

DD2D - Matryoshka approach

0

Generated by Doxygen 1.8.3.1

Wed Jun 5 2013 17:20:43

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	18
5.2.2.1	Dislocation	18
5.2.2.2	Dislocation	18
5.2.3	Member Function Documentation	18
5.2.3.1	calculateRotationMatrix	18
5.2.3.2	forcePeachKoehler	19
5.2.3.3	getBurgers	19
5.2.3.4	getLineVector	20
5.2.3.5	getPosition	20
5.2.3.6	getPosition	20
5.2.3.7	getX	20
5.2.3.8	getY	21
5.2.3.9	getZ	21
5.2.3.10	idealTimeIncrement	21
5.2.3.11	isMobile	22
5.2.3.12	setBurgers	22
5.2.3.13	setLineVector	22
5.2.3.14	setMobile	23
5.2.3.15	setPinned	23
5.2.3.16	setPosition	23
5.2.3.17	setPosition	23
5.2.3.18	setPosition	24
5.2.3.19	setX	24
5.2.3.20	setY	24
5.2.3.21	setZ	24
5.2.3.22	stressField	25
5.2.3.23	stressFieldLocal	25
5.2.4	Field Documentation	26
5.2.4.1	bmag	26
5.2.4.2	bvec	26
5.2.4.3	lvec	26
5.2.4.4	mobile	26
5.2.4.5	pos	26
5.2.4.6	rotationMatrix	27
5.3	DislocationSource Class Reference	27
5.3.1	Detailed Description	30
5.3.2	Constructor & Destructor Documentation	30
5.3.2.1	DislocationSource	30

5.3.2.2	DislocationSource	30
5.3.3	Member Function Documentation	31
5.3.3.1	dipoleNucleationLength	31
5.3.3.2	getBurgers	31
5.3.3.3	getBurgersMag	32
5.3.3.4	getIterationCount	32
5.3.3.5	getLineVector	32
5.3.3.6	getNumIterations	32
5.3.3.7	getPosition	33
5.3.3.8	getPosition	33
5.3.3.9	getTauCritical	33
5.3.3.10	getX	33
5.3.3.11	getY	34
5.3.3.12	getZ	34
5.3.3.13	ifEmitDipole	34
5.3.3.14	incrementIterationCount	34
5.3.3.15	resetIterationCounter	35
5.3.3.16	setBurgers	35
5.3.3.17	setBurgersMagnitude	35
5.3.3.18	setLineVector	35
5.3.3.19	setNumIterations	35
5.3.3.20	setPosition	36
5.3.3.21	setPosition	36
5.3.3.22	setPosition	36
5.3.3.23	setTauCritical	37
5.3.3.24	setX	37
5.3.3.25	setY	37
5.3.3.26	setZ	37
5.3.3.27	stressField	38
5.3.4	Field Documentation	38
5.3.4.1	bmag	38
5.3.4.2	bvec	38
5.3.4.3	countIterations	38
5.3.4.4	lvec	39
5.3.4.5	mobile	39
5.3.4.6	nIterations	39
5.3.4.7	pos	39
5.3.4.8	rotationMatrix	39
5.3.4.9	tauCritical	39
5.4	Matrix33 Class Reference	39

5.4.1	Detailed Description	41
5.4.2	Constructor & Destructor Documentation	41
5.4.2.1	Matrix33	41
5.4.2.2	Matrix33	41
5.4.2.3	Matrix33	42
5.4.2.4	Matrix33	42
5.4.3	Member Function Documentation	42
5.4.3.1	adjugate	42
5.4.3.2	getValue	43
5.4.3.3	operator!	43
5.4.3.4	operator*	44
5.4.3.5	operator*	44
5.4.3.6	operator*	45
5.4.3.7	operator*=	45
5.4.3.8	operator*=	45
5.4.3.9	operator+	46
5.4.3.10	operator+=	46
5.4.3.11	operator-	46
5.4.3.12	operator-=	47
5.4.3.13	operator~	47
5.4.3.14	setValue	48
5.4.3.15	transpose	48
5.4.4	Field Documentation	48
5.4.4.1	x	48
5.5	RotationMatrix Class Reference	49
5.5.1	Detailed Description	50
5.5.2	Constructor & Destructor Documentation	50
5.5.2.1	RotationMatrix	50
5.5.2.2	RotationMatrix	51
5.5.2.3	RotationMatrix	51
5.5.3	Member Function Documentation	51
5.5.3.1	adjugate	51
5.5.3.2	getValue	52
5.5.3.3	operator!	52
5.5.3.4	operator*	53
5.5.3.5	operator*	53
5.5.3.6	operator*	54
5.5.3.7	operator*=	54
5.5.3.8	operator*=	54
5.5.3.9	operator+	55

5.5.3.10	operator+=	55
5.5.3.11	operator-	55
5.5.3.12	operator-=	56
5.5.3.13	operator~	56
5.5.3.14	setValue	57
5.5.3.15	transpose	57
5.5.4	Field Documentation	57
5.5.4.1	x	57
5.6	SlipPlane Class Reference	58
5.6.1	Detailed Description	60
5.6.2	Constructor & Destructor Documentation	60
5.6.2.1	SlipPlane	60
5.6.2.2	SlipPlane	60
5.6.3	Member Function Documentation	61
5.6.3.1	calculateDislocationForces	61
5.6.3.2	calculateDislocationStresses	61
5.6.3.3	calculateRotationMatrix	62
5.6.3.4	calculateTimeIncrement	62
5.6.3.5	calculateVelocities	64
5.6.3.6	distanceFromExtremity	65
5.6.3.7	getAxis	65
5.6.3.8	getDislocation	66
5.6.3.9	getDislocationList	66
5.6.3.10	getDislocationSource	67
5.6.3.11	getDislocationSourceList	67
5.6.3.12	getExtremity	67
5.6.3.13	getNormal	68
5.6.3.14	getNumDislocations	68
5.6.3.15	getNumDislocationSources	68
5.6.3.16	getPosition	69
5.6.3.17	getRotationMatrix	69
5.6.3.18	getSlipPlaneStress_global	69
5.6.3.19	getSlipPlaneStress_local	70
5.6.3.20	moveDislocations	71
5.6.3.21	setDislocationList	71
5.6.3.22	setDislocationSourceList	71
5.6.3.23	setExtremities	72
5.6.3.24	setNormal	72
5.6.3.25	setPosition	72
5.6.3.26	sortDislocations	72

5.6.4	Field Documentation	73
5.6.4.1	dislocationForces	73
5.6.4.2	dislocations	73
5.6.4.3	dislocationSources	73
5.6.4.4	dislocationStresses	73
5.6.4.5	dislocationVelocities	73
5.6.4.6	dt	74
5.6.4.7	extremities	74
5.6.4.8	normalVector	74
5.6.4.9	position	74
5.6.4.10	rotationMatrix	74
5.7	Strain Class Reference	74
5.7.1	Detailed Description	76
5.7.2	Constructor & Destructor Documentation	76
5.7.2.1	Strain	76
5.7.2.2	Strain	77
5.7.2.3	Strain	77
5.7.3	Member Function Documentation	78
5.7.3.1	adjugate	78
5.7.3.2	getPrincipalStrains	78
5.7.3.3	getShearStrains	79
5.7.3.4	getValue	79
5.7.3.5	operator!	79
5.7.3.6	operator*	80
5.7.3.7	operator*	80
5.7.3.8	operator*	81
5.7.3.9	operator*=	81
5.7.3.10	operator*=	81
5.7.3.11	operator+	82
5.7.3.12	operator+=	82
5.7.3.13	operator-	82
5.7.3.14	operator-=	83
5.7.3.15	operator~	83
5.7.3.16	populateMatrix	84
5.7.3.17	rotate	84
5.7.3.18	setValue	84
5.7.3.19	transpose	85
5.7.4	Field Documentation	85
5.7.4.1	principalStrains	85
5.7.4.2	shearStrains	85

5.7.4.3	x	85
5.8	Stress Class Reference	85
5.8.1	Detailed Description	87
5.8.2	Constructor & Destructor Documentation	87
5.8.2.1	Stress	87
5.8.2.2	Stress	88
5.8.2.3	Stress	88
5.8.3	Member Function Documentation	89
5.8.3.1	adjugate	89
5.8.3.2	getPrincipalStresses	89
5.8.3.3	getShearStresses	90
5.8.3.4	getValue	90
5.8.3.5	operator!	90
5.8.3.6	operator*	91
5.8.3.7	operator*	91
5.8.3.8	operator*	92
5.8.3.9	operator*=	92
5.8.3.10	operator*=	92
5.8.3.11	operator+	93
5.8.3.12	operator+=	93
5.8.3.13	operator-	93
5.8.3.14	operator-=	94
5.8.3.15	operator~	94
5.8.3.16	populateMatrix	95
5.8.3.17	rotate	95
5.8.3.18	setValue	95
5.8.3.19	transpose	96
5.8.4	Field Documentation	96
5.8.4.1	principalStresses	96
5.8.4.2	shearStresses	96
5.8.4.3	x	96
5.9	Vector3d Class Reference	96
5.9.1	Detailed Description	97
5.9.2	Constructor & Destructor Documentation	98
5.9.2.1	Vector3d	98
5.9.2.2	Vector3d	98
5.9.2.3	Vector3d	98
5.9.3	Member Function Documentation	98
5.9.3.1	getValue	98
5.9.3.2	getVector	99

5.9.3.3	magnitude	99
5.9.3.4	normalize	100
5.9.3.5	operator*	100
5.9.3.6	operator*	100
5.9.3.7	operator*=	101
5.9.3.8	operator+	101
5.9.3.9	operator+=	101
5.9.3.10	operator-	102
5.9.3.11	operator-=	102
5.9.3.12	operator^	102
5.9.3.13	operator^=	103
5.9.3.14	setValue	103
5.9.3.15	setVector	103
5.9.3.16	sum	103
5.9.4	Field Documentation	104
5.9.4.1	x	104
6	File Documentation	105
6.1	constants.h File Reference	105
6.1.1	Detailed Description	106
6.1.2	Macro Definition Documentation	106
6.1.2.1	PI	106
6.1.2.2	SQRT2	106
6.1.2.3	SQRT3	106
6.1.2.4	SQRT5	106
6.2	defect.cpp File Reference	107
6.2.1	Detailed Description	107
6.3	defect.h File Reference	108
6.3.1	Detailed Description	109
6.4	dislocation.cpp File Reference	110
6.4.1	Detailed Description	110
6.5	dislocation.h File Reference	111
6.5.1	Detailed Description	112
6.6	dislocationDefaults.h File Reference	112
6.6.1	Detailed Description	113
6.6.2	Macro Definition Documentation	114
6.6.2.1	DEFAULT_BURGERS_0	114
6.6.2.2	DEFAULT_BURGERS_1	114
6.6.2.3	DEFAULT_BURGERS_2	114
6.6.2.4	DEFAULT_BURGERS_MAGNITUDE	114

6.6.2.5	DEFAULT_LINEVECTOR_0	114
6.6.2.6	DEFAULT_LINEVECTOR_1	114
6.6.2.7	DEFAULT_LINEVECTOR_2	115
6.6.2.8	DEFAULT_POSITION_0	115
6.6.2.9	DEFAULT_POSITION_1	115
6.6.2.10	DEFAULT_POSITION_2	115
6.7	dislocationSource.cpp File Reference	115
6.7.1	Detailed Description	116
6.8	dislocationSource.h File Reference	117
6.8.1	Detailed Description	118
6.9	dislocationSourceDefaults.h File Reference	118
6.9.1	Detailed Description	119
6.9.2	Macro Definition Documentation	120
6.9.2.1	DEFAULT_NITERATIONS	120
6.9.2.2	DEFAULT_TAU_CRITICAL	120
6.10	main.cpp File Reference	120
6.10.1	Function Documentation	121
6.10.1.1	main	121
6.11	mainpage.dox File Reference	121
6.12	matrix33.cpp File Reference	122
6.12.1	Detailed Description	122
6.13	matrix33.h File Reference	122
6.13.1	Detailed Description	124
6.14	rotationMatrix.cpp File Reference	124
6.14.1	Detailed Description	125
6.15	rotationMatrix.h File Reference	125
6.15.1	Detailed Description	127
6.16	slipPlane.cpp File Reference	128
6.16.1	Detailed Description	128
6.17	slipPlane.h File Reference	129
6.17.1	Detailed Description	130
6.18	slipPlaneDefaults.h File Reference	130
6.18.1	Detailed Description	132
6.18.2	Macro Definition Documentation	132
6.18.2.1	DEFAULT_SLIPPLANE_EXTREMITY1_0	132
6.18.2.2	DEFAULT_SLIPPLANE_EXTREMITY1_1	132
6.18.2.3	DEFAULT_SLIPPLANE_EXTREMITY1_2	132
6.18.2.4	DEFAULT_SLIPPLANE_EXTREMITY2_0	132
6.18.2.5	DEFAULT_SLIPPLANE_EXTREMITY2_1	133
6.18.2.6	DEFAULT_SLIPPLANE_EXTREMITY2_2	133

6.18.2.7	DEFAULT_SLIPPLANE_NORMALVECTOR_0	133
6.18.2.8	DEFAULT_SLIPPLANE_NORMALVECTOR_1	133
6.18.2.9	DEFAULT_SLIPPLANE_NORMALVECTOR_2	133
6.18.2.10	DEFAULT_SLIPPLANE_POSITION_0	133
6.18.2.11	DEFAULT_SLIPPLANE_POSITION_1	133
6.18.2.12	DEFAULT_SLIPPLANE_POSITION_2	133
6.19	strain.cpp File Reference	133
6.19.1	Detailed Description	134
6.20	strain.h File Reference	135
6.20.1	Detailed Description	136
6.21	stress.cpp File Reference	136
6.21.1	Detailed Description	137
6.22	stress.h File Reference	138
6.22.1	Detailed Description	139
6.23	vector3d.cpp File Reference	140
6.23.1	Detailed Description	140
6.24	vector3d.h File Reference	140
6.24.1	Detailed Description	142

Index

142

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	27
Matrix33	39
RotationMatrix	49
Strain	74
Stress	85
SlipPlane	58
Vector3d	96

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	27
Matrix33	Matrix33 class representing a 3x3 square matrix	39
RotationMatrix	RotationMatrix class to represent a rotation matrix	49
SlipPlane	SlipPlane class representing a slip plane in the simulation	58
Strain	Strain class to represent the strain tensor	74
Stress	Stress class to represent the stress tensor	85
Vector3d	Vector3d class representing a single 3-dimensional vector in the simulation	96

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	105
defect.cpp	Definition of member functions of the Defect class	107
defect.h	Definition of the Defect class	108
dislocation.cpp	Definition of constructors and member functions of the Dislocation class	110
dislocation.h	Definition of the Dislocation class	111
dislocationDefaults.h	Definition of certain default values for members of the Dislocation class	112
dislocationSource.cpp	Definition of the member functions of the DislocationSource class	115
dislocationSource.h	Definition of the DislocationSource class	117
dislocationSourceDefaults.h	Definition of certain default values for members of the DislocationSource class	118
main.cpp	120
matrix33.cpp	Definition of the member functions and operators of the Matrix33 class	122
matrix33.h	Definition of the Matrix33 class	122
rotationMatrix.cpp	Definition of the RotationMatrix class member functions	124
rotationMatrix.h	Definition of the RotationMatrix class	125
slipPlane.cpp	Definition of the member functions of the SlipPlane class	128
slipPlane.h	Definition of the SlipPlane class	129
slipPlaneDefaults.h	Definition of certain default values for members of the SlipPlane class	130
strain.cpp	Definition of the member functions if the Strain class	133
strain.h	Definition of the Strain class	135

stress.cpp	
Definition of the member functions if the Stress class	136
stress.h	
Definition of the Stress class	138
vector3d.cpp	
Definition of member functions and operators of the Vector3d class	140
vector3d.h	
Definition of the Vector3d class	140

Chapter 5

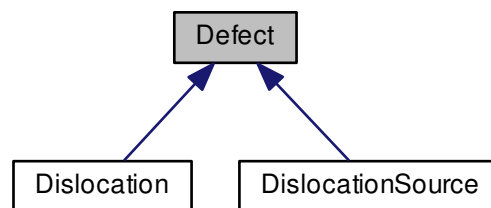
Data Structure Documentation

5.1 Defect Class Reference

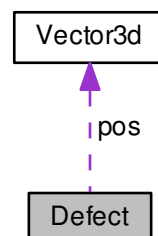
Class [Defect](#) representing a generic defect in a material.

```
#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.
- void [getPosition](#) (double *a) const
Returns the array position in a pre-allocated array.
- [Vector3d](#) [getPosition](#) () const
Returns the position vector of the defect.
- double [getX](#) () const
Returns the X-coordinate of the defect.
- double [getY](#) () const
Returns the Y-coordinate of the defect.
- double [getZ](#) () const
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     for (int i=0; i<3; i++)
47     {
48         this->pos.setValue (i, p[i]);
49     }
50 }
```

5.1.3 Member Function Documentation

5.1.3.1 void Defect::getPosition (double * a) const

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 122 of file defect.cpp.

```
123 {  
124   a = this->pos.getVector ();  
125 }
```

5.1.3.2 Vector3d Defect::getPosition () const

Returns the position vector of the defect.

