67
5.7.3	Member Function Documentation	67
5.7.3.1	adjugate	67

5.7.3.2	getPrincipalStrains	68
5.7.3.3	getShearStrains	68
5.7.3.4	getValue	68
5.7.3.5	operator!	69
5.7.3.6	operator*	69
5.7.3.7	operator*	70
5.7.3.8	operator*	70
5.7.3.9	operator*=	71
5.7.3.10	operator*=	71
5.7.3.11	operator+	71
5.7.3.12	operator+=	72
5.7.3.13	operator-	72
5.7.3.14	operator-=	72
5.7.3.15	operator^	73
5.7.3.16	operator~	73
5.7.3.17	populateMatrix	73
5.7.3.18	rotate	74
5.7.3.19	setValue	74
5.7.4	Field Documentation	74
5.7.4.1	principalStrains	74
5.7.4.2	shearStrains	74
5.7.4.3	x	75
5.8	Stress Class Reference	75
5.8.1	Detailed Description	77
5.8.2	Constructor & Destructor Documentation	77
5.8.2.1	Stress	77
5.8.2.2	Stress	77
5.8.3	Member Function Documentation	77
5.8.3.1	adjugate	77
5.8.3.2	getPrincipalStresses	78
5.8.3.3	getShearStresses	78
5.8.3.4	getValue	79
5.8.3.5	operator!	79
5.8.3.6	operator*	80
5.8.3.7	operator*	80
5.8.3.8	operator*	80
5.8.3.9	operator*=	81
5.8.3.10	operator*=	81
5.8.3.11	operator+	81
5.8.3.12	operator+=	82

5.8.3.13	operator-	82
5.8.3.14	operator=	83
5.8.3.15	operator^	83
5.8.3.16	operator~	83
5.8.3.17	populateMatrix	84
5.8.3.18	rotate	84
5.8.3.19	setValue	84
5.8.4	Field Documentation	85
5.8.4.1	principalStresses	85
5.8.4.2	shearStresses	85
5.8.4.3	x	85
5.9	Vector3d Class Reference	85
5.9.1	Detailed Description	86
5.9.2	Constructor & Destructor Documentation	86
5.9.2.1	Vector3d	86
5.9.2.2	Vector3d	87
5.9.2.3	Vector3d	87
5.9.3	Member Function Documentation	87
5.9.3.1	getValue	87
5.9.3.2	getVector	88
5.9.3.3	magnitude	88
5.9.3.4	normalize	88
5.9.3.5	operator*	89
5.9.3.6	operator*	89
5.9.3.7	operator*=	90
5.9.3.8	operator+	90
5.9.3.9	operator+=	90
5.9.3.10	operator-	90
5.9.3.11	operator=	91
5.9.3.12	operator^	91
5.9.3.13	operator^=	91
5.9.3.14	setValue	92
5.9.3.15	setVector	92
5.9.3.16	sum	92
5.9.4	Field Documentation	93
5.9.4.1	x	93
6	File Documentation	95
6.1	constants.h File Reference	95
6.1.1	Detailed Description	96

6.1.2	Macro Definition Documentation	96
6.1.2.1	PI	96
6.1.2.2	SQRT2	96
6.1.2.3	SQRT3	96
6.1.2.4	SQRT5	96
6.2	defect.cpp File Reference	97
6.2.1	Detailed Description	97
6.3	defect.h File Reference	98
6.3.1	Detailed Description	99
6.4	dislocation.cpp File Reference	100
6.4.1	Detailed Description	100
6.5	dislocation.h File Reference	101
6.5.1	Detailed Description	102
6.6	dislocationDefaults.h File Reference	102
6.6.1	Detailed Description	104
6.6.2	Macro Definition Documentation	104
6.6.2.1	DEFAULT_BURGERS_0	104
6.6.2.2	DEFAULT_BURGERS_1	104
6.6.2.3	DEFAULT_BURGERS_2	104
6.6.2.4	DEFAULT_BURGERS_MAGNITUDE	104
6.6.2.5	DEFAULT_LINEVECTOR_0	104
6.6.2.6	DEFAULT_LINEVECTOR_1	104
6.6.2.7	DEFAULT_LINEVECTOR_2	105
6.6.2.8	DEFAULT_POSITION_0	105
6.6.2.9	DEFAULT_POSITION_1	105
6.6.2.10	DEFAULT_POSITION_2	105
6.7	dislocationSource.cpp File Reference	105
6.7.1	Detailed Description	106
6.8	dislocationSource.h File Reference	107
6.8.1	Detailed Description	108
6.9	dislocationSourceDefaults.h File Reference	108
6.9.1	Detailed Description	109
6.9.2	Macro Definition Documentation	110
6.9.2.1	DEFAULT_NITERATIONS	110
6.9.2.2	DEFAULT_TAU_CRITICAL	110
6.10	mainpage.dox File Reference	110
6.11	matrix33.cpp File Reference	110
6.11.1	Detailed Description	110
6.12	matrix33.h File Reference	111
6.12.1	Detailed Description	112

6.13	rotationMatrix.cpp File Reference	113
6.13.1	Detailed Description	113
6.14	rotationMatrix.h File Reference	113
6.14.1	Detailed Description	114
6.15	slipPlane.cpp File Reference	115
6.15.1	Detailed Description	115
6.15.2	Function Documentation	116
6.15.2.1	SlipPlane	116
6.16	slipPlane.h File Reference	116
6.16.1	Detailed Description	118
6.17	strain.cpp File Reference	118
6.17.1	Detailed Description	119
6.18	strain.h File Reference	120
6.18.1	Detailed Description	121
6.19	stress.cpp File Reference	121
6.19.1	Detailed Description	122
6.20	stress.h File Reference	123
6.20.1	Detailed Description	124
6.21	vector3d.cpp File Reference	125
6.21.1	Detailed Description	125
6.22	vector3d.h File Reference	125
6.22.1	Detailed Description	127

Chapter 1

Main Page

The files in this program provide a hierarchical data structure system for carrying out dislocation dynamics simulations in two dimensions. The base class is [Defect](#), which represents a generic defect in a metallic crystal. All other defects, such as dislocations, dislocation sources, precipitates, etc., are represented by their own classes which inherit certain functions from the [Defect](#) class.

The goal of carrying out these simulations in two dimensions is to be able to simulate plastic deformation of up to a few percent. Current three dimensional dislocation dynamics simulations are computationally expensive. This approach hopes to sacrifice some of the precision in order to gain in speed and flexibility.

The program is under development now, with the data structures being defined. When it will be complete, it is intended to have data structures nested within each other, hence the name [Matryoshka](#). For example, a polycrystal is a collection of grains; a grain is a collection of slip systems; a slip system is a collection of slip planes; a slip plane is a collection of dislocations, dislocation sources and other defects. This program will also take advantage of the functionality provided by the C++ STL to manage lists of various objects in the simulation. Once the base simulations execute successfully, other defects will be introduced.

To view the hierarchical structure, go to the section labeled Data Structures > Class Hierarchy. A good place to start would be the [Defect](#) class, which is the generic base class for most of the entities present in the simulation.

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

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

Defect	9
Dislocation	15
DislocationSource	25
Matrix33	37
RotationMatrix	47
Strain	65
Stress	75
SlipPlane	55
Vector3d	85

Chapter 3

Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

Defect	Class Defect representing a generic defect in a material	9
Dislocation	Dislocation class representing a dislocation in the simulation	15
DislocationSource	DislocationSource class representing a source of dislocations in the simulation	25
Matrix33	Matrix33 class representing a 3x3 square matrix	37
RotationMatrix	RotationMatrix class to represent a rotation matrix	47
SlipPlane	SlipPlane class representing a slip plane in the simulation	55
Strain	Strain class to represent the strain tensor	65
Stress	Stress class to represent the stress tensor	75
Vector3d	Vector3d class representing a single 3-dimensional vector in the simulation	85

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

constants.h	Definition of constants used in the program	95
defect.cpp	Definition of member functions of the Defect class	97
defect.h	Definition of the Defect class	98
dislocation.cpp	Definition of constructors and member functions of the Dislocation class	100
dislocation.h	Definition of the Dislocation class	101
dislocationDefaults.h	Definition of certain default values for members of the Dislocation class	102
dislocationSource.cpp	Definition of the member functions of the DislocationSource class	105
dislocationSource.h	Definition of the DislocationSource class	107
dislocationSourceDefaults.h	Definition of certain default values for members of the DislocationSource class	108
matrix33.cpp	Definition of the member functions and operators of the Matrix33 class	110
matrix33.h	Definition of the Matrix33 class	111
rotationMatrix.cpp	Definition of the RotationMatrix class member functions	113
rotationMatrix.h	Definition of the RotationMatrix class	113
slipPlane.cpp	Definition of the member functions of the SlipPlane class	115
slipPlane.h	Definition of the SlipPlane class	116
strain.cpp	Definition of the member functions if the Strain class	118
strain.h	Definition of the Strain class	120
stress.cpp	Definition of the member functions if the Stress class	121
stress.h	Definition of the Stress class	123

vector3d.cpp	
Definition of member functions and operators of the Vector3d class	125
vector3d.h	
Definition of the Vector3d class	125

Chapter 5

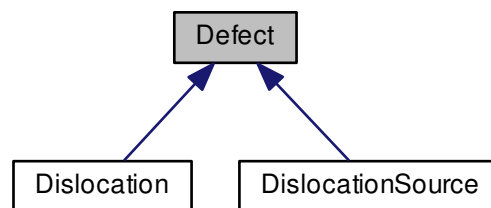
Data Structure Documentation

5.1 Defect Class Reference

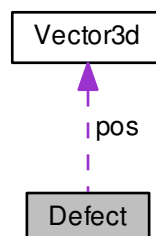
Class [Defect](#) representing a generic defect in a material.

```
#include <defect.h>
```

Inheritance diagram for Defect:



Collaboration diagram for Defect:



Public Member Functions

- [Defect](#) ()
Default constructor.
- [Defect](#) (double x, double y, double z)
Constructor specifying the position.
- [Defect](#) (double *p)
Constructor specifying the position.
- void [setPosition](#) (double *a)
Sets the position of the defect.
- void [setPosition](#) (double x, double y, double z)
Sets the position of the defect.
- void [setPosition](#) ([Vector3d](#) a)
Sets the position of the defect.
- void [setX](#) (double x)
Sets the X-coordinate of the defect.
- void [setY](#) (double y)
Sets the Y-coordinate of the defect.
- void [setZ](#) (double z)
Sets the Z-coordinate of the defect.
- double * [getPosition](#) ()
Returns in an array the position.
- void [getPosition](#) (double *a)
Returns the array position in a pre-allocated array.
- double [getX](#) ()
Returns the X-coordinate of the defect.
- double [getY](#) ()
Returns the Y-coordinate of the defect.
- double [getZ](#) ()
Returns the Z-coordinate of the defect.
- virtual [Stress stressField](#) ([Vector3d](#) p, double mu, double nu)
Virtual function for calculating the stress field.

Protected Attributes

- [Vector3d](#) pos
Position vector of the defect in 2D space.

5.1.1 Detailed Description

Class [Defect](#) representing a generic defect in a material.

Defines the [Defect](#) class representing an defect in the simulation. This is simply a generic description class with virtual functions. Later classes like dislocations, precipitates, boundaries etc will inherit from this class.

Definition at line 20 of file defect.h.

5.1.2 Constructor & Destructor Documentation

5.1.2.1 Defect::Defect ()

Default constructor.

Creates the object with position (0.0, 0.0, 0.0).

Definition at line 17 of file defect.cpp.

```
18 {
19     for (int i=0; i<3; i++)
20     {
21         this->pos.setValue(i, 0.0);
22     }
23 }
```

5.1.2.2 Defect::Defect (double x, double y, double z)

Constructor specifying the position.

The object is initialized with the position specified by the arguments (x, y, z).

Parameters

x	X-coordinate of the defect.
y	Y-coordinate of the defect
z	Z-coordinate of the defect.

Definition at line 32 of file defect.cpp.

```
33 {
34     this->pos.setValue (0, x);
35     this->pos.setValue (1, y);
36     this->pos.setValue (2, z);
37 }
```

5.1.2.3 Defect::Defect (double * p)

Constructor specifying the position.

The object is initialized with the position specified in the array pointed to by the argument.

Parameters

p	Pointer to the array containing the coordinates of the defect.
---	----------------------------------------------------------------

Definition at line 44 of file defect.cpp.

```
45 {
46     this->pos.setValue (p);
47 }
```

5.1.3 Member Function Documentation

5.1.3.1 double * Defect::getPosition ()

Returns in an array the position.

The position of the defect is saved in an array and a pointer to its first term is returned.

Returns

Pointer to the first term of the array containing the position of the defect.

Definition at line 119 of file defect.cpp.

```
120 {  
121     return (this->pos.getVector ());  
122 }
```

5.1.3.2 void Defect::getPosition (double * a)

Returns the array position in a pre-allocated array.

Returns in the array provided in the argument the position of the defect. The array must be pre-allocated.

Parameters

a	Pointer to the location where the defect coordinates are to be populated.
----------	---------------------------------------------------------------------------

Definition at line 129 of file defect.cpp.

```
130 {  
131     a = this->pos.getVector ();  
132 }
```

5.1.3.3 double Defect::getX ()

Returns the X-coordinate of the defect.

Returns

X-coordinate of the defect.

Definition at line 138 of file defect.cpp.

```
139 {  
140     return (this->getValue (0));  
141 }
```

5.1.3.4 double Defect::getY ()

Returns the Y-coordinate of the defect.

Returns

Y-coordinate of the defect.

Definition at line 147 of file defect.cpp.

```
148 {  
149     return (this->pos.getValue (1));  
150 }
```


5.1.3.5 double Defect::getZ ()

Returns the Z-coordinate of the defect.

Returns

Z-coordinate of the defect.

Definition at line 156 of file defect.cpp.

```
157 {
158     return (this->pos.getValue (2));
159 }
```

5.1.3.6 void Defect::setPosition (double * a)

Sets the position of the defect.

The position of the defect is set to the co-ordinates present in the array pointed to by the argument. Sets the position of the defect as the values in the array pointed to by the argument.

Parameters

a	Pointer to the array containing the coordinates of the defect.
----------	----------------------------------------------------------------

Definition at line 56 of file defect.cpp.

```
57 {
58     this->pos.setVector (a);
59 }
```

5.1.3.7 void Defect::setPosition (double x, double y, double z)

Sets the position of the defect.

The position of the defect is set to the co-ordinates specified by the arguments (x, y, z). Sets the position of the defect as the coordinates provided as arguments.

Parameters

x	X-coordinate of the defect.
y	Y-coordinate of the defect.
z	Z-coordinate of the defect.

Definition at line 69 of file defect.cpp.

```
70 {
71     this->pos.setValue (0, x);
72     this->pos.setValue (1, y);
73     this->pos.setValue (2, z);
74 }
```

5.1.3.8 void Defect::setPosition (Vector3d a)

Sets the position of the defect.

The position of the defect is set to the position vector given by the argument a.

Parameters

<i>a</i>	Position vector of the defect.
----------	--------------------------------

Definition at line 81 of file defect.cpp.

```
82 {  
83   this->position = a;  
84 }
```

5.1.3.9 void Defect::setX (double *x*)

Sets the X-coordinate of the defect.

Parameters

<i>x</i>	X-coordinate of the defect.
----------	-----------------------------

Definition at line 90 of file defect.cpp.

```
91 {  
92   this->pos.setValue (0, x);  
93 }
```

5.1.3.10 void Defect::setY (double *y*)

Sets the Y-coordinate of the defect.

Parameters

<i>y</i>	Y-coordinate of the defect.
----------	-----------------------------

Definition at line 99 of file defect.cpp.

```
100 {  
101   this->pos.setValue (1, y);  
102 }
```

5.1.3.11 void Defect::setZ (double *z*)

Sets the Z-coordinate of the defect.

Parameters

<i>z</i>	Z-coordinate of the defect.
----------	-----------------------------

Definition at line 108 of file defect.cpp.

```
109 {  
110   this->pos.setValue (2, z);  
111 }
```

5.1.3.12 virtual Stress Defect::stressField (Vector3d *p*, double *mu*, double *nu*) [inline], [virtual]

Virtual function for calculating the stress field.

Returns the value of the stress field of the given defect at the position given by the argument. This is a virtual function and always returns a zero matrix. Classes which inherit this function should have their own implementations of this function to override its behaviour.

Parameters

p	Position vector of the the point where the stress field is to be calculated.
μ	Shear modulus in Pascals.
ν	Poisson's ratio.

Returns

[Stress](#) field value at the position p .

Reimplemented in [Dislocation](#).

Definition at line 136 of file defect.h.

```

137 {
138     // This virtual function returns a zero matrix.
139     // Inheriting classes will have functions implementing this in their own way
140     // They will override this behaviour.
141     Stress s;
142     return (s);
143 }
```

5.1.4 Field Documentation

5.1.4.1 Vector3d Defect::pos [protected]

Position vector of the defect in 2D space.

Definition at line 26 of file defect.h.

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

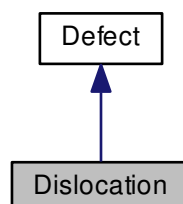
- [defect.h](#)
- [defect.cpp](#)

5.2 Dislocation Class Reference

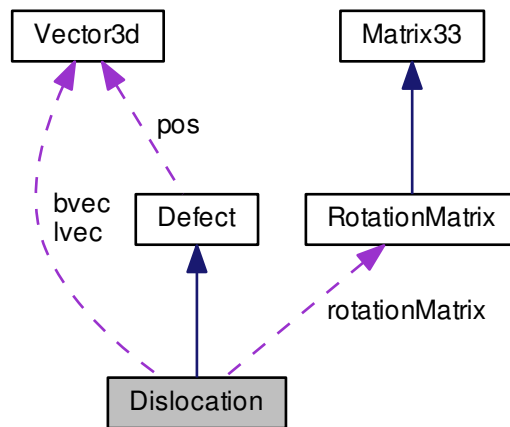
[Dislocation](#) class representing a dislocation in the simulation.

```
#include <dislocation.h>
```

Inheritance diagram for Dislocation:



Collaboration diagram for Dislocation:



Public Member Functions

- [Dislocation](#) ()
Default constructor.
- [Dislocation](#) ([Vector3d](#) burgers, [Vector3d](#) line, [Vector3d](#) position, double bm, bool m)
Constructor that explicitly specifies all parameters.
- void [setBurgers](#) ([Vector3d](#) burgers)
Sets the Burgers vector of the dislocation.
- void [setLineVector](#) ([Vector3d](#) line)
Sets the line vector of the dislocation.
- void [setMobile](#) ()
Sets the dislocation as mobile.
- void [setPinned](#) ()
Sets the dislocation as pinned.
- [Vector3d](#) [getBurgers](#) ()
Gets the Burgers vector of the dislocation.
- [Vector3d](#) [getLineVector](#) ()
Gets the line vector of the dislocation.
- void [calculateRotationMatrix](#) ()
Calculate the roation matrix.
- [Stress](#) [stressField](#) ([Vector3d](#) p, double mu, double nu)
Calculates the stress field due to this dislocation at the position given as argument.
- [Stress](#) [stressFieldLocal](#) ([Vector3d](#) p, double mu, double nu)
Calculates the stress field due to the dislocation in the local co-ordinate system.
- void [setPosition](#) (double *a)
Sets the position of the defect.
- void [setPosition](#) (double x, double y, double z)
Sets the position of the defect.
- void [setPosition](#) ([Vector3d](#) a)

- Sets the position of the defect.*
 - void [setX](#) (double x)
- Sets the X-coordinate of the defect.*
 - void [setY](#) (double y)
- Sets the Y-coordinate of the defect.*
 - void [setZ](#) (double z)
- Sets the Z-coordinate of the defect.*
 - double * [getPosition](#) ()
- Returns in an array the position.*
 - void [getPosition](#) (double *a)
- Returns the array position in a pre-allocated array.*
 - double [getX](#) ()
- Returns the X-coordinate of the defect.*
 - double [getY](#) ()
- Returns the Y-coordinate of the defect.*
 - double [getZ](#) ()
- Returns the Z-coordinate of the defect.*

Protected Attributes

- [Vector3d bvec](#)
 - Burgers vector of the dislocation.*
- [Vector3d lvec](#)
 - Line vector of the dislocation.*
- bool [mobile](#)
 - Boolean term indicating mobility.*
- double [bmag](#)
 - Magnitude of the Burgers vector in metres.*
- [RotationMatrix rotationMatrix](#)
 - The rotation matrix for rotating from the global to the local co-ordinate system and vice-versa.*
- [Vector3d pos](#)
 - Position vector of the defect in 2D space.*

5.2.1 Detailed Description

[Dislocation](#) class representing a dislocation in the simulation.

The [Dislocation](#) class represents a dislocation in the simulation. The class inherits from the [Defect](#) class. A dislocation has several properties like a Burgers vector, line vector, etc. which will all be declared here.

Definition at line 21 of file [dislocation.h](#).

5.2.2 Constructor & Destructor Documentation

5.2.2.1 [Dislocation::Dislocation](#) ()

Default constructor.

Initializes the dislocation with the following default parameters: Position: (0.0, 0.0, 0.0) Burgers vector: Default value set in defaults file. Line vector: Default value set in defaults file. Burgers vector magnitude: Default value set in the defaults file. Mobile: true.

Initializes the dislocation with the following default parameters: Position: (0.0, 0.0, 0.0) Burgers vector: Default value set in defaults file. Line vector: Default value set in defaults file. Burgers vector magnitude: Default value set in teh defaults file. Mobile: true.

Definition at line 21 of file dislocation.cpp.

```

22 {
23   this->setPosition ( 0.0, 0.0, 0.0 );
24   this->setBurgers ( Vector3d ( DEFAULT_BURGERS_0,
    DEFAULT_BURGERS_1, DEFAULT_BURGERS_2 ) );
25   this->setLineVector ( Vector3d ( DEFAULT_LINEVECTOR_0,
    DEFAULT_LINEVECTOR_1, DEFAULT_LINEVECTOR_2 ) );
26   this->bmag = DEFAULT_BURGERS_MAGNITUDE;
27   this->mobile = true;
28   this->calculateRotationMatrix ();
29 }
```

5.2.2.2 Dislocation::Dislocation (Vector3d burgers, Vector3d line, Vector3d position, double bm, bool m)

Constructor that explicitly specifies all parameters.

All parameters: Burgers vector, line vector, position, are specified.

Parameters

<i>burgers</i>	Burgers vector.
<i>line</i>	Line vector.
<i>position</i>	Position of the dislocation.
<i>bm</i>	Magnitude of the Burgers vector in metres.
<i>m</i>	Mobility (true/false).

Definition at line 40 of file dislocation.cpp.

```

41 {
42   this->bvec = burgers;
43   this->lvec = line;
44   this->pos = position;
45   this->mobile = m;
46   this->bmag = bm;
47   this->calculateRotationMatrix ();
48 }
```

5.2.3 Member Function Documentation

5.2.3.1 void Dislocation::calculateRotationMatrix ()

Calculate the roation matrix.

This function calculates the rotation matrix for this dislocation using the global and local co-ordinate systems. The matrix rotationMatrix is for rotation from the old (unprimed, global) to the new (primed, dislocation) system.

Definition at line 109 of file dislocation.cpp.

```

110 {
111   Vector3d globalSystem[3];    // Global co-ordinate systems
112   Vector3d localSystem[3];    // Dislocation co-ordinate system
113
114   // Vectors of the global co-ordinate system
115   globalSystem[0] = Vector3d (1.0, 0.0, 0.0);
116   globalSystem[1] = Vector3d (0.0, 1.0, 0.0);
117   globalSystem[2] = Vector3d (0.0, 0.0, 1.0);
118
119   // Vectors of the dislocation co-ordinate system
120   localSystem[0] = bvec.normalize ();
121   localSystem[2] = lvec.normalize ();
122   localSystem[1] = (lvec ^ bvec).normalize ();
123
124   // Calculate rotation matrix
```

```

125   this->rotationMatrix = RotationMatrix (globalSystem, localSystem);
126 }

```

5.2.3.2 Vector3d Dislocation::getBurgers ()

Gets the Burgers vector of the dislocation.

Returns

Burgers vector in a variable of type [Vector3d](#).

Definition at line 90 of file dislocation.cpp.

```

91 {
92   return ( this->bvec );
93 }

```

5.2.3.3 Vector3d Dislocation::getLineVector ()

Gets the line vector of the dislocation.

Returns

Line vector in a variable of type [Vector3d](#).

Definition at line 99 of file dislocation.cpp.

```

100 {
101   return ( this->lvec );
102 }

```

5.2.3.4 double * Defect::getPosition () [inherited]

Returns in an array the position.

The position of the defect is saved in an array and a pointer to its first term is returned.

Returns

Pointer to the first term of the array containing the position of the defect.

Definition at line 119 of file defect.cpp.

```

120 {
121   return (this->pos.getVector ());
122 }

```

5.2.3.5 void Defect::getPosition (double * a) [inherited]

Returns the array position in a pre-allocated array.

Returns in the array provided in the argument the position of the defect. The array must be pre-allocated.

Parameters

<i>a</i>	Pointer to the location where the defect coordinates are to be populated.
----------	---------------------------------------------------------------------------

Definition at line 129 of file defect.cpp.

```
130 {
131     a = this->pos.getVector ();
132 }
```

5.2.3.6 double Defect::getX () [inherited]

Returns the X-coordinate of the defect.

Returns

X-coordinate of the defect.

Definition at line 138 of file defect.cpp.

```
139 {
140     return (this->getValue (0));
141 }
```

5.2.3.7 double Defect::getY () [inherited]

Returns the Y-coordinate of the defect.

Returns

Y-coordinate of the defect.

Definition at line 147 of file defect.cpp.

```
148 {
149     return (this->pos.getValue (1));
150 }
```

5.2.3.8 double Defect::getZ () [inherited]

Returns the Z-coordinate of the defect.

Returns

Z-coordinate of the defect.

Definition at line 156 of file defect.cpp.

```
157 {
158     return (this->pos.getValue (2));
159 }
```

5.2.3.9 void Dislocation::setBurgers (Vector3d *burgers*)

Sets the Burgers vector of the dislocation.

Parameters

<i>burgers</i>	Burgers vector of the dislocation.
----------------	------------------------------------

Definition at line 54 of file dislocation.cpp.

```
55 {
56     this->bvec = burgers;
57 }
```

5.2.3.10 void Dislocation::setLineVector (Vector3d *line*)

Sets the line vector of the dislocation.

Parameters

<i>line</i>	Line vector of the dislocation.
-------------	---------------------------------

Definition at line 62 of file dislocation.cpp.

```
63 {
64     this->lvec = line;
65 }
```

5.2.3.11 void Dislocation::setMobile ()

Sets the dislocation as mobile.

Sets the flag mobile to true.

Definition at line 71 of file dislocation.cpp.

```
72 {
73     this->mobile = true;
74 }
```

5.2.3.12 void Dislocation::setPinned ()

Sets the dislocation as pinned.

Sets the flag mobile to false.

Definition at line 80 of file dislocation.cpp.

```
81 {
82     this->mobile = false;
83 }
```

5.2.3.13 void Defect::setPosition (double * *a*) [inherited]

Sets the position of the defect.

The position of the defect is set to the co-ordinates present in the array pointed to by the argument. Sets the position of the defect as the values in the array pointed to by the argument.

Parameters

<i>a</i>	Pointer to the array containing the coordinates of the defect.
----------	----------------------------------------------------------------

Definition at line 56 of file defect.cpp.

```
57 {
```

```

58  this->pos.setVector (a);
59  }

```

5.2.3.14 void Defect::setPosition (double x, double y, double z) [inherited]

Sets the position of the defect.

The position of the defect is set to the co-ordinates specified by the arguments (x, y, z). Sets the position of the defect as the coordinates provided as arguments.

Parameters

x	X-coordinate of the defect.
y	Y-coordinate of the defect.
z	Z-coordinate of the defect.

Definition at line 69 of file defect.cpp.

```

70 {
71  this->pos.setValue (0, x);
72  this->pos.setValue (1, y);
73  this->pos.setValue (2, z);
74 }

```

5.2.3.15 void Defect::setPosition (Vector3d a) [inherited]

Sets the position of the defect.

The position of the defect is set to the position vector given by the argument a.

Parameters

a	Position vector of the defect.
---	--------------------------------

Definition at line 81 of file defect.cpp.

```

82 {
83  this->position = a;
84 }

```

5.2.3.16 void Defect::setX (double x) [inherited]

Sets the X-coordinate of the defect.

Parameters

x	X-coordinate of the defect.
---	-----------------------------

Definition at line 90 of file defect.cpp.

```

91 {
92  this->pos.setValue (0, x);
93 }

```

5.2.3.17 void Defect::setY (double y) [inherited]

Sets the Y-coordinate of the defect.

Parameters

<i>y</i>	Y-coordinate of the defect.
----------	-----------------------------

Definition at line 99 of file defect.cpp.

```
100 {
101     this->pos.setValue (1, y);
102 }
```

5.2.3.18 void Defect::setZ (double z) [inherited]

Sets the Z-coordinate of the defect.

Parameters

<i>z</i>	Z-coordinate of the defect.
----------	-----------------------------

Definition at line 108 of file defect.cpp.

```
109 {
110     this->pos.setValue (2, z);
111 }
```

5.2.3.19 Stress Dislocation::stressField (Vector3d p, double mu, double nu) [virtual]

Calculates the stress field due to this dislocation at the position given as argument.

The stress field of the dislocation is calculated at the position indicated by the argument.

Parameters

<i>p</i>	Position vector of the point where the stress field is to be calculated.
<i>mu</i>	Shear modulus in Pascals.
<i>nu</i>	Poisson's ratio.

Returns

[Stress](#) tensor, expressed in the global co-ordinate system, giving the value of the stress field at position p.

Reimplemented from [Defect](#).

Definition at line 137 of file dislocation.cpp.

```
138 {
139     double principalStresses[3];
140     double shearStresses[3];
141     Vector3d r; // Vector joining the present dislocation to the point p
142
143     r = p - this->pos; // Still in global coordinate system
144     Vector3d rLocal = this->rotationMatrix * r; // Rotated to local co-ordinate
        system
145
146     // Calculate the stress field in the local co-ordinate system
147     Stress sLocal = this->stressFieldLocal (rLocal, mu, nu);
148
149     // Calculate the stress field in the global co-ordinate system
150     Stress sGlobal = (this->rotationMatrix) * sLocal * (^ (this->
        rotationMatrix));
151
152     return (sGlobal);
153 }
```

5.2.3.20 Stress Dislocation::stressFieldLocal (Vector3d *p*, double *mu*, double *nu*)

Calculates the stress field due to the dislocation in the local co-ordinate system.

The stress field due to the dislocation is calculated at the position indicated by the argument. The stress tensor is expressed in the dislocation's local co-ordinate system.

Parameters

<i>p</i>	Position vector of the point where the stress field is to be calculated. This position vector is calculated in the local co-ordinate system, taking the dislocation as the origin.
<i>mu</i>	Shear modulus in Pascals.
<i>nu</i>	Poisson's ratio.

Returns

[Stress](#) tensor, expressed in the dislocation's local co-ordinate system.

Definition at line 163 of file dislocation.cpp.

```

164 {
165     double D = ( mu * this->bm ) / ( 2.0 * PI * ( 1.0 - nu ) ); // Constant for all components of the
        stress tensor
166
167     double x, y, denominator;      // Terms that appear repeatedly in the stress tensor
168
169     x = p.getValue (0);
170     y = p.getValue (1);
171     denominator = pow ( ((x*x) + (y*y)), 2);
172
173     principalStresses[0] = -1.0 * D * y * ( (3.0*x*x) + (y*y) ) / denominator;
174     principalStresses[1] = D * y * ( (x*x) - (y*y) ) / denominator;
175     principalStresses[2] = nu * ( principalStresses[0] + principalStresses[1] );
176
177     shearStresses[0] = D * x * ( (x*x) - (y*y) ) / denominator;
178     shearStresses[1] = 0.0;
179     shearStresses[2] = 0.0;
180
181     return (Stress(principalStresses, shearStresses));
182 }
```

5.2.4 Field Documentation

5.2.4.1 double Dislocation::bmag [protected]

Magnitude of the Burgers vector in metres.

The magnitude of the Burgers vector is useful for several calculations such as stress field around the dislocation.

Definition at line 44 of file dislocation.h.

5.2.4.2 Vector3d Dislocation::bvec [protected]

Burgers vector of the dislocation.

Definition at line 27 of file dislocation.h.

5.2.4.3 Vector3d Dislocation::lvec [protected]

Line vector if the dislocation.

Definition at line 32 of file dislocation.h.

5.2.4.4 `bool Dislocation::mobile` [protected]

Boolean term indicating mobility.

For mobile dislocations this term is true and for pinned dislocations it is false.

Definition at line 38 of file `dislocation.h`.

5.2.4.5 `Vector3d Defect::pos` [protected], [inherited]

Position vector of the defect in 2D space.

Definition at line 26 of file `defect.h`.

5.2.4.6 `RotationMatrix Dislocation::rotationMatrix` [protected]

The rotation matrix for rotating from the global to the local co-ordinate system and vice-versa.

This is the rotation matrix that represents the relationship between the global and local co-ordinate systems. It is used to convert tensors and vectors between the two systems. The rotation matrix needs to be calculated once and may be refreshed periodically if lattice rotation is implemented. In the absence of lattice rotation, the matrix will remain invariant.

Definition at line 50 of file `dislocation.h`.

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

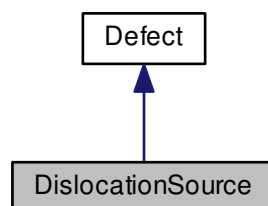
- [dislocation.h](#)
- [dislocation.cpp](#)

5.3 DislocationSource Class Reference

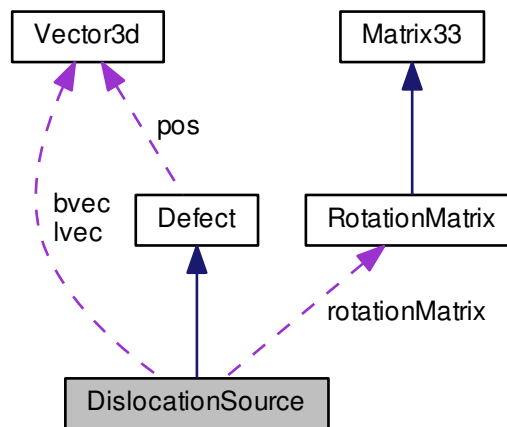
[DislocationSource](#) class representing a source of dislocations in the simulation.

```
#include <dislocationSource.h>
```

Inheritance diagram for DislocationSource:



Collaboration diagram for DislocationSource:



Public Member Functions

- [DislocationSource](#) ()
Default constructor.
- [DislocationSource](#) ([Vector3d](#) burgers, [Vector3d](#) line, [Vector3d](#) position, double bm, double tau, int nlter)
Constructor that explicitly specifies all parameters.
- void [setBurgers](#) ([Vector3d](#) burgers)
Sets the Burgers vector of the dislocation.
- void [setLineVector](#) ([Vector3d](#) line)
Sets the line vector of the dislocation.
- void [setBurgersMagnitude](#) (double bm)
Set the magnitude of the Burgers vector.
- void [setTauCritical](#) (double tauC)
Set the critical shear stres for dipole emission.
- void [setNumIterations](#) (int nlter)
Set the number of iterations before a dipole is emitted.
- void [resetIterationCounter](#) ()
Sets the iteration counter to zero.
- [Vector3d](#) [getBurgers](#) ()
Returns the Burgers vector of the dislocations in the dipole.
- [Vector3d](#) [getLineVector](#) ()
Returns the line vector of the dislocations in the dipole.
- double [getBurgersMag](#) ()
Returns the magnitude of the Burgers vector.
- double [getTauCritical](#) ()
Returns the critical shear stress value for dipole emission.
- int [getNumIterations](#) ()
Returns the number if iterations that the dislocation source must spend experiencing a shear stress greater than the critical value before it can emit a dislocation dipole.
- int [getIterationCount](#) ()

- Get the count of the iterations spent at higher than critical shear stress.*

 - double `dipoleNucleationLength` (double tau, double mu, double nu)
The nucleation length of the dipole.
- void `incrementIterationCount` ()
Increments the variable countIterations by 1.
- bool `ifEmitDipole` ()
Checks if the dislocation source has experienced higher than critical shear stress for long enough to emit a dipole.
- void `setPosition` (double *a)
Sets the position of the defect.
- void `setPosition` (double x, double y, double z)
Sets the position of the defect.
- void `setPosition` (Vector3d a)
Sets the position of the defect.
- void `setX` (double x)
Sets the X-coordinate of the defect.
- void `setY` (double y)
Sets the Y-coordinate of the defect.
- void `setZ` (double z)
Sets the Z-coordinate of the defect.
- double * `getPosition` ()
Returns in an array the position.
- void `getPosition` (double *a)
Returns the array position in a pre-allocated array.
- double `getX` ()
Returns the X-coordinate of the defect.
- double `getY` ()
Returns the Y-coordinate of the defect.
- double `getZ` ()
Returns the Z-coordinate of the defect.
- virtual `Stress stressField` (Vector3d p, double mu, double nu)
Virtual function for calculating the stress field.

Protected Attributes

- Vector3d `bvec`
Burgers vector of the dislocation.
- Vector3d `lvec`
Line vector of the dislocation.
- bool `mobile`
Boolean term indicating mobility.
- double `bmag`
Magnitude of the Burgers vector in metres.
- RotationMatrix `rotationMatrix`
The rotation matrix for rotating from the global to the local co-ordinate system and vice-versa.
- double `tauCritical`
Critical stress for the emission of a dislocation dipole.
- int `nIterations`
Number of iterations before a dipole is emitted.
- int `countIterations`
Counter variable for the number of consecutive iterations the dislocation source has experienced a shear stress greater than its critical value.
- Vector3d `pos`
Position vector of the defect in 2D space.

5.3.1 Detailed Description

[DislocationSource](#) class representing a source of dislocations in the simulation.

This class inherits from the [Defect](#) class. This object is basically the representation of a Frank-Read source emitting dislocation dipoles. When the dislocation source experiences a shear stress greater than a critical value for a certain amount of time (or number of iterations), it emits a dislocation dipole with a length that is a function of the applied stress. The properties of this class and the member functions will be declared here.

Definition at line 22 of file `dislocationSource.h`.

5.3.2 Constructor & Destructor Documentation

5.3.2.1 `DislocationSource::DislocationSource ()`

Default constructor.

Initializes the dislocation with the default parameters provided in the files [dislocationDefaults.h](#) and [dislocationSourceDefaults.h](#).

Definition at line 17 of file `dislocationSource.cpp`.

```
18 {
19     this->setPosition ( Vector3d ( DEFAULT_POSITION_0,
20     DEFAULT_POSITION_1, DEFAULT_POSITION_2 ) );
21     this->setBurgers ( Vector3d ( DEFAULT_BURGERS_0,
22     DEFAULT_BURGERS_1, DEFAULT_BURGERS_2 ) );
23     this->setLineVector ( Vector3d ( DEFAULT_LINEVECTOR_0,
24     DEFAULT_LINEVECTOR_1, DEFAULT_LINEVECTOR_2 ) );
25     this->bmag = DEFAULT_BURGERS_MAGNITUDE;
26     this->tauCritical = DEFAULT_TAU_CRITICAL;
27     this->nIterations = DEFAULT_NITERATIONS;
28     this->countIterations = 0;
29 }
```

5.3.2.2 `DislocationSource::DislocationSource (Vector3d burgers, Vector3d line, Vector3d position, double bm, double tau, int nlter)`

Constructor that explicitly specifies all parameters.

All parameters: Burgers vector, line vector, position, are specified.

Parameters

<i>burgers</i>	Burgers vector.
<i>line</i>	Line vector.
<i>position</i>	Position of the dislocation source.
<i>bm</i>	Magnitude of the Burgers vector in metres.
<i>tau</i>	Critical shear stress value.
<i>nlter</i>	Number of iterations of experiencing critical stress before a dipole is emitted.

All parameters: Burgers vector, line vector, position, are specified.

Parameters

<i>burgers</i>	Burgers vector.
<i>line</i>	Line vector.
<i>position</i>	Position of the dislocation.
<i>bm</i>	Magnitude of the Burgers vector in metres.
<i>tau</i>	Critical shear stress value.
<i>nlter</i>	Number of iterations of experiencing critical stress before a dipole is emitted.

Definition at line 38 of file dislocationSource.cpp.

```

39 {
40     this->bvec    = burgers;
41     this->lvec    = line;
42     this->pos     = position;
43     this->bmag    = bm;
44     this->tauCritical = tau;
45     this->nIterations = nIter;
46     this->countIterations = 0;
47 }
```

5.3.3 Member Function Documentation

5.3.3.1 double DislocationSource::dipoleNucleationLength (double *tau*, double *mu*, double *nu*)

The nucleation length of the dipole.

When a dislocation source has experienced a shear stress greater than the critical value for a certain amount of time, it emits a dislocation dipole. In three dimensions, this is equivalent to a dislocation loop emitted by a Frank--Read source. The length of the dipole (or diameter of the loop in 3D) is such that the interaction force between the two dislocations (or line tension in 3D) balances out the applied shear stress.

Parameters

<i>tau</i>	The shear stress experienced by the dislocation source.
<i>mu</i>	Shear modulus of the material, in Pa.
<i>nu</i>	Poisson's ratio.

Returns

The length of the dislocation dipole.

Definition at line 167 of file dislocationSource.cpp.

```

168 {
169     double L = 0.0;
170
171     if (tau >= tauCritical)
172     {
173         L = (mu * this->bmag) / ( 2.0 * PI * (1.0 - nu) * this->tauCritical );
174     }
175     return (L);
176 }
177 }
```

5.3.3.2 Vector3d DislocationSource::getBurgers ()

Returns the Burgers vector of the dislocations in the dipole.

Returns

The Burgers vector of the dislocations in the dipole.

Definition at line 108 of file dislocationSource.cpp.

```

109 {
110     return (this->bvec);
111 }
```

5.3.3.3 double DislocationSource::getBurgersMag ()

Returns the magnitude of the Burgers vector.

Returns

The magnitude of the Burgers vector.

Definition at line 126 of file dislocationSource.cpp.

```
127 {  
128     return (this->bmag);  
129 }
```

5.3.3.4 int DislocationSource::getIterationCount ()

Get the count of the iterations spent at higher than critical shear stress.

Returns

Number of iterations spent at higher than critical shear stress.

Definition at line 153 of file dislocationSource.cpp.

```
154 {  
155     return (this->countIterations);  
156 }
```

5.3.3.5 Vector3d DislocationSource::getLineVector ()

Returns the line vector of the dislocations in the dipole.

Returns

The line vector of the dislocations in the dipole.

Definition at line 117 of file dislocationSource.cpp.

```
118 {  
119     return (this->lvec);  
120 }
```

5.3.3.6 int DislocationSource::getNumIterations ()

Returns the number if iterations that the dislocation source must spend experiencing a shear stress greater than the critical value before it can emit a dislocation dipole.

Returns

The number if iterations that the dislocation source must spend experiencing a shear stress greater than the critical value before it can emit a dislocation dipole.

Definition at line 144 of file dislocationSource.cpp.

```
145 {  
146     return (this->nIterations);  
147 }
```

5.3.3.7 double * Defect::getPosition () [inherited]

Returns in an array the position.

The position of the defect is saved in an array and a pointer to its first term is returned.

Returns

Pointer to the first term of the array containing the position of the defect.

Definition at line 119 of file defect.cpp.

```
120 {
121     return (this->pos.getVector ());
122 }
```

5.3.3.8 void Defect::getPosition (double * a) [inherited]

Returns the array position in a pre-allocated array.

Returns in the array provided in the argument the position of the defect. The array must be pre-allocated.

Parameters

a	Pointer to the location where the defect coordinates are to be populated.
----------	---------------------------------------------------------------------------

Definition at line 129 of file defect.cpp.

```
130 {
131     a = this->pos.getVector ();
132 }
```

5.3.3.9 double DislocationSource::getTauCritical ()

Returns the critical shear stress value for dipole emission.

Returns

The critical shear stress value for dipole emission.

Definition at line 135 of file dislocationSource.cpp.

```
136 {
137     return (this->tauCritical);
138 }
```

5.3.3.10 double Defect::getX () [inherited]

Returns the X-coordinate of the defect.

Returns

X-coordinate of the defect.

Definition at line 138 of file defect.cpp.

```
139 {
140     return (this->getValue (0));
141 }
```

5.3.3.11 double Defect::getY () [inherited]

Returns the Y-coordinate of the defect.

Returns

Y-coordinate of the defect.

Definition at line 147 of file defect.cpp.

```
148 {  
149     return (this->pos.getValue (1));  
150 }
```

5.3.3.12 double Defect::getZ () [inherited]

Returns the Z-coordinate of the defect.

Returns

Z-coordinate of the defect.

Definition at line 156 of file defect.cpp.

```
157 {  
158     return (this->pos.getValue (2));  
159 }
```

5.3.3.13 bool DislocationSource::ifEmitDipole ()

Checks if the dislocation source has experienced higher than critical shear stress for long enough to emit a dipole.

The number of iterations for which the dislocation source must experience a shear stress higher than the critical value is given in the member nIterations. When the counter variable countIterations reaches this value, the source is ready to emit a dipole, so a true value is returned. In other cases, false is returned.

Returns

The boolean result of whether the count of iterations is greater than the limiting number of iterations provided at input.

Definition at line 192 of file dislocationSource.cpp.

```
193 {  
194     return ( this->countIterations >= this->nIterations );  
195 }
```

5.3.3.14 void DislocationSource::incrementIterationCount ()

Increments the variable countIterations by 1.

Definition at line 182 of file dislocationSource.cpp.

```
183 {  
184     this->countIterations++;  
185 }
```

5.3.3.15 void DislocationSource::resetIterationCounter ()

Sets the iteration counter to zero.

Definition at line 98 of file dislocationSource.cpp.

```
99 {
100     this->countIterations = 0;
101 }
```

5.3.3.16 void DislocationSource::setBurgers (Vector3d burgers)

Sets the Burgers vector of the dislocation.

Parameters

<i>burgers</i>	Burgers vector of the dislocation.
----------------	------------------------------------

Definition at line 54 of file dislocationSource.cpp.

```
55 {
56     this->bvec = burgers;
57 }
```

5.3.3.17 void DislocationSource::setBurgersMagnitude (double bm)

Set the magnitude of the Burgers vector.

Parameters

<i>bm</i>	Magnitude of the Burgers vector.
-----------	----------------------------------

Definition at line 72 of file dislocationSource.cpp.

```
73 {
74     this->bmag = bm;
75 }
```

5.3.3.18 void DislocationSource::setLineVector (Vector3d line)

Sets the line vector of the dislocation.

Parameters

<i>line</i>	Line vector of the dislocation.
-------------	---------------------------------

Definition at line 63 of file dislocationSource.cpp.

```
64 {
65     this->lvec = line;
66 }
```

5.3.3.19 void DislocationSource::setNumIterations (int nIter)

Set the number of iterations before a dipole is emitted.

Parameters

<i>nIter</i>	Number of iterations spent at a high shear stress value before a dislocation dipole is emitted.
--------------	-------------------------------------------------------------------------------------------------

Definition at line 90 of file dislocationSource.cpp.

```

91 {
92     this->nIterations = nIter;
93 }
```

5.3.3.20 void Defect::setPosition (double * a) [inherited]

Sets the position of the defect.

The position of the defect is set to the co-ordinates present in the array pointed to by the argument. Sets the position of the defect as the values in the array pointed to by the argument.

Parameters

<i>a</i>	Pointer to the array containing the coordinates of the defect.
----------	----------------------------------------------------------------

Definition at line 56 of file defect.cpp.

```

57 {
58     this->pos.setVector (a);
59 }
```

5.3.3.21 void Defect::setPosition (double x, double y, double z) [inherited]

Sets the position of the defect.

The position of the defect is set to the co-ordinates specified by the arguments (x, y, z). Sets the position of the defect as the coordinates provided as arguments.

Parameters

<i>x</i>	X-coordinate of the defect.
<i>y</i>	Y-coordinate of the defect.
<i>z</i>	Z-coordinate of the defect.

Definition at line 69 of file defect.cpp.

```

70 {
71     this->pos.setValue (0, x);
72     this->pos.setValue (1, y);
73     this->pos.setValue (2, z);
74 }
```

5.3.3.22 void Defect::setPosition (Vector3d a) [inherited]

Sets the position of the defect.

The position of the defect is set to the position vector given by the argument a.

Parameters

<i>a</i>	Position vector of the defect.
----------	--------------------------------

Definition at line 81 of file defect.cpp.

```

82 {
83     this->position = a;
84 }

```

5.3.3.23 void DislocationSource::setTauCritical (double *tauC*)

Set the critical shear stress for dipole emission.

Parameters

<i>tauC</i>	Critical shear stress for dipole emission.
-------------	--------------------------------------------

Definition at line 81 of file dislocationSource.cpp.

```

82 {
83     this->tauCritical = tauC;
84 }

```

5.3.3.24 void Defect::setX (double *x*) [inherited]

Sets the X-coordinate of the defect.

Parameters

<i>x</i>	X-coordinate of the defect.
----------	-----------------------------

Definition at line 90 of file defect.cpp.

```

91 {
92     this->pos.setValue (0, x);
93 }

```

5.3.3.25 void Defect::setY (double *y*) [inherited]

Sets the Y-coordinate of the defect.

Parameters

<i>y</i>	Y-coordinate of the defect.
----------	-----------------------------

Definition at line 99 of file defect.cpp.

```

100 {
101     this->pos.setValue (1, y);
102 }

```

5.3.3.26 void Defect::setZ (double *z*) [inherited]

Sets the Z-coordinate of the defect.

Parameters

<i>z</i>	Z-coordinate of the defect.
----------	-----------------------------

Definition at line 108 of file defect.cpp.

```

109 {
110     this->pos.setValue (2, z);
111 }

```

5.3.3.27 virtual Stress Defect::stressField (Vector3d p, double mu, double nu) [inline],[virtual],[inherited]

Virtual function for calculating the stress field.

Returns the value of the stress field of the given defect at the position given by the argument. This is a virtual function and always returns a zero matrix. Classes which inherit this function should have their own implementations of this function to override its behaviour.

Parameters

<i>p</i>	Position vector of the the point where the stress field is to be calculated.
<i>mu</i>	Shear modulus in Pascals.
<i>nu</i>	Poisson's ratio.

Returns

[Stress](#) field value at the position p.

Reimplemented in [Dislocation](#).