Returns

The position vector of the defect, in a variable of type [Vector3d](#).

Definition at line 131 of file defect.cpp.

```
132 {  
133   return (this->pos);  
134 }
```

5.1.3.3 double Defect::getX () const

Returns the X-coordinate of the defect.

Returns

X-coordinate of the defect.

Definition at line 140 of file defect.cpp.

```
141 {  
142   return (this->pos.getValue (0));  
143 }
```

5.1.3.4 double Defect::getY () const

Returns the Y-coordinate of the defect.

Returns

Y-coordinate of the defect.

Definition at line 149 of file defect.cpp.

```
150 {  
151   return (this->pos.getValue (1));  
152 }
```


5.1.3.5 double Defect::getZ () const

Returns the Z-coordinate of the defect.

Returns

Z-coordinate of the defect.

Definition at line 158 of file defect.cpp.

```
159 {
160     return (this->pos.getValue (2));
161 }
```

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 59 of file defect.cpp.

```
60 {
61     this->pos.setVector (a);
62 }
```

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 72 of file defect.cpp.

```
73 {
74     this->pos.setValue (0, x);
75     this->pos.setValue (1, y);
76     this->pos.setValue (2, z);
77 }
```

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 84 of file defect.cpp.

```
85 {
86     this->pos = a;
87 }
```

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 93 of file defect.cpp.

```
94 {
95     this->pos.setValue (0, x);
96 }
```

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 102 of file defect.cpp.

```
103 {
104     this->pos.setValue (1, y);
105 }
```

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 111 of file defect.cpp.

```
112 {
113     this->pos.setValue (2, z);
114 }
```

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 135 of file defect.h.

```

136 {
137     // This virtual function returns a zero matrix.
138     // Inheriting classes will have functions implementing this in their own way
139     // They will override this behaviour.
140     Stress s;
141     return (s);
142 }
```

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:

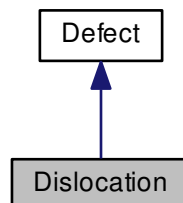
- [defect.h](#)
- [defect.cpp](#)

5.2 Dislocation Class Reference

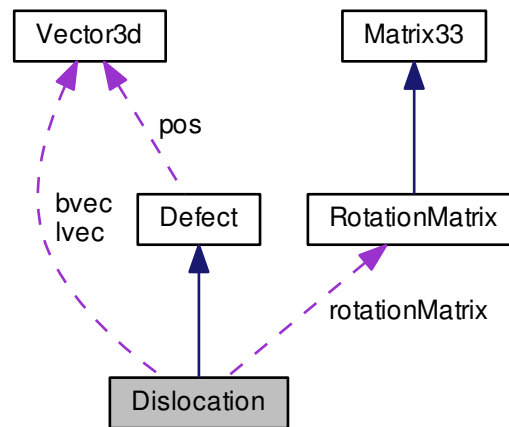
[Dislocation](#) class representing a dislocation in the simulation.

```
#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](#) () const
Gets the Burgers vector of the dislocation.
- [Vector3d](#) [getLineVector](#) () const
Gets the line vector of the dislocation.
- bool [isMobile](#) () const
Returns whether the dislocation is mobile or pinned.
- 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) const
Calculates the stress field due to the dislocation in the local co-ordinate system.
- [Vector3d](#) [forcePeachKoehler](#) ([Stress](#) sigma, double tau_crss) const
Calculate the Peach-Koehler force acting on the dislocation due the stress.
- double [idealTimeIncrement](#) ([Vector3d](#) v0, double minDistance, [Defect](#) d, [Vector3d](#) v1)

- Returns the ideal time increment for the dislocation.*
- 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.
- void [getPosition](#) (double *a) const
Returns the array position in a pre-allocated array.
- [Vector3d](#) [getPosition](#) () const
Returns the position vector of the defect.
- double [getX](#) () const
Returns the X-coordinate of the defect.
- double [getY](#) () const
Returns the Y-coordinate of the defect.
- double [getZ](#) () const
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,
25     DEFAULT_BURGERS_1, DEFAULT_BURGERS_2 ) );
26   this->setLineVector ( Vector3d ( DEFAULT_LINEVECTOR_0,
27     DEFAULT_LINEVECTOR_1, DEFAULT_LINEVECTOR_2 ) );
28   this->bmag = DEFAULT_BURGERS_MAGNITUDE;
29   this->mobile = true;
30   this->calculateRotationMatrix ();
31 }
```

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 118 of file dislocation.cpp.

```
119 {
120   Vector3d *globalSystem = new Vector3d[3];    // Global co-ordinate systems
```

```

121  Vector3d *localSystem = new Vector3d[3];    // Dislocation co-ordinate system
122
123  // Vectors of the global co-ordinate system
124  globalSystem[0] = Vector3d (1.0, 0.0, 0.0);
125  globalSystem[1] = Vector3d (0.0, 1.0, 0.0);
126  globalSystem[2] = Vector3d (0.0, 0.0, 1.0);
127
128  // Vectors of the dislocation co-ordinate system
129  localSystem[0] = bvec.normalize ();
130  localSystem[2] = lvec.normalize ();
131  localSystem[1] = (lvec ^ bvec).normalize ();
132
133  // Calculate rotation matrix
134  this->rotationMatrix = RotationMatrix (globalSystem, localSystem);
135
136  // Release memory
137  delete (globalSystem); globalSystem = NULL;
138  delete (localSystem); localSystem = NULL;
139 }

```

5.2.3.2 Vector3d Dislocation::forcePeachKoehler (Stress *sigma*, double *tau_crss*) const

Calculate the Peach-Koehler force acting on the dislocation due the stress.

This function calculates the Peach-Koehler force in the dislocation due to the stress (expressed in the global co-ordinate system) provided as argument. The force returned is also in the global co-ordinate system. This function checks if the xy component of the stress tensor expressed in the dislocation's local co-ordinate system, is greater than *tau_crss*. If it is, the force is calculated using the Peach-Koehler equation, otherwise, the force on the dislocation is zero.

Parameters

<i>sigma</i>	The stress tensor, expressed in the global co-ordinate system.
<i>tau_crss</i>	Critical Resolved Shear Stress in Pa.

Returns

The Peach-Koehler force on the dislocation, expressed in the global co-ordinate system.

Definition at line 208 of file dislocation.cpp.

```

209 {
210  // Stress in the local co-ordinate system
211  Stress sigmaLocal = sigma.rotate(this->rotationMatrix);
212  Vector3d force;
213
214  // Check for CRSS condition
215  if (sigmaLocal.getValue(0,1) >= tau_crss)
216  {
217      Vector3d force = sigma * ((this->bvec)^(this->lvec));
218  }
219
220  return (force);
221 }

```

5.2.3.3 Vector3d Dislocation::getBurgers () const

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 }

```

5.2.3.4 Vector3d Dislocation::getLineVector () const

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 }
```

5.2.3.5 void Defect::getPosition (double * a) const [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 122 of file defect.cpp.

```
123 {
124     a = this->pos.getVector ();
125 }
```

5.2.3.6 Vector3d Defect::getPosition () const [inherited]

Returns the position vector of the defect.

Returns

The position vector of the defect, in a variable of type [Vector3d](#).

Definition at line 131 of file defect.cpp.

```
132 {
133     return (this->pos);
134 }
```

5.2.3.7 double Defect::getX () const [inherited]

Returns the X-coordinate of the defect.

Returns

X-coordinate of the defect.

Definition at line 140 of file defect.cpp.

```
141 {
142     return (this->pos.getValue (0));
143 }
```


5.2.3.8 `double Defect::getY () const` [inherited]

Returns the Y-coordinate of the defect.

Returns

Y-coordinate of the defect.

Definition at line 149 of file defect.cpp.

```
150 {
151     return (this->pos.getValue (1));
152 }
```

5.2.3.9 `double Defect::getZ () const` [inherited]

Returns the Z-coordinate of the defect.

Returns

Z-coordinate of the defect.

Definition at line 158 of file defect.cpp.

```
159 {
160     return (this->pos.getValue (2));
161 }
```

5.2.3.10 `double Dislocation::idealTimeIncrement (Vector3d v0, double minDistance, Defect d, Vector3d v1)`

Returns the ideal time increment for the dislocation.

A dislocation is not allowed to approach another defect beyond a certain distance, specified by the argument minDistance. This function calculates the ideal time increment for this dislocation to not collide with the defect.

Parameters

<i>v0</i>	Velocity of the dislocation.
<i>minDistance</i>	Minimum distance of approach to the defect.
<i>d</i>	The defect for which the present dislocation's time increment is to be calculated.
<i>v1</i>	Velocity of the other defect.

Returns

The ideal time increment for this dislocation.

Definition at line 232 of file dislocation.cpp.

```
233 {
234     double norm_v0 = v0.magnitude();
235     if (norm_v0 == 0.0)
236     {
237         // This dislocation is not moving
238         return (1000.0);
239     }
240
241     // Positions
242     Vector3d p0 = this->getPosition();
243     Vector3d p1 = d.getPosition();
244     Vector3d p01 = p1 - p0;
245     double norm_p01 = p01.magnitude();
246 }
```

```

247  if (norm_p01 == 0.0)
248  {
249      // The dislocation is lying on top of the obstacle - so it should not move
250      return (0.0);
251  }
252  else
253  {
254      // Find out if the dislocation is approaching the defect or not
255
256      // Velocities
257      Vector3d v01 = v1 - v0;
258      double norm_v01 = v01.magnitude();
259      double dotProduct = v01 * p01;
260      double cosine = dotProduct/(norm_v01 * norm_p01);
261      if (cosine < 0.0)
262      {
263          // The dislocation is approaching the other defect
264          return ( (norm_p01 - minDistance)/norm_v01 );
265      }
266      else
267      {
268          // They are diverging
269          // So any time increment will do
270          return (1000.0);
271      }
272  }
273 }

```

5.2.3.11 bool Dislocation::isMobile () const

Returns whether the dislocation is mobile or pinned.

Returns

Returns true if the dislocation is mobile, false if pinned.

Definition at line 108 of file dislocation.cpp.

```

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

```

5.2.3.12 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 }

```

5.2.3.13 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 }
```

5.2.3.14 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 }
```

5.2.3.15 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 }
```

5.2.3.16 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 59 of file defect.cpp.

```
60 {
61     this->pos.setVector (a);
62 }
```

5.2.3.17 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 72 of file defect.cpp.

```
73 {  
74     this->pos.setValue (0, x);  
75     this->pos.setValue (1, y);  
76     this->pos.setValue (2, z);  
77 }
```

5.2.3.18 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 84 of file defect.cpp.

```
85 {  
86     this->pos = a;  
87 }
```

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

Sets the X-coordinate of the defect.

Parameters

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

Definition at line 93 of file defect.cpp.

```
94 {  
95     this->pos.setValue (0, x);  
96 }
```

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

Sets the Y-coordinate of the defect.

Parameters

y	Y-coordinate of the defect.
---	-----------------------------

Definition at line 102 of file defect.cpp.

```
103 {  
104     this->pos.setValue (1, y);  
105 }
```

5.2.3.21 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 111 of file defect.cpp.

```
112 {
113     this->pos.setValue (2, z);
114 }
```

5.2.3.22 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 150 of file dislocation.cpp.

```
151 {
152     double principalStresses[3];
153     double shearStresses[3];
154     Vector3d r; // Vector joining the present dislocation to the point p
155
156     r = p - this->pos; // Still in global coordinate system
157     Vector3d rLocal = this->rotationMatrix * r; // Rotated to local co-ordinate
158         system
159
160     // Calculate the stress field in the local co-ordinate system
161     Stress sLocal = this->stressFieldLocal (rLocal, mu, nu);
162
163     // Calculate the stress field in the global co-ordinate system
164     //Stress sGlobal = (this->rotationMatrix) * sLocal * (this->rotationMatrix.transpose());
165     Stress sGlobal = sLocal.rotate (this->rotationMatrix);
166
167     return (sGlobal);
168 }
```

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

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 177 of file dislocation.cpp.

```

178 {
179     double D = ( mu * this->bmag ) / ( 2.0 * PI * ( 1.0 - nu ) );    // Constant for all components of
        the stress tensor
180
181     double x, y, denominator;    // Terms that appear repeatedly in the stress tensor
182
183     x = p.getValue (0);
184     y = p.getValue (1);
185     denominator = pow ( ((x*x) + (y*y)), 2);
186
187     double principalStresses[3], shearStresses[3];
188
189     principalStresses[0] = -1.0 * D * y * ( (3.0*x*x) + (y*y) ) / denominator;
190     principalStresses[1] = D * y * ( (x*x) - (y*y) ) / denominator;
191     principalStresses[2] = nu * ( principalStresses[0] + principalStresses[1] );
192
193     shearStresses[0] = D * x * ( (x*x) - (y*y) ) / denominator;
194     shearStresses[1] = 0.0;
195     shearStresses[2] = 0.0;
196
197     return (Stress(principalStresses, shearStresses));
198 }
```

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 of 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:

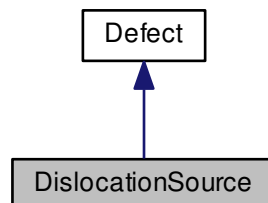
- [dislocation.h](#)
- [dislocation.cpp](#)

5.3 DislocationSource Class Reference

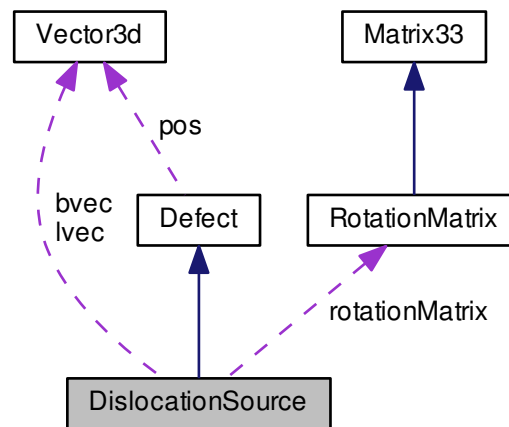
[DislocationSource](#) class representing a source of dislocations in the simulation.

```
#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](#) () const
Returns the Burgers vector of the dislocations in the dipole.
- [Vector3d](#) [getLineVector](#) () const
Returns the line vector of the dislocations in the dipole.
- double [getBurgersMag](#) () const
Returns the magnitude of the Burgers vector.
- double [getTauCritical](#) () const
Returns the critical shear stress value for dipole emission.
- int [getNumIterations](#) () const
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](#) () const

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

 - double `dipoleNucleationLength` (double tau, double mu, double nu) const

The nucleation length of the dipole.
- void `incrementIterationCount` ()

Increments the variable countIterations by 1.
- bool `ifEmitDipole` () const

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.
- void `getPosition` (double *a) const

Returns the array position in a pre-allocated array.
- Vector3d `getPosition` () const

Returns the position vector of the defect.
- double `getX` () const

Returns the X-coordinate of the defect.
- double `getY` () const

Returns the Y-coordinate of the defect.
- double `getZ` () const

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*) const

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 () const

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 () const

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 () const

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 () const

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 () const

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 void Defect::getPosition (double * a) const [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 122 of file defect.cpp.

```
123 {
124     a = this->pos.getVector ();
125 }
```

5.3.3.8 Vector3d Defect::getPosition () const [inherited]

Returns the position vector of the defect.

Returns

The position vector of the defect, in a variable of type [Vector3d](#).

Definition at line 131 of file defect.cpp.

```
132 {
133     return (this->pos);
134 }
```

5.3.3.9 double DislocationSource::getTauCritical () const

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 () const [inherited]

Returns the X-coordinate of the defect.

Returns

X-coordinate of the defect.

Definition at line 140 of file defect.cpp.

```
141 {
142     return (this->pos.getValue (0));
143 }
```

5.3.3.11 double Defect::getY () const [inherited]

Returns the Y-coordinate of the defect.

Returns

Y-coordinate of the defect.

Definition at line 149 of file defect.cpp.

```
150 {  
151     return (this->pos.getValue (1));  
152 }
```

5.3.3.12 double Defect::getZ () const [inherited]

Returns the Z-coordinate of the defect.

Returns

Z-coordinate of the defect.

Definition at line 158 of file defect.cpp.

```
159 {  
160     return (this->pos.getValue (2));  
161 }
```

5.3.3.13 bool DislocationSource::ifEmitDipole () const

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 59 of file defect.cpp.

```

60 {
61     this->pos.setVector (a);
62 }
```

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 72 of file defect.cpp.

```

73 {
74     this->pos.setValue (0, x);
75     this->pos.setValue (1, y);
76     this->pos.setValue (2, z);
77 }
```

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 84 of file defect.cpp.


```

85 {
86     this->pos = a;
87 }

```

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 93 of file defect.cpp.

```

94 {
95     this->pos.setValue (0, x);
96 }

```

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 102 of file defect.cpp.

```

103 {
104     this->pos.setValue (1, y);
105 }

```

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 111 of file defect.cpp.