Definition at line 136 of file defect.h.

```

137 {
138     // This virtual function returns a zero matrix.
139     // Inheriting classes will have functions implementing this in their own way
140     // They will override this behaviour.
141     Stress s;
142     return (s);
143 }

```

5.3.4 Field Documentation

5.3.4.1 double DislocationSource::bmag [protected]

Magnitude of the Burgers vector in metres.

The magnitude of the Burgers vector is useful for several calculations such as stress field around the dislocation.

Definition at line 45 of file dislocationSource.h.

5.3.4.2 Vector3d DislocationSource::bvec [protected]

Burgers vector of the dislocation.

Definition at line 28 of file dislocationSource.h.

5.3.4.3 int DislocationSource::countIterations [protected]

Counter variable for the number of consecutive iterations the dislocation source has experienced a shear stress greater than its critical value.

A dislocation source needs to experience a shear stress higher than a critical value, given by tauCritical, for a certain amount of time before it is triggered and it emits a dislocation dipole. This limiting number of iterations is given by the variable nIterations, and this variable countIterations is a counter variable. Once this limit is reached, a dipole is emitted and this counter variable is set to zero.

Definition at line 69 of file dislocationSource.h.

5.3.4.4 Vector3d DislocationSource::lvec [protected]

Line vector if the dislocation.

Definition at line 33 of file dislocationSource.h.

5.3.4.5 bool DislocationSource::mobile [protected]

Boolean term indicating mobility.

For mobile dislocations this term is true and for pinned dislocations it is false.

Definition at line 39 of file dislocationSource.h.

5.3.4.6 int DislocationSource::nIterations [protected]

Number of iterations before a dipole is emitted.

A dislocation dipole source needs to experience a certain critical level of shear stress for a certain amount of time before it can emit a dipole. The amount of time is represented instead by a number of iterations nIterations.

Definition at line 63 of file dislocationSource.h.

5.3.4.7 Vector3d Defect::pos [protected], [inherited]

Position vector of the defect in 2D space.

Definition at line 26 of file defect.h.

5.3.4.8 RotationMatrix DislocationSource::rotationMatrix [protected]

The rotation matrix for rotating from the global to the local co-ordinate system and vice-versa.

This is the rotation matrix that represents the relationship between the global and local co-ordinate systems. It is used to convert tensors and vectors between the two systems. The rotation matrix needs to be calculated once and may be refreshed periodically if lattice rotation is implemented. In the absence of lattice rotation, the matrix will remain invariant.

Definition at line 51 of file dislocationSource.h.

5.3.4.9 double DislocationSource::tauCritical [protected]

Critical stress for the emission of a dislocation dipole.

A dislocation dipole source needs to experience a certain critical level of shear stress for a certain amount of time before it can emit a dipole. This critical stress is given by tauCritical.

Definition at line 57 of file dislocationSource.h.

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

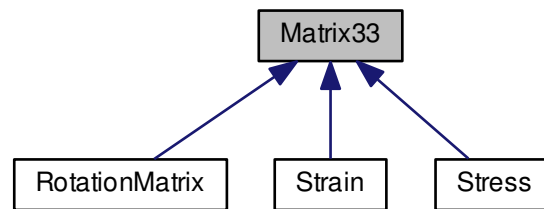
- [dislocationSource.h](#)
- [dislocationSource.cpp](#)

5.4 Matrix33 Class Reference

[Matrix33](#) class representing a 3x3 square matrix.

```
#include <matrix33.h>
```

Inheritance diagram for Matrix33:



Public Member Functions

- [Matrix33](#) ()
Default constructor.
- [Matrix33](#) (double **a)
Constructor with the values provided in a 3x3 matrix.
- [Matrix33](#) ([Vector3d](#) a)
Constructor to create the matrix from the dyadic product of a vector with itself.
- [Matrix33](#) ([Vector3d](#) a, [Vector3d](#) b)
Constructor with the vectors, the product of which will result in the matrix.
- void [setValue](#) (int row, int column, double value)
Function to set the value of an element indicated by its position.
- double [getValue](#) (int row, int column)
Returns the value of the element located by the row and column indices provided.
- [Matrix33](#) [adjugate](#) ()
Returns the adjugate matrix of the present matrix.
- [Matrix33](#) [operator+](#) (const [Matrix33](#) &) const
Operator for addition of two matrices.
- void [operator+=](#) (const [Matrix33](#) &)
Operator for reflexive addition of two matrices.
- [Matrix33](#) [operator-](#) (const [Matrix33](#) &) const
Operator for the subtraction of two matrices.
- void [operator-=](#) (const [Matrix33](#) &)
Operator for reflexive subtraction of two matrices.
- [Matrix33](#) [operator*](#) (const double &) const
Operator for scaling the matrix by a scalar.
- void [operator*=](#) (const double &)
Operator for reflexive scaling of the matrix by a scalar.
- [Matrix33](#) [operator*](#) (const [Matrix33](#) &) const
Operator for the multiplication of two matrices.
- void [operator*=](#) (const [Matrix33](#) &)
Operator for reflexive multiplication of two matrices.
- [Vector3d](#) [operator*](#) (const [Vector3d](#) &) const
Operator for the multiplication of a matrix with a vector.
- [Matrix33](#) [operator^](#) () const

- Transpose.*
- double `operator~` () const
- Determinant.*
- `Matrix33 operator!` () const
- Inverse.*

Protected Attributes

- double `x` [3][3]
- Array containing the elements of the matrix.*

5.4.1 Detailed Description

`Matrix33` class representing a 3x3 square matrix.

This class represents a 3x3 square matrix. The member functions and operators define various operations that may be carried out on the matrix.

Definition at line 20 of file `matrix33.h`.

5.4.2 Constructor & Destructor Documentation

5.4.2.1 `Matrix33::Matrix33 ()`

Default constructor.

Initializes the matrix with all elements equal to 0.0.

Definition at line 17 of file `matrix33.cpp`.

```

18 {
19     int i, j;
20
21     for (i=0; i<3; i++)
22     {
23         for (j=0; j<3; j++)
24         {
25             this->x[i][j] = 0.0;
26         }
27     }
28 }
```

5.4.2.2 `Matrix33::Matrix33 (double ** a)`

Constructor with the values provided in a 3x3 matrix.

Populated the matrix with data present in corresponding elements of the provided 3x3 array.

Parameters

<code>a</code>	Pointer to the two-dimensional 3x3 array.
----------------	-------------------------------------------

Definition at line 35 of file `matrix33.cpp`.

```

36 {
37     int i, j;
38
39     for (i=0; i<3; i++)
40     {
41         for (j=0; j<3; j++)
42         {
43             this->x[i][j] = a[i][j];
44         }
45     }
46 }
```

```

44     }
45 }
46 }

```

5.4.2.3 Matrix33::Matrix33 (Vector3d a)

Constructor to create the matrix from the dyadic product of a vector with itself.

The matrix is created by performing the dyadic product of the provided vector with itself.

Parameters

<i>a</i>	The vector whose dyadic product results in the matrix.
----------	--------------------------------------------------------

Definition at line 53 of file matrix33.cpp.

```

54 {
55     int i, j;
56
57     for (i=0; i<3; i++)
58     {
59         for (j=0; j<3; j++)
60         {
61             this->x[i][j] = a.x[i] * a.x[j];
62         }
63     }
64 }

```

5.4.2.4 Matrix33::Matrix33 (Vector3d a, Vector3d b)

Constructor with the vectors, the product of which will result in the matrix.

The matrix is created from the product the first vector with the second.

Parameters

<i>a</i>	First vector.
<i>b</i>	Second vector.

Definition at line 72 of file matrix33.cpp.

```

73 {
74     int i, j;
75
76     for (i=0; i<3; i++)
77     {
78         for (j=0; j<3; j++)
79         {
80             this->x[i][j] = a.x[i] * b.x[j];
81         }
82     }
83 }

```

5.4.3 Member Function Documentation

5.4.3.1 Matrix33 Matrix33::adjugate ()

Returns the adjugate matrix of the present matrix.

The adjugate matrix of the present matrix is returned. The adjugate matrix is calculated by evaluating the determinant of the cofactor matrix of each element, and then replacing the corresponding element position by the value of the determinant. This operation is useful in calculating the inverse of a matrix.

Returns

The adjugate matrix of the present matrix.

Definition at line 129 of file matrix33.cpp.

```

130 {
131     Matrix33 adj;
132
133     adj.x[0][0] = (this->x[1][1]*this->x[2][2]) - (this->x[1][2]*this->x[2][1]);
134     adj.x[0][1] = (this->x[1][2]*this->x[2][0]) - (this->x[1][0]*this->x[2][2]);
135     adj.x[0][2] = (this->x[1][0]*this->x[2][1]) - (this->x[1][1]*this->x[2][0]);
136
137     adj.x[1][0] = (this->x[2][1]*this->x[0][2]) - (this->x[0][1]*this->x[2][2]);
138     adj.x[1][1] = (this->x[2][2]*this->x[0][0]) - (this->x[2][0]*this->x[0][2]);
139     adj.x[1][2] = (this->x[2][0]*this->x[0][1]) - (this->x[2][1]*this->x[0][0]);
140
141     adj.x[2][0] = (this->x[0][1]*this->x[1][2]) - (this->x[0][2]*this->x[1][1]);
142     adj.x[2][1] = (this->x[0][2]*this->x[1][0]) - (this->x[0][0]*this->x[1][2]);
143     adj.x[2][2] = (this->x[0][0]*this->x[1][1]) - (this->x[0][1]*this->x[1][0]);
144
145     return (adj);
146 }
```

5.4.3.2 double Matrix33::getValue (int row, int column)

Returns the value of the element located by the row and column indices provided.

The value of the element indicated by the arguments row and column is returned. The values of row and column must correspond to array indices, and thus can be one of 0, 1 and 2. In any other case 0.0 is returned.

Parameters

<i>row</i>	Row index of the element.
<i>column</i>	Column index of the element.

Returns

Value of the element located at the given position.

Definition at line 111 of file matrix33.cpp.

```

112 {
113     if (row>=0 && row<3)
114     {
115         if (column>=0 && column<3)
116         {
117             return (this->x[row][column]);
118         }
119     }
120
121     return (0.0);
122 }
```

5.4.3.3 Matrix33 Matrix33::operator! () const

Inverse.

Returns in a new matrix the inverse of the current matrix. If the current matrix is non-invertible, a zero matrix is returned.

Returns

Matrix with the inverse of the current matrix. If the current matrix is non-invertible, a zero matrix is returned.

Definition at line 370 of file matrix33.cpp.

```

371 {
372     Matrix33 r;    // Result matrix
373
374     double determinant = ~(*this);
375
376     if (determinant == 0.0)
377     {
378         // The matrix is non-invertible
379         return (r);    // Zero matrix
380     }
381
382     // If we are still here, the matrix is invertible
383
384     // Transpose
385     Matrix33 tr = ^(*this);
386
387     // Find Adjugate matrix
388     Matrix33 adj = tr.adjugate();
389
390     // Calculate the inverse by dividing the adjugate matrix by the determinant
391     r = adj * (1.0/determinant);
392
393     return (r);
394 }

```

5.4.3.4 Matrix33 Matrix33::operator* (const double & p) const

Operator for scaling the matrix by a scalar.

Scales the current matrix by the scalar provided and returns the result in a third matrix.

Returns

Matrix containing the result of scaling the current matrix by the scalar provided as argument.

Definition at line 233 of file matrix33.cpp.

```

234 {
235     int i, j;
236     Matrix33 r;
237
238     for (i=0; i<3; i++)
239     {
240         for (j=0; j<3; j++)
241         {
242             r.x[i][j] = this->x[i][j] * p;
243         }
244     }
245
246     return (r);
247 }

```

5.4.3.5 Matrix33 Matrix33::operator* (const Matrix33 & p) const

Operator for the multiplication of two matrices.

Multiplies the current matrix with another 3x3 matrix and returns the result in a new matrix.

Returns

The result of the multiplication of the current matrix with the one provided as argument.

Definition at line 271 of file matrix33.cpp.

```

272 {
273     int i, j, k;
274     Matrix33 r;
275
276     for (i=0; i<3; i++)
277     {
278         for (j=0; j<3; j++)

```

```

279     {
280         r.x[i][j] = 0.0;
281         for (k=0; k<3; k++)
282         {
283             r.x[i][j] += this->x[i][k] * p.x[k][j];
284         }
285     }
286 }
287
288 return (r);
289 }

```

5.4.3.6 Vector3d Matrix33::operator* (const Vector3d & v) const

Operator for the multiplication of a matrix with a vector.

Returns in a vector the result of the multiplication of the current matrix with the provided vector.

Returns

The vector resulting from the multiplication of the current matrix with a vector.

Definition at line 311 of file matrix33.cpp.

```

312 {
313     Vector3d r(0.0, 0.0, 0.0);
314     int i, j;
315
316     for (i=0; i<3; i++)
317     {
318         for (j=0; j<3; j++)
319         {
320             r[i] += this->x[i][j] * v.x[j];
321         }
322     }
323
324     return (r);
325 }

```

5.4.3.7 void Matrix33::operator*=(const double & p)

Operator for reflexive scaling of the matrix by a scalar.

Scales the current matrix by the scalar provided and populates the current matrix elements with the result.

Definition at line 253 of file matrix33.cpp.

```

254 {
255     int i, j;
256
257     for (i=0; i<3; i++)
258     {
259         for (j=0; j<3; j++)
260         {
261             this->x[i][j] *= p;
262         }
263     }
264 }

```

5.4.3.8 void Matrix33::operator*=(const Matrix33 & p)

Operator for reflexive multiplication of two matrices.

Multiplies the current matrix with another 3x3 matrix and populates the elements of the current matrix with the result.

Definition at line 295 of file matrix33.cpp.

```

296 {
297     Matrix33* r = new Matrix33;
298
299     *r = (*this) * p;
300     *this = *r;
301
302     delete(r);
303     r = NULL;
304 }

```

5.4.3.9 Matrix33 Matrix33::operator+ (const Matrix33 & p) const

Operator for addition of two matrices.

Adds the current matrix to the provided matrix and returns a third matrix with the result.

Returns

Matrix containing the sum of the present matrix and the one provided.

Definition at line 155 of file matrix33.cpp.

```

156 {
157     int i, j;
158     Matrix33 r;
159
160     for (i=0; i<3; i++)
161     {
162         for (j=0; j<3; j++)
163         {
164             r.x[i][j] = this->x[i][j] + p.x[i][j];
165         }
166     }
167
168     return (r);
169 }

```

5.4.3.10 void Matrix33::operator+= (const Matrix33 & p)

Operator for reflexive addition of two matrices.

Adds the current matrix to the provided matrix and populates the current matrix elements with the result.

Definition at line 175 of file matrix33.cpp.

```

176 {
177     int i, j;
178
179     for (i=0; i<3; i++)
180     {
181         for (j=0; j<3; j++)
182         {
183             this->x[i][j] += p.x[i][j];
184         }
185     }
186 }

```

5.4.3.11 Matrix33 Matrix33::operator- (const Matrix33 & p) const

Operator for the subtraction of two matrices.

Subtracts the given matrix from the current matrix and returns the result in a new matrix.

Returns

Matrix containing the result of subtracting the provided matrix from the current matrix.

Definition at line 194 of file matrix33.cpp.

```

195 {
196     int i, j;
197     Matrix33 r;
198
199     for (i=0; i<3; i++)
200     {
201         for (j=0; j<3; j++)
202         {
203             r.x[i][j] = this->x[i][j] - p.x[i][j];
204         }
205     }
206
207     return (r);
208 }
```

5.4.3.12 void Matrix33::operator-= (const Matrix33 & p)

Operator for reflexive subtraction of two matrices.

Subtracts the given matrix from the current matrix and populates the current matrix with the result.

Definition at line 214 of file matrix33.cpp.

```

215 {
216     int i, j;
217
218     for (i=0; i<3; i++)
219     {
220         for (j=0; j<3; j++)
221         {
222             this->x[i][j] -= p.x[i][j];
223         }
224     }
225 }
```

5.4.3.13 Matrix33 Matrix33::operator^ () const

Transpose.

Performs the transpose of the current matrix.

Returns

A new matrix with the transpose of the current matrix.

Definition at line 333 of file matrix33.cpp.

```

334 {
335     Matrix33 r;
336     int i, j;
337
338     for (i=0; i<3; i++)
339     {
340         for (j=0; j<3; j++)
341         {
342             r.x[i][j] = this->x[j][i];
343         }
344     }
345
346     return (r);
347 }
```

5.4.3.14 double Matrix33::operator~ () const

Determinant.

Calculates the determinant of the current matrix.

Returns

Returns the determinant of the current matrix.

Definition at line 354 of file matrix33.cpp.

```

355 {
356     double d = 0.0;
357
358     d += this->x[0][0] * ( (this->x[1][1]*this->x[2][2]) - (this->x[2][1]*this->
x[1][2]) );
359     d += this->x[0][1] * ( (this->x[1][2]*this->x[2][0]) - (this->x[1][0]*this->
x[2][2]) );
360     d += this->x[0][2] * ( (this->x[1][0]*this->x[2][1]) - (this->x[2][0]*this->
x[1][1]) );
361
362     return (d);
363 }
```

5.4.3.15 void Matrix33::setValue (int row, int column, double value)

Function to set the value of an element indicated by its position.

The element indicated by the arguments row and column is set to the value provided. The values of row and column must correspond to array indices, and thus can be one of 0, 1 and 2. In any other case 0.0 is returned.

Parameters

<i>row</i>	Row index of the element.
<i>column</i>	Column index of the element.
<i>value</i>	Value that the element is to be set to.

Definition at line 93 of file matrix33.cpp.

```

94 {
95     if (row>=0 && row<3)
96     {
97         if (column>=0 && column<3)
98         {
99             this->x[row][column] = value;
100         }
101     }
102 }
```

5.4.4 Field Documentation

5.4.4.1 double Matrix33::x[3][3] [protected]

Array containing the elements of the matrix.

Definition at line 26 of file matrix33.h.

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

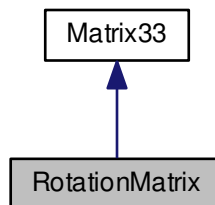
- [matrix33.h](#)
- [matrix33.cpp](#)

5.5 RotationMatrix Class Reference

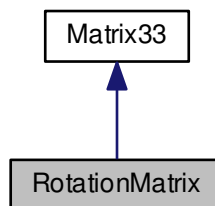
[RotationMatrix](#) class to represent a rotation matrix.

```
#include <rotationMatrix.h>
```

Inheritance diagram for RotationMatrix:



Collaboration diagram for RotationMatrix:



Public Member Functions

- [RotationMatrix](#) ()
Default constructor.
- [RotationMatrix](#) ([Vector3d](#) *unPrimed, [Vector3d](#) *primed)
Defines the rotation matrix based on two co-ordinate systems.
- void [setValue](#) (int row, int column, double value)
Function to set the value of an element indicated by its position.
- double [getValue](#) (int row, int column)
Returns the value of the element located by the row and column indices provided.
- [Matrix33](#) [adjugate](#) ()
Returns the adjugate matrix of the present matrix.
- [Matrix33](#) [operator+](#) (const [Matrix33](#) &) const
Operator for addition of two matrices.
- void [operator+=](#) (const [Matrix33](#) &)
Operator for reflexive addition of two matrices.

- **Matrix33 operator-** (const **Matrix33** &) const
Operator for the subtraction of two matrices.
- void **operator-=** (const **Matrix33** &)
Operator for reflexive subtraction of two matrices.
- **Matrix33 operator*** (const double &) const
Operator for scaling the matrix by a scalar.
- **Matrix33 operator*** (const **Matrix33** &) const
Operator for the multiplication of two matrices.
- **Vector3d operator*** (const **Vector3d** &) const
Operator for the multiplication of a matrix with a vector.
- void **operator*=** (const double &)
Operator for reflexive scaling of the matrix by a scalar.
- void **operator*=** (const **Matrix33** &)
Operator for reflexive multiplication of two matrices.
- **Matrix33 operator^** () const
Transpose.
- double **operator~** () const
Determinant.
- **Matrix33 operator!** () const
Inverse.

Protected Attributes

- double **x** [3][3]
Array containing the elements of the matrix.

5.5.1 Detailed Description

RotationMatrix class to represent a rotation matrix.

The member functions of this class create a rotation matrix for carrying out rotations in 3D and transformation of axes.

Definition at line 19 of file rotationMatrix.h.

5.5.2 Constructor & Destructor Documentation

5.5.2.1 RotationMatrix::RotationMatrix ()

Default constructor.

Initializes the rotation matrix with a unit matrix.

Definition at line 14 of file rotationMatrix.cpp.

```

15 {
16     int i, j;
17
18     for ( i=0; i<3; i++ ) {
19         for ( j=0; j<3; j++ ) {
20             if ( i==j ) {
21                 this->setValue ( i, j, 1.0 );
22             }
23             else {
24                 this->setValue ( i, j, 0.0 );
25             }
26         }
27     }
28 }
```

5.5.2.2 RotationMatrix::RotationMatrix (Vector3d * *unPrimed*, Vector3d * *primed*)

Defines the rotation matrix based on two co-ordinate systems.

The rotation matrix is created using the axes of the two co-ordinate systems provided as arguments. The vectors must be normalized to be unit vectors.

Parameters

<i>unPrimed</i>	Pointer to the array containing the three axes vectors of the unprimed (old) system.
<i>primed</i>	Pointer to the array containing the three axes vectors of the primed (new) system.

Definition at line 36 of file rotationMatrix.cpp.

```

37 {
38     int i, j;
39
40     for ( i=0; i<3; i++ ) {
41         for ( j=0; j<3; j++ ) {
42             this->setValue ( i, j, primed[i]*unPrimed[j] );
43         }
44     }
45 }
```

5.5.3 Member Function Documentation

5.5.3.1 Matrix33 Matrix33::adjugate () [inherited]

Returns the adjugate matrix of the present matrix.

The adjugate matrix of the present matrix is returned. The adjugate matrix is calculated by evaluating the determinant of the cofactor matrix of each element, and then replacing the corresponding element position by the value of the determinant. This operation is useful in calculating the inverse of a matrix.

Returns

The adjugate matrix of the present matrix.

Definition at line 129 of file matrix33.cpp.

```

130 {
131     Matrix33 adj;
132
133     adj.x[0][0] = (this->x[1][1]*this->x[2][2]) - (this->x[1][2]*this->x[2][1]);
134     adj.x[0][1] = (this->x[1][2]*this->x[2][0]) - (this->x[1][0]*this->x[2][2]);
135     adj.x[0][2] = (this->x[1][0]*this->x[2][1]) - (this->x[1][1]*this->x[2][0]);
136
137     adj.x[1][0] = (this->x[2][1]*this->x[0][2]) - (this->x[0][1]*this->x[2][2]);
138     adj.x[1][1] = (this->x[2][2]*this->x[0][0]) - (this->x[2][0]*this->x[0][2]);
139     adj.x[1][2] = (this->x[2][0]*this->x[0][1]) - (this->x[2][1]*this->x[0][0]);
140
141     adj.x[2][0] = (this->x[0][1]*this->x[1][2]) - (this->x[0][2]*this->x[1][1]);
142     adj.x[2][1] = (this->x[0][2]*this->x[1][0]) - (this->x[0][0]*this->x[1][2]);
143     adj.x[2][2] = (this->x[0][0]*this->x[1][1]) - (this->x[0][1]*this->x[1][0]);
144
145     return (adj);
146 }
```

5.5.3.2 double Matrix33::getValue (int *row*, int *column*) [inherited]

Returns the value of the element located by the row and column indices provided.

The value of the element indicated by the arguments row and column is returned. The values of row and column must correspond to array indices, and thus can be one of 0, 1 and 2. In any other case 0.0 is returned.

Parameters

<i>row</i>	Row index of the element.
<i>column</i>	Column index of the element.

Returns

Value of the element located at the given position.

Definition at line 111 of file matrix33.cpp.

```

112 {
113     if (row>=0 && row<3)
114     {
115         if (column>=0 && column<3)
116         {
117             return (this->x[row][column]);
118         }
119     }
120
121     return (0.0);
122 }
```

5.5.3.3 Matrix33 Matrix33::operator! () const [inherited]

Inverse.

Returns in a new matrix the inverse of the current matrix. If the current matrix is non-invertible, a zero matrix is returned.

Returns

Matrix with the inverse of the current matrix. If the current matrix is non-invertible, a zero matrix is returned.

Definition at line 370 of file matrix33.cpp.

```

371 {
372     Matrix33 r;    // Result matrix
373
374     double determinant = ~(*this);
375
376     if (determinant == 0.0)
377     {
378         // The matrix is non-invertible
379         return (r);    // Zero matrix
380     }
381
382     // If we are still here, the matrix is invertible
383
384     // Transpose
385     Matrix33 tr = ^(*this);
386
387     // Find Adjugate matrix
388     Matrix33 adj = tr.adjugate();
389
390     // Calculate the inverse by dividing the adjugate matrix by the determinant
391     r = adj * (1.0/determinant);
392
393     return (r);
394 }
```

5.5.3.4 Matrix33 Matrix33::operator* (const double & p) const [inherited]

Operator for scaling the matrix by a scalar.

Scales the current matrix by the scalar provided and returns the result in a third matrix.

Returns

Matrix containing the result of scaling the current matrix by the scalar provided as argument.

Definition at line 233 of file matrix33.cpp.

```

234 {
235     int i, j;
236     Matrix33 r;
237
238     for (i=0; i<3; i++)
239     {
240         for (j=0; j<3; j++)
241         {
242             r.x[i][j] = this->x[i][j] * p;
243         }
244     }
245
246     return (r);
247 }
```

5.5.3.5 Matrix33 Matrix33::operator* (const Matrix33 & p) const [inherited]

Operator for the multiplication of two matrices.

Multiplies the current matrix with another 3x3 matrix and returns the result in a new matrix.

Returns

The result of the multiplication of the current matrix with the one provided as argument.

Definition at line 271 of file matrix33.cpp.

```

272 {
273     int i, j, k;
274     Matrix33 r;
275
276     for (i=0; i<3; i++)
277     {
278         for (j=0; j<3; j++)
279         {
280             r.x[i][j] = 0.0;
281             for (k=0; k<3; k++)
282             {
283                 r.x[i][j] += this->x[i][k] * p.x[k][j];
284             }
285         }
286     }
287
288     return (r);
289 }
```

5.5.3.6 Vector3d Matrix33::operator* (const Vector3d & v) const [inherited]

Operator for the multiplication of a matrix with a vector.

Returns in a vector the result of the multiplication of the current matrix with the provided vector.

Returns

The vector resulting from the multiplication of the current matrix with a vector.

Definition at line 311 of file matrix33.cpp.

```

312 {
313     Vector3d r(0.0, 0.0, 0.0);
314     int i, j;
315
316     for (i=0; i<3; i++)
```

```

317     {
318         for (j=0; j<3; j++)
319         {
320             r[i] += this->x[i][j] * v.x[j];
321         }
322     }
323
324     return (r);
325 }

```

5.5.3.7 void Matrix33::operator*=(const double & p) [inherited]

Operator for reflexive scaling of the matrix by a scalar.

Scales the current matrix by the scalar provided and populates the current matrix elements with the result.

Definition at line 253 of file matrix33.cpp.

```

254 {
255     int i, j;
256
257     for (i=0; i<3; i++)
258     {
259         for (j=0; j<3; j++)
260         {
261             this->x[i][j] *= p;
262         }
263     }
264 }

```

5.5.3.8 void Matrix33::operator*=(const Matrix33 & p) [inherited]

Operator for reflexive multiplication of two matrices.

Multiplies the current matrix with another 3x3 matrix and populates the elements of the current matrix with the result.

Definition at line 295 of file matrix33.cpp.

```

296 {
297     Matrix33* r = new Matrix33;
298
299     *r = (*this) * p;
300     *this = *r;
301
302     delete(r);
303     r = NULL;
304 }

```

5.5.3.9 Matrix33 Matrix33::operator+ (const Matrix33 & p) const [inherited]

Operator for addition of two matrices.

Adds the current matrix to the provided matrix and returns a third matrix with the result.

Returns

Matrix containing the sum of the present matrix and the one provided.

Definition at line 155 of file matrix33.cpp.

```

156 {
157     int i, j;
158     Matrix33 r;
159
160     for (i=0; i<3; i++)
161     {
162         for (j=0; j<3; j++)
163         {

```



```

164         r.x[i][j] = this->x[i][j] + p.x[i][j];
165     }
166 }
167
168 return (r);
169 }

```

5.5.3.10 void Matrix33::operator+=(const Matrix33 & p) [inherited]

Operator for reflexive addition of two matrices.

Adds the current matrix to the provided matrix and populates the current matrix elements with the result.

Definition at line 175 of file matrix33.cpp.

```

176 {
177     int i, j;
178
179     for (i=0; i<3; i++)
180     {
181         for (j=0; j<3; j++)
182         {
183             this->x[i][j] += p.x[i][j];
184         }
185     }
186 }

```

5.5.3.11 Matrix33 Matrix33::operator- (const Matrix33 & p) const [inherited]

Operator for the subtraction of two matrices.

Subtracts the given matrix from the current matrix and returns the result in a new matrix.

Returns

Matrix containing the result of subtracting the provided matrix from the current matrix.

Definition at line 194 of file matrix33.cpp.

```

195 {
196     int i, j;
197     Matrix33 r;
198
199     for (i=0; i<3; i++)
200     {
201         for (j=0; j<3; j++)
202         {
203             r.x[i][j] = this->x[i][j] - p.x[i][j];
204         }
205     }
206
207     return (r);
208 }

```

5.5.3.12 void Matrix33::operator-= (const Matrix33 & p) [inherited]

Operator for reflexive subtraction of two matrices.

Subtracts the given matrix from the current matrix and populates the current matrix with the result.

Definition at line 214 of file matrix33.cpp.

```

215 {
216     int i, j;
217
218     for (i=0; i<3; i++)
219     {
220         for (j=0; j<3; j++)

```

```

221     {
222         this->x[i][j] -= p.x[i][j];
223     }
224 }
225 }

```

5.5.3.13 Matrix33 Matrix33::operator^() const [inherited]

Transpose.

Performs the transpose of the current matrix.

Returns

A new matrix with the transpose of the current matrix.

Definition at line 333 of file matrix33.cpp.

```

334 {
335     Matrix33 r;
336     int i, j;
337
338     for (i=0; i<3; i++)
339     {
340         for (j=0; j<3; j++)
341         {
342             r.x[i][j] = this->x[j][i];
343         }
344     }
345
346     return (r);
347 }

```

5.5.3.14 double Matrix33::operator~() const [inherited]

Determinant.

Calculates the determinant of the current matrix.

Returns

Returns the determinant of the current matrix.

Definition at line 354 of file matrix33.cpp.

```

355 {
356     double d = 0.0;
357
358     d += this->x[0][0] * ( (this->x[1][1]*this->x[2][2]) - (this->x[2][1]*this->
359     x[1][2]) );
360     d += this->x[0][1] * ( (this->x[1][2]*this->x[2][0]) - (this->x[1][0]*this->
361     x[2][2]) );
362     d += this->x[0][2] * ( (this->x[1][0]*this->x[2][1]) - (this->x[2][0]*this->
363     x[1][1]) );
364
365     return (d);
366 }

```

5.5.3.15 void Matrix33::setValue (int row, int column, double value) [inherited]

Function to set the value of an element indicated by its position.

The element indicated by the arguments row and column is set to the value provided. The values of row and column must correspond to array indices, and thus can be one of 0, 1 and 2. In any other case 0.0 is returned.

Parameters

<i>row</i>	Row index of the element.
<i>column</i>	Column index of the element.
<i>value</i>	Value that the element is to be set to.

Definition at line 93 of file matrix33.cpp.

```

94 {
95     if (row>=0 && row<3)
96     {
97         if (column>=0 && column<3)
98         {
99             this->x[row][column] = value;
100         }
101     }
102 }
```

5.5.4 Field Documentation

5.5.4.1 `double Matrix33::x[3][3]` `[protected]`, `[inherited]`

Array containing the elements of the matrix.

Definition at line 26 of file matrix33.h.

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

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

5.6 SlipPlane Class Reference

[SlipPlane](#) class representing a slip plane in the simulation.

```
#include <slipPlane.h>
```


- bool [getDislocation](#) (int i, [Dislocation](#) *d)
Get the dislocation on the slip plane indicated by the index provided as argument.
- std::vector< [Dislocation](#) > [getDislocationList](#) ()
Get the entire vector container which holds the dislocations lying on this slip plane.
- bool [getDislocationSource](#) (int i, [DislocationSource](#) *dSource)
Get the dislocation source on the slip plane indicated by the index provided as argument.
- std::vector< [DislocationSource](#) > [getDislocationSourceList](#) ()
Get the entire vector container which holds the dislocation sources lying on this slip plane.
- [RotationMatrix](#) [getRotationMatrix](#) ()
Get the rotation matrix for this slip plane.
- [Vector3d](#) [getAxis](#) (int i)
Get the axis (expressed in the global co-ordinate system) of the slip plane's local co-ordinate system, as indicated by the argument. (0, 1, 2)=(x, y, z).
- void [calculateRotationMatrix](#) ()
Calculates the rotation matrix for this slip plane.

Protected Attributes

- [Vector3d](#) [extremities](#) [2]
The extremities of the slip plane.
- [Vector3d](#) [normalVector](#)
The normal vector to the slip plane.
- [Vector3d](#) [position](#)
The position vector of the slip plane.
- std::vector< [Dislocation](#) > [dislocations](#)
STL vector container with dislocations.
- std::vector< [DislocationSource](#) > [dislocationSources](#)
STL vector container with dislocation sources.
- [RotationMatrix](#) [rotationMatrix](#)
Rotation matrix for co-ordinate system transformations.

5.6.1 Detailed Description

[SlipPlane](#) class representing a slip plane in the simulation.

This is the definition of the class [SlipPlane](#). It represents a slip plane in the simulation. A slip plane is considered to be a collection of defects, such as dislocations and dislocation sources. In these simulations in two dimensions, the slip plane becomes a straight line. Its attributes are: position vectors of the extremities, normal vector (since we are concerned with the cubic system here, the normal vector's indices are the same as those of the plane), and the collection of defects.

Definition at line 26 of file [slipPlane.h](#).

5.6.2 Constructor & Destructor Documentation

5.6.2.1 [SlipPlane::SlipPlane](#) ()

Default constructor.

The slip plane is initialized with default parameters.

5.6.2.2 SlipPlane::SlipPlane (Vector3d * ends, Vector3d normal, Vector3d pos, std::vector< Dislocation > dislocationList, std::vector< DislocationSource > dislocationSourceList)

Constructor that specifies all members explicitly.

The slip plane is initialized with parameters specified in the arguments.

Parameters

<i>ends</i>	Pointer to an array of type Vector3d , containing the position vectors of the extremities of the slip plane in consecutive locations.
<i>normal</i>	The normal vector of the slip plane.
<i>pos</i>	The position vector of the slip plane. (This parameter is useful for locating the slip plane within a slip system)
<i>dislocationList</i>	A vector container of type Dislocation containing the dislocations lying on this slip plane.
<i>dislocation-SourceList</i>	A vector container of type DislocationSource containing the dislocation sources lying on this slip plane.

Definition at line 28 of file slipPlane.cpp.

```

29 {
30     this->setExtremities (ends);
31     this->setNormal (normal);
32     this->setPosition (pos);
33     this->setDislocationList (dislocationList);
34     this->setDislocationSourceList (dislocationSourceList);
35
36     this->calculateRotationMatrix ();
37 }
```

5.6.3 Member Function Documentation

5.6.3.1 void SlipPlane::calculateRotationMatrix ()

Calculates the rotation matrix for this slip plane.

The slip plane has a local co-ordinate system whose axes are the following: z-axis||normal vector and x-axis||slip plane vector (vector joining the extremities). The rotation matrix is calculated in order to carry out transformations between the global and local co-ordinate systems.

Definition at line 234 of file slipPlane.cpp.

```

235 {
236     Vector3d *unPrimed = new Vector3d[3]; // Old system (global)
237     Vector3d *primed   = new Vector3d[3]; // New system (local)
238
239     int i, j;
240
241     // Prepare the global and local systems
242     for (i=0; i<3; i++)
243     {
244         for (j=0; j<3; j++)
245         {
246             unPrimed[i].setValue(j, (double) (i==j));
247         }
248         primed[i] = this->getAxis(i);
249     }
250
251
252     // Calculate the rotationMatrix
253     this->rotationMatrix = RotationMatrix(unPrimed, primed);
254
255     // Free memory
256     delete(unPrimed);    unPrimed = NULL;
257     delete(primed);      primed = NULL;
258 }
```

5.6.3.2 Vector3d SlipPlane::getAxis (int *i*)

Get the axis (expressed in the global co-ordinate system) of the slip plane's local co-ordinate system, as indicated by the argument. (0, 1, 2)=(x, y, z).

Parameters

<i>i</i>	Index of the axis that is to be returned. (0, 1, 2)=(x, y, z).
----------	----------------------------------------------------------------

Returns

The desired axis of the slip plane's local co-ordinate system, expressed in the global co-ordinate system. In case of invalid argument, a zero vector is returned.

Definition at line 204 of file slipPlane.cpp.

```

205 {
206     Vector3d = axis;
207
208     if (i==2)
209     {
210         // Z-axis
211         axis = this->normalVector;
212     }
213
214     if (i==0)
215     {
216         // X-axis
217         axis = (this->extremities[1] - this->extremities[0]);
218     }
219
220     if (i==1)
221     {
222         // Y-axis = Z x X
223         axis = this->getAxis(2) ^ this->getAxis(0);
224     }
225
226     return ( axis.normaize() );
227 }
```

5.6.3.3 bool SlipPlane::getDislocation (int *i*, Dislocation * *d*)

Get the dislocation on the slip plane indicated by the index provided as argument.

The slip plane contains several dislocations that are stored in a vector container. This function returns the dislocation in that vector that corresponds to the index provided as argument.

Parameters

<i>i</i>	Index of the required dislocation in the vector. This value should be greater than or equal to 0 and less than the number of dislocations on the slip plane.
<i>d</i>	Pointer to the memory location where the required dislocation is to be stored. Space in memory must be pre-allocated.

Returns

True if the provided index is greater than or equal to 0 and less than the number of dislocations on the slip plane (the memory location pointed to by *d* is populated with the [Dislocation](#) data). Otherwise, the return value is false.

Definition at line 139 of file slipPlane.cpp.

```

140 {
141     if (i>=0 && i<this->dislocations.size ())
142     {
```

```

143     *d = this->dislocations[i];
144     return (true);
145 }
146 else
147 {
148     return (false);
149 }
150 }

```

5.6.3.4 std::vector< Dislocation > SlipPlane::getDislocationList ()

Get the entire vector container which holds the dislocations lying on this slip plane.

Returns

The vector of dislocations lying on this slip plane.

Definition at line 156 of file slipPlane.cpp.

```

157 {
158     return (this->dislocations);
159 }

```

5.6.3.5 bool SlipPlane::getDislocationSource (int i, DislocationSource * dSource)

Get the dislocation source on the slip plane indicated by the index provided as argument.

The slip plane contains several dislocation sources that are stored in a vector container. This function returns the dislocation source in that vector that corresponds to the index provided as argument.

Parameters

<i>i</i>	Index of the required dislocation source in the vector. This value should be greater than or equal to 0 and less than the number of dislocation sources on the slip plane.
<i>dSource</i>	Pointer to the memory location where the required dislocation source is to be stored. Space in memory must be pre-allocated.

Returns

True if the provided index is greater than or equal to 0 and less than the number of dislocation sources on the slip plane (the memory location pointed to by d is populated with the [DislocationSource](#) data). Otherwise, the return value is false.

Definition at line 168 of file slipPlane.cpp.

```

169 {
170     if (i>=0 && i<this->dislocationSources.size ())
171     {
172         *dSource = this->dislocationSources[i];
173         return (true);
174     }
175     else
176     {
177         return (false);
178     }
179 }

```

5.6.3.6 std::vector< DislocationSource > SlipPlane::getDislocationSourceList ()

Get the entire vector container which holds the dislocation sources lying on this slip plane.

Returns

The vector of dislocation sources lying on this slip plane.

Definition at line 185 of file slipPlane.cpp.

```
186 {
187     return (this->dislocationSources);
188 }
```

5.6.3.7 Vector3d * SlipPlane::getExtremities ()

Get the position vectors of the extremities of the slip plane.

Returns

Pointer to an array containing the position vectors of the two extremities of the slip plane, variables of type [Vector3d](#).

Definition at line 108 of file slipPlane.cpp.

```
109 {
110     return (this->extremities);
111 }
```

5.6.3.8 Vector3d SlipPlane::getExtremity (int i)

Get the position vector of the extremity whose index is provided as argument.

Parameters

<i>i</i>	Index of the extremity. Possible values: 0, 1
----------	-----------------------------------------------

Returns

Position vector of the extremity indicated by the argument, returned as a variable of type [Vector3d](#).

Definition at line 92 of file slipPlane.cpp.

```
93 {
94     if (i==0 || i==1)
95     {
96         return (this->extremities[i]);
97     }
98     else
99     {
100         return (Vector3d());
101     }
102 }
```

5.6.3.9 Vector3d SlipPlane::getNormal ()

Get the normal vector of the slip plane.

Returns

The normal vector of the slip plane, in a variable of type [Vector3d](#).

Definition at line 117 of file slipPlane.cpp.

```

118 {
119     return (this->normalVector);
120 }

```

5.6.3.10 Vector3d SlipPlane::getPosition ()

Get the position vector of the slip plane.

This function returns the position vector of the slip plane. The position vector is redundant because the slip plane is completely defined by its extremities and the normal vector. Nevertheless, this value can be useful to locate the slip plane within a slip system.

Returns

Position vector of the slip plane, in a variable of type [Vector3d](#).

Definition at line 127 of file slipPlane.cpp.

```

128 {
129     return (this->position);
130 }

```

5.6.3.11 RotationMatrix SlipPlane::getRotationMatrix ()

Get the rotation matrix for this slip plane.

Returns

The rotation matrix of this slip plane, in a variable of type [RotationMatrix](#).

Definition at line 194 of file slipPlane.cpp.

```

195 {
196     return (this->rotationMatrix);
197 }

```

5.6.3.12 void SlipPlane::setDislocationList (std::vector< Dislocation > dislocationList)

Set the list of dislocations of the slip plane.

Parameters

<i>dislocationList</i>	A vector container of type Dislocation containing the dislocations lying on this slip plane.
------------------------	--------------------------------------------------------------------------------------------------------------

Definition at line 72 of file slipPlane.cpp.

```

73 {
74     this->dislocations = dislocationList;
75 }

```

5.6.3.13 void SlipPlane::setDislocationSourceList (std::vector< DislocationSource > dislocationSourceList)

Set the list of dislocation sources on the slip plane.

Parameters

<i>dislocation-SourceList</i>	A vector container of type DislocationSource containing the dislocation sources lying on this slip plane.
-------------------------------	---------------------------------------------------------------------------------------------------------------------------

Definition at line 81 of file slipPlane.cpp.

```
82 {
83     this->dislocationSources = dislocationSourceList;
84 }
```

5.6.3.14 void SlipPlane::setExtremities (Vector3d * ends)

Set the extremities of the slip plane.

Parameters

<i>ends</i>	Pointer to an array of type Vector3d , containing the position vectors of the extremities of the slip plane in consecutive locations.
-------------	-------------------------------------------------------------------------------------------------------------------------------------------------------

Definition at line 44 of file slipPlane.cpp.

```
45 {
46     this->extremities[0] = *(ends);
47     this->extremities[1] = *(ends+1);
48 }
```

5.6.3.15 void SlipPlane::setNormal (Vector3d normal)

Set the normal vector of the slip plane.

Parameters

<i>normal</i>	The normal vector of the slip plane.
---------------	--------------------------------------

Definition at line 54 of file slipPlane.cpp.

```
55 {
56     this->normalVector = normal;
57 }
```

5.6.3.16 void SlipPlane::setPosition (Vector3d pos)

Set the position of the slip plane.

Parameters

<i>pos</i>	The position vector of the slip plane. (This parameter is useful for locating the slip plane within a slip system)
------------	--------------------------------------------------------------------------------------------------------------------

Definition at line 63 of file slipPlane.cpp.

```
64 {
65     this->position = pos;
66 }
```

5.6.4 Field Documentation

5.6.4.1 `std::vector<Dislocation> SlipPlane::dislocations` [protected]

STL vector container with dislocations.

A slip plane may contain several dislocations. These are stored in this vector container dislocations.

Definition at line 51 of file slipPlane.h.

5.6.4.2 `std::vector<DislocationSource> SlipPlane::dislocationSources` [protected]

STL vector container with dislocation sources.

A slip plane may contain several dislocation sources. These are stored in this vector container dislocationSources.

Definition at line 57 of file slipPlane.h.

5.6.4.3 `Vector3d SlipPlane::extremities[2]` [protected]

The extremities of the slip plane.

The slip plane is represented as a straight line in these two dimensional simulations. The position vectors of the two ends are given here.

Definition at line 33 of file slipPlane.h.

5.6.4.4 `Vector3d SlipPlane::normalVector` [protected]

The normal vector to the slip plane.

This is the vector normal to the slip plane. Since we are concerned with the cubic system here, the indices of the normal vector are the same as those of the slip plane.

Definition at line 39 of file slipPlane.h.

5.6.4.5 `Vector3d SlipPlane::position` [protected]

The position vector of the slip plane.

This position vector is redundant because the combination of the position vectors of the extremities and the normal vector define the slip plane completely. However, this vector, position, is useful to locate the slip plane in a given slip system.

Definition at line 45 of file slipPlane.h.

5.6.4.6 `RotationMatrix SlipPlane::rotationMatrix` [protected]

Rotation matrix for co-ordinate system transformations.

The slip plane's local co-ordinate system is defined as follows: z-axis||NormalVector; x-axis||slipPlane line. The rotation matrix is created using this convention.

Definition at line 63 of file slipPlane.h.

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

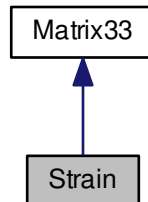
- [slipPlane.h](#)
- [slipPlane.cpp](#)

5.7 Strain Class Reference

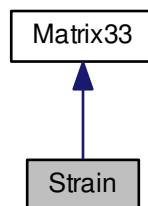
[Strain](#) class to represent the strain tensor.

```
#include <strain.h>
```

Inheritance diagram for Strain:



Collaboration diagram for Strain:



Public Member Functions

- [Strain](#) ()
Default constructor.
- [Strain](#) (double *principal, double *shear)
Constructor specifying the principal and shear strains.
- void [populateMatrix](#) ()
Construct the strain tensor from the principal and shear strains.
- double * [getPrincipalStrains](#) ()
Get the principal strains.
- double * [getShearStrains](#) ()
Get the shear strains.
- [Strain rotate](#) ([RotationMatrix](#) alpha)
Rotate the strain tensor from one coordinate system to another.
- void [setValue](#) (int row, int column, double value)
Function to set the value of an element indicated by its position.

- double `getValue` (int row, int column)
Returns the value of the element located by the row and column indices provided.
- `Matrix33 adjugate` ()
Returns the adjugate matrix of the present matrix.
- `Matrix33 operator+` (const `Matrix33` &) const
Operator for addition of two matrices.
- void `operator+=` (const `Matrix33` &)
Operator for reflexive addition of two matrices.
- `Matrix33 operator-` (const `Matrix33` &) const
Operator for the subtraction of two matrices.
- void `operator-=` (const `Matrix33` &)
Operator for reflexive subtraction of two matrices.
- `Matrix33 operator*` (const double &) const
Operator for scaling the matrix by a scalar.
- `Matrix33 operator*` (const `Matrix33` &) const
Operator for the multiplication of two matrices.
- `Vector3d operator*` (const `Vector3d` &) const
Operator for the multiplication of a matrix with a vector.
- void `operator*=` (const double &)
Operator for reflexive scaling of the matrix by a scalar.
- void `operator*=` (const `Matrix33` &)
Operator for reflexive multiplication of two matrices.
- `Matrix33 operator^` () const
Transpose.
- double `operator~` () const
Determinant.
- `Matrix33 operator!` () const
Inverse.

Protected Attributes

- double `principalStrains` [3]
- double `shearStrains` [3]
- double `x` [3][3]
Array containing the elements of the matrix.

5.7.1 Detailed Description

`Strain` class to represent the strain tensor.

The member functions of this class construct the symmetric strain tensor and operate on it.

Definition at line 21 of file `strain.h`.

5.7.2 Constructor & Destructor Documentation

5.7.2.1 `Strain::Strain` ()

Default constructor.

Initializes the strain tensor with zeros.

Definition at line 16 of file `strain.cpp`.

```

17 {
18     int i, j;
19
20     for (i=0; i<3; i++)
21     {
22         principalStrains [i] = 0.0;
23         shearStrains [i] = 0.0;
24     }
25
26     this->populateMatrix ();
27 }

```

5.7.2.2 Strain::Strain (double * *principal*, double * *shear*)

Constructor specifying the principal and shear strains.

The principal and shear strains are provided in the arguments and the symmetrical strain tensor is constructed using them.

Parameters

<i>principal</i>	Pointer to the array containing principal strains.
<i>shear</i>	Pointer to the array containing shear strains.

Definition at line 35 of file strain.cpp.

```

36 {
37     int i;
38
39     for (i=0; i<3; i++)
40     {
41         this->principalStrains [i] = principal [i];
42         this->shearStrains [i] = shear [i];
43     }
44
45     this->populateMatrix ();
46 }

```

5.7.3 Member Function Documentation

5.7.3.1 Matrix33 Matrix33::adjugate () [inherited]

Returns the adjugate matrix of the present matrix.

The adjugate matrix of the present matrix is returned. The adjugate matrix is calculated by evaluating the determinant of the cofactor matrix of each element, and then replacing the corresponding element position by the value of the determinant. This operation is useful in calculating the inverse of a matrix.

Returns

The adjugate matrix of the present matrix.