```

112 {
113     this->pos.setValue (2, z);
114 }

```

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 135 of file defect.h.

```

136 {
137     // This virtual function returns a zero matrix.
138     // Inheriting classes will have functions implementing this in their own way
139     // They will override this behaviour.
140     Stress s;
141     return (s);
142 }

```

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:

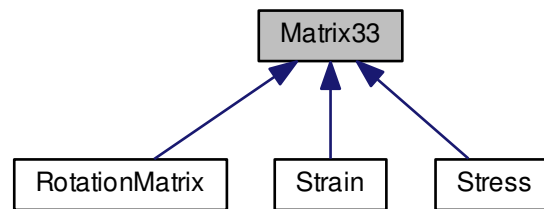
- [dislocationSource.h](#)
- [dislocationSource.cpp](#)

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) const
Returns the value of the element located by the row and column indices provided.
- [Matrix33](#) [adjugate](#) () const
Returns the adjugate matrix of the present matrix.
- [Matrix33](#) [transpose](#) () const
Returns the transpose 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.

- 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     for (i=0; i<3; i++)
57     {
58         for (j=0; j<3; j++)
59         {
60             this->x[i][j] = a.getValue(i) * a.getValue(j);
61         }
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     for (i=0; i<3; i++)
76     {
77         for (j=0; j<3; j++)
78         {
79             this->x[i][j] = a.getValue(i) * b.getValue(j);
80         }
81     }
82 }
83 }

```

5.4.3 Member Function Documentation

5.4.3.1 Matrix33 Matrix33::adjugate () const

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.setValue(0, 0, ((this->x[1][1]*this->x[2][2]) - (this->x[1][2]*this->x[2][1])));
134     adj.setValue(0, 1, ((this->x[1][2]*this->x[2][0]) - (this->x[1][0]*this->x[2][2])));
135     adj.setValue(0, 2, ((this->x[1][0]*this->x[2][1]) - (this->x[1][1]*this->x[2][0])));
136
137     adj.setValue(1, 0, ((this->x[2][1]*this->x[0][2]) - (this->x[0][1]*this->x[2][2])));
138     adj.setValue(1, 1, ((this->x[2][2]*this->x[0][0]) - (this->x[2][0]*this->x[0][2])));
139     adj.setValue(1, 2, ((this->x[2][0]*this->x[0][1]) - (this->x[2][1]*this->x[0][0])));
140
141     adj.setValue(2, 0, ((this->x[0][1]*this->x[1][2]) - (this->x[0][2]*this->x[1][1])));
142     adj.setValue(2, 1, ((this->x[0][2]*this->x[1][0]) - (this->x[0][0]*this->x[1][2])));
143     adj.setValue(2, 2, ((this->x[0][0]*this->x[1][1]) - (this->x[1][0]*this->x[0][1])));
144
145     return (adj);
146 }
```

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

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 374 of file matrix33.cpp.

```

375 {
376     Matrix33 r;    // Result matrix
377
378     double determinant = ~(*this);
379
380     if (determinant == 0.0)
381     {
382         // The matrix is non-invertible
383         return (r);    // Zero matrix
384     }
385
386     // If we are still here, the matrix is invertible
387
388     // Transpose
389     Matrix33 tr = this->transpose();
390
391     // Find Adjugate matrix
392     Matrix33 adj = tr.adjugate();
393
394     // Calculate the inverse by dividing the adjugate matrix by the determinant
395     r = adj * (1.0/determinant);
396
397     return (r);
398 }

```

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 253 of file matrix33.cpp.

```

254 {
255     int i, j;
256     Matrix33 r;
257
258     for (i=0; i<3; i++)
259     {
260         for (j=0; j<3; j++)
261         {
262             r.setValue(i, j, (this->x[i][j] * p));
263         }
264     }
265
266     return (r);
267 }

```

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 291 of file matrix33.cpp.

```

292 {
293     int i, j, k;
294     Matrix33 r;
295     double s;
296
297     for (i=0; i<3; i++)
298     {

```



```

299         for (j=0; j<3; j++)
300         {
301             s = 0.0;
302             for (k=0; k<3; k++)
303             {
304                 s += this->x[i][k] * p.getValue(k, j);
305             }
306             r.setValue (i, j, s);
307         }
308     }
309
310     return (r);
311 }

```

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 333 of file matrix33.cpp.

```

334 {
335     Vector3d r(0.0, 0.0, 0.0);
336     double s;
337     int i, j;
338
339     for (i=0; i<3; i++)
340     {
341         s = 0.0;
342         for (j=0; j<3; j++)
343         {
344             s += this->x[i][j] * v.getValue(j);
345         }
346         r.setValue (i, s);
347     }
348
349     return (r);
350 }

```

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 273 of file matrix33.cpp.

```

274 {
275     int i, j;
276
277     for (i=0; i<3; i++)
278     {
279         for (j=0; j<3; j++)
280         {
281             this->x[i][j] *= p;
282         }
283     }
284 }

```

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 317 of file matrix33.cpp.

```

318 {
319     Matrix33* r = new Matrix33;
320
321     *r = (*this) * p;
322     *this = *r;
323
324     delete(r);
325     r = NULL;
326 }

```

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 175 of file matrix33.cpp.

```

176 {
177     int i, j;
178     Matrix33 r;
179
180     for (i=0; i<3; i++)
181     {
182         for (j=0; j<3; j++)
183         {
184             r.setValue(i, j, (this->x[i][j] + p.getValue(i, j)));
185         }
186     }
187
188     return (r);
189 }

```

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 195 of file matrix33.cpp.

```

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

```

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 214 of file matrix33.cpp.

```

215 {
216     int i, j;
217     Matrix33 r;
218
219     for (i=0; i<3; i++)
220     {
221         for (j=0; j<3; j++)
222         {
223             r.setValue(i, j, (this->x[i][j] - p.getValue(i, j)));
224         }
225     }
226
227     return (r);
228 }
```

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 234 of file matrix33.cpp.

```

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

5.4.3.13 double Matrix33::operator~() const

Determinant.

Calculates the determinant of the current matrix.

Returns

Returns the determinant of the current matrix.

Definition at line 358 of file matrix33.cpp.

```

359 {
360     double d = 0.0;
361
362     d += this->x[0][0] * ( (this->x[1][1]*this->x[2][2]) - (this->x[2][1]*this->
363         x[1][2]) );
364     d += this->x[0][1] * ( (this->x[1][2]*this->x[2][0]) - (this->x[1][0]*this->
365         x[2][2]) );
366     d += this->x[0][2] * ( (this->x[1][0]*this->x[2][1]) - (this->x[2][0]*this->
367         x[1][1]) );
368
369     return (d);
370 }
```

5.4.3.14 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 }
```

5.4.3.15 Matrix33 Matrix33::transpose () const

Returns the transpose of the present matrix.

The transpose of a matrix is another matrix having rows identical to the columns of the present matrix, and vice-versa.

Returns

The transpose of the present matrix.

Definition at line 153 of file matrix33.cpp.

```

154 {
155     Matrix33 tr;
156     int i, j;
157     for (i=0; i<3; i++)
158     {
159         for (j=0; j<3; j++)
160         {
161             tr.setValue (i, j, this->x[j][i]);
162         }
163     }
164     return (tr);
165 }
166 }
```

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:

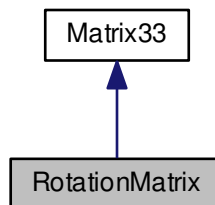
- [matrix33.h](#)
- [matrix33.cpp](#)

5.5 RotationMatrix Class Reference

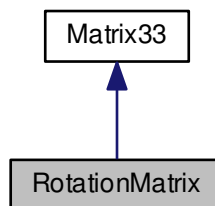
[RotationMatrix](#) class to represent a rotation matrix.

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

Inheritance diagram for RotationMatrix:



Collaboration diagram for RotationMatrix:



Public Member Functions

- [RotationMatrix](#) ()
Default constructor.
- [RotationMatrix](#) ([Matrix33](#) m)
Constructor specifying the matrix.
- [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) const
Returns the value of the element located by the row and column indices provided.
- [Matrix33](#) [adjugate](#) () const
Returns the adjugate matrix of the present matrix.
- [Matrix33](#) [transpose](#) () const
Returns the transpose 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.
- 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 16 of file [rotationMatrix.cpp](#).

```

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

```

```

27         }
28     }
29 }
30 }

```

5.5.2.2 RotationMatrix::RotationMatrix (Matrix33 m)

Constructor specifying the matrix.

The rotation matrix is provided as the matrix m.

Parameters

<i>m</i>	The matrix m which is equal to the rotation matrix.
----------	---

Definition at line 37 of file rotationMatrix.cpp.

```

38 {
39     int i, j;
40
41     for (i=0; i<3; i++)
42     {
43         for (j=0; j<3; j++)
44         {
45             this->setValue (i, j, (m.getValue(i,j)));
46         }
47     }
48 }

```

5.5.2.3 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 56 of file rotationMatrix.cpp.

```

57 {
58     int i, j;
59
60     for ( i=0; i<3; i++ ) {
61         for ( j=0; j<3; j++ ) {
62             this->setValue ( i, j, primed[i]*unPrimed[j] );
63         }
64     }
65 }

```

5.5.3 Member Function Documentation

5.5.3.1 Matrix33 Matrix33::adjugate () const [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.setValue(0, 0, ((this->x[1][1]*this->x[2][2]) - (this->x[1][2]*this->x[2][1])));
134     adj.setValue(0, 1, ((this->x[1][2]*this->x[2][0]) - (this->x[1][0]*this->x[2][2])));
135     adj.setValue(0, 2, ((this->x[1][0]*this->x[2][1]) - (this->x[1][1]*this->x[2][0])));
136
137     adj.setValue(1, 0, ((this->x[2][1]*this->x[0][2]) - (this->x[0][1]*this->x[2][2])));
138     adj.setValue(1, 1, ((this->x[2][2]*this->x[0][0]) - (this->x[2][0]*this->x[0][2])));
139     adj.setValue(1, 2, ((this->x[2][0]*this->x[0][1]) - (this->x[2][1]*this->x[0][0])));
140
141     adj.setValue(2, 0, ((this->x[0][1]*this->x[1][2]) - (this->x[0][2]*this->x[1][1])));
142     adj.setValue(2, 1, ((this->x[0][2]*this->x[1][0]) - (this->x[0][0]*this->x[1][2])));
143     adj.setValue(2, 2, ((this->x[0][0]*this->x[1][1]) - (this->x[1][0]*this->x[0][1])));
144
145     return (adj);
146 }
```

5.5.3.2 double Matrix33::getValue (int row, int column) const [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 374 of file matrix33.cpp.


```

375 {
376     Matrix33 r;    // Result matrix
377
378     double determinant = ~(*this);
379
380     if (determinant == 0.0)
381     {
382         // The matrix is non-invertible
383         return (r);    // Zero matrix
384     }
385
386     // If we are still here, the matrix is invertible
387
388     // Transpose
389     Matrix33 tr = this->transpose();
390
391     // Find Adjugate matrix
392     Matrix33 adj = tr.adjugate();
393
394     // Calculate the inverse by dividing the adjugate matrix by the determinant
395     r = adj * (1.0/determinant);
396
397     return (r);
398 }

```

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 253 of file matrix33.cpp.

```

254 {
255     int i, j;
256     Matrix33 r;
257
258     for (i=0; i<3; i++)
259     {
260         for (j=0; j<3; j++)
261         {
262             r.setValue(i, j, (this->x[i][j] * p));
263         }
264     }
265
266     return (r);
267 }

```

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 291 of file matrix33.cpp.

```

292 {
293     int i, j, k;
294     Matrix33 r;
295     double s;
296
297     for (i=0; i<3; i++)
298     {

```

```

299     for (j=0; j<3; j++)
300     {
301         s = 0.0;
302         for (k=0; k<3; k++)
303         {
304             s += this->x[i][k] * p.getValue(k,j);
305         }
306         r.setValue (i, j, s);
307     }
308 }
309
310 return (r);
311 }

```

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 333 of file matrix33.cpp.

```

334 {
335     Vector3d r(0.0, 0.0, 0.0);
336     double s;
337     int i, j;
338
339     for (i=0; i<3; i++)
340     {
341         s = 0.0;
342         for (j=0; j<3; j++)
343         {
344             s += this->x[i][j] * v.getValue(j);
345         }
346         r.setValue (i, s);
347     }
348
349     return (r);
350 }

```

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 273 of file matrix33.cpp.

```

274 {
275     int i, j;
276
277     for (i=0; i<3; i++)
278     {
279         for (j=0; j<3; j++)
280         {
281             this->x[i][j] *= p;
282         }
283     }
284 }

```

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 317 of file matrix33.cpp.

```

318 {
319     Matrix33* r = new Matrix33;
320
321     *r = (*this) * p;
322     *this = *r;
323
324     delete(r);
325     r = NULL;
326 }

```

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 175 of file matrix33.cpp.

```

176 {
177     int i, j;
178     Matrix33 r;
179
180     for (i=0; i<3; i++)
181     {
182         for (j=0; j<3; j++)
183         {
184             r.setValue(i, j, (this->x[i][j] + p.getValue(i, j)));
185         }
186     }
187
188     return (r);
189 }

```

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 195 of file matrix33.cpp.

```

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

```

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 214 of file matrix33.cpp.

```

215 {
216     int i, j;
217     Matrix33 r;
218
219     for (i=0; i<3; i++)
220     {
221         for (j=0; j<3; j++)
222         {
223             r.setValue(i, j, (this->x[i][j] - p.getValue(i, j)));
224         }
225     }
226
227     return (r);
228 }
```

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 234 of file matrix33.cpp.

```

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

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

Determinant.

Calculates the determinant of the current matrix.

Returns

Returns the determinant of the current matrix.

Definition at line 358 of file matrix33.cpp.

```

359 {
360     double d = 0.0;
361
362     d += this->x[0][0] * ( (this->x[1][1]*this->x[2][2]) - (this->x[2][1]*this->
x[1][2]) );
363     d += this->x[0][1] * ( (this->x[1][2]*this->x[2][0]) - (this->x[1][0]*this->
x[2][2]) );
364     d += this->x[0][2] * ( (this->x[1][0]*this->x[2][1]) - (this->x[2][0]*this->
x[1][1]) );
365
366     return (d);
367 }
```

5.5.3.14 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 }
```

5.5.3.15 Matrix33 Matrix33::transpose () const [inherited]

Returns the transpose of the present matrix.

The transpose of a matrix is another matrix having rows identical to the columns of the present matrix, and vice-versa.

Returns

The transpose of the present matrix.

Definition at line 153 of file matrix33.cpp.

```

154 {
155     Matrix33 tr;
156     int i, j;
157     for (i=0; i<3; i++)
158     {
159         for (j=0; j<3; j++)
160         {
161             tr.setValue (i, j, this->x[j][i]);
162         }
163     }
164     return (tr);
165 }
166 }
```

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:

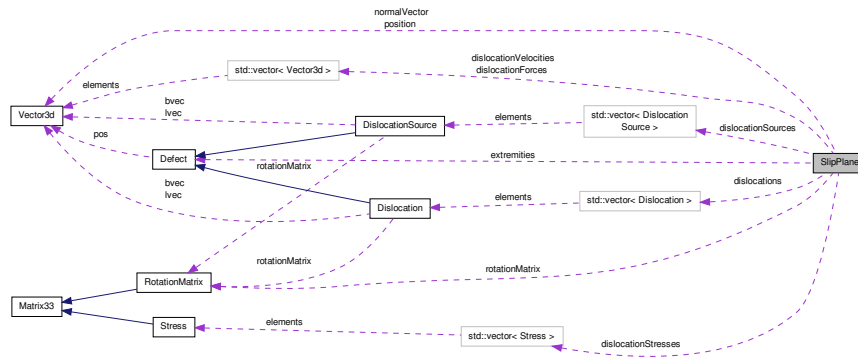
- [rotationMatrix.h](#)
- [rotationMatrix.cpp](#)

5.6 SlipPlane Class Reference

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

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

Collaboration diagram for SlipPlane:



Public Member Functions

- [SlipPlane](#) ()
Default constructor.
- [SlipPlane](#) ([Vector3d](#) *ends, [Vector3d](#) normal, [Vector3d](#) pos, [std::vector](#)< [Dislocation](#) > dislocationList, [std::vector](#)< [DislocationSource](#) > dislocationSourceList)
Constructor that specifies all members explicitly.
- void [setExtremities](#) ([Vector3d](#) *ends)
Set the extremities of the slip plane.
- void [setNormal](#) ([Vector3d](#) normal)
Set the normal vector of the slip plane.
- void [setPosition](#) ([Vector3d](#) pos)
Set the position of the slip plane.
- void [setDislocationList](#) ([std::vector](#)< [Dislocation](#) > dislocationList)
Set the list of dislocations of the slip plane.
- void [setDislocationSourceList](#) ([std::vector](#)< [DislocationSource](#) > dislocationSourceList)
Set the list of dislocation sources on the slip plane.
- [Vector3d](#) [getExtremity](#) (int i) const
Get the position vector of the extremity whose index is provided as argument.
- [Vector3d](#) [getNormal](#) () const
Get the normal vector of the slip plane.
- [Vector3d](#) [getPosition](#) () const
Get the position vector of the slip plane.
- bool [getDislocation](#) (int i, [Dislocation](#) *d) const
Get the dislocation on the slip plane indicated by the index provided as argument.
- [std::vector](#)< [Dislocation](#) > [getDislocationList](#) () const
Get the entire vector container which holds the dislocations lying on this slip plane.
- int [getNumDislocations](#) () const
Get the number of dislocations.
- bool [getDislocationSource](#) (int i, [DislocationSource](#) *dSource) const
Get the dislocation source on the slip plane indicated by the index provided as argument.

- `int getNumDislocationSources () const`
Get the number of dislocation sources.
- `std::vector< DislocationSource > getDislocationSourceList () const`
Get the entire vector container which holds the dislocation sources lying on this slip plane.
- `RotationMatrix getRotationMatrix () const`
Get the rotation matrix for this slip plane.
- `Vector3d getAxis (int i) const`
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.
- `void calculateDislocationStresses (Stress appliedStress, double mu, double nu)`
Calculates the total stress field experienced by each dislocation and stored it in the STL vector container dislocationStresses.
- `void calculateDislocationForces (double tau_crss)`
This function populates the STL vector container dislocationForces with the Peach-Koehler force experienced by each dislocation.
- `void calculateVelocities (double B)`
Calculates the velocities of dislocations and stores them in the std::vector container velocities.
- `void calculateTimeIncrement (double minDistance, double minDt)`
Calculate the time increment based on the velocities of the dislocations.
- `void moveDislocations ()`
Displaces the dislocations according to their velocities and the time increment.
- `double distanceFromExtremity (Vector3d pos, int n)`
The distance of the point pos from the n^{th} extremity is returned.
- `void sortDislocations ()`
Sorts the dislocations present on the slip plane in the ascending order of distance from the first extremity.
- `std::vector< Stress > getSlipPlaneStress_global (std::vector< Vector3d > points, Stress appliedStress, double mu, double nu)`
Returns a vector containing the stress values at different points along a slip plane.
- `std::vector< Stress > getSlipPlaneStress_local (std::vector< Vector3d > points, Stress appliedStress, double mu, double nu)`
Returns a vector containing the stress values at different points along a slip plane.

Protected Attributes

- `Defect 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< Stress > dislocationStresses`
STL vector container with the stress fields of dislocations.
- `std::vector< Vector3d > dislocationForces`
The Peach-Koehler force experienced by each dislocation.
- `std::vector< Vector3d > dislocationVelocities`
STL vector container with dislocation velocities.
- `std::vector< DislocationSource > dislocationSources`

STL vector container with dislocation sources.

- double `dt`

Time increment for the slip plane.

- [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 29 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 specified in the file [slipPlaneDefaults.h](#).

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

```
18 {
19     // Initialize the default variables.
20     Vector3d pos(DEFAULT_SLIPPLANE_POSITION_0,
21                 DEFAULT_SLIPPLANE_POSITION_1,
22                 DEFAULT_SLIPPLANE_POSITION_2);
23     Vector3d normal(DEFAULT_SLIPPLANE_NORMALVECTOR_0,
24                     DEFAULT_SLIPPLANE_NORMALVECTOR_1,
25                     DEFAULT_SLIPPLANE_NORMALVECTOR_2);
26     Vector3d ends[2];
27     ends[0] = Vector3d(DEFAULT_SLIPPLANE_EXTREMITY1_0,
28                        DEFAULT_SLIPPLANE_EXTREMITY1_1,
29                        DEFAULT_SLIPPLANE_EXTREMITY1_2);
30     ends[1] = Vector3d(DEFAULT_SLIPPLANE_EXTREMITY2_0,
31                        DEFAULT_SLIPPLANE_EXTREMITY2_1,
32                        DEFAULT_SLIPPLANE_EXTREMITY2_2);
33     std::vector<Dislocation> dislocationList(1, Dislocation());
34     std::vector<DislocationSource> dislocationSourceList(1, DislocationSource());
35
36     *this = SlipPlane(ends, normal, pos, dislocationList, dislocationSourceList);
37 }
```

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

Definition at line 48 of file slipPlane.cpp.

```

49 {
50     this->setExtremities (ends);
51     this->setNormal (normal);
52     this->setPosition (pos);
53     this->setDislocationList (dislocationList);
54     this->setDislocationSourceList (dislocationSourceList);
55
56     // Fill the vectors and stresses with zero vectors and stresses
57     int nDisl = this->getNumDislocations ();
58     this->dislocationStresses.resize(nDisl, Stress ());
59     this->dislocationVelocities.resize(nDisl, Vector3d());
60     this->dislocationForces.resize(nDisl, Vector3d());
61
62     // Time increment
63     this->dt = 0;
64
65     this->calculateRotationMatrix ();
66 }
```

5.6.3 Member Function Documentation

5.6.3.1 void SlipPlane::calculateDislocationForces (double *tau_crss*)

This function populates the STL vector container `dislocationForces` with the Peach-Koehler force experienced by each dislocation.

This function calculates the Peach-Koehler force experienced by each dislocation using the function [Dislocation::forcePeachKoehler](#) and the STL vector [SlipPlane::dislocationStresses](#). The argument `tau_crss` is the Critical Resolved Shear [Stress](#) in Pa.

Parameters

<i>tau_crss</i>	Critical Resolved Shear Stress in Pa.
-----------------	---

Definition at line 345 of file slipPlane.cpp.

```

346 {
347     std::vector<Dislocation>::iterator d; // Iterator for dislocations
348     std::vector<Vector3d>::iterator f;    // Iterator for forces
349     std::vector<Stress>::iterator s;      // Iterator for stresses
350
351     s = this->dislocationStresses.begin();
352     f = this->dislocationForces.begin();
353
354     for (d = this->dislocations.begin(); d!=this->dislocations.end(); d++)
355     {
356         *f = d->forcePeachKoehler (*s, tau_crss);
357         s++;
358         f++;
359     }
360 }
```

5.6.3.2 void SlipPlane::calculateDislocationStresses (Stress *appliedStress*, double *mu*, double *nu*)

Calculates the total stress field experienced by each dislocation and stored it in the STL vector container `dislocationStresses`.

The total stress field is calculated as a superposition of the applied stress field and the stress fields experienced by each dislocation due to every other dislocation in the simulation.

Parameters

<i>appliedStress</i>	The stress applied externally.
<i>mu</i>	Shear modulus of the material.
<i>nu</i>	Poisson's ratio.

Definition at line 313 of file slipPlane.cpp.

```

314 {
315     std::vector<Dislocation>::iterator d1; // Iterator for each dislocation
316     std::vector<Dislocation>::iterator d2; // Nested iterator
317     std::vector<Stress>::iterator s;      // Iterator for the Stress
318
319     Vector3d p;                          // Position vector
320
321     s = this->dislocationStresses.begin();
322     for (d1=this->dislocations.begin(); d1!=this->dislocations.end(); d1++)
323     {
324         *s = appliedStress;
325         p = d1->getPosition();
326         for (d2 = this->dislocations.begin(); d2!=this->dislocations.end(); d2++)
327         {
328             if (d1==d2)
329             {
330                 continue;
331             }
332             else
333             {
334                 *s = *s + d2->stressField(p, mu, nu);
335             }
336         }
337     }
338 }
```

5.6.3.3 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 280 of file slipPlane.cpp.

```

281 {
282     Vector3d *unPrimed = new Vector3d[3]; // Old system (global)
283     Vector3d *primed   = new Vector3d[3]; // New system (local)
284
285     int i, j;
286
287     // Prepare the global and local systems
288     for (i=0; i<3; i++)
289     {
290         for (j=0; j<3; j++)
291         {
292             unPrimed[i].setValue(j, (double) (i==j));
293         }
294         primed[i] = this->getAxis(i);
295     }
296
297     // Calculate the rotationMatrix
298     this->rotationMatrix = RotationMatrix(unPrimed, primed);
299
300     // Free memory
301     delete(unPrimed);    unPrimed = NULL;
302     delete(primed);      primed = NULL;
303
304 }
```

5.6.3.4 void SlipPlane::calculateTimeIncrement (double minDistance, double minDt)

Calculate the time increment based on the velocities of the dislocations.

In order to avoid the collision of dislocations with similar sign of Burgers vector, it is important to specify a minimum distance of approach between dislocations. When a dislocation reaches this limit, it is pinned. The velocities of the dislocations all being different, a time increment needs to be evaluated, which will limit the distance traveled by the dislocations in a given iteration.

Parameters

<i>minDistance</i>	Minimum distance of approach between dislocations having Burgers vectors of the same sign.
<i>minDt</i>	The smallest time step permissible. Dislocations having time steps smaller than this are made immobile for the present iteration.

Definition at line 416 of file slipPlane.cpp.

```

417 {
418     // Get the number of dislocations
419     int nDisl = this->dislocations.size();
420
421     // Vector of time increments
422     std::vector<double> timeIncrement(nDisl, 1000.0);
423
424     // Position vectors
425     Vector3d p0, p1;
426     double norm_p0l;
427
428     // Velocity vectors
429     Vector3d v0, v1;
430     double norm_v0l;
431
432     int i;          // Counter for the loop
433     double t1, t2;
434     double dtMin;   // Minimum time increment
435
436     // For the first dislocation, the time increment has to be calculated
437     // for approach to both a dislocation and the slip plane extremity.
438     // Time for slip plane extremity
439     t1 = this->dislocations[0].idealTimeIncrement(this->
dislocationVelocities[0],
440
441                                     minDistance,
442                                     this->extremities[0],
443                                     Vector3d(0.0, 0.0, 0.0));
444     t2 = this->dislocations[0].idealTimeIncrement(this->
dislocationVelocities[0],
445
446                                     minDistance,
447                                     this->dislocations[1],
448                                     this->dislocationVelocities[1]);
449
450     // Choose the smaller of the two
451     timeIncrement[0] = t1 < t2 ? t1:t2;
452     if (timeIncrement[0] < minDt)
453     {
454         // This dislocation should not move in this iteration because it might collide with the next defect
455         timeIncrement[0] = minDt;
456         this->dislocationVelocities[0] = Vector3d(0.0, 0.0, 0.0);
457
458         // The other defect is a slip plane extremity
459         // This dislocation will not move any more
460         this->dislocations[0].setPinned();
461     }
462
463     for (i=1; i<(nDisl-1); i++)
464     {
465         t1 = this->dislocations[i].idealTimeIncrement(this->
dislocationVelocities[i],
466
467                                     minDistance,
468                                     this->dislocations[i-1],
469                                     this->dislocationVelocities[i-1])
470
471         ;
472         t2 = this->dislocations[i].idealTimeIncrement(this->
dislocationVelocities[i],
473
474                                     minDistance,
475                                     this->dislocations[i+1],
476                                     this->dislocationVelocities[i+1])
477
478         ;
479         timeIncrement[i] = t1 < t2 ? t1:t2;
480
481         if (timeIncrement[i] < minDt)
482         {
483             // This dislocation should not move in this iteration because it might collide with the next
484             defect
485             timeIncrement[i] = minDt;
486             this->dislocationVelocities[i] = Vector3d(0.0, 0.0, 0.0);
487         }
488     }
489 }

```

```

478     }
479
480     // For the last dislocation, the time increment has to be calculated
481     // for approach to both a dislocation and the slip plane extremity.
482     // Time for slip plane extremity
483     i=nDisl-1;
484     t1 = this->dislocations[i].idealTimeIncrement(this->
dislocationVelocities[i],
485                                     minDistance,
486                                     this->extremities[1],
487                                     Vector3d(0.0, 0.0, 0.0));
488     t2 = this->dislocations[i].idealTimeIncrement(this->
dislocationVelocities[i],
489                                     minDistance,
490                                     this->dislocations[i-1],
491                                     this->dislocationVelocities[i-1]);
492     // Choose the smaller of the two
493     timeIncrement[i] = t1 < t2 ? t1:t2;
494
495     if (timeIncrement[i] < minDt)
496     {
497         // This dislocation should not move in this iteration because it might collide with the next defect
498         timeIncrement[i] = minDt;
499         this->dislocationVelocities[i] = Vector3d(0.0, 0.0, 0.0);
500
501         // The other defect is a slip plane extremity
502         // This dislocation will not move any more
503         this->dislocations[i].setPinned();
504     }
505
506     dtMin = 1000;
507     for (i=0; i<nDisl; i++)
508     {
509         if (timeIncrement[i] < dtMin)
510         {
511             dtMin = timeIncrement[i];
512         }
513     }
514
515     this->dt = dtMin;
516 }

```

5.6.3.5 void SlipPlane::calculateVelocities (double B)

Calculates the velocities of dislocations and stores them in the std::vector container velocities.

The velocities of the dislocations are calculated and stored in the std::vector container called velocities. The velocities are calculated using the proportionality law between them and the Peach-Koehler force, using the drag coefficient B as the constant of proportionality. param B The drag coefficient.

Definition at line 367 of file slipPlane.cpp.

```

368 {
369     std::vector<Dislocation>::iterator d; // Iterator for dislocations
370     std::vector<Vector3d>::iterator f;    // Iterator for forces
371     std::vector<Vector3d>::iterator v;    // Iterator for velocities
372
373     Vector3d p0, p1, p01;
374     double norm_v, norm_p01, cosine;
375
376     d = this->dislocations.begin();
377     f = this->dislocationForces.begin();
378     v = this->dislocationVelocities.begin();
379
380     while (v != this->dislocationVelocities.end())
381     {
382         if (d->isMobile())
383         {
384             // Velocity directly proportional to Peach-Koehler force
385             (*v) = (*f) * (1.0/B);
386             norm_v = v->magnitude();
387
388             if (norm_v > 0.0)
389             {
390                 // Project the velocity on to the slip plane line
391                 p0 = this->extremities[0].getPosition();
392                 p1 = this->extremities[1].getPosition();
393                 p01 = p1 - p0;
394                 norm_p01 = p01.magnitude();
395
396                 cosine = ((*v) * p01)/(norm_v * norm_p01);

```

```

397         (*v) *= cosine;
398     }
399 }
400 else
401 {
402     *v = Vector3d(0.0, 0.0, 0.0);
403 }
404 d++;
405 f++;
406 v++;
407 }
408 }

```

5.6.3.6 double SlipPlane::distanceFromExtremity (Vector3d pos, int n)

The distance of the point pos from the n^{th} extremity is returned.

Parameters

<i>pos</i>	Position vector of the point whose distance is to be calculated.
<i>n</i>	Index of the extremity. Can be only 0 or 1. In all other cases 0.0 is returned.

Returns

Distance of the point pos from the n^{th} extremity of the slip plane.

Definition at line 547 of file slipPlane.cpp.

```

548 {
549     if (n!=0 && n!=1)
550     {
551         return (0.0);
552     }
553
554     Vector3d r = this->extremities[n].getPosition();
555     return ( (r-pos).magnitude() );
556 }

```

5.6.3.7 Vector3d SlipPlane::getAxis (int i) const

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 242 of file slipPlane.cpp.

```

243 {
244     Vector3d axis;
245
246     if (i==2)
247     {
248         // Z-axis
249         axis = this->normalVector;
250     }
251
252     if (i==0)
253     {

```

```

254     // X-axis
255     Vector3d *e1 = new Vector3d;
256     Vector3d *e2 = new Vector3d;
257
258     *e1 = this->extremities[0].getPosition();
259     *e2 = this->extremities[1].getPosition();
260     axis = ((*e2) - (*e1));
261
262     delete(e1);  e1 = NULL;
263     delete(e2);  e2 = NULL;
264 }
265
266 if (i==1)
267 {
268     // Y-axis = Z x X
269     axis = this->getAxis(2) ^ this->getAxis(0);
270 }
271
272 return ( axis.normalize() );
273 }

```

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

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 159 of file slipPlane.cpp.

```

160 {
161     if (i>=0 && i<this->dislocations.size ())
162     {
163         *d = this->dislocations[i];
164         return (true);
165     }
166     else
167     {
168         return (false);
169     }
170 }

```

5.6.3.9 std::vector< Dislocation > SlipPlane::getDislocationList () const

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 176 of file slipPlane.cpp.

```

177 {
178     return (this->dislocations);
179 }

```

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

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 197 of file slipPlane.cpp.

```

198 {
199     if (i>=0 && i<this->dislocationSources.size ())
200     {
201         *dSource = this->dislocationSources[i];
202         return (true);
203     }
204     else
205     {
206         return (false);
207     }
208 }
```

5.6.3.11 std::vector< DislocationSource > SlipPlane::getDislocationSourceList () const

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 214 of file slipPlane.cpp.

```

215 {
216     return (this->dislocationSources);
217 }
```

5.6.3.12 Vector3d SlipPlane::getExtremity (int *i*) const

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 121 of file slipPlane.cpp.

```
122 {  
123     if (i==0 || i==1)  
124     {  
125         return (this->extremities[i].getPosition());  
126     }  
127     else  
128     {  
129         return (Vector3d());  
130     }  
131 }
```

5.6.3.13 Vector3d SlipPlane::getNormal () const

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 137 of file slipPlane.cpp.

```
138 {  
139     return (this->normalVector);  
140 }
```

5.6.3.14 int SlipPlane::getNumDislocations () const

Get the number of dislocations.