Definition at line 129 of file matrix33.cpp.

```

130 {
131     Matrix33 adj;
132
133     adj.x[0][0] = (this->x[1][1]*this->x[2][2]) - (this->x[1][2]*this->x[2][1]);
134     adj.x[0][1] = (this->x[1][2]*this->x[2][0]) - (this->x[1][0]*this->x[2][2]);
135     adj.x[0][2] = (this->x[1][0]*this->x[2][1]) - (this->x[1][1]*this->x[2][0]);
136
137     adj.x[1][0] = (this->x[2][1]*this->x[0][2]) - (this->x[0][1]*this->x[2][2]);
138     adj.x[1][1] = (this->x[2][2]*this->x[0][0]) - (this->x[2][0]*this->x[0][2]);
139     adj.x[1][2] = (this->x[2][0]*this->x[0][1]) - (this->x[2][1]*this->x[0][0]);
140
141     adj.x[2][0] = (this->x[0][1]*this->x[1][2]) - (this->x[0][2]*this->x[1][1]);
142     adj.x[2][1] = (this->x[0][2]*this->x[1][0]) - (this->x[0][0]*this->x[1][2]);
143     adj.x[2][2] = (this->x[0][0]*this->x[1][1]) - (this->x[1][0]*this->x[0][1]);
144
145     return (adj);
146 }

```

5.7.3.2 `double * Strain::getPrincipalStrains ()`

Get the principal strains.

Returns a 3-member array with the principal strains: s11 s22 s33.

Returns

3-member array with the principal strains.

Definition at line 68 of file strain.cpp.

```
69 {
70     double p[3];
71     int i;
72
73     for (i=0; i<3; i++)
74     {
75         p[i] = this->principalStrains[i];
76     }
77
78     return (p);
79 }
```

5.7.3.3 `double * Strain::getShearStrains ()`

Get the shear strains.

Returns a 3-member array with the shear strains: s12 s13 s23.

Returns

3-member array with the shear strains.

Definition at line 86 of file strain.cpp.

```
87 {
88     double s[3];
89     int i;
90
91     for (i=0; i<3; i++)
92     {
93         s[i] = this->shearStrains[i];
94     }
95
96     return (s);
97 }
```

5.7.3.4 `double Matrix33::getValue (int row, int column)` [inherited]

Returns the value of the element located by the row and column indices provided.

The value of the element indicated by the arguments row and column is returned. The values of row and column must correspond to array indices, and thus can be one of 0, 1 and 2. In any other case 0.0 is returned.

Parameters

<i>row</i>	Row index of the element.
<i>column</i>	Column index of the element.

Returns

Value of the element located at the given position.

Definition at line 111 of file matrix33.cpp.

```

112 {
113     if (row>=0 && row<3)
114     {
115         if (column>=0 && column<3)
116         {
117             return (this->x[row][column]);
118         }
119     }
120
121     return (0.0);
122 }
```

5.7.3.5 Matrix33 Matrix33::operator! () const [inherited]

Inverse.

Returns in a new matrix the inverse of the current matrix. If the current matrix is non-invertible, a zero matrix is returned.

Returns

Matrix with the inverse of the current matrix. If the current matrix is non-invertible, a zero matrix is returned.

Definition at line 370 of file matrix33.cpp.

```

371 {
372     Matrix33 r;    // Result matrix
373
374     double determinant = ~(*this);
375
376     if (determinant == 0.0)
377     {
378         // The matrix is non-invertible
379         return (r);    // Zero matrix
380     }
381
382     // If we are still here, the matrix is invertible
383
384     // Transpose
385     Matrix33 tr = ^(*this);
386
387     // Find Adjugate matrix
388     Matrix33 adj = tr.adjugate();
389
390     // Calculate the inverse by dividing the adjugate matrix by the determinant
391     r = adj * (1.0/determinant);
392
393     return (r);
394 }
```

5.7.3.6 Matrix33 Matrix33::operator*(const double & p) const [inherited]

Operator for scaling the matrix by a scalar.

Scales the current matrix by the scalar provided and returns the result in a third matrix.

Returns

Matrix containing the result of scaling the current matrix by the scalar provided as argument.

Definition at line 233 of file matrix33.cpp.

```

234 {
235     int i, j;
236     Matrix33 r;
237
238     for (i=0; i<3; i++)
239     {
240         for (j=0; j<3; j++)
241         {
242             r.x[i][j] = this->x[i][j] * p;
243         }
244     }
245
246     return (r);
247 }

```

5.7.3.7 Matrix33 Matrix33::operator* (const Matrix33 & p) const [inherited]

Operator for the multiplication of two matrices.

Multiplies the current matrix with another 3x3 matrix and returns the result in a new matrix.

Returns

The result of the multiplication of the current matrix with the one provided as argument.

Definition at line 271 of file matrix33.cpp.

```

272 {
273     int i, j, k;
274     Matrix33 r;
275
276     for (i=0; i<3; i++)
277     {
278         for (j=0; j<3; j++)
279         {
280             r.x[i][j] = 0.0;
281             for (k=0; k<3; k++)
282             {
283                 r.x[i][j] += this->x[i][k] * p.x[k][j];
284             }
285         }
286     }
287
288     return (r);
289 }

```

5.7.3.8 Vector3d Matrix33::operator* (const Vector3d & v) const [inherited]

Operator for the multiplication of a matrix with a vector.

Returns in a vector the result of the multiplication of the current matrix with the provided vector.

Returns

The vector resulting from the multiplication of the current matrix with a vector.

Definition at line 311 of file matrix33.cpp.

```

312 {
313     Vector3d r(0.0, 0.0, 0.0);
314     int i, j;
315
316     for (i=0; i<3; i++)
317     {
318         for (j=0; j<3; j++)
319         {
320             r[i] += this->x[i][j] * v.x[j];
321         }
322     }
323
324     return (r);
325 }

```

5.7.3.9 void Matrix33::operator*=(const double & p) [inherited]

Operator for reflexive scaling of the matrix by a scalar.

Scales the current matrix by the scalar provided and populates the current matrix elements with the result.

Definition at line 253 of file matrix33.cpp.

```

254 {
255     int i, j;
256
257     for (i=0; i<3; i++)
258     {
259         for (j=0; j<3; j++)
260         {
261             this->x[i][j] *= p;
262         }
263     }
264 }
```

5.7.3.10 void Matrix33::operator*=(const Matrix33 & p) [inherited]

Operator for reflexive multiplication of two matrices.

Multiplies the current matrix with another 3x3 matrix and populates the elements of the current matrix with the result.

Definition at line 295 of file matrix33.cpp.

```

296 {
297     Matrix33* r = new Matrix33;
298
299     *r = (*this) * p;
300     *this = *r;
301
302     delete(r);
303     r = NULL;
304 }
```

5.7.3.11 Matrix33 Matrix33::operator+(const Matrix33 & p) const [inherited]

Operator for addition of two matrices.

Adds the current matrix to the provided matrix and returns a third matrix with the result.

Returns

Matrix containing the sum of the present matrix and the one provided.

Definition at line 155 of file matrix33.cpp.

```

156 {
157     int i, j;
158     Matrix33 r;
159
160     for (i=0; i<3; i++)
161     {
162         for (j=0; j<3; j++)
163         {
164             r.x[i][j] = this->x[i][j] + p.x[i][j];
165         }
166     }
167
168     return (r);
169 }
```

5.7.3.12 void Matrix33::operator+=(const Matrix33 & p) [inherited]

Operator for reflexive addition of two matrices.

Adds the current matrix to the provided matrix and populates the current matrix elements with the result.

Definition at line 175 of file matrix33.cpp.

```

176 {
177     int i, j;
178
179     for (i=0; i<3; i++)
180     {
181         for (j=0; j<3; j++)
182         {
183             this->x[i][j] += p.x[i][j];
184         }
185     }
186 }
```

5.7.3.13 Matrix33 Matrix33::operator- (const Matrix33 & p) const [inherited]

Operator for the subtraction of two matrices.

Subtracts the given matrix from the current matrix and returns the result in a new matrix.

Returns

Matrix containing the result of subtracting the provided matrix from the current matrix.

Definition at line 194 of file matrix33.cpp.

```

195 {
196     int i, j;
197     Matrix33 r;
198
199     for (i=0; i<3; i++)
200     {
201         for (j=0; j<3; j++)
202         {
203             r.x[i][j] = this->x[i][j] - p.x[i][j];
204         }
205     }
206
207     return (r);
208 }
```

5.7.3.14 void Matrix33::operator-= (const Matrix33 & p) [inherited]

Operator for reflexive subtraction of two matrices.

Subtracts the given matrix from the current matrix and populates the current matrix with the result.

Definition at line 214 of file matrix33.cpp.

```

215 {
216     int i, j;
217
218     for (i=0; i<3; i++)
219     {
220         for (j=0; j<3; j++)
221         {
222             this->x[i][j] -= p.x[i][j];
223         }
224     }
225 }
```

5.7.3.15 Matrix33::operator[^]() const [inherited]

Transpose.

Performs the transpose of the current matrix.

Returns

A new matrix with the transpose of the current matrix.

Definition at line 333 of file matrix33.cpp.

```

334 {
335     Matrix33 r;
336     int i, j;
337
338     for (i=0; i<3; i++)
339     {
340         for (j=0; j<3; j++)
341         {
342             r.x[i][j] = this->x[j][i];
343         }
344     }
345
346     return (r);
347 }
```

5.7.3.16 double Matrix33::operator~() const [inherited]

Determinant.

Calculates the determinant of the current matrix.

Returns

Returns the determinant of the current matrix.

Definition at line 354 of file matrix33.cpp.

```

355 {
356     double d = 0.0;
357
358     d += this->x[0][0] * ( (this->x[1][1]*this->x[2][2]) - (this->x[2][1]*this->
359     x[1][2]) );
360     d += this->x[0][1] * ( (this->x[1][2]*this->x[2][0]) - (this->x[1][0]*this->
361     x[2][2]) );
362     d += this->x[0][2] * ( (this->x[1][0]*this->x[2][1]) - (this->x[2][0]*this->
363     x[1][1]) );
364
365     return (d);
366 }
```

5.7.3.17 void Strain::populateMatrix ()

Construct the strain tensor from the principal and shear strains.

Takes the values in principalStrains and shearStrains and constructs the symmetrical strain matrix.

Definition at line 52 of file strain.cpp.

```

53 {
54     this->x[0][0] = this->principalStrains [0];
55     this->x[1][1] = this->principalStrains [1];
56     this->x[2][2] = this->principalStrains [2];
57
58     this->x[0][1] = this->x[1][0] = this->shearStrains [0];
59     this->x[0][2] = this->x[2][0] = this->shearStrains [1];
60     this->x[1][2] = this->x[2][1] = this->shearStrains [2];
61 }
```

5.7.3.18 Strain Strain::rotate (RotationMatrix *alpha*)

Rotate the strain tensor from one coordinate system to another.

Rotates the present strain matrix from one coordinate system to another using the rotation matrix supplied. The result is returned in a new [Strain](#) matrix.

Parameters

<i>alpha</i>	Rotation matrix.
--------------	------------------

Returns

Rotated strain tensor.

Definition at line 105 of file strain.cpp.

```

106 {
107     Matrix33 alphaT = ^alpha; // Transpose
108     Strain sNew;
109
110     sNew = alpha * (*this) * alphaT; // Rotate the strain matrix
111
112     return (sNew);
113 }
```

5.7.3.19 void Matrix33::setValue (int *row*, int *column*, double *value*) [inherited]

Function to set the value of an element indicated by its position.

The element indicated by the arguments row and column is set to the value provided. The values of row and column must correspond to array indices, and thus can be one of 0, 1 and 2. In any other case 0.0 is returned.

Parameters

<i>row</i>	Row index of the element.
<i>column</i>	Column index of the element.
<i>value</i>	Value that the element is to be set to.

Definition at line 93 of file matrix33.cpp.

```

94 {
95     if (row>=0 && row<3)
96     {
97         if (column>=0 && column<3)
98         {
99             this->x[row][column] = value;
100         }
101     }
102 }
```

5.7.4 Field Documentation

5.7.4.1 double Strain::principalStrains[3] [protected]

The three principal strains: s11, s22, s33.

Definition at line 27 of file strain.h.

5.7.4.2 double Strain::shearStrains[3] [protected]

The three shear strains: s12, s13, s23,

Definition at line 31 of file strain.h.

5.7.4.3 `double Matrix33::x[3][3]` `[protected], [inherited]`

Array containing the elements of the matrix.

Definition at line 26 of file matrix33.h.

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

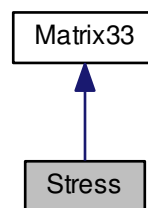
- [strain.h](#)
- [strain.cpp](#)

5.8 Stress Class Reference

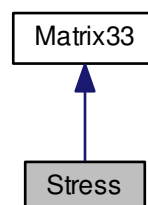
[Stress](#) class to represent the stress tensor.

```
#include <stress.h>
```

Inheritance diagram for Stress:



Collaboration diagram for Stress:



Public Member Functions

- [Stress](#) ()

Default constructor.

- **Stress** (double *principal, double *shear)

Constructor specifying the principal and shear stresses.

- void **populateMatrix** ()

Construct the stress tensor from the principal and shear stresses.

- double * **getPrincipalStresses** ()

Get the principal stresses.

- double * **getShearStresses** ()

Get the shear stresses.

- **Stress rotate** (**RotationMatrix** alpha)

Rotate the stress tensor from one coordinate system to another.

- void **setValue** (int row, int column, double value)

Function to set the value of an element indicated by its position.

- double **getValue** (int row, int column)

Returns the value of the element located by the row and column indices provided.

- **Matrix33 adjugate** ()

Returns the adjugate matrix of the present matrix.

- **Matrix33 operator+** (const **Matrix33** &) const

Operator for addition of two matrices.

- void **operator+=** (const **Matrix33** &)

Operator for reflexive addition of two matrices.

- **Matrix33 operator-** (const **Matrix33** &) const

Operator for the subtraction of two matrices.

- void **operator-=** (const **Matrix33** &)

Operator for reflexive subtraction of two matrices.

- **Matrix33 operator*** (const double &) const

Operator for scaling the matrix by a scalar.

- **Matrix33 operator*** (const **Matrix33** &) const

Operator for the multiplication of two matrices.

- **Vector3d operator*** (const **Vector3d** &) const

Operator for the multiplication of a matrix with a vector.

- void **operator*=** (const double &)

Operator for reflexive scaling of the matrix by a scalar.

- void **operator*=** (const **Matrix33** &)

Operator for reflexive multiplication of two matrices.

- **Matrix33 operator[^]** () const

Transpose.

- double **operator~** () const

Determinant.

- **Matrix33 operator!** () const

Inverse.

Protected Attributes

- double **principalStresses** [3]
- double **shearStresses** [3]
- double **x** [3][3]

Array containing the elements of the matrix.

5.8.1 Detailed Description

[Stress](#) class to represent the stress tensor.

The member functions of this class construct the symmetric stress tensor and operate on it.

Definition at line 21 of file stress.h.

5.8.2 Constructor & Destructor Documentation

5.8.2.1 Stress::Stress ()

Default constructor.

Initializes the stress tensor with zeros.

Definition at line 16 of file stress.cpp.

```

17 {
18     int i, j;
19
20     for (i=0; i<3; i++)
21     {
22         principalStresses [i] = 0.0;
23         shearStresses [i] = 0.0;
24     }
25
26     this->populateMatrix ();
27 }
```

5.8.2.2 Stress::Stress (double * *principal*, double * *shear*)

Constructor specifying the principal and shear stresses.

The principal and shear stresses are provided in the arguments and the symmetrical stress tensor is constructed using them.

Parameters

<i>principal</i>	Pointer to the array containing principal stresses.
<i>shear</i>	Pointer to the array containing shear stresses.

Definition at line 35 of file stress.cpp.

```

36 {
37     int i;
38
39     for (i=0; i<3; i++)
40     {
41         this->principalStresses [i] = principal [i];
42         this->shearStresses [i] = shear [i];
43     }
44
45     this->populateMatrix ();
46 }
```

5.8.3 Member Function Documentation

5.8.3.1 Matrix33 Matrix33::adjugate () [inherited]

Returns the adjugate matrix of the present matrix.

The adjugate matrix of the present matrix is returned. The adjugate matrix is calculated by evaluating the determinant of the cofactor matrix of each element, and then replacing the corresponding element position by the value of the determinant. This operation is useful in calculating the inverse of a matrix.

Returns

The adjugate matrix of the present matrix.

Definition at line 129 of file matrix33.cpp.

```

130 {
131     Matrix33 adj;
132
133     adj.x[0][0] = (this->x[1][1]*this->x[2][2]) - (this->x[1][2]*this->x[2][1]);
134     adj.x[0][1] = (this->x[1][2]*this->x[2][0]) - (this->x[1][0]*this->x[2][2]);
135     adj.x[0][2] = (this->x[1][0]*this->x[2][1]) - (this->x[1][1]*this->x[2][0]);
136
137     adj.x[1][0] = (this->x[2][1]*this->x[0][2]) - (this->x[0][1]*this->x[2][2]);
138     adj.x[1][1] = (this->x[2][2]*this->x[0][0]) - (this->x[2][0]*this->x[0][2]);
139     adj.x[1][2] = (this->x[2][0]*this->x[0][1]) - (this->x[2][1]*this->x[0][0]);
140
141     adj.x[2][0] = (this->x[0][1]*this->x[1][2]) - (this->x[0][2]*this->x[1][1]);
142     adj.x[2][1] = (this->x[0][2]*this->x[1][0]) - (this->x[0][0]*this->x[1][2]);
143     adj.x[2][2] = (this->x[0][0]*this->x[1][1]) - (this->x[1][0]*this->x[0][1]);
144
145     return (adj);
146 }
```

5.8.3.2 double * Stress::getPrincipalStresses ()

Get the principal stresses.

Returns a 3-member array with the principal stresses: s11 s22 s33.

Returns

3-member array with the principal stresses.

Definition at line 68 of file stress.cpp.

```

69 {
70     double p[3];
71     int i;
72
73     for (i=0; i<3; i++)
74     {
75         p[i] = this->principalStresses[i];
76     }
77
78     return (p);
79 }
```

5.8.3.3 double * Stress::getShearStresses ()

Get the shear stresses.

Returns a 3-member array with the shear stresses: s12 s13 s23.

Returns

3-member array with the shear stresses.

Definition at line 86 of file stress.cpp.

```

87 {
88     double s[3];
89     int i;
90
91     for (i=0; i<3; i++)
92     {
93         s[i] = this->shearStresses[i];
94     }
95
96     return (s);
97 }
```

5.8.3.4 `double Matrix33::getValue (int row, int column)` [inherited]

Returns the value of the element located by the row and column indices provided.

The value of the element indicated by the arguments row and column is returned. The values of row and column must correspond to array indices, and thus can be one of 0, 1 and 2. In any other case 0.0 is returned.

Parameters

<i>row</i>	Row index of the element.
<i>column</i>	Column index of the element.

Returns

Value of the element located at the given position.

Definition at line 111 of file matrix33.cpp.

```

112 {
113     if (row>=0 && row<3)
114     {
115         if (column>=0 && column<3)
116         {
117             return (this->x[row][column]);
118         }
119     }
120
121     return (0.0);
122 }
```

5.8.3.5 `Matrix33 Matrix33::operator! () const` [inherited]

Inverse.

Returns in a new matrix the inverse of the current matrix. If the current matrix is non-invertible, a zero matrix is returned.

Returns

Matrix with the inverse of the current matrix. If the current matrix is non-invertible, a zero matrix is returned.

Definition at line 370 of file matrix33.cpp.

```

371 {
372     Matrix33 r;    // Result matrix
373
374     double determinant = ~(*this);
375
376     if (determinant == 0.0)
377     {
378         // The matrix is non-invertible
379         return (r);    // Zero matrix
380     }
381
382     // If we are still here, the matrix is invertible
383
384     // Transpose
385     Matrix33 tr = ^(*this);
386
387     // Find Adjugate matrix
388     Matrix33 adj = tr.adjugate();
389
390     // Calculate the inverse by dividing the adjugate matrix by the determinant
391     r = adj * (1.0/determinant);
392
393     return (r);
394 }
```

5.8.3.6 Matrix33 Matrix33::operator* (const double & p) const [inherited]

Operator for scaling the matrix by a scalar.

Scales the current matrix by the scalar provided and returns the result in a third matrix.

Returns

Matrix containing the result of scaling the current matrix by the scalar provided as argument.

Definition at line 233 of file matrix33.cpp.

```

234 {
235     int i, j;
236     Matrix33 r;
237
238     for (i=0; i<3; i++)
239     {
240         for (j=0; j<3; j++)
241         {
242             r.x[i][j] = this->x[i][j] * p;
243         }
244     }
245
246     return (r);
247 }
```

5.8.3.7 Matrix33 Matrix33::operator* (const Matrix33 & p) const [inherited]

Operator for the multiplication of two matrices.

Multiplies the current matrix with another 3x3 matrix and returns the result in a new matrix.

Returns

The result of the multiplication of the current matrix with the one provided as argument.

Definition at line 271 of file matrix33.cpp.

```

272 {
273     int i, j, k;
274     Matrix33 r;
275
276     for (i=0; i<3; i++)
277     {
278         for (j=0; j<3; j++)
279         {
280             r.x[i][j] = 0.0;
281             for (k=0; k<3; k++)
282             {
283                 r.x[i][j] += this->x[i][k] * p.x[k][j];
284             }
285         }
286     }
287
288     return (r);
289 }
```

5.8.3.8 Vector3d Matrix33::operator* (const Vector3d & v) const [inherited]

Operator for the multiplication of a matrix with a vector.

Returns in a vector the result of the multiplication of the current matrix with the provided vector.

Returns

The vector resulting from the multiplication of the current matrix with a vector.

Definition at line 311 of file matrix33.cpp.

```

312 {
313     Vector3d r(0.0, 0.0, 0.0);
314     int i, j;
315
316     for (i=0; i<3; i++)
317     {
318         for (j=0; j<3; j++)
319         {
320             r[i] += this->x[i][j] * v.x[j];
321         }
322     }
323
324     return (r);
325 }
```

5.8.3.9 void Matrix33::operator*=(const double & p) [inherited]

Operator for reflexive scaling of the matrix by a scalar.

Scales the current matrix by the scalar provided and populates the current matrix elements with the result.

Definition at line 253 of file matrix33.cpp.

```

254 {
255     int i, j;
256
257     for (i=0; i<3; i++)
258     {
259         for (j=0; j<3; j++)
260         {
261             this->x[i][j] *= p;
262         }
263     }
264 }
```

5.8.3.10 void Matrix33::operator*=(const Matrix33 & p) [inherited]

Operator for reflexive multiplication of two matrices.

Multiplies the current matrix with another 3x3 matrix and populates the elements of the current matrix with the result.

Definition at line 295 of file matrix33.cpp.

```

296 {
297     Matrix33* r = new Matrix33;
298
299     *r = (*this) * p;
300     *this = *r;
301
302     delete(r);
303     r = NULL;
304 }
```

5.8.3.11 Matrix33 Matrix33::operator+ (const Matrix33 & p) const [inherited]

Operator for addition of two matrices.

Adds the current matrix to the provided matrix and returns a third matrix with the result.

Returns

Matrix containing the sum of the present matrix and the one provided.

Definition at line 155 of file matrix33.cpp.

```

156 {
157     int i, j;
158     Matrix33 r;
159
160     for (i=0; i<3; i++)
161     {
162         for (j=0; j<3; j++)
163         {
164             r.x[i][j] = this->x[i][j] + p.x[i][j];
165         }
166     }
167
168     return (r);
169 }
```

5.8.3.12 void Matrix33::operator+=(const Matrix33 & p) [inherited]

Operator for reflexive addition of two matrices.

Adds the current matrix to the provided matrix and populates the current matrix elements with the result.

Definition at line 175 of file matrix33.cpp.

```

176 {
177     int i, j;
178
179     for (i=0; i<3; i++)
180     {
181         for (j=0; j<3; j++)
182         {
183             this->x[i][j] += p.x[i][j];
184         }
185     }
186 }
```

5.8.3.13 Matrix33 Matrix33::operator- (const Matrix33 & p) const [inherited]

Operator for the subtraction of two matrices.

Subtracts the given matrix from the current matrix and returns the result in a new matrix.

Returns

Matrix containing the result of subtracting the provided matrix from the current matrix.

Definition at line 194 of file matrix33.cpp.

```

195 {
196     int i, j;
197     Matrix33 r;
198
199     for (i=0; i<3; i++)
200     {
201         for (j=0; j<3; j++)
202         {
203             r.x[i][j] = this->x[i][j] - p.x[i][j];
204         }
205     }
206
207     return (r);
208 }
```

5.8.3.14 void Matrix33::operator-= (const Matrix33 & p) [inherited]

Operator for reflexive subtraction of two matrices.

Subtracts the given matrix from the current matrix and populates the current matrix with the result.

Definition at line 214 of file matrix33.cpp.

```

215 {
216     int i, j;
217
218     for (i=0; i<3; i++)
219     {
220         for (j=0; j<3; j++)
221         {
222             this->x[i][j] -= p.x[i][j];
223         }
224     }
225 }
```

5.8.3.15 Matrix33 Matrix33::operator^() const [inherited]

Transpose.

Performs the transpose of the current matrix.

Returns

A new matrix with the transpose of the current matrix.

Definition at line 333 of file matrix33.cpp.

```

334 {
335     Matrix33 r;
336     int i, j;
337
338     for (i=0; i<3; i++)
339     {
340         for (j=0; j<3; j++)
341         {
342             r.x[i][j] = this->x[j][i];
343         }
344     }
345
346     return (r);
347 }
```

5.8.3.16 double Matrix33::operator~() const [inherited]

Determinant.

Calculates the determinant of the current matrix.

Returns

Returns the determinant of the current matrix.

Definition at line 354 of file matrix33.cpp.

```

355 {
356     double d = 0.0;
357
358     d += this->x[0][0] * ( (this->x[1][1]*this->x[2][2]) - (this->x[2][1]*this->
x[1][2]) );
359     d += this->x[0][1] * ( (this->x[1][2]*this->x[2][0]) - (this->x[1][0]*this->
x[2][2]) );
360     d += this->x[0][2] * ( (this->x[1][0]*this->x[2][1]) - (this->x[2][0]*this->
x[1][1]) );
361
362     return (d);
363 }
```

5.8.3.17 void Stress::populateMatrix ()

Construct the stress tensor from the principal and shear stresses.

Takes the values in principalStresses and shearStresses and constructs the symmetrical stress matrix.

Definition at line 52 of file stress.cpp.

```

53 {
54     this->x[0][0] = this->principalStresses [0];
55     this->x[1][1] = this->principalStresses [1];
56     this->x[2][2] = this->principalStresses [2];
57
58     this->x[0][1] = this->x[1][0] = this->shearStresses [0];
59     this->x[0][2] = this->x[2][0] = this->shearStresses [1];
60     this->x[1][2] = this->x[2][1] = this->shearStresses [2];
61 }
```

5.8.3.18 Stress Stress::rotate (RotationMatrix alpha)

Rotate the stress tensor from one coordinate system to another.

Rotates the present stress matrix from one coordinate system to another using the rotation matrix supplied. The result is returned in a new [Stress](#) matrix.

Parameters

<i>alpha</i>	Rotation matrix.
--------------	------------------

Returns

Rotated stress tensor.

Definition at line 105 of file stress.cpp.

```

106 {
107     Matrix33 alphaT = ^alpha; // Transpose
108     Stress sNew;
109
110     sNew = alpha * (*this) * alphaT; // Rotate the stress matrix
111
112     return (sNew);
113 }
```

5.8.3.19 void Matrix33::setValue (int row, int column, double value) [inherited]

Function to set the value of an element indicated by its position.

The element indicated by the arguments row and column is set to the value provided. The values of row and column must correspond to array indices, and thus can be one of 0, 1 and 2. In any other case 0.0 is returned.

Parameters

<i>row</i>	Row index of the element.
<i>column</i>	Column index of the element.
<i>value</i>	Value that the element is to be set to.

Definition at line 93 of file matrix33.cpp.

```

94 {
95     if (row>=0 && row<3)
96     {
97         if (column>=0 && column<3)
98         {
```



```

99         this->x[row][column] = value;
100     }
101 }
102 }
```

5.8.4 Field Documentation

5.8.4.1 `double Stress::principalStresses[3]` `[protected]`

The three principal stresses: s11, s22, s33.

Definition at line 27 of file stress.h.

5.8.4.2 `double Stress::shearStresses[3]` `[protected]`

The three shear stresses: s12, s13, s23,

Definition at line 31 of file stress.h.

5.8.4.3 `double Matrix33::x[3][3]` `[protected]`, `[inherited]`

Array containing the elements of the matrix.

Definition at line 26 of file matrix33.h.

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

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

5.9 Vector3d Class Reference

[Vector3d](#) class representing a single 3-dimensional vector in the simulation.

```
#include <vector3d.h>
```

Public Member Functions

- [Vector3d](#) ()
Default constructor.
- [Vector3d](#) (double *a)
Constructor with values provided in an array.
- [Vector3d](#) (double a1, double a2, double a3)
Constructor with values provided explicitly.
- void [setValue](#) (int index, double value)
Function to set the value of an element of the vector.
- void [setVector](#) (double *a)
Function to set the value of the entire vector using an array.
- double [getValue](#) (int index)
Function to get the value of an element of the vector.
- double * [getVector](#) ()
Function to get the values of the elements of the vector in an array.
- double [sum](#) ()
Computes the sum of the elements of the vector.

- double `magnitude` ()
Computes the magnitude of the vector.
- `Vector3d normalize` ()
Returns the vector normalized to be a unit vector.
- `Vector3d operator+` (const `Vector3d` &) const
Operator for addition of two vectors.
- void `operator+=` (const `Vector3d` &)
Operator for reflexive addition of two vectors.
- `Vector3d operator-` (const `Vector3d` &) const
Operator for the subtraction of two vectors.
- void `operator-=` (const `Vector3d` &)
Operator for reflexive subtraction of two vectors.
- `Vector3d operator*` (const double &) const
Operator for scaling the vector by a scalar.
- void `operator*=` (const double &)
Operator for reflexive scaling of the vector by a scalar.
- double `operator*` (const `Vector3d` &) const
Operator for the scalar product of two vectors.
- `Vector3d operator^` (const `Vector3d` &) const
Operator for the vector product of two vectors.
- void `operator^=` (const `Vector3d` &)
Operator for reflexive vector product of two vectors.

Protected Attributes

- double `x` [3]
The elements of the vector.

5.9.1 Detailed Description

`Vector3d` class representing a single 3-dimensional vector in the simulation.

This class represents a vector in 3D space. The member functions and operators define various operations on the vector and its interactions with other data types.

Definition at line 20 of file `vector3d.h`.

5.9.2 Constructor & Destructor Documentation

5.9.2.1 `Vector3d::Vector3d` ()

Default constructor.

Initializes the vector with all elements equal to 0.0.

Definition at line 16 of file `vector3d.cpp`.

```

17 {
18     this->x[0] = 0.0;
19     this->x[1] = 0.0;
20     this->x[2] = 0.0;
21 }
```

5.9.2.2 Vector3d::Vector3d (double * a)

Constructor with values provided in an array.

Initializes the vector with the values provided in the array.

Parameters

<i>a</i>	Pointer to the array containing the elements of the vector
----------	------------------------------------------------------------

Definition at line 28 of file vector3d.cpp.

```

29 {
30     this->x[0] = a[0];
31     this->x[1] = a[1];
32     this->x[2] = a[2];
33 }
```

5.9.2.3 Vector3d::Vector3d (double a1, double a2, double a3)

Constructor with values provided explicitly.

Initializes the vector with the three values provided as arguments.

Parameters

<i>a1</i>	Value of the first element of the vector.
<i>a2</i>	Value of the second element of the vector.
<i>a3</i>	Value of the third element of the vector.

Definition at line 42 of file vector3d.cpp.

```

43 {
44     this->x[0] = a1;
45     this->x[1] = a2;
46     this->x[2] = a3;
47 }
```

5.9.3 Member Function Documentation**5.9.3.1 double Vector3d::getValue (int index)**

Function to get the value of an element of the vector.

Returns the value of the element at the position indicated by the argument index.

Parameters

<i>index</i>	Index of the element whose value is to be got.
--------------	------------------------------------------------

Returns

The value of the element of the vector at the position

Definition at line 83 of file vector3d.cpp.

```

84 {
85     if (index >= 0 && index < 3)
86     {
87         return (this->x[index]);
88     }
89     else
```

```
90     {  
91         return (0);  
92     }  
93 }
```

5.9.3.2 double * Vector3d::getVector ()

Function to get the values of the elements of the vector in an array.

The vector is returned in an array.

Returns

Pointer to the first term of an array containing the elements of the vector.

Definition at line 100 of file vector3d.cpp.

```
101 {  
102     double* a = new double[3];  
103  
104     a[0] = this->x[0];  
105     a[1] = this->x[1];  
106     a[2] = this->x[2];  
107  
108     return (a);  
109 }
```

5.9.3.3 double Vector3d::magnitude ()

Computes the magnitude of the vector.

Computes the magnitude of the vector. Basically the square root of the sum of the squares of the vector elements.

Returns

The magnitude of the vector.

Definition at line 134 of file vector3d.cpp.

```
135 {  
136     double s = 0.0;  
137     int i;  
138  
139     for (i=0; i<3; i++)  
140     {  
141         s += this->x[i] * this->x[i];  
142     }  
143  
144     return ( sqrt (s) );  
145 }
```

5.9.3.4 Vector3d Vector3d::normalize ()

Returns the vector normalized to be a unit vector.

This function normalizes a vector by dividing its elements by the magnitude. In case the magnitude is zero, a zero vector is returned.

Returns

Normalized vector.

Definition at line 152 of file vector3d.cpp.

```

153 {
154     double m = this->magnitude ();
155
156     if (m==0.0)
157     {
158         return (Vector3d ());
159     }
160     else
161     {
162         return ((*this) * (1.0/m));
163     }
164 }
```

5.9.3.5 Vector3d Vector3d::operator* (const double & p) const

Operator for scaling the vector by a scalar.

Scales the current vector by the scalar provided and returns the result in a third vector.

Returns

Vector containing the result of scaling the current vector by the scala provided as argument.

Definition at line 239 of file vector3d.cpp.

```

240 {
241     Vector3d r(0.0, 0.0, 0.0);
242     int i;
243
244     for (i=0; i<3; i++)
245     {
246         r.x[i] = this->x[i] * p;
247     }
248
249     return (r);
250 }
```

5.9.3.6 double Vector3d::operator* (const Vector3d & p) const

Operator for the scalar product of two vectors.

Performs the scalar product or dot product of the current vector with the one provided as argument and returns the result.

Returns

Scalar value of the scalar product of dot product of the current vector with the one provided as argument.

Definition at line 271 of file vector3d.cpp.

```

272 {
273     double s = 0.0;
274     int i;
275
276     for (i=0; i<3; i++)
277     {
278         s += this->x[i] * p.x[i];
279     }
280
281     return (s);
282 }
```

5.9.3.7 void Vector3d::operator*= (const double & p)

Operator for reflexive scaling of the vector by a scalar.

Scales the current vector by the scalar provided and populates the current vector elements with the result.

Definition at line 256 of file vector3d.cpp.

```

257 {
258     int i;
259
260     for (i=0; i<3; i++)
261     {
262         this->x[i] *= p;
263     }
264 }
```

5.9.3.8 Vector3d Vector3d::operator+ (const Vector3d & p) const

Operator for addition of two vectors.

Adds the current vector to the provided vector and returns a third vector with the result.

Returns

Vector containing the sum of the current vector with the one provided as argument.

Definition at line 173 of file vector3d.cpp.

```

174 {
175     Vector3d r (0.0, 0.0, 0.0);
176     int i;
177
178     for (i=0; i<3; i++)
179     {
180         r.x[i] = this->x[i] + p.x[i];
181     }
182
183     return (r);
184 }
```

5.9.3.9 void Vector3d::operator+= (const Vector3d & p)

Operator for reflexive addition of two vectors.

Adds the current vector to the provided vector and populates the current vector elements with the result.

Definition at line 190 of file vector3d.cpp.

```

191 {
192     int i;
193
194     for (i=0; i<3; i++)
195     {
196         this->x[i] += p.x[i];
197     }
198 }
```

5.9.3.10 Vector3d Vector3d::operator- (const Vector3d & p) const

Operator for the subtraction of two vectors.

Subtracts the given vector from the current vector and returns the result in a new vector.

Returns

Vector containing the result of subtracting the vector provided as argument from the current vector.

Definition at line 206 of file vector3d.cpp.

```

207 {
208     Vector3d r(0.0, 0.0, 0.0);
209     int i;
210
211     for (i=0; i<3; i++)
212     {
213         r.x[i] = this->x[i] - p.x[i];
214     }
215
216     return (r);
217 }
```

5.9.3.11 void Vector3d::operator-= (const Vector3d & p)

Operator for reflexive subtraction of two vectors.

Subtracts the given vector from the current vector and populates the current vector with the result.

Definition at line 223 of file vector3d.cpp.

```

224 {
225     int i;
226
227     for (i=0; i<3; i++)
228     {
229         this->x[i] -= p.x[i];
230     }
231 }
```

5.9.3.12 Vector3d Vector3d::operator^ (const Vector3d & p) const

Operator for the vector product of two vectors.

Evaluates the vector product of the current vector with the provided vector and returns the result in a third vector.

Returns

Vector containing the result of the vector product of the current vector with the one provided as argument.

Definition at line 289 of file vector3d.cpp.

```

290 {
291     Vector3d r(0.0, 0.0, 0.0);
292
293     r.x[0] = (this->x[1] * p.x[2]) - (this->x[2] * p.x[1]);
294     r.x[1] = (this->x[2] * p.x[0]) - (this->x[0] * p.x[2]);
295     r.x[2] = (this->x[0] * p.x[1]) - (this->x[1] * p.x[0]);
296
297     return (r);
298 }
```

5.9.3.13 void Vector3d::operator^= (const Vector3d & p)

Operator for reflexive vector product of two vectors.

Evaluates the vector product of the current vector and the one provided, and populates the result in the current vector.

Definition at line 304 of file vector3d.cpp.

```

305 {
306     Vector3d* r = Vector3d(0.0, 0.0, 0.0);
307
308     r->x[0] = (this->x[1] * p.x[2]) - (this->x[2] * p.x[1]);
309     r->x[1] = (this->x[2] * p.x[0]) - (this->x[0] * p.x[2]);
310     r->x[2] = (this->x[0] * p.x[1]) - (this->x[1] * p.x[0]);
311
312     *this = *r;
313
314     delete (r);
315     r = NULL;
316 }

```

5.9.3.14 void Vector3d::setValue (int *index*, double *value*)

Function to set the value of an element of the vector.

Sets the value of the element indicated by the index argument.

Parameters

<i>index</i>	Index of the element whose value is to be set.
<i>value</i>	Value that is to be given to the element.

Definition at line 56 of file vector3d.cpp.

```

57 {
58     if (index >= 0 && index < 3)
59     {
60         this->x[index] = value;
61     }
62 }

```

5.9.3.15 void Vector3d::setVector (double * *a*)

Function to set the value of the entire vector using an array.

Sets the values of the elements of the vector to values in the array pointed to by the argument *a*.

Parameters

<i>a</i>	Pointer of the array containing the values of the elements of the vector.
----------	---------------------------------------------------------------------------

Definition at line 69 of file vector3d.cpp.

```

70 {
71     this->x[0] = a[0];
72     this->x[1] = a[1];
73     this->x[2] = a[2];
74 }

```

5.9.3.16 double Vector3d::sum ()

Computes the sum of the elements of the vector.

Sums the elements of the vector and returns the result.

Returns

The sum of the elements of the vector.

Definition at line 116 of file vector3d.cpp.


```
117 {  
118     double s = 0.0;  
119     int i;  
120  
121     for (i=0; i<3; i++)  
122     {  
123         s += this->x[i];  
124     }  
125  
126     return (s);  
127 }
```

5.9.4 Field Documentation

5.9.4.1 double Vector3d::x[3] [protected]

The elements of the vector.

Definition at line 26 of file vector3d.h.

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

- [vector3d.h](#)
- [vector3d.cpp](#)

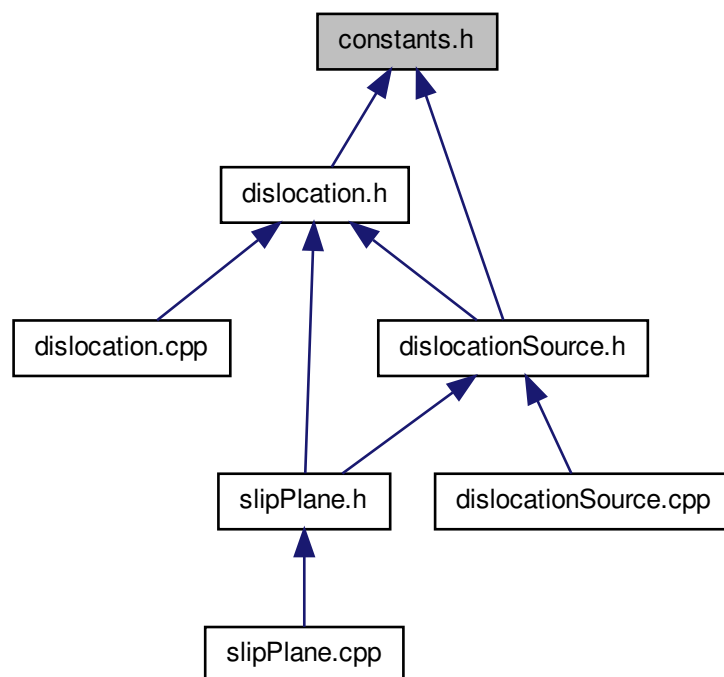
Chapter 6

File Documentation

6.1 constants.h File Reference

Definition of constants used in the program.

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



Macros

- #define `PI` 3.141592654
The irrational number pi.
- #define `SQRT2` 1.414213562

The square root of 2.

- `#define SQRT3 1.732050808`

The square root of 3.

- `#define SQRT5 2.236067978`

The square root of 5.

6.1.1 Detailed Description

Definition of constants used in the program.

Author

Adhish Majumdar

Version

0.0

Date

26/04/2013

This file defines the values of various constants used in the program.

Definition in file [constants.h](#).

6.1.2 Macro Definition Documentation

6.1.2.1 `#define PI 3.141592654`

The irrational number pi.

Definition at line 16 of file constants.h.

6.1.2.2 `#define SQRT2 1.414213562`

The square root of 2.

Definition at line 21 of file constants.h.

6.1.2.3 `#define SQRT3 1.732050808`

The square root of 3.

Definition at line 26 of file constants.h.

6.1.2.4 `#define SQRT5 2.236067978`

The square root of 5.

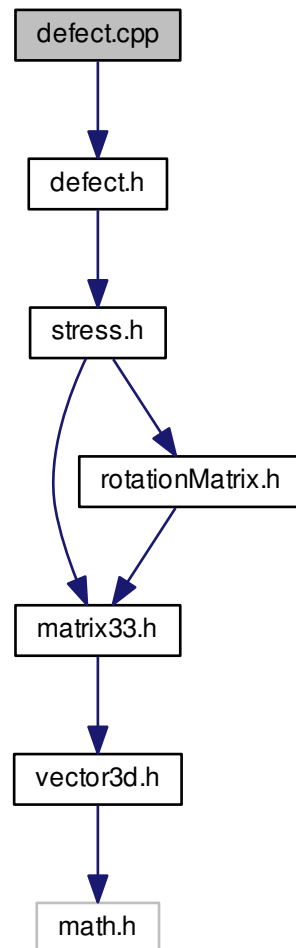
Definition at line 31 of file constants.h.

6.2 defect.cpp File Reference

Definition of member functions of the [Defect](#) class.

```
#include "defect.h"
```

Include dependency graph for defect.cpp:



6.2.1 Detailed Description

Definition of member functions of the [Defect](#) class.

Author

Adhish Majumdar

Version

0.0

Date

22/04/2013

This file defines the member functions of the [Defect](#) class representing a single defect in the simulation.

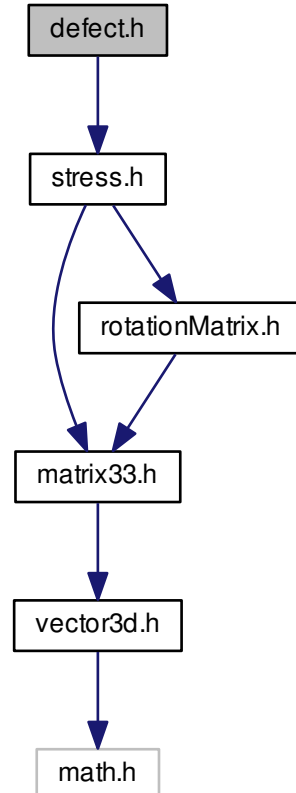
Definition in file [defect.cpp](#).

6.3 defect.h File Reference

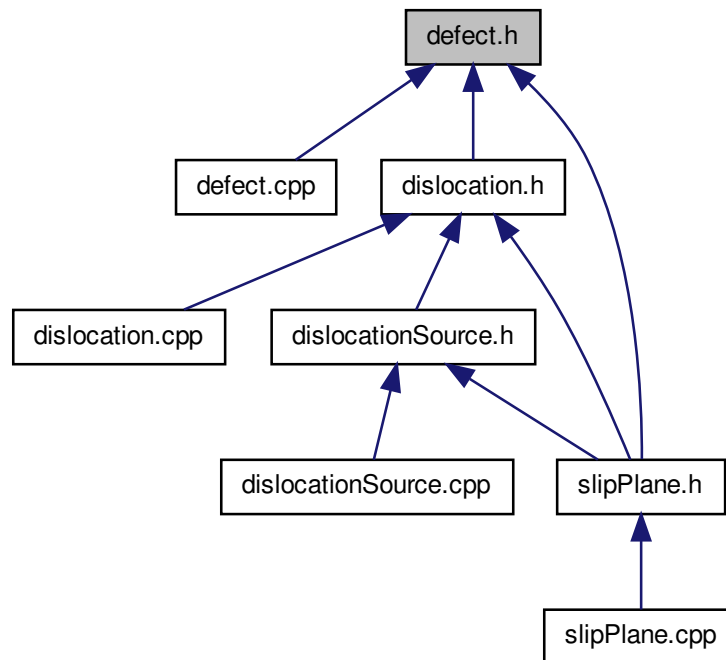
Definition of the [Defect](#) class.

```
#include "stress.h"
```

Include dependency graph for defect.h:



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



Data Structures

- class [Defect](#)

Class [Defect](#) representing a generic defect in a material.

6.3.1 Detailed Description

Definition of the [Defect](#) class.

Author

Adhish Majumdar

Version

0.0

Date

27/05/2013

This file defines the [Defect](#) class representing an defect in the simulation. This is simply a generic description class with virtual functions. Later classes like dislocations, precipitates, boundaries etc will inherit from this class.

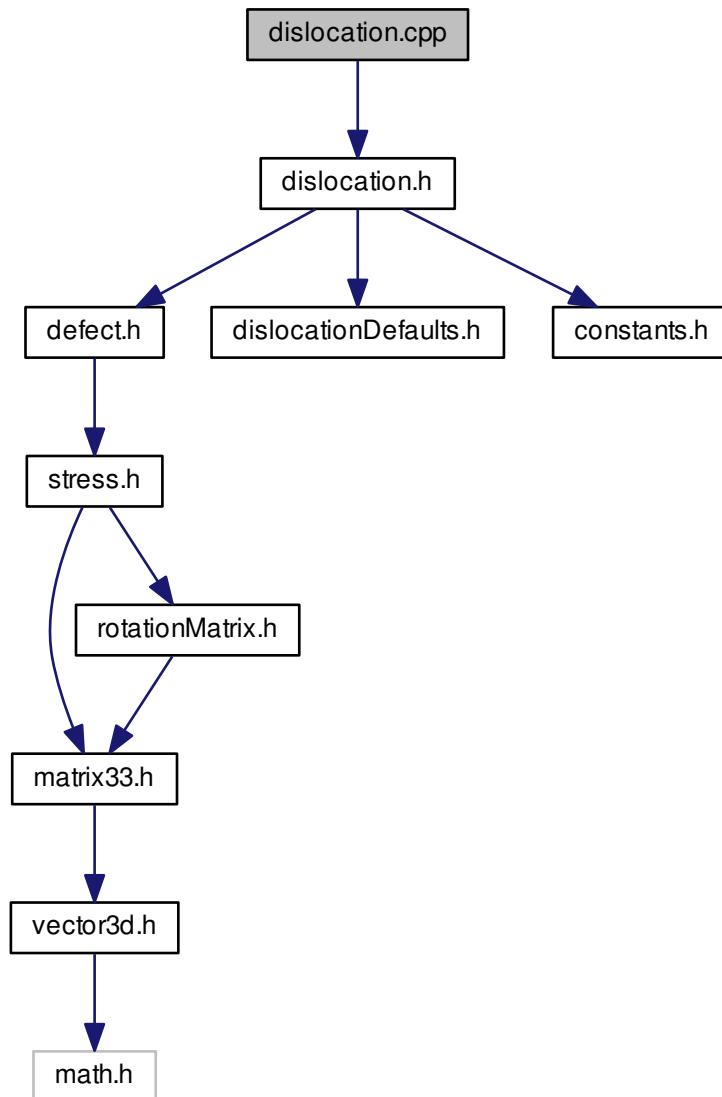
Definition in file [defect.h](#).

6.4 dislocation.cpp File Reference

Definition of constructors and member functions of the [Dislocation](#) class.

```
#include "dislocation.h"
```

Include dependency graph for dislocation.cpp:



6.4.1 Detailed Description

Definition of constructors and member functions of the [Dislocation](#) class.

Author

Adhish Majumdar

Version

0.0

Date

29/04/2013

This file defines the constructors and member functions of the [Dislocation](#) class. This class inherits from the [Defect](#) class.

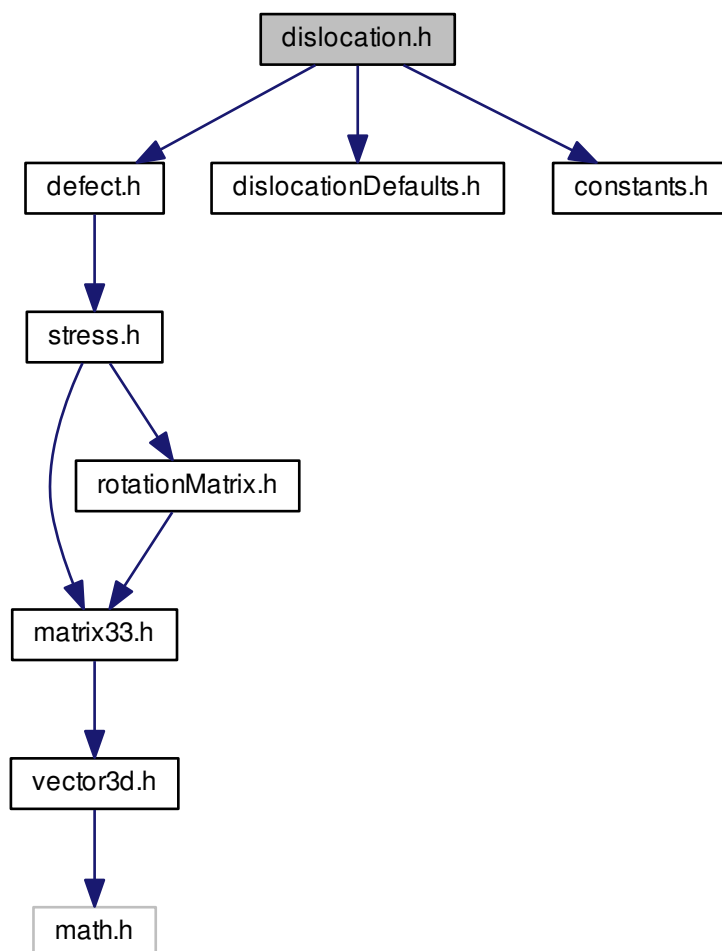
Definition in file [dislocation.cpp](#).