Returns

The number of dislocations on the slip plane.

Definition at line 185 of file slipPlane.cpp.

```
186 {  
187     return (this->dislocations.size ());  
188 }
```

5.6.3.15 int SlipPlane::getNumDislocationSources () const

Get the number of dislocation sources.

Returns

The number of dislocation sources on the slip plane.

Definition at line 223 of file slipPlane.cpp.

```
224 {  
225     return (this->dislocationSources.size ());  
226 }
```


5.6.3.16 Vector3d SlipPlane::getPosition () const

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 147 of file slipPlane.cpp.

```
148 {
149     return (this->position);
150 }
```

5.6.3.17 RotationMatrix SlipPlane::getRotationMatrix () const

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 232 of file slipPlane.cpp.

```
233 {
234     return (this->rotationMatrix);
235 }
```

5.6.3.18 std::vector< Stress > SlipPlane::getSlipPlaneStress_global (std::vector< Vector3d > points, Stress appliedStress, double mu, double nu)

Returns a vector containing the stress values at different points along a slip plane.

The stress field (expressed in the global co-ordinate system) is calculated at points along the slip plane given as argument. This function only takes into account the dislocations present on itself for calculating the stress field.

Parameters

<i>points</i>	STL vector container with position vectors (Vector3d) of points at which the stress field is to be calculated.
<i>appliedStress</i>	The externally applied stress (in the global co-ordinate system).
<i>mu</i>	Shear modulus of the material in Pa.
<i>nu</i>	Poisson's ratio.

Returns

STL vector container with the full stress tensor expressing the stress field (in the global co-ordinate system) at the points provided as input.

Definition at line 600 of file slipPlane.cpp.

```
601 {
602     // Initialize the vector for holding Stress values
603     std::vector<Stress> stressVector(points.size(), Stress());
604 }
```

```

605 // Iterator for the points
606 std::vector<Vector3d>::iterator p = points.begin();
607
608 // Iterator for the stress
609 std::vector<Stress>::iterator s = stressVector.begin();
610
611 // Temporary variable for stress
612 Stress sTemp;
613
614 while (p != points.end())
615 {
616     sTemp = appliedStress;
617     // Iterator for the dislocations
618     std::vector<Dislocation>::iterator d = this->dislocations.begin();
619     while (d != this->dislocations.end())
620     {
621         sTemp += d->stressField (*p, mu, nu);
622         d++;
623     }
624     *s = sTemp;
625     s++;
626     p++;
627 }
628
629
630 return (stressVector);
631 }

```

5.6.3.19 `std::vector< Stress > SlipPlane::getSlipPlaneStress_local (std::vector< Vector3d > points, Stress appliedStress, double mu, double nu)`

Returns a vector containing the stress values at different points along a slip plane.

The stress field (expressed in the local co-ordinate system) is calculated at points along the slip plane given as argument. This function only takes into account the dislocations present on itself for calculating the stress field.

Parameters

<i>points</i>	STL vector container with position vectors (Vector3d) of points at which the stress field is to be calculated.
<i>appliedStress</i>	The externally applied stress (in the global co-ordinate system).
<i>mu</i>	Shear modulus of the material in Pa.
<i>nu</i>	Poisson's ratio.

Returns

STL vector container with the full stress tensor expressing the stress field (in the local co-ordinate system) at the points provided as input.

Definition at line 642 of file slipPlane.cpp.

```

643 {
644     // Initialize the vector for holding Stress values
645     std::vector<Stress> stressVector(points.size(), Stress());
646
647     // Iterator for the points
648     std::vector<Vector3d>::iterator p = points.begin();
649
650     // Iterator for the stress
651     std::vector<Stress>::iterator s = stressVector.begin();
652
653     // Temporary variable for stress
654     Stress sTemp;
655
656     while (p != points.end())
657     {
658         sTemp = appliedStress;
659         // Iterator for the dislocations
660         std::vector<Dislocation>::iterator d = this->dislocations.begin();
661         while (d != this->dislocations.end())
662         {
663             sTemp += d->stressField (*p, mu, nu);
664             d++;

```

```

665         }
666
667         // Convert to local co-ordinate system
668         *s = sTemp.rotate(this->rotationMatrix);
669
670         s++;
671         p++;
672     }
673
674     return (stressVector);
675 }

```

5.6.3.20 void SlipPlane::moveDislocations ()

Displaces the dislocations according to their velocities and the time increment.

Definition at line 521 of file slipPlane.cpp.

```

522 {
523     std::vector<Dislocation>::iterator d;
524     std::vector<Vector3d>::iterator v;
525     Vector3d p;
526
527     d = this->dislocations.begin();
528     v = this->dislocationVelocities.begin();
529
530     while (d != this->dislocations.end())
531     {
532         p = d->getPosition();
533         p += (*v) * (this->dt);
534         d->setPosition(p);
535
536         d++;
537         v++;
538     }
539 }

```

5.6.3.21 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 101 of file slipPlane.cpp.

```

102 {
103     this->dislocations = dislocationList;
104 }

```

5.6.3.22 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 110 of file slipPlane.cpp.

```

111 {
112     this->dislocationSources = dislocationSourceList;
113 }

```

5.6.3.23 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 73 of file slipPlane.cpp.

```

74 {
75     this->extremities[0].setPosition(ends[0]);
76     this->extremities[1].setPosition(ends[1]);
77 }
```

5.6.3.24 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 83 of file slipPlane.cpp.

```

84 {
85     this->normalVector = normal;
86 }
```

5.6.3.25 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 92 of file slipPlane.cpp.

```

93 {
94     this->position = pos;
95 }
```

5.6.3.26 void SlipPlane::sortDislocations ()

Sorts the dislocations present on the slip plane in the ascending order of distance from the first extremity.

The dislocations present on the slip plane are sorted in ascending order of distance from the first extremity of the slip plane.

Definition at line 562 of file slipPlane.cpp.

```

563 {
564     int nDisl = this->dislocations.size();
565     int i, j;
566     double di, dj;
567     Vector3d pi, pj;
```

```

568     Dislocation temp;
569
570     for (i=0; i<nDisl-1; i++)
571     {
572         for (j=i+1; j<nDisl; j++)
573         {
574             pi = this->dislocations[i].getPosition();
575             di = this->distanceFromExtremity(pi, 0);
576
577             pj = this->dislocations[j].getPosition();
578             dj = this->distanceFromExtremity(pj, 0);
579
580             if (dj < di)
581             {
582                 // Swap the two
583                 temp = this->dislocations[i];
584                 this->dislocations[i] = this->dislocations[j];
585                 this->dislocations[j] = temp;
586             }
587         }
588     }
589 }

```

5.6.4 Field Documentation

5.6.4.1 `std::vector<Vector3d> SlipPlane::dislocationForces` [protected]

The Peach-Koehler force experienced by each dislocation.

This vector container stores the Peach-Koehler force experienced by each dislocation. They are calculated in each iteration by the function `calculateDislocationForces(tau_crss)`.

Definition at line 66 of file `slipPlane.h`.

5.6.4.2 `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 54 of file `slipPlane.h`.

5.6.4.3 `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 78 of file `slipPlane.h`.

5.6.4.4 `std::vector<Stress> SlipPlane::dislocationStresses` [protected]

STL vector container with the stress fields of dislocations.

The stress fields experienced by the dislocations, expressed in the global co-ordinate system, are stored in this vector with positions corresponding to the positions of dislocations in the vector `dislocations`.

Definition at line 60 of file `slipPlane.h`.

5.6.4.5 `std::vector<Vector3d> SlipPlane::dislocationVelocities` [protected]

STL vector container with dislocation velocities.

The dislocations on this slip plane will have a velocity associated with them. These velocity vectors are stored in this container. The order is the same as the order of the dislocations.

Definition at line 72 of file `slipPlane.h`.

5.6.4.6 `double SlipPlane::dt` `[protected]`

Time increment for the slip plane.

A time increment is calculated for each slip plane based on the distances traveled by the dislocations.

Definition at line 84 of file `slipPlane.h`.

5.6.4.7 `Defect 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 36 of file `slipPlane.h`.

5.6.4.8 `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 42 of file `slipPlane.h`.

5.6.4.9 `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 48 of file `slipPlane.h`.

5.6.4.10 `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 90 of file `slipPlane.h`.

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

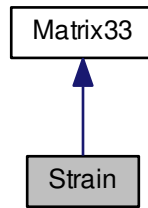
- [slipPlane.h](#)
- [slipPlane.cpp](#)

5.7 Strain Class Reference

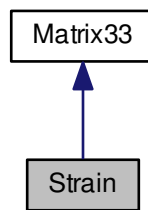
[Strain](#) class to represent the strain tensor.

```
#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.
- [Strain](#) ([Matrix33](#) m)
Constructor specifying the full matrix.
- void [populateMatrix](#) ()
Construct the strain tensor from the principal and shear strains.
- [Vector3d](#) [getPrincipalStrains](#) () const
Get the principal strains.
- [Vector3d](#) [getShearStrains](#) () const
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) const
Returns the value of the element located by the row and column indices provided.

- [Matrix33 adjugate](#) () const
Returns the adjugate matrix of the present matrix.
- [Matrix33 transpose](#) () const
Returns the transpose 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.
- 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 }

```

5.7.2.3 Strain::Strain (Matrix33 *m*)

Constructor specifying the full matrix.

This constructor accepts the full strain matrix as input and extracts the principal and shear strain components.

Parameters

<i>m</i>	Matrix33 variable containing the full strain tensor.
----------	--

Definition at line 53 of file strain.cpp.

```

54 {
55     int i, j;
56     bool symmetry = true;
57
58     // Verify symmetry
59     for (i=0; i<3; i++)
60     {
61         for (j=0; j<3; j++)
62         {
63             if (m.getValue(i,j) != m.getValue(j,i))
64             {
65                 symmetry = false;
66                 break;
67             }
68         }
69     }
70
71     if (symmetry)
72     {
73         // The matrix is symmetrical

```

```

74     this->principalStrains [0] = m.getValue(0,0);
75     this->principalStrains [1] = m.getValue(1,1);
76     this->principalStrains [2] = m.getValue(2,2);
77
78     this->shearStrains [0] = m.getValue(0,1);
79     this->shearStrains [1] = m.getValue(0,2);
80     this->shearStrains [2] = m.getValue(1,2);
81 }
82 else
83 {
84     // The matrix is asymmetrical
85     // A zero matrix will be returned
86     for (i=0; i<3; i++)
87     {
88         this->principalStrains[i] = 0.0;
89         this->shearStrains[i] = 0.0;
90     }
91 }
92
93 this->populateMatrix ();
94 }

```

5.7.3 Member Function Documentation

5.7.3.1 Matrix33 Matrix33::adjugate () const [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.setValue(0, 0, ((this->x[1][1]*this->x[2][2]) - (this->x[1][2]*this->x[2][1])));
134     adj.setValue(0, 1, ((this->x[1][2]*this->x[2][0]) - (this->x[1][0]*this->x[2][2])));
135     adj.setValue(0, 2, ((this->x[1][0]*this->x[2][1]) - (this->x[1][1]*this->x[2][0])));
136
137     adj.setValue(1, 0, ((this->x[2][1]*this->x[0][2]) - (this->x[0][1]*this->x[2][2])));
138     adj.setValue(1, 1, ((this->x[2][2]*this->x[0][0]) - (this->x[2][0]*this->x[0][2])));
139     adj.setValue(1, 2, ((this->x[2][0]*this->x[0][1]) - (this->x[2][1]*this->x[0][0])));
140
141     adj.setValue(2, 0, ((this->x[0][1]*this->x[1][2]) - (this->x[0][2]*this->x[1][1])));
142     adj.setValue(2, 1, ((this->x[0][2]*this->x[1][0]) - (this->x[0][0]*this->x[1][2])));
143     adj.setValue(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 Vector3d Strain::getPrincipalStrains () const

Get the principal strains.

Returns a vector of type [Vector3d](#) with the principal strains: s11 s22 s33.

Returns

[Vector3d](#) variable with the principal strains.

Definition at line 116 of file strain.cpp.

```

117 {
118     return ( Vector3d (this->principalStrains [0],
119                     this->principalStrains [1],
120                     this->principalStrains [2] ) );
121 }

```

5.7.3.3 Vector3d Strain::getShearStrains () const

Get the shear strains.

Returns a vector of type [Vector3d](#) with the shear strains: s12 s13 s23.

Returns

[Vector3d](#) variable with the shear strains.

Definition at line 128 of file strain.cpp.

```
129 {
130     return ( Vector3d (this->shearStrains [0],
131                     this->shearStrains [1],
132                     this->shearStrains [2] ) );
133 }
```

5.7.3.4 double Matrix33::getValue (int row, int column) const *[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 374 of file matrix33.cpp.

```
375 {
```

```

376  Matrix33 r;    // Result matrix
377
378  double determinant = ~(*this);
379
380  if (determinant == 0.0)
381  {
382      // The matrix is non-invertible
383      return (r);    // Zero matrix
384  }
385
386  // If we are still here, the matrix is invertible
387
388  // Transpose
389  Matrix33 tr = this->transpose();
390
391  // Find Adjugate matrix
392  Matrix33 adj = tr.adjugate();
393
394  // Calculate the inverse by dividing the adjugate matrix by the determinant
395  r = adj * (1.0/determinant);
396
397  return (r);
398 }

```

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 253 of file matrix33.cpp.

```

254 {
255     int i, j;
256     Matrix33 r;
257
258     for (i=0; i<3; i++)
259     {
260         for (j=0; j<3; j++)
261         {
262             r.setValue(i, j, (this->x[i][j] * p));
263         }
264     }
265
266     return (r);
267 }

```

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 291 of file matrix33.cpp.

```

292 {
293     int i, j, k;
294     Matrix33 r;
295     double s;
296
297     for (i=0; i<3; i++)
298     {
299         for (j=0; j<3; j++)

```

```

300     {
301         s = 0.0;
302         for (k=0; k<3; k++)
303         {
304             s += this->x[i][k] * p.getValue(k, j);
305         }
306         r.setValue (i, j, s);
307     }
308 }
309
310 return (r);
311 }

```

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 333 of file matrix33.cpp.

```

334 {
335     Vector3d r(0.0, 0.0, 0.0);
336     double s;
337     int i, j;
338
339     for (i=0; i<3; i++)
340     {
341         s = 0.0;
342         for (j=0; j<3; j++)
343         {
344             s += this->x[i][j] * v.getValue(j);
345         }
346         r.setValue (i, s);
347     }
348
349     return (r);
350 }

```

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 273 of file matrix33.cpp.

```

274 {
275     int i, j;
276
277     for (i=0; i<3; i++)
278     {
279         for (j=0; j<3; j++)
280         {
281             this->x[i][j] *= p;
282         }
283     }
284 }

```

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 317 of file matrix33.cpp.

```

318 {
319     Matrix33* r = new Matrix33;
320
321     *r = (*this) * p;
322     *this = *r;
323
324     delete(r);
325     r = NULL;
326 }

```

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 175 of file matrix33.cpp.

```

176 {
177     int i, j;
178     Matrix33 r;
179
180     for (i=0; i<3; i++)
181     {
182         for (j=0; j<3; j++)
183         {
184             r.setValue(i, j, (this->x[i][j] + p.getValue(i, j)));
185         }
186     }
187
188     return (r);
189 }

```

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 195 of file matrix33.cpp.

```

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

```

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 214 of file matrix33.cpp.

```

215 {
216     int i, j;
217     Matrix33 r;
218
219     for (i=0; i<3; i++)
220     {
221         for (j=0; j<3; j++)
222         {
223             r.setValue(i, j, (this->x[i][j] - p.getValue(i, j)));
224         }
225     }
226
227     return (r);
228 }
```

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 234 of file matrix33.cpp.

```

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

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

Determinant.

Calculates the determinant of the current matrix.

Returns

Returns the determinant of the current matrix.

Definition at line 358 of file matrix33.cpp.

```

359 {
360     double d = 0.0;
361
362     d += this->x[0][0] * ( (this->x[1][1]*this->x[2][2]) - (this->x[2][1]*this->
363         x[1][2]) );
364     d += this->x[0][1] * ( (this->x[1][2]*this->x[2][0]) - (this->x[1][0]*this->
365         x[2][2]) );
366     d += this->x[0][2] * ( (this->x[1][0]*this->x[2][1]) - (this->x[2][0]*this->
367         x[1][1]) );
368
369     return (d);
370 }
```

5.7.3.16 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 100 of file strain.cpp.

```

101 {
102     this->x[0][0] = this->principalStrains [0];
103     this->x[1][1] = this->principalStrains [1];
104     this->x[2][2] = this->principalStrains [2];
105
106     this->x[0][1] = this->x[1][0] = this->shearStrains [0];
107     this->x[0][2] = this->x[2][0] = this->shearStrains [1];
108     this->x[1][2] = this->x[2][1] = this->shearStrains [2];
109 }
```

5.7.3.17 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 141 of file strain.cpp.

```

142 {
143     // Transpose
144     RotationMatrix alphaT (alpha.transpose());
145
146     // Rotate the strain matrix
147     Strain sNew = Strain (alpha * (*this) * alphaT);
148
149     return (sNew);
150 }
```

5.7.3.18 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 }

```

5.7.3.19 Matrix33 Matrix33::transpose () const [inherited]

Returns the transpose of the present matrix.

The transpose of a matrix is another matrix having rows identical to the columns of the present matrix, and vice-versa.

Returns

The transpose of the present matrix.

Definition at line 153 of file matrix33.cpp.

```

154 {
155     Matrix33 tr;
156     int i, j;
157
158     for (i=0; i<3; i++)
159     {
160         for (j=0; j<3; j++)
161         {
162             tr.setValue (i, j, this->x[j][i]);
163         }
164     }
165     return (tr);
166 }

```

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:

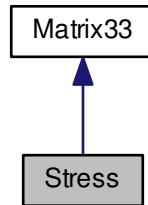
- [strain.h](#)
- [strain.cpp](#)

5.8 Stress Class Reference

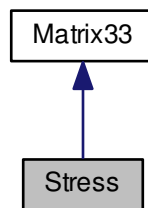
[Stress](#) class to represent the stress tensor.

```
#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.
- [Stress](#) ([Matrix33](#) m)
Constructor specifying the full matrix.
- void [populateMatrix](#) ()
Construct the stress tensor from the principal and shear stresses.
- [Vector3d](#) [getPrincipalStresses](#) () const
Get the principal stresses.
- [Vector3d](#) [getShearStresses](#) () const
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) const

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

 - `Matrix33 adjugate () const`
Returns the adjugate matrix of the present matrix.
 - `Matrix33 transpose () const`
Returns the transpose 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.
 - `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 }

```

5.8.2.3 Stress::Stress (Matrix33 *m*)

Constructor specifying the full matrix.

This constructor accepts the full stress matrix as input and extracts the principal and shear stress components.

Parameters

<i>m</i>	Matrix33 variable containing the full stress tensor.
----------	--

Definition at line 53 of file stress.cpp.

```

54 {
55     int i, j;
56     bool symmetry = true;
57
58     // Verify symmetry
59     for (i=0; i<3; i++)
60     {
61         for (j=0; j<3; j++)
62         {
63             if (m.getValue(i,j) != m.getValue(j,i))
64             {
65                 symmetry = false;
66                 break;
67             }
68         }
69     }
70
71     if (symmetry)
72     {
73         // The matrix is symmetrical

```

```

74     this->principalStresses [0] = m.getValue(0,0);
75     this->principalStresses [1] = m.getValue(1,1);
76     this->principalStresses [2] = m.getValue(2,2);
77
78     this->shearStresses [0] = m.getValue(0,1);
79     this->shearStresses [1] = m.getValue(0,2);
80     this->shearStresses [2] = m.getValue(1,2);
81 }
82 else
83 {
84     // The matrix is asymmetrical
85     // A zero matrix will be returned
86     for (i=0; i<3; i++)
87     {
88         this->principalStresses[i] = 0.0;
89         this->shearStresses[i] = 0.0;
90     }
91 }
92
93 this->populateMatrix ();
94 }

```

5.8.3 Member Function Documentation

5.8.3.1 Matrix33 Matrix33::adjugate () const [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.setValue(0, 0, ((this->x[1][1]*this->x[2][2]) - (this->x[1][2]*this->x[2][1])));
134     adj.setValue(0, 1, ((this->x[1][2]*this->x[2][0]) - (this->x[1][0]*this->x[2][2])));
135     adj.setValue(0, 2, ((this->x[1][0]*this->x[2][1]) - (this->x[1][1]*this->x[2][0])));
136
137     adj.setValue(1, 0, ((this->x[2][1]*this->x[0][2]) - (this->x[0][1]*this->x[2][2])));
138     adj.setValue(1, 1, ((this->x[2][2]*this->x[0][0]) - (this->x[2][0]*this->x[0][2])));
139     adj.setValue(1, 2, ((this->x[2][0]*this->x[0][1]) - (this->x[2][1]*this->x[0][0])));
140
141     adj.setValue(2, 0, ((this->x[0][1]*this->x[1][2]) - (this->x[0][2]*this->x[1][1])));
142     adj.setValue(2, 1, ((this->x[0][2]*this->x[1][0]) - (this->x[0][0]*this->x[1][2])));
143     adj.setValue(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 Vector3d Stress::getPrincipalStresses () const

Get the principal stresses.

Returns a vector of type [Vector3d](#) with the principal stresses: s11 s22 s33.

Returns

[Vector3d](#) variable with the principal stresses.

Definition at line 117 of file stress.cpp.

```

118 {
119     return ( Vector3d (this->principalStresses [0],
120                     this->principalStresses [1],
121                     this->principalStresses [2] ) );
122 }

```

5.8.3.3 Vector3d Stress::getShearStresses () const

Get the shear stresses.

Returns a vector of type [Vector3d](#) with the shear stresses: s12 s13 s23.

Returns

[Vector3d](#) variable with the shear stresses.

Definition at line 129 of file stress.cpp.

```
130 {
131     return ( Vector3d (this->shearStresses [0],
132                     this->shearStresses [1],
133                     this->shearStresses [2] ) );
134 }
```

5.8.3.4 double Matrix33::getValue (int row, int column) const [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 374 of file matrix33.cpp.

```
375 {
```

```

376  Matrix33 r;    // Result matrix
377
378  double determinant = ~(*this);
379
380  if (determinant == 0.0)
381  {
382      // The matrix is non-invertible
383      return (r);    // Zero matrix
384  }
385
386  // If we are still here, the matrix is invertible
387
388  // Transpose
389  Matrix33 tr = this->transpose();
390
391  // Find Adjugate matrix
392  Matrix33 adj = tr.adjugate();
393
394  // Calculate the inverse by dividing the adjugate matrix by the determinant
395  r = adj * (1.0/determinant);
396
397  return (r);
398 }

```

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 253 of file matrix33.cpp.

```

254 {
255     int i, j;
256     Matrix33 r;
257
258     for (i=0; i<3; i++)
259     {
260         for (j=0; j<3; j++)
261         {
262             r.setValue(i, j, (this->x[i][j] * p));
263         }
264     }
265
266     return (r);
267 }

```

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 291 of file matrix33.cpp.

```

292 {
293     int i, j, k;
294     Matrix33 r;
295     double s;
296
297     for (i=0; i<3; i++)
298     {
299         for (j=0; j<3; j++)

```

```

300     {
301         s = 0.0;
302         for (k=0; k<3; k++)
303         {
304             s += this->x[i][k] * p.getValue(k, j);
305         }
306         r.setValue (i, j, s);
307     }
308 }
309
310 return (r);
311 }

```

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 333 of file matrix33.cpp.

```

334 {
335     Vector3d r(0.0, 0.0, 0.0);
336     double s;
337     int i, j;
338
339     for (i=0; i<3; i++)
340     {
341         s = 0.0;
342         for (j=0; j<3; j++)
343         {
344             s += this->x[i][j] * v.getValue(j);
345         }
346         r.setValue (i, s);
347     }
348
349     return (r);
350 }

```

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 273 of file matrix33.cpp.

```

274 {
275     int i, j;
276
277     for (i=0; i<3; i++)
278     {
279         for (j=0; j<3; j++)
280         {
281             this->x[i][j] *= p;
282         }
283     }
284 }

```

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 317 of file matrix33.cpp.


```

318 {
319     Matrix33* r = new Matrix33;
320
321     *r = (*this) * p;
322     *this = *r;
323
324     delete(r);
325     r = NULL;
326 }

```

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 175 of file matrix33.cpp.

```

176 {
177     int i, j;
178     Matrix33 r;
179
180     for (i=0; i<3; i++)
181     {
182         for (j=0; j<3; j++)
183         {
184             r.setValue(i, j, (this->x[i][j] + p.getValue(i, j)));
185         }
186     }
187
188     return (r);
189 }

```

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 195 of file matrix33.cpp.

```

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

```

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 214 of file matrix33.cpp.

```

215 {
216     int i, j;
217     Matrix33 r;
218
219     for (i=0; i<3; i++)
220     {
221         for (j=0; j<3; j++)
222         {
223             r.setValue(i, j, (this->x[i][j] - p.getValue(i, j)));
224         }
225     }
226
227     return (r);
228 }
```

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 234 of file matrix33.cpp.

```

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

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

Determinant.

Calculates the determinant of the current matrix.

Returns

Returns the determinant of the current matrix.

Definition at line 358 of file matrix33.cpp.

```

359 {
360     double d = 0.0;
361
362     d += this->x[0][0] * ( (this->x[1][1]*this->x[2][2]) - (this->x[2][1]*this->
x[1][2]) );
363     d += this->x[0][1] * ( (this->x[1][2]*this->x[2][0]) - (this->x[1][0]*this->
x[2][2]) );
364     d += this->x[0][2] * ( (this->x[1][0]*this->x[2][1]) - (this->x[2][0]*this->
x[1][1]) );
365
366     return (d);
367 }
```

5.8.3.16 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 101 of file stress.cpp.

```

102 {
103     this->x[0][0] = this->principalStresses [0];
104     this->x[1][1] = this->principalStresses [1];
105     this->x[2][2] = this->principalStresses [2];
106
107     this->x[0][1] = this->x[1][0] = this->shearStresses [0];
108     this->x[0][2] = this->x[2][0] = this->shearStresses [1];
109     this->x[1][2] = this->x[2][1] = this->shearStresses [2];
110 }
```

5.8.3.17 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 142 of file stress.cpp.

```

143 {
144     // Transpose
145     RotationMatrix alphaT (alpha.transpose());
146
147     // Rotate the stress matrix
148     Stress sNew = Stress (alpha * (*this) * alphaT);
149
150     return (sNew);
151 }
```

5.8.3.18 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 }

```

5.8.3.19 Matrix33 Matrix33::transpose () const [inherited]

Returns the transpose of the present matrix.

The transpose of a matrix is another matrix having rows identical to the columns of the present matrix, and vice-versa.

Returns

The transpose of the present matrix.

Definition at line 153 of file matrix33.cpp.

```

154 {
155     Matrix33 tr;
156     int i, j;
157
158     for (i=0; i<3; i++)
159     {
160         for (j=0; j<3; j++)
161         {
162             tr.setValue (i, j, this->x[j][i]);
163         }
164     }
165     return (tr);
166 }

```

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) const
Function to get the value of an element of the vector.
- double * [getVector](#) () const
Function to get the values of the elements of the vector in an array.
- double [sum](#) () const
Computes the sum of the elements of the vector.
- double [magnitude](#) () const
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*=
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^=
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 21 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) const

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 () const

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 () const

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.setValue(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.getValue(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.setValue(i, (this->x[i] + p.getValue(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.setValue(i, (this->x[i] - p.getValue(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.getValue(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.setValue(0, ((this->x[1] * p.getValue(2)) - (this->x[2] * p.
294         getValue(1))));
295     r.setValue(1, ((this->x[2] * p.getValue(0)) - (this->x[0] * p.
296         getValue(2))));
297     r.setValue(2, ((this->x[0] * p.getValue(1)) - (this->x[1] * p.
298         getValue(0))));
299
300     return (r);
301 }
```

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 = new Vector3d(0.0, 0.0, 0.0);
307
308     *r = (*this)^p;
309     *this = *r;
310     delete (r);
311     r = NULL;
312 }
```

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 () const

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 27 of file vector3d.h.

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

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

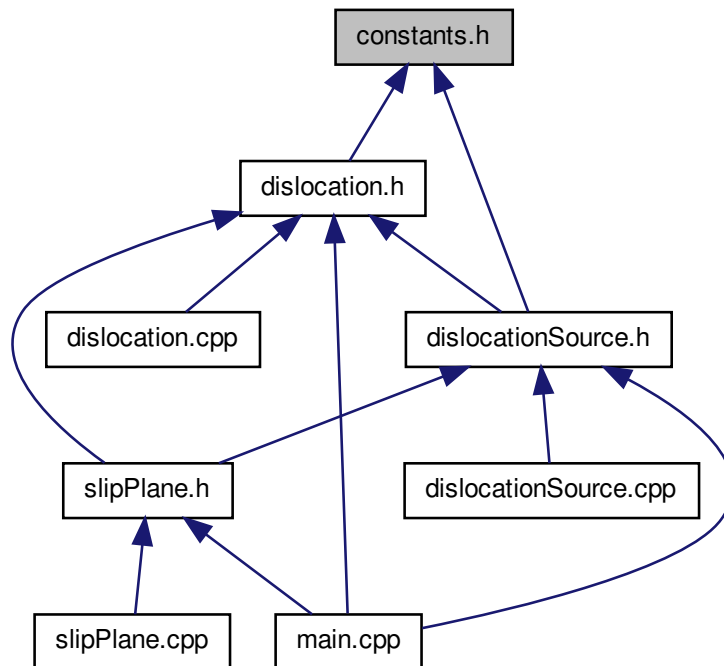
Chapter 6

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.

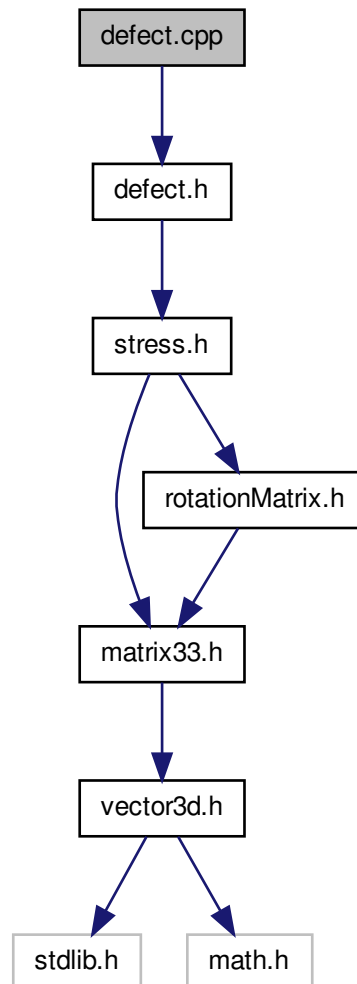
Definition at line 31 of file constants.h.

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

1.0

Date

04/06/2013

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

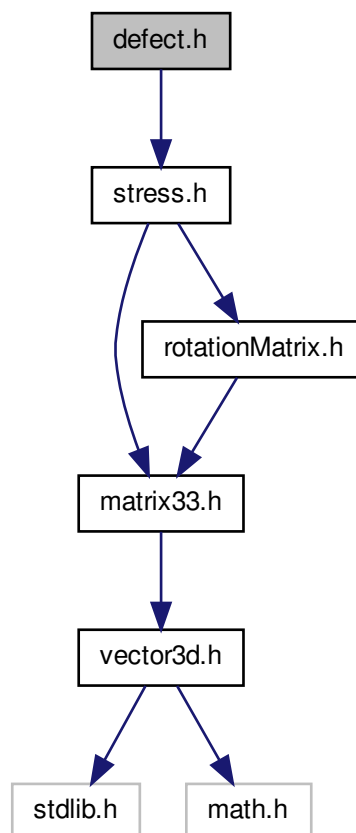
Definition in file [defect.cpp](#).