6.5 dislocation.h File Reference

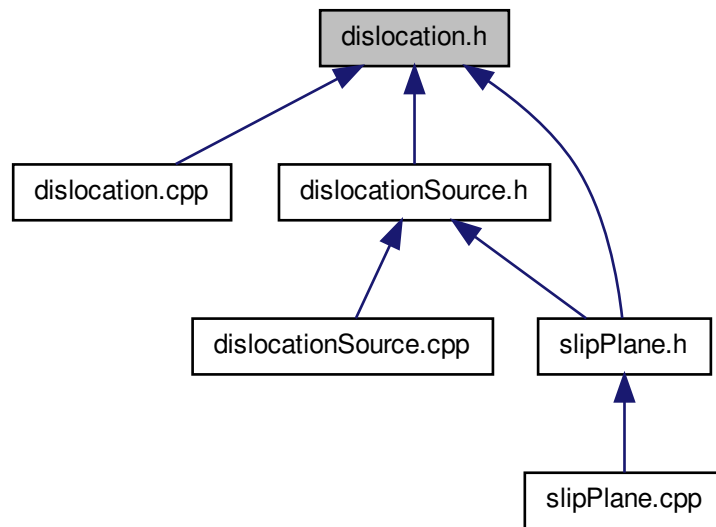
Definition of the [Dislocation](#) class.

```
#include "defect.h"
#include "dislocationDefaults.h"
#include "constants.h"
```

Include dependency graph for dislocation.h:



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



Data Structures

- class [Dislocation](#)
[Dislocation](#) class representing a dislocation in the simulation.

6.5.1 Detailed Description

Definition of the [Dislocation](#) class.

Author

Adhish Majumdar

Version

0.0

Date

29/04/2013

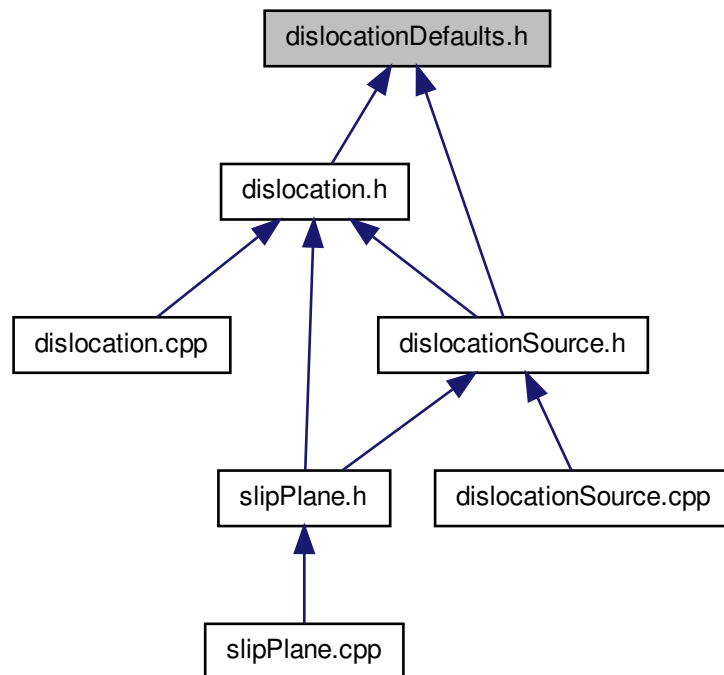
This file defines the [Dislocation](#) class representing a dislocation in the simulation. This class inherits from the [Defect](#) class.

Definition in file [dislocation.h](#).

6.6 dislocationDefaults.h File Reference

Definition of certain default values for members of the [Dislocation](#) class.

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



Macros

- `#define` `DEFAULT_POSITION_0` 0.0
Default value of the position vector x-coordinate.
- `#define` `DEFAULT_POSITION_1` 0.0
Default value of the position vector y-coordinate.
- `#define` `DEFAULT_POSITION_2` 0.0
Default value of the position vector z-coordinate.
- `#define` `DEFAULT_BURGERS_MAGNITUDE` 5.0e-09
Default value of the magnitude of the Burgers vector.
- `#define` `DEFAULT_BURGERS_0` 1.0
Default value of the Burgers vector x-coordinate.
- `#define` `DEFAULT_BURGERS_1` 1.0
Default value of the Burgers vector y-coordinate.
- `#define` `DEFAULT_BURGERS_2` 0.0
Default value of the Burgers vector z-coordinate.
- `#define` `DEFAULT_LINEVECTOR_0` 1.0
Default value of the line vector x-coordinate.
- `#define` `DEFAULT_LINEVECTOR_1` -1.0
Default value of the line vector y-coordinate.
- `#define` `DEFAULT_LINEVECTOR_2` -2.0
Default value of the line vector z-coordinate.

6.6.1 Detailed Description

Definition of certain default values for members of the [Dislocation](#) class.

Author

Adhish Majumdar

Version

0.0

Date

26/04/2013

This file defines some default values for members of the [Dislocation](#) class representing a dislocation in the simulation.

Definition in file [dislocationDefaults.h](#).

6.6.2 Macro Definition Documentation

6.6.2.1 `#define DEFAULT_BURGERS_0 1.0`

Default value of the Burgers vector x-coordinate.

Definition at line 34 of file [dislocationDefaults.h](#).

6.6.2.2 `#define DEFAULT_BURGERS_1 1.0`

Default value of the Burgers vector y-coordinate.

Definition at line 38 of file [dislocationDefaults.h](#).

6.6.2.3 `#define DEFAULT_BURGERS_2 0.0`

Default value of the Burgers vector z-coordinate.

Definition at line 42 of file [dislocationDefaults.h](#).

6.6.2.4 `#define DEFAULT_BURGERS_MAGNITUDE 5.0e-09`

Default value of the magnitude of the Burgers vector.

Definition at line 29 of file [dislocationDefaults.h](#).

6.6.2.5 `#define DEFAULT_LINEVECTOR_0 1.0`

Default value of the line vector x-coordinate.

Definition at line 47 of file [dislocationDefaults.h](#).

6.6.2.6 `#define DEFAULT_LINEVECTOR_1 -1.0`

Default value of the line vector y-coordinate.

Definition at line 51 of file [dislocationDefaults.h](#).

6.6.2.7 `#define DEFAULT_LINEVECTOR_2 -2.0`

Default value of the line vector z-coordinate.

Definition at line 55 of file dislocationDefaults.h.

6.6.2.8 `#define DEFAULT_POSITION_0 0.0`

Default value of the position vector x-coordinate.

Definition at line 16 of file dislocationDefaults.h.

6.6.2.9 `#define DEFAULT_POSITION_1 0.0`

Default value of the position vector y-coordinate.

Definition at line 20 of file dislocationDefaults.h.

6.6.2.10 `#define DEFAULT_POSITION_2 0.0`

Default value of the position vector z-coordinate.

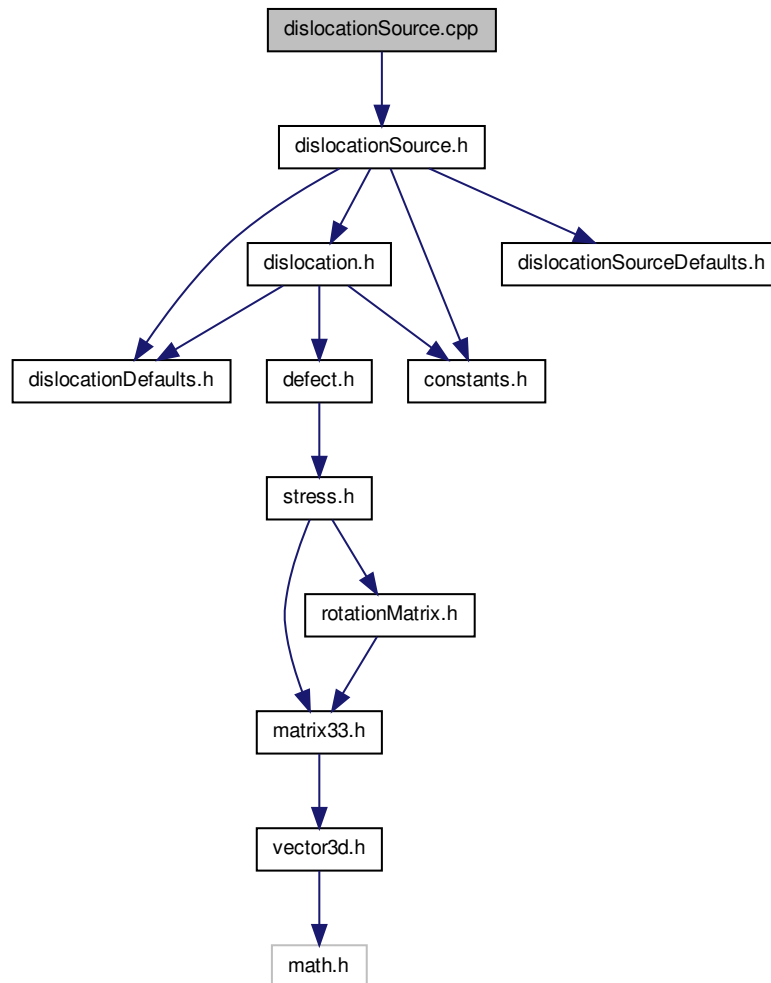
Definition at line 24 of file dislocationDefaults.h.

6.7 dislocationSource.cpp File Reference

Definition of the member functions of the [DislocationSource](#) class.

```
#include "dislocationSource.h"
```

Include dependency graph for dislocationSource.cpp:



6.7.1 Detailed Description

Definition of the member functions of the [DislocationSource](#) class.

Author

Adhish Majumdar

Version

0.0

Date

27/05/2013

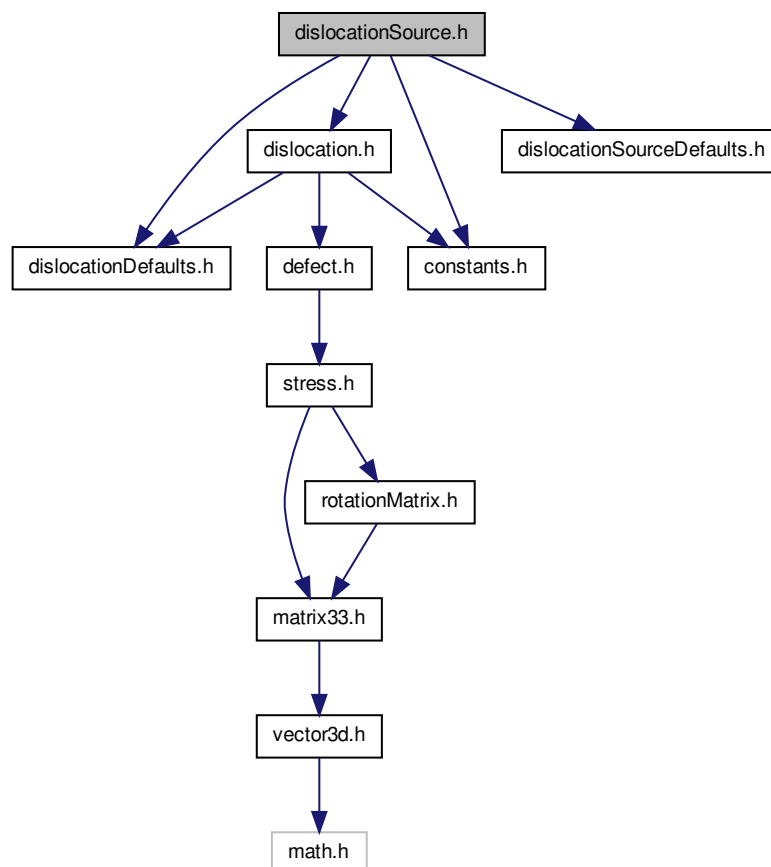
This file defines the member functions of the [DislocationSource](#) class representing a source of dislocations in the simulation. This class inherits from the [Defect](#) class. This object is basically the representation of a Frank-Read source emitting dislocation dipoles. When the dislocation source experiences a shear stress greater than a critical value for a certain amount of time (or number of iterations), it emits a dislocation dipole with a length that is a function of the applied stress.

Definition in file [dislocationSource.cpp](#).

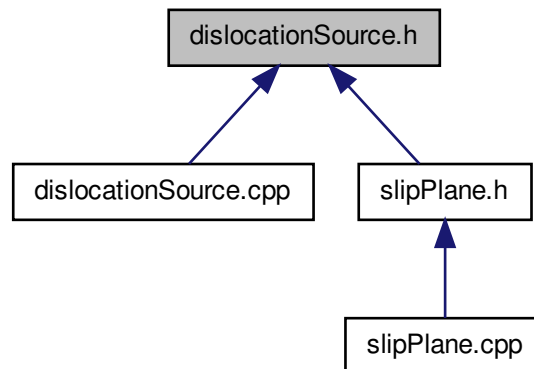
6.8 dislocationSource.h File Reference

Definition of the [DislocationSource](#) class.

```
#include "dislocation.h"  
#include "constants.h"  
#include "dislocationDefaults.h"  
#include "dislocationSourceDefaults.h"  
Include dependency graph for dislocationSource.h:
```



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



Data Structures

- class [DislocationSource](#)
[DislocationSource](#) class representing a source of dislocations in the simulation.

6.8.1 Detailed Description

Definition of the [DislocationSource](#) class.

Author

Adhish Majumdar

Version

0.0

Date

27/05/2013

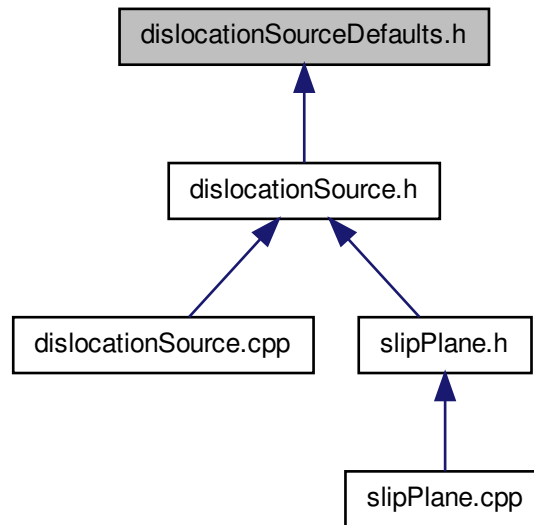
This file defines the [DislocationSource](#) class representing a source of dislocations in the simulation. This class inherits from the [Defect](#) class. This object is basically the representation of a Frank-Read source emitting dislocation dipoles. When the dislocation source experiences a shear stress greater than a critical value for a certain amount of time (or number of iterations), it emits a dislocation dipole with a length that is a function of the applied stress.

Definition in file [dislocationSource.h](#).

6.9 dislocationSourceDefaults.h File Reference

Definition of certain default values for members of the [DislocationSource](#) class.

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



Macros

- `#define` [DEFAULT_TAU_CRITICAL](#) 1.0e09
Default value of the critical shear stress for a dislocation source to emit a dipole.
- `#define` [DEFAULT_NITERATIONS](#) 10
Default value of the number of iterations required for a dislocation source to emit a dipole.

6.9.1 Detailed Description

Definition of certain default values for members of the [DislocationSource](#) class.

Author

Adhish Majumdar

Version

0.0

Date

02/05/2013

This file defines some default values for members of the [DislocationSource](#) class representing a dislocation dipole source in the simulation.

Definition in file [dislocationSourceDefaults.h](#).

6.9.2 Macro Definition Documentation

6.9.2.1 `#define DEFAULT_ITERATIONS 10`

Default value of the number of iterations required for a dislocation source to emit a dipole.

The dislocation source must experience a shear stress greater than the critical value in order to emit a dipole. This time is expressed in terms of the number of iterations here.

Definition at line 23 of file `dislocationSourceDefaults.h`.

6.9.2.2 `#define DEFAULT_TAU_CRITICAL 1.0e09`

Default value of the critical shear stress for a dislocation source to emit a dipole.

Default value of the critical shear stress for a dislocation source to emit a dipole. The number is expressed in Pa.

Definition at line 17 of file `dislocationSourceDefaults.h`.

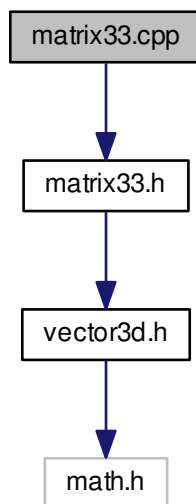
6.10 `mainpage.dox` File Reference

6.11 `matrix33.cpp` File Reference

Definition of the member functions and operators of the [Matrix33](#) class.

```
#include "matrix33.h"
```

Include dependency graph for `matrix33.cpp`:



6.11.1 Detailed Description

Definition of the member functions and operators of the [Matrix33](#) class.

Author

Adhish Majumdar

Version

0.0

Date

22/04/2013

This file defines the member functions and operators of the [Matrix33](#) class representing a 3x3 matrix in the simulation.

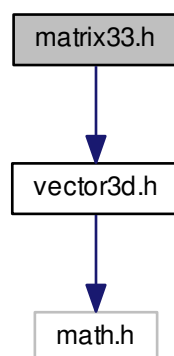
Definition in file [matrix33.cpp](#).

6.12 matrix33.h File Reference

Definition of the [Matrix33](#) class.

```
#include "vector3d.h"
```

Include dependency graph for matrix33.h:



6.13 rotationMatrix.cpp File Reference

Definition of the [RotationMatrix](#) class member functions.

6.13.1 Detailed Description

Definition of the [RotationMatrix](#) class member functions.

Author

Adhish Majumdar

Version

0.0

Date

25/04/2013

This file defines member functions of the [RotationMatrix](#) class for carrying out 3D rotations and axes transformations.

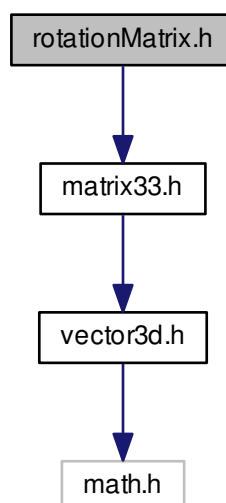
Definition in file [rotationMatrix.cpp](#).

6.14 rotationMatrix.h File Reference

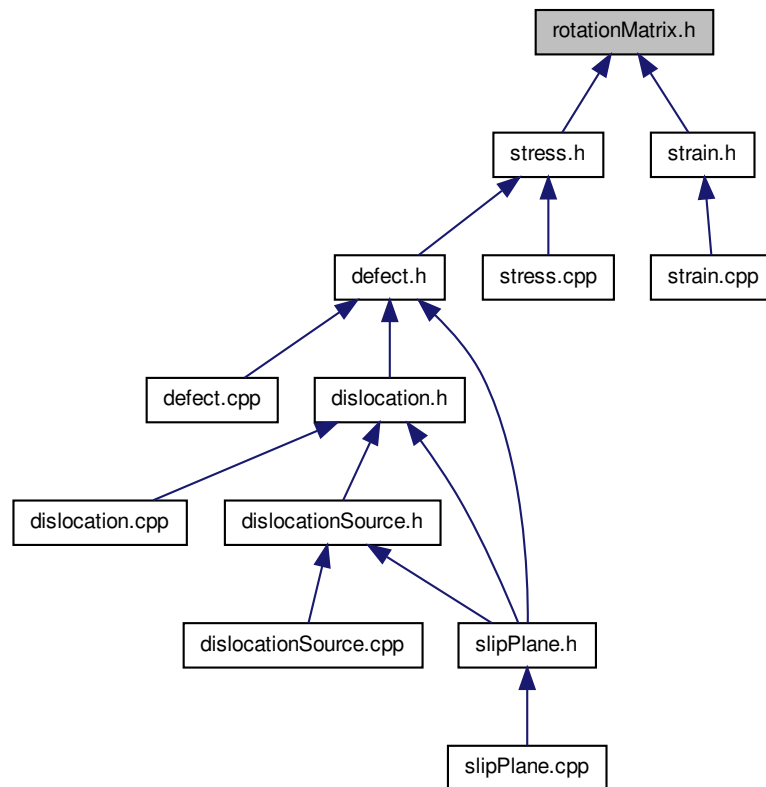
Definition of the [RotationMatrix](#) class.

```
#include "matrix33.h"
```

Include dependency graph for rotationMatrix.h:



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



Data Structures

- class [RotationMatrix](#)
[RotationMatrix](#) class to represent a rotation matrix.

6.14.1 Detailed Description

Definition of the [RotationMatrix](#) class.

Author

Adhish Majumdar

Version

0.0

Date

25/04/2013

This file defines the [RotationMatrix](#) class for carrying out 3D rotations and axes transformations.

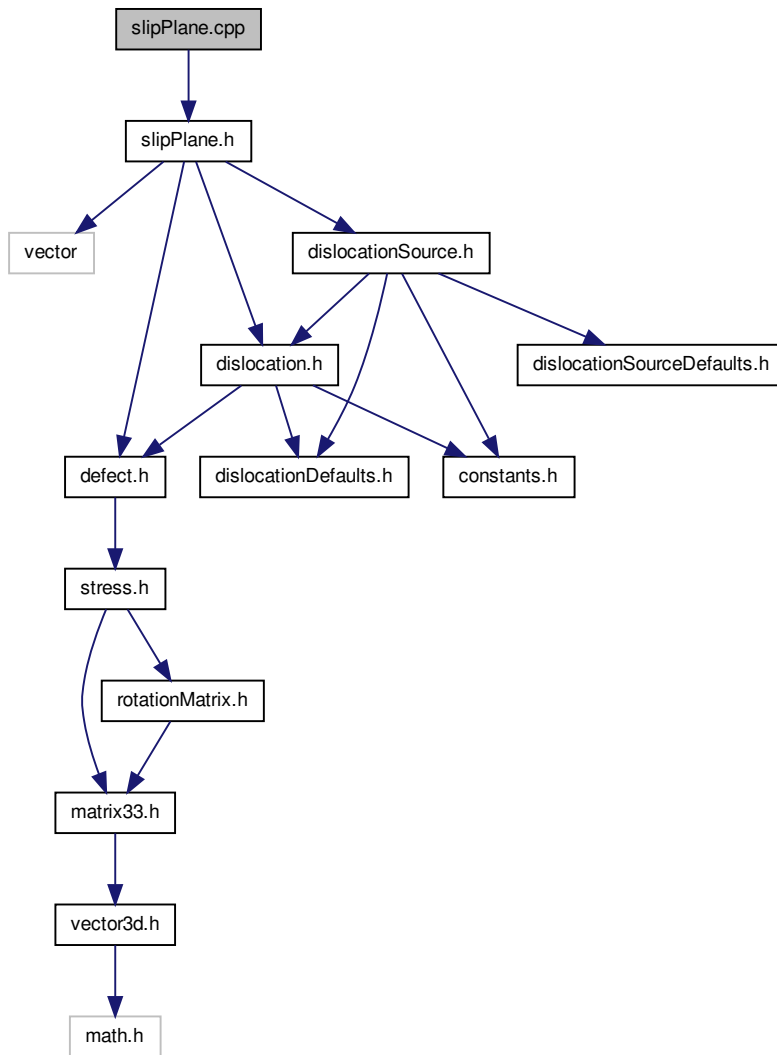
Definition in file [rotationMatrix.h](#).

6.15 slipPlane.cpp File Reference

Definition of the member functions of the [SlipPlane](#) class.

```
#include "slipPlane.h"
```

Include dependency graph for slipPlane.cpp:



Functions

- [SlipPlane](#) ()
Default constructor.

6.15.1 Detailed Description

Definition of the member functions of the [SlipPlane](#) class.

Author

Adhish Majumdar

Version

0.0

Date

28/05/2013

This file defines the member functions of the [SlipPlane](#) class.

Definition in file [slipPlane.cpp](#).

6.15.2 Function Documentation

6.15.2.1 [SlipPlane](#) ()

Default constructor.