6.3 defect.h File Reference

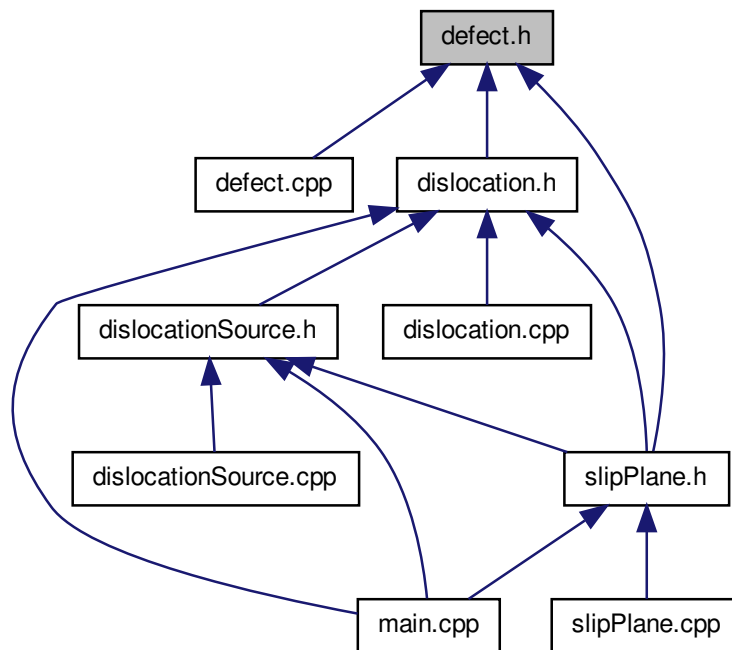
Definition of the [Defect](#) class.

```
#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

1.0

Date

04/06/2013

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.

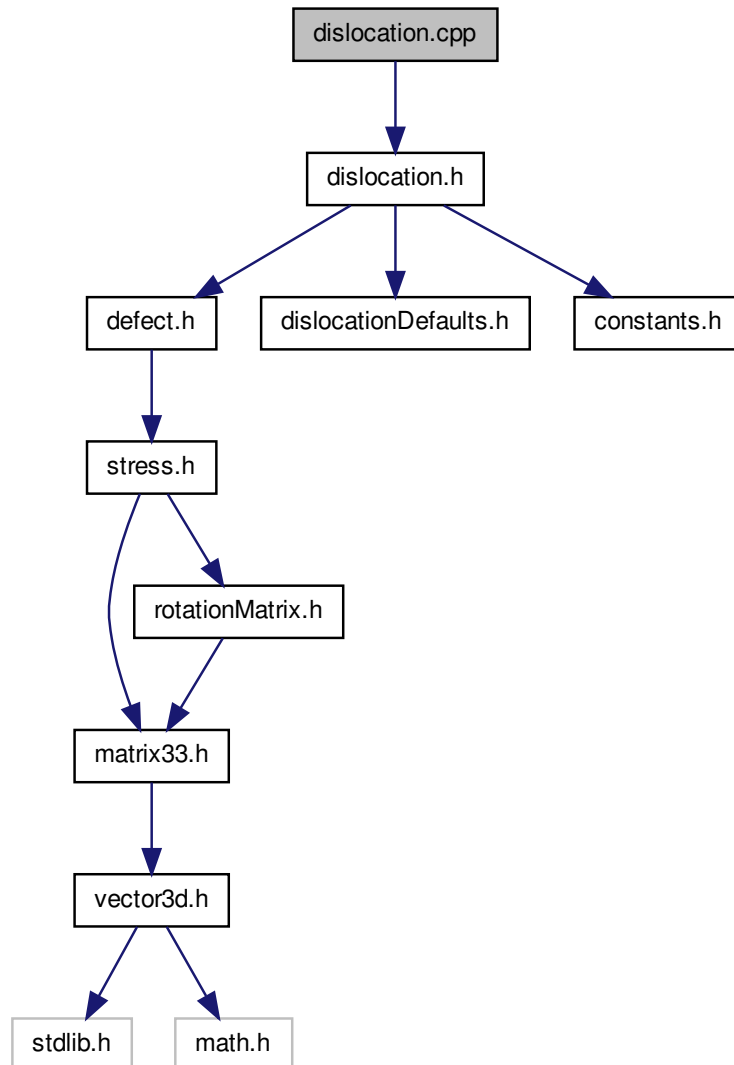
Definition in file [defect.h](#).

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

1.0

Date

04/06/2013

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

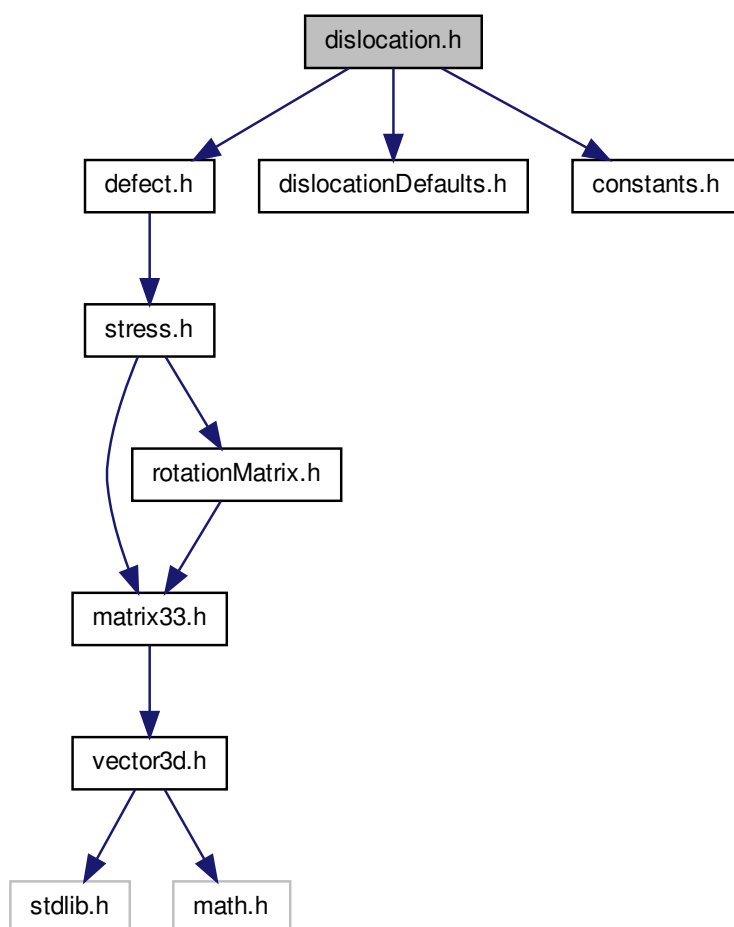
Definition in file [dislocation.cpp](#).

6.5 dislocation.h File Reference

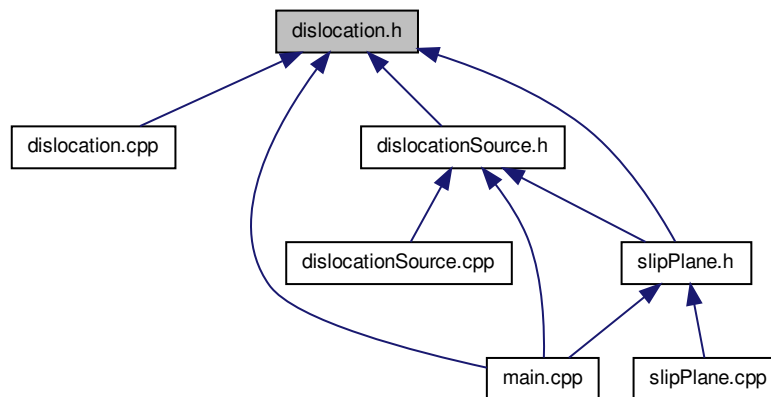
Definition of the [Dislocation](#) class.

```
#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

1.0

Date

04/06/2013

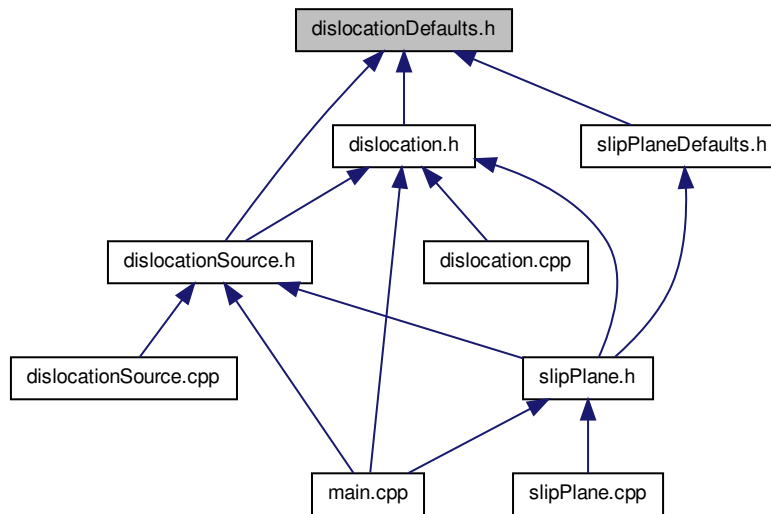
This file defines the [Dislocation](#) class representing a dislocation in the simulation. This class inherits from the [Defect](#) class.

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.

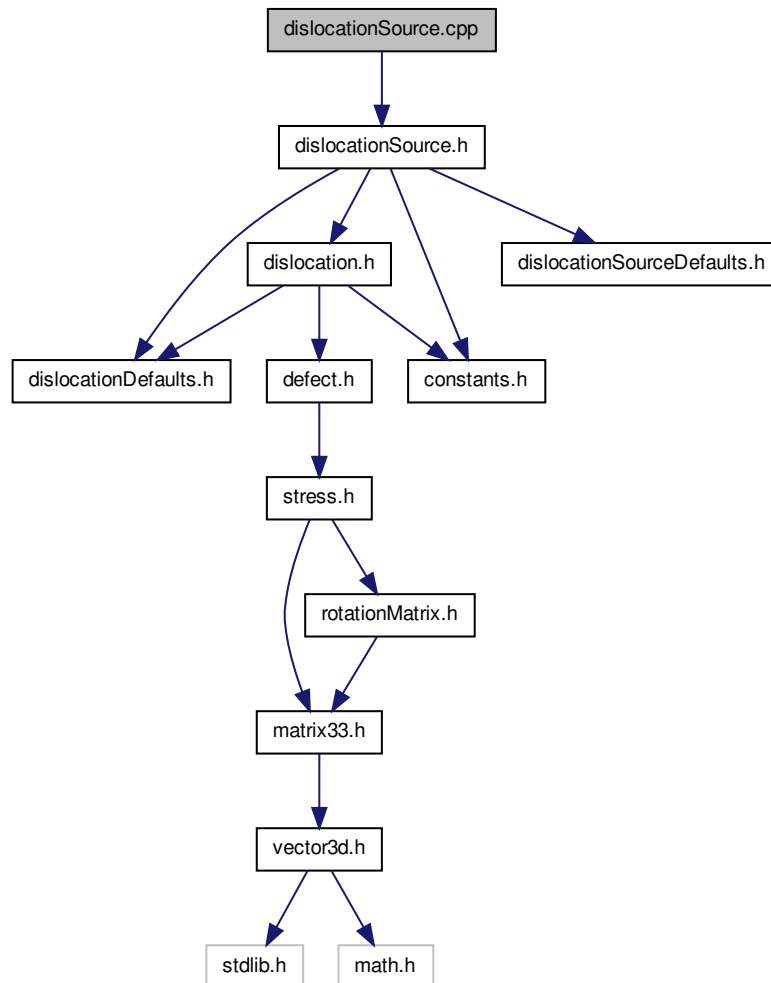
Definition at line 24 of file dislocationDefaults.h.

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

05/06/2013

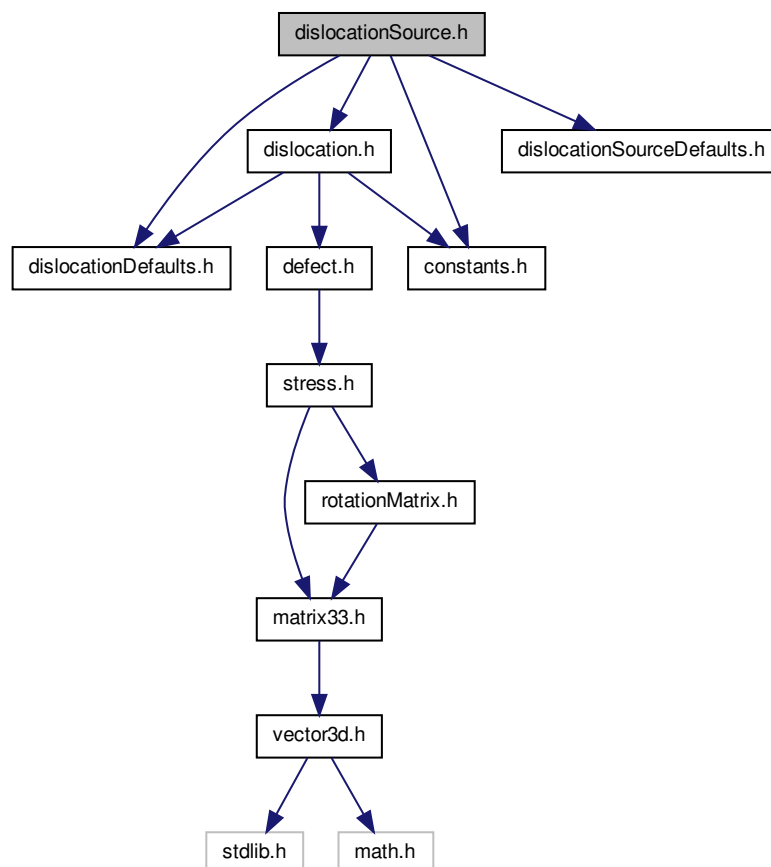
This file defines the member functions of the [DislocationSource](#) class representing a source of dislocations in the simulation. This class inherits from the [Defect](#) class. This object is basically the representation of a Frank-Read source emitting dislocation dipoles. When the dislocation source 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](#).

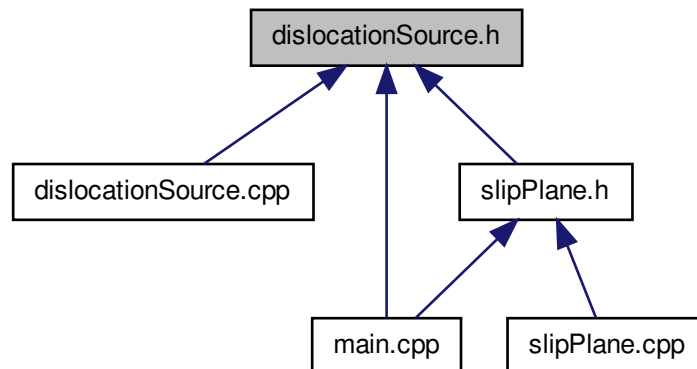
6.8 dislocationSource.h File Reference

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

05/06/2013

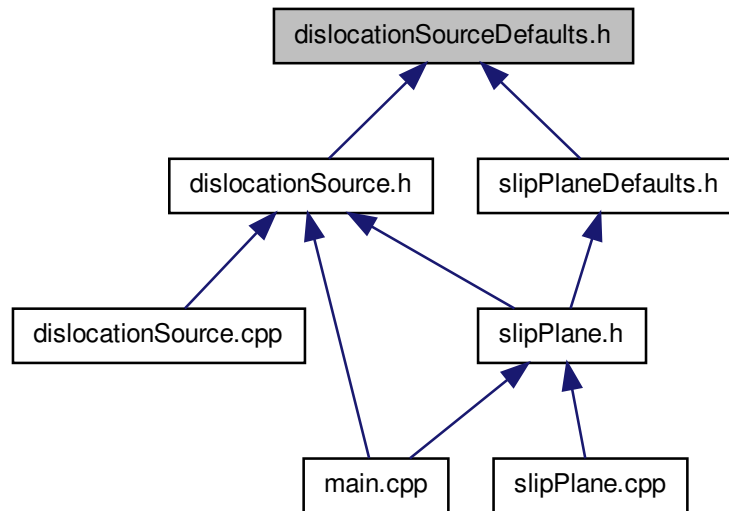
This file defines the [DislocationSource](#) class representing a source of dislocations in the simulation. This class inherits from the [Defect](#) class. This object is basically the representation of a Frank-Read source emitting dislocation dipoles. When the dislocation source experiences a shear 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_NITERATIONS 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.

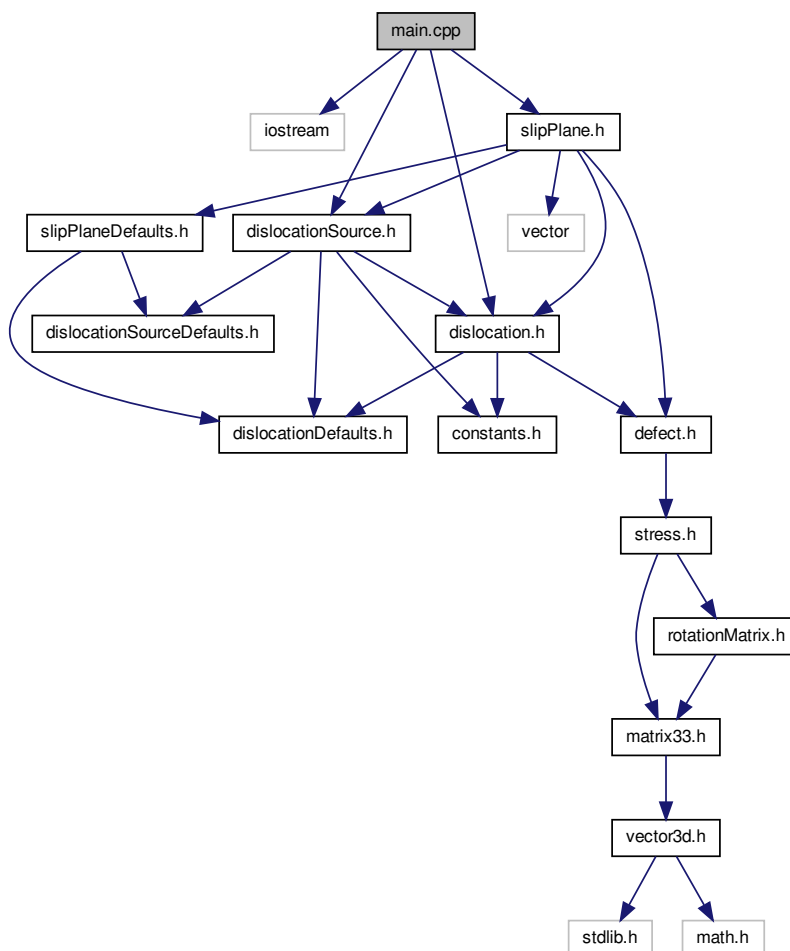
Default value of the critical shear stress for a dislocation source to emit a dipole. The number is expressed in Pa.