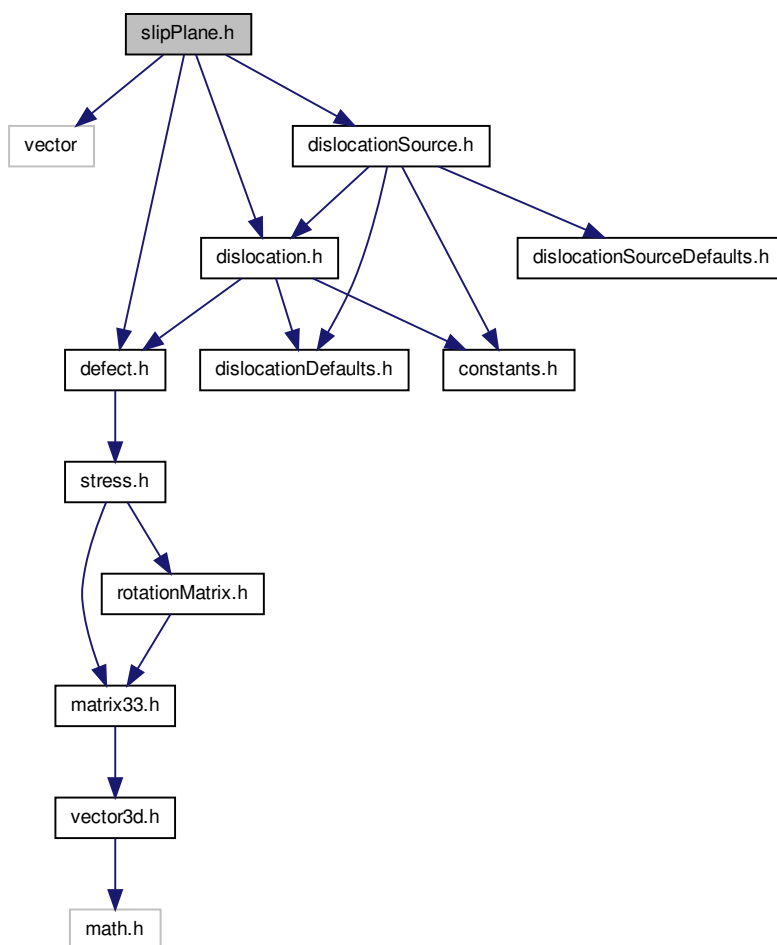
The slip plane is initialized with default parameters.

6.16 [slipPlane.h](#) File Reference

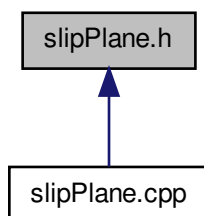
Definition of the [SlipPlane](#) class.

```
#include <vector>
#include "defect.h"
#include "dislocation.h"
#include "dislocationSource.h"
```


Include dependency graph for slipPlane.h:



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



Data Structures

- class [SlipPlane](#)

[SlipPlane](#) class representing a slip plane in the simulation.

6.16.1 Detailed Description

Definition of the [SlipPlane](#) class.

Author

Adhish Majumdar

Version

0.0

Date

28/05/2013

This file defines the [SlipPlane](#) class representing a slip plane in the simulation.

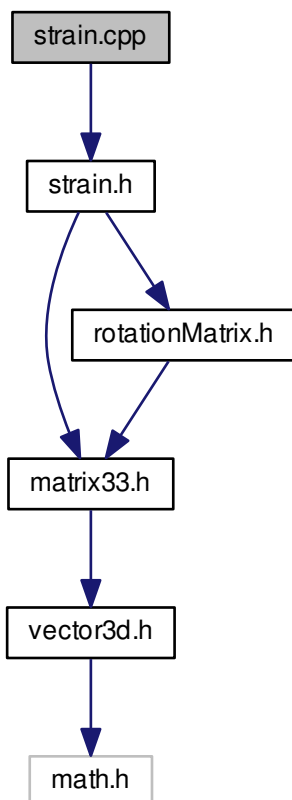
Definition in file [slipPlane.h](#).

6.17 strain.cpp File Reference

Definition of the member functions if the [Strain](#) class.

```
#include "strain.h"
```

Include dependency graph for strain.cpp:



6.17.1 Detailed Description

Definition of the member functions if the [Strain](#) class.

Author

Adhish Majumdar

Version

0.0

Date

22/04/2013

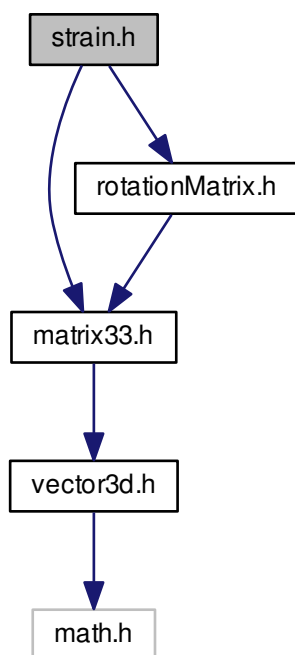
This file defines the member functions of the [Strain](#) class for the strain tensor.

Definition in file [strain.cpp](#).

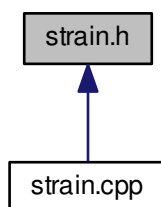
6.18 strain.h File Reference

Definition of the [Strain](#) class.

```
#include "matrix33.h"  
#include "rotationMatrix.h"  
Include dependency graph for strain.h:
```



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



Data Structures

- class [Strain](#)

Strain class to represent the strain tensor.

6.18.1 Detailed Description

Definition of the [Strain](#) class.

Author

Adhish Majumdar

Version

0.0

Date

25/04/2013

This file defines the [Strain](#) class for the strain tensor.

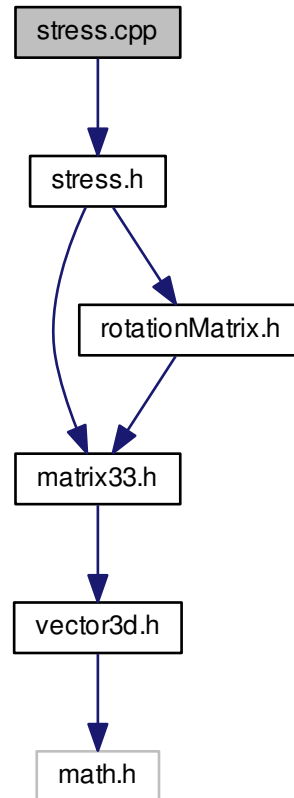
Definition in file [strain.h](#).

6.19 stress.cpp File Reference

Definition of the member functions if the [Stress](#) class.

```
#include "stress.h"
```

Include dependency graph for stress.cpp:



6.19.1 Detailed Description

Definition of the member functions of the [Stress](#) class.

Author

Adhish Majumdar

Version

0.0

Date

22/04/2013

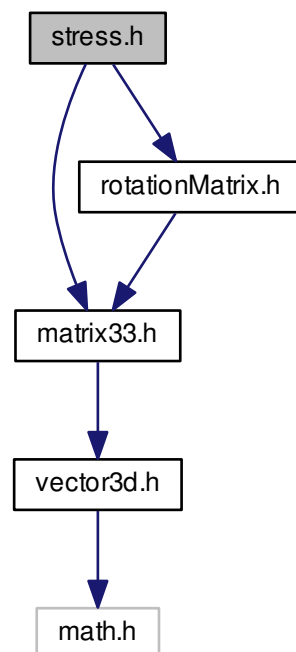
This file defines the member functions of the [Stress](#) class for the stress tensor.

Definition in file [stress.cpp](#).

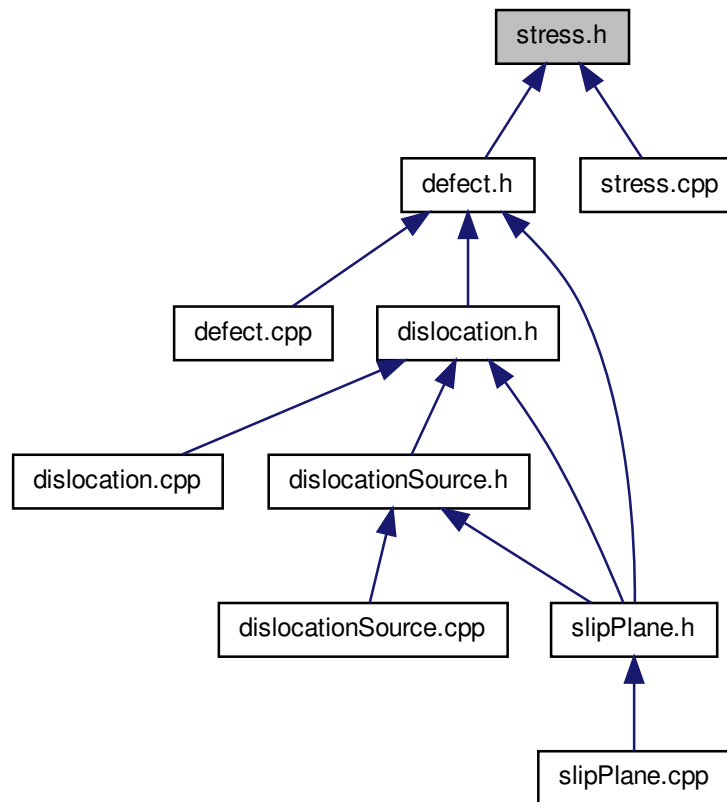
6.20 stress.h File Reference

Definition of the [Stress](#) class.

```
#include "matrix33.h"  
#include "rotationMatrix.h"  
Include dependency graph for stress.h:
```



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



Data Structures

- class [Stress](#)
[Stress](#) class to represent the stress tensor.

6.20.1 Detailed Description

Definition of the [Stress](#) class.

Author

Adhish Majumdar

Version

0.0

Date

22/04/2013

This file defines the [Stress](#) class for the stress tensor.

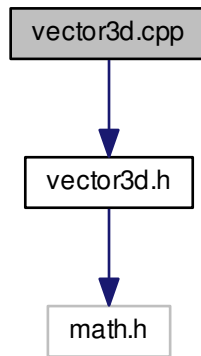
Definition in file [stress.h](#).

6.21 vector3d.cpp File Reference

Definition of member functions and operators of the [Vector3d](#) class.

```
#include "vector3d.h"
```

Include dependency graph for vector3d.cpp:



6.21.1 Detailed Description

Definition of member functions and operators of the [Vector3d](#) class.

Author

Adhish Majumdar

Version

0.0

Date

29/04/2013

This file defines the member functions and operators of the [Vector3d](#) class representing a single 3-dimensional vector in the simulation.

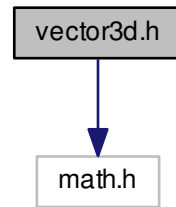
Definition in file [vector3d.cpp](#).

6.22 vector3d.h File Reference

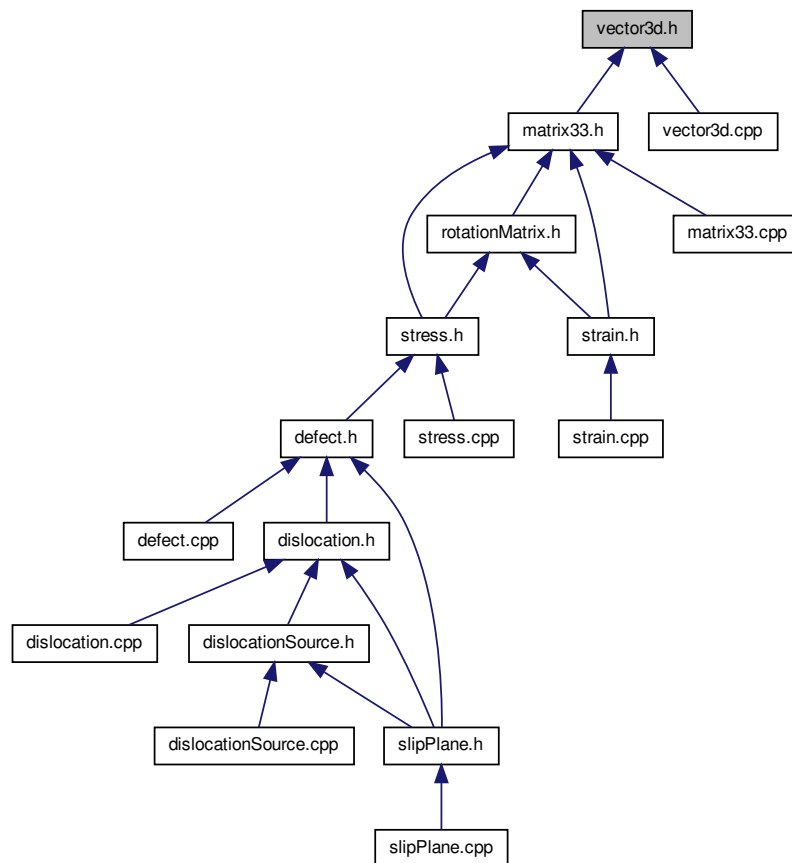
Definition of the [Vector3d](#) class.

```
#include <math.h>
```

Include dependency graph for vector3d.h:



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



Data Structures

- class [Vector3d](#)

[Vector3d](#) class representing a single 3-dimensional vector in the simulation.

6.22.1 Detailed Description

Definition of the [Vector3d](#) class.

Author

Adhish Majumdar

Version

0.0

Date

29/04/2013

This file defines the [Vector3d](#) class representing a single 3-dimensional vector in the simulation.

Definition in file [vector3d.h](#).

Index

- adjugate
 - Matrix33, [40](#)
 - RotationMatrix, [49](#)
 - Strain, [67](#)
 - Stress, [77](#)
- bmag
 - Dislocation, [24](#)
 - DislocationSource, [36](#)
- bvec
 - Dislocation, [24](#)
 - DislocationSource, [36](#)
- calculateRotationMatrix
 - Dislocation, [18](#)
 - SlipPlane, [58](#)
- constants.h, [95](#)
 - PI, [96](#)
 - SQRT2, [96](#)
 - SQRT3, [96](#)
 - SQRT5, [96](#)
- countIterations
 - DislocationSource, [36](#)
- DEFAULT_BURGERS_0
 - dislocationDefaults.h, [104](#)
- DEFAULT_BURGERS_1
 - dislocationDefaults.h, [104](#)
- DEFAULT_BURGERS_2
 - dislocationDefaults.h, [104](#)
- DEFAULT_NITERATIONS
 - dislocationSourceDefaults.h, [110](#)
- DEFAULT_POSITION_0
 - dislocationDefaults.h, [105](#)
- DEFAULT_POSITION_1
 - dislocationDefaults.h, [105](#)
- DEFAULT_POSITION_2
 - dislocationDefaults.h, [105](#)
- Defect, [9](#)
 - Defect, [11](#)
 - getPosition, [11](#), [12](#)
 - getX, [12](#)
 - getY, [12](#)
 - getZ, [12](#)
 - pos, [15](#)
 - setPosition, [13](#)
 - setX, [14](#)
 - setY, [14](#)
 - setZ, [14](#)
 - stressField, [14](#)
- defect.cpp, [97](#)
- defect.h, [98](#)
- dipoleNucleationLength
 - DislocationSource, [29](#)
- Dislocation, [15](#)
 - bmag, [24](#)
 - bvec, [24](#)
 - calculateRotationMatrix, [18](#)
 - Dislocation, [17](#), [18](#)
 - getBurgers, [19](#)
 - getLineVector, [19](#)
 - getPosition, [19](#)
 - getX, [20](#)
 - getY, [20](#)
 - getZ, [20](#)
 - lvec, [24](#)
 - mobile, [24](#)
 - pos, [25](#)
 - rotationMatrix, [25](#)
 - setBurgers, [20](#)
 - setLineVector, [21](#)
 - setMobile, [21](#)
 - setPinned, [21](#)
 - setPosition, [21](#), [22](#)
 - setX, [22](#)
 - setY, [22](#)
 - setZ, [23](#)
 - stressField, [23](#)
 - stressFieldLocal, [23](#)
- dislocation.cpp, [100](#)
- dislocation.h, [101](#)
- dislocationDefaults.h, [102](#)
 - DEFAULT_BURGERS_0, [104](#)
 - DEFAULT_BURGERS_1, [104](#)
 - DEFAULT_BURGERS_2, [104](#)
 - DEFAULT_POSITION_0, [105](#)
 - DEFAULT_POSITION_1, [105](#)
 - DEFAULT_POSITION_2, [105](#)
- DislocationSource, [25](#)
 - bmag, [36](#)
 - bvec, [36](#)
 - countIterations, [36](#)
 - dipoleNucleationLength, [29](#)
 - DislocationSource, [28](#)
 - DislocationSource, [28](#)
 - getBurgers, [29](#)
 - getBurgersMag, [29](#)
 - getIterationCount, [30](#)
 - getLineVector, [30](#)

- getNumIterations, 30
- getPosition, 30, 31
- getTauCritical, 31
- getX, 31
- getY, 31
- getZ, 32
- ifEmitDipole, 32
- incrementIterationCount, 32
- lvec, 36
- mobile, 37
- nIterations, 37
- pos, 37
- resetIterationCounter, 32
- rotationMatrix, 37
- setBurgers, 33
- setBurgersMagnitude, 33
- setLineVector, 33
- setNumIterations, 33
- setPosition, 34
- setTauCritical, 35
- setX, 35
- setY, 35
- setZ, 35
- stressField, 36
- tauCritical, 37
- dislocationSource.cpp, 105
- dislocationSource.h, 107
- dislocationSourceDefaults.h, 108
- dislocationSources
 - SlipPlane, 64
- dislocations
 - SlipPlane, 63
- extremities
 - SlipPlane, 64
- getAxis
 - SlipPlane, 58
- getBurgers
 - Dislocation, 19
 - DislocationSource, 29
- getBurgersMag
 - DislocationSource, 29
- getDislocation
 - SlipPlane, 59
- getDislocationList
 - SlipPlane, 60
- getDislocationSource
 - SlipPlane, 60
- getDislocationSourceList
 - SlipPlane, 60
- getExtremities
 - SlipPlane, 61
- getExtremity
 - SlipPlane, 61
- getIterationCount
 - DislocationSource, 30
- getLineVector
 - Dislocation, 19
- DislocationSource, 30
- getNormal
 - SlipPlane, 61
- getNumIterations
 - DislocationSource, 30
- getPosition
 - Defect, 11, 12
 - Dislocation, 19
 - DislocationSource, 30, 31
 - SlipPlane, 62
- getPrincipalStrains
 - Strain, 68
- getPrincipalStresses
 - Stress, 78
- getRotationMatrix
 - SlipPlane, 62
- getShearStrains
 - Strain, 68
- getShearStresses
 - Stress, 78
- getTauCritical
 - DislocationSource, 31
- getValue
 - Matrix33, 41
 - RotationMatrix, 49
 - Strain, 68
 - Stress, 78
 - Vector3d, 87
- getVector
 - Vector3d, 88
- getX
 - Defect, 12
 - Dislocation, 20
 - DislocationSource, 31
- getY
 - Defect, 12
 - Dislocation, 20
 - DislocationSource, 31
- getZ
 - Defect, 12
 - Dislocation, 20
 - DislocationSource, 32
- ifEmitDipole
 - DislocationSource, 32
- incrementIterationCount
 - DislocationSource, 32
- lvec
 - Dislocation, 24
 - DislocationSource, 36
- magnitude
 - Vector3d, 88
- mainpage.dox, 110
- Matrix33, 37
 - adjugate, 40
 - getValue, 41
 - Matrix33, 39, 40

- operator*, 42, 43
- operator*=: 43
- operator~, 45
- operator[^], 45
- operator+, 44
- operator+=, 44
- operator-, 44
- operator-=, 45
- setValue, 46
- x, 46
- matrix33.cpp, 110
- matrix33.h, 111
- mobile
 - Dislocation, 24
 - DislocationSource, 37
- nIterations
 - DislocationSource, 37
- normalVector
 - SlipPlane, 64
- normalize
 - Vector3d, 88
- operator*
 - Matrix33, 42, 43
 - RotationMatrix, 50, 51
 - Strain, 69, 70
 - Stress, 79, 80
 - Vector3d, 89
- operator*=
 - Matrix33, 43
 - RotationMatrix, 52
 - Strain, 70, 71
 - Stress, 81
 - Vector3d, 89
- operator~
 - Matrix33, 45
 - RotationMatrix, 54
 - Strain, 73
 - Stress, 83
- operator[^]
 - Matrix33, 45
 - RotationMatrix, 54
 - Strain, 72
 - Stress, 83
 - Vector3d, 91
- operator[^]=
 - Vector3d, 91
- operator+
 - Matrix33, 44
 - RotationMatrix, 52
 - Strain, 71
 - Stress, 81
 - Vector3d, 90
- operator+=
 - Matrix33, 44
 - RotationMatrix, 53
 - Strain, 71
 - Stress, 82
- Vector3d, 90
- operator-
 - Matrix33, 44
 - RotationMatrix, 53
 - Strain, 72
 - Stress, 82
 - Vector3d, 90
- operator-=
 - Matrix33, 45
 - RotationMatrix, 53
 - Strain, 72
 - Stress, 82
 - Vector3d, 91
- PI
 - constants.h, 96
- populateMatrix
 - Strain, 73
 - Stress, 83
- pos
 - Defect, 15
 - Dislocation, 25
 - DislocationSource, 37
- position
 - SlipPlane, 64
- principalStrains
 - Strain, 74
- principalStresses
 - Stress, 85
- resetIterationCounter
 - DislocationSource, 32
- rotate
 - Strain, 73
 - Stress, 84
- RotationMatrix, 47
 - adjugate, 49
 - getValue, 49
 - operator*, 50, 51
 - operator*=: 52
 - operator~, 54
 - operator[^], 54
 - operator+, 52
 - operator+=, 53
 - operator-, 53
 - operator-=, 53
 - RotationMatrix, 48
 - RotationMatrix, 48
 - setValue, 54
 - x, 55
- rotationMatrix
 - Dislocation, 25
 - DislocationSource, 37
 - SlipPlane, 64
- rotationMatrix.cpp, 113
- rotationMatrix.h, 113
- SQRT2
 - constants.h, 96

- SQRT3
 - constants.h, 96
- SQRT5
 - constants.h, 96
- setBurgers
 - Dislocation, 20
 - DislocationSource, 33
- setBurgersMagnitude
 - DislocationSource, 33
- setDislocationList
 - SlipPlane, 62
- setDislocationSourceList
 - SlipPlane, 62
- setExtremities
 - SlipPlane, 63
- setLineVector
 - Dislocation, 21
 - DislocationSource, 33
- setMobile
 - Dislocation, 21
- setNormal
 - SlipPlane, 63
- setNumIterations
 - DislocationSource, 33
- setPinned
 - Dislocation, 21
- setPosition
 - Defect, 13
 - Dislocation, 21, 22
 - DislocationSource, 34
 - SlipPlane, 63
- setTauCritical
 - DislocationSource, 35
- setValue
 - Matrix33, 46
 - RotationMatrix, 54
 - Strain, 74
 - Stress, 84
 - Vector3d, 92
- setVector
 - Vector3d, 92
- setX
 - Defect, 14
 - Dislocation, 22
 - DislocationSource, 35
- setY
 - Defect, 14
 - Dislocation, 22
 - DislocationSource, 35
- setZ
 - Defect, 14
 - Dislocation, 23
 - DislocationSource, 35
- shearStrains
 - Strain, 74
- shearStresses
 - Stress, 85
- SlipPlane, 55
 - calculateRotationMatrix, 58
 - dislocationSources, 64
 - dislocations, 63
 - extremities, 64
 - getAxis, 58
 - getDislocation, 59
 - getDislocationList, 60
 - getDislocationSource, 60
 - getDislocationSourceList, 60
 - getExtremities, 61
 - getExtremity, 61
 - getNormal, 61
 - getPosition, 62
 - getRotationMatrix, 62
 - normalVector, 64
 - position, 64
 - rotationMatrix, 64
 - setDislocationList, 62
 - setDislocationSourceList, 62
 - setExtremities, 63
 - setNormal, 63
 - setPosition, 63
 - SlipPlane, 57
 - SlipPlane, 57
 - slipPlane.cpp, 116
- slipPlane.cpp, 115
 - SlipPlane, 116
- slipPlane.h, 116
- Strain, 65
 - adjugate, 67
 - getPrincipalStrains, 68
 - getShearStrains, 68
 - getValue, 68
 - operator*, 69, 70
 - operator*=: 70, 71
 - operator~, 73
 - operator[^], 72
 - operator+, 71
 - operator+=, 71
 - operator-, 72
 - operator-=, 72
 - populateMatrix, 73
 - principalStrains, 74
 - rotate, 73
 - setValue, 74
 - shearStrains, 74
 - Strain, 66, 67
 - x, 75
- strain.cpp, 118
- strain.h, 120
- Stress, 75
 - adjugate, 77
 - getPrincipalStresses, 78
 - getShearStresses, 78
 - getValue, 78
 - operator*, 79, 80
 - operator*=: 81
 - operator~, 83

- operator[^], 83
- operator+, 81
- operator+=, 82
- operator-, 82
- operator-=, 82
- populateMatrix, 83
- principalStresses, 85
- rotate, 84
- setValue, 84
- shearStresses, 85
- Stress, 77
- x, 85
- stress.cpp, 121
- stress.h, 123
- stressField
 - Defect, 14
 - Dislocation, 23
 - DislocationSource, 36
- stressFieldLocal
 - Dislocation, 23
- sum
 - Vector3d, 92
- tauCritical
 - DislocationSource, 37
- Vector3d, 85
 - getValue, 87
 - getVector, 88
 - magnitude, 88
 - normalize, 88
 - operator*, 89
 - operator*=, 89
 - operator[^], 91
 - operator[^]==, 91
 - operator+, 90
 - operator+=, 90
 - operator-, 90
 - operator-=, 91
 - setValue, 92
 - setVector, 92
 - sum, 92
 - Vector3d, 86, 87
 - x, 93
- vector3d.cpp, 125
- vector3d.h, 125
- x
 - Matrix33, 46
 - RotationMatrix, 55
 - Strain, 75
 - Stress, 85
 - Vector3d, 93