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

6.10 `main.cpp` File Reference

```
#include <iostream>
#include "dislocation.h"
#include "dislocationSource.h"
#include "slipPlane.h"
```

Include dependency graph for main.cpp:



Functions

- int [main](#) ()

6.10.1 Function Documentation

6.10.1.1 int main ()

Definition at line 7 of file main.cpp.

```

8 {
9     return (0);
10 }
```

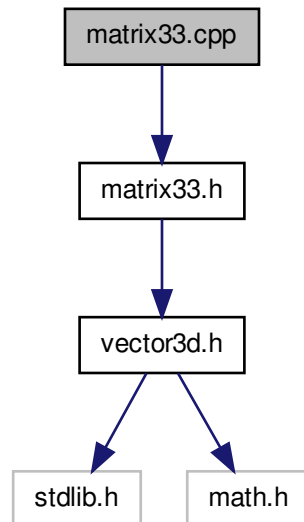
6.11 mainpage.dox File Reference

6.12 matrix33.cpp File Reference

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

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

Include dependency graph for matrix33.cpp:



6.12.1 Detailed Description

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

Author

Adhish Majumdar

Version

1.0

Date

04/06/2013

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

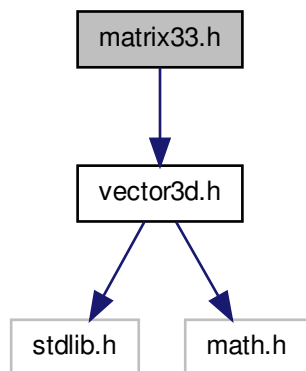
Definition in file [matrix33.cpp](#).

6.13 matrix33.h File Reference

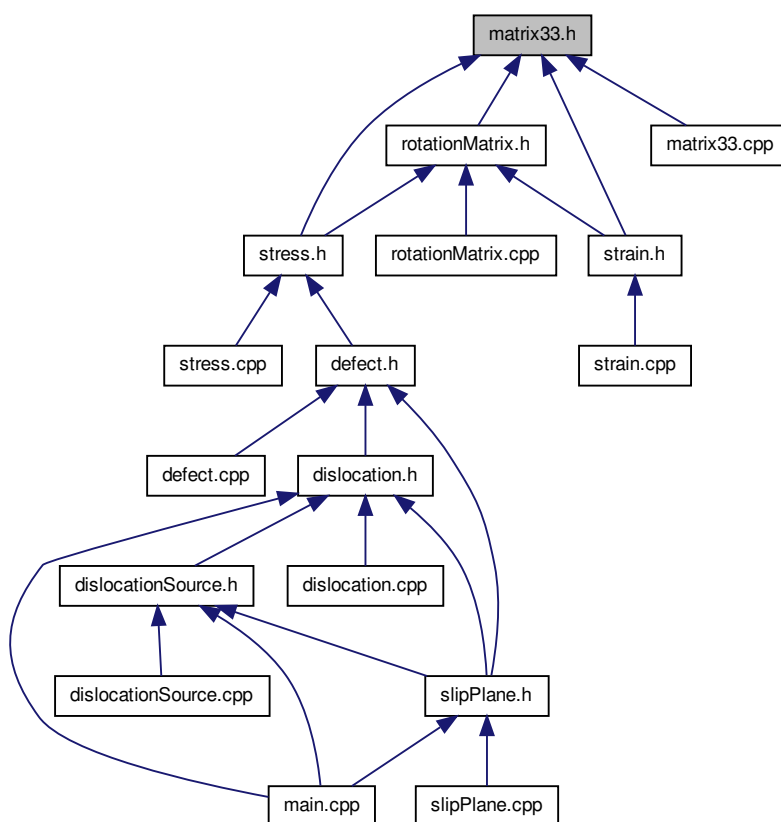
Definition of the [Matrix33](#) class.

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

Include dependency graph for matrix33.h:



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



Data Structures

- class [Matrix33](#)

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

6.13.1 Detailed Description

Definition of the [Matrix33](#) class.

Author

Adhish Majumdar

Version

1.0

Date

04/06/2013

This file defines the [Matrix33](#) class representing a 3x3 matrix in the simulation.

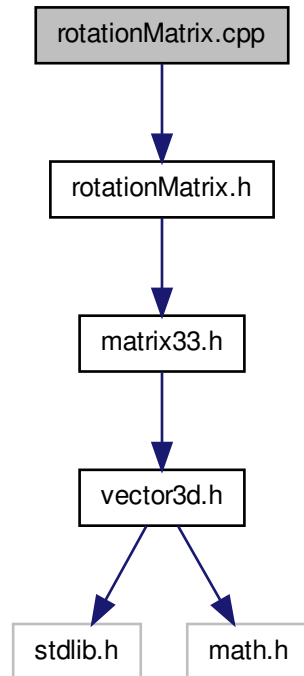
Definition in file [matrix33.h](#).

6.14 rotationMatrix.cpp File Reference

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


```
#include "rotationMatrix.h"
```

Include dependency graph for rotationMatrix.cpp:



6.14.1 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.

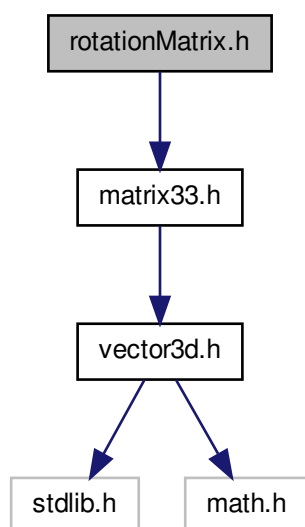
Definition in file [rotationMatrix.cpp](#).

6.15 rotationMatrix.h File Reference

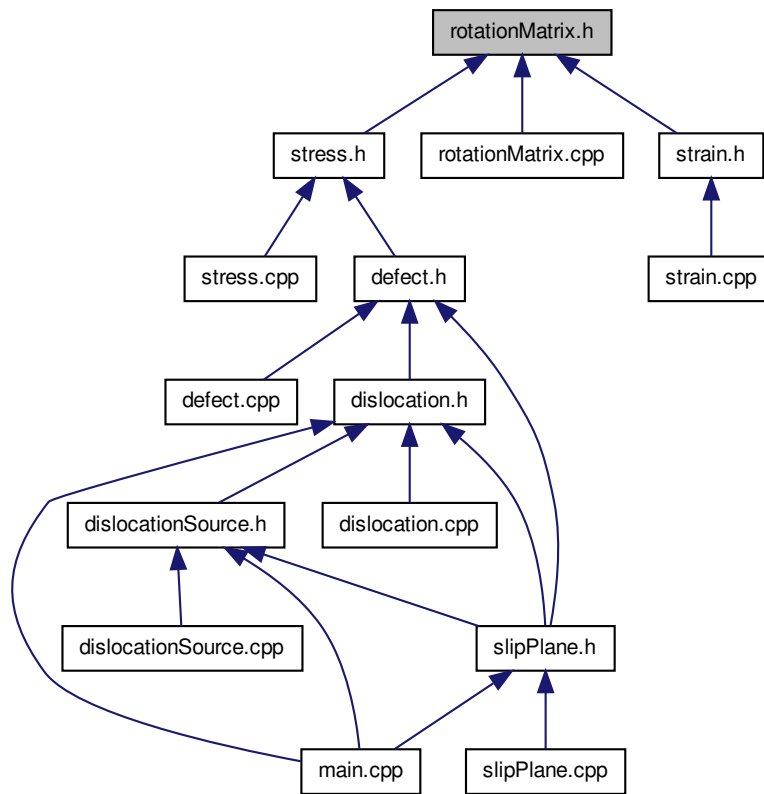
Definition of the [RotationMatrix](#) class.

```
#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.

6.15.1 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.

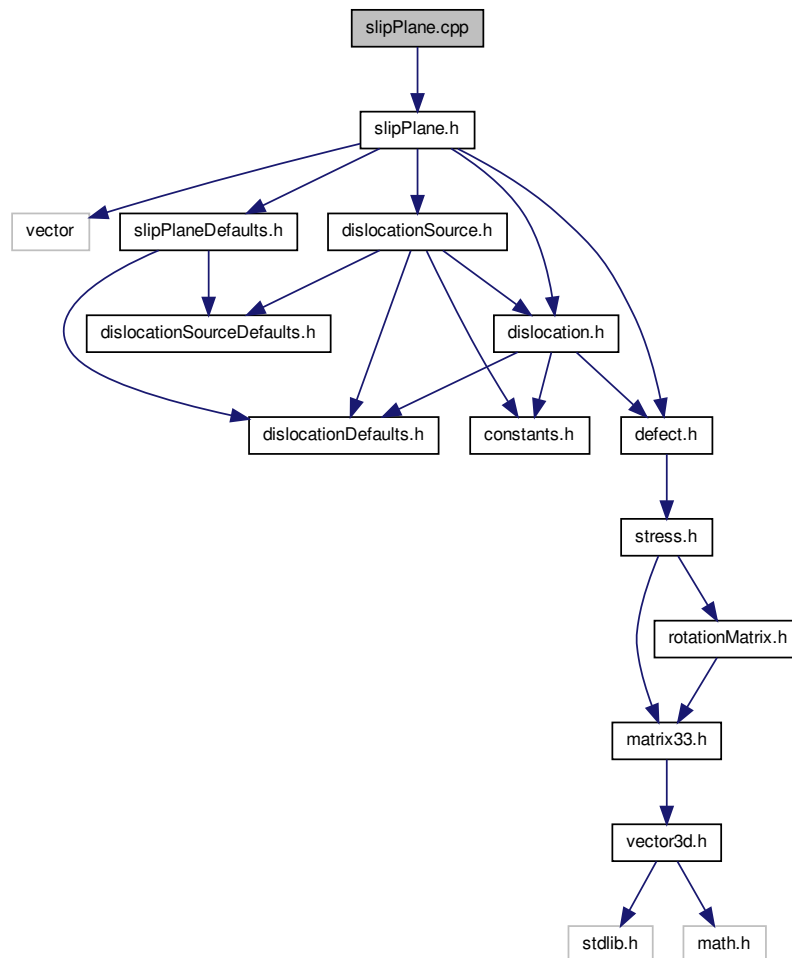
Definition in file [rotationMatrix.h](#).

6.16 slipPlane.cpp File Reference

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

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

Include dependency graph for slipPlane.cpp:



6.16.1 Detailed Description

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

Author

Adhish Majumdar

Version

0.0

Date

03/06/2013

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

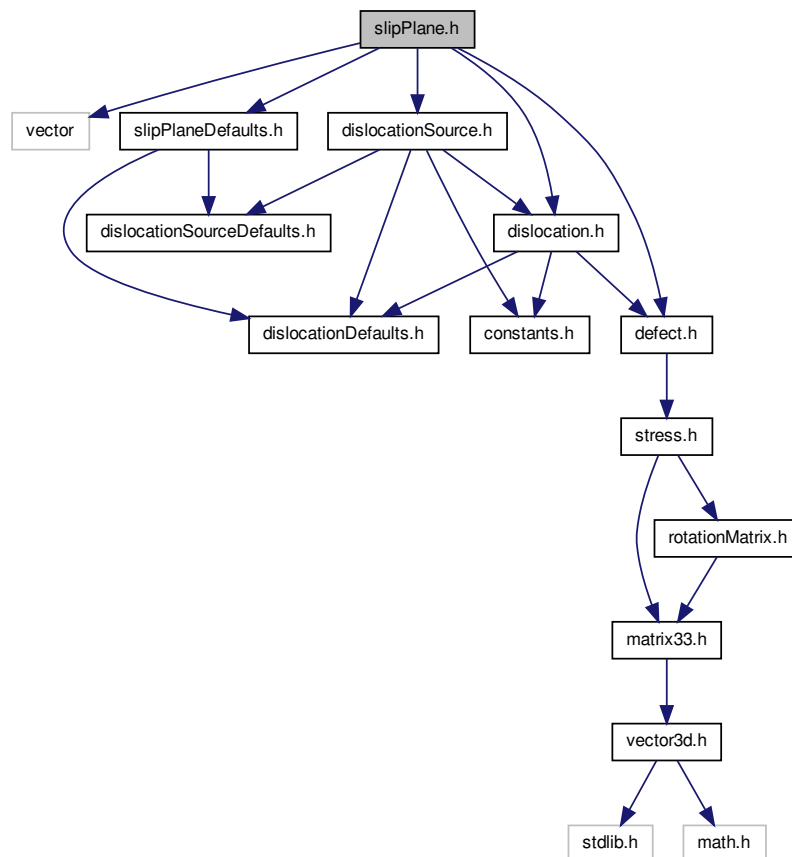
Definition in file [slipPlane.cpp](#).

6.17 slipPlane.h File Reference

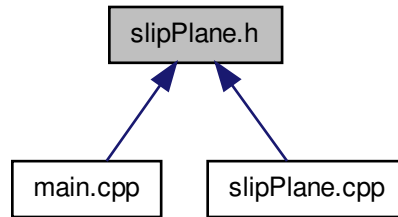
Definition of the [SlipPlane](#) class.

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

6.17.1 Detailed Description

Definition of the [SlipPlane](#) class.

Author

Adhish Majumdar

Version

0.0

Date

03/06/2013

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

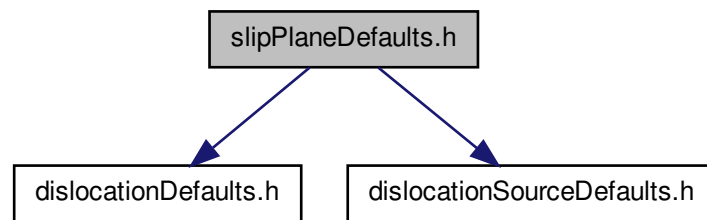
Definition in file [slipPlane.h](#).

6.18 slipPlaneDefaults.h File Reference

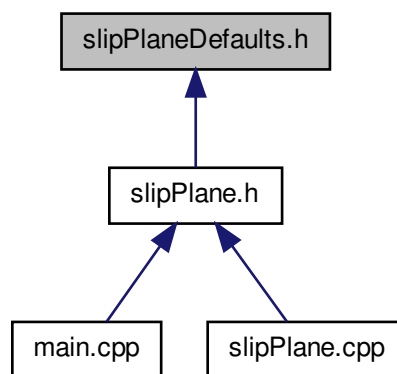
Definition of certain default values for members of the [SlipPlane](#) class.

```
#include "dislocationDefaults.h"
#include "dislocationSourceDefaults.h"
```

Include dependency graph for slipPlaneDefaults.h:



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



Macros

- `#define` [DEFAULT_SLIPPLANE_POSITION_0](#) 0.0
Default value of the position vector x-coordinate.
- `#define` [DEFAULT_SLIPPLANE_POSITION_1](#) 0.0
Default value of the position vector y-coordinate.
- `#define` [DEFAULT_SLIPPLANE_POSITION_2](#) 0.0
Default value of the position vector z-coordinate.
- `#define` [DEFAULT_SLIPPLANE_NORMALVECTOR_0](#) 1.0
Default value of the normal vector x-coordinate.
- `#define` [DEFAULT_SLIPPLANE_NORMALVECTOR_1](#) 1.0
Default value of the normal vector y-coordinate.
- `#define` [DEFAULT_SLIPPLANE_NORMALVECTOR_2](#) 1.0
Default value of the normal vector z-coordinate.
- `#define` [DEFAULT_SLIPPLANE_EXTREMITY1_0](#) 5.0e-06

- Default value of the position vector of extremity 1 x-coordinate.*
 - `#define` [DEFAULT_SLIPPLANE_EXTREMITY1_1](#) 0.0
- Default value of the position vector of extremity 1 y-coordinate.*
 - `#define` [DEFAULT_SLIPPLANE_EXTREMITY1_2](#) 0.0
- Default value of the position vector of extremity 1 z-coordinate.*
 - `#define` [DEFAULT_SLIPPLANE_EXTREMITY2_0](#) 0.0
- Default value of the position vector of extremity 2 x-coordinate.*
 - `#define` [DEFAULT_SLIPPLANE_EXTREMITY2_1](#) 5.0e-6
- Default value of the position vector of extremity 2 y-coordinate.*
 - `#define` [DEFAULT_SLIPPLANE_EXTREMITY2_2](#) 0.0
- Default value of the position vector of extremity 2 z-coordinate.*

6.18.1 Detailed Description

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

Author

Adhish Majumdar

Version

0.0

Date

31/05/2013

This file defines some default values for members of the [SlipPlane](#) class representing a slip plane in the simulation.
Definition in file [slipPlaneDefaults.h](#).

6.18.2 Macro Definition Documentation

6.18.2.1 `#define` [DEFAULT_SLIPPLANE_EXTREMITY1_0](#) 5.0e-06

Default value of the position vector of extremity 1 x-coordinate.
Definition at line 50 of file [slipPlaneDefaults.h](#).

6.18.2.2 `#define` [DEFAULT_SLIPPLANE_EXTREMITY1_1](#) 0.0

Default value of the position vector of extremity 1 y-coordinate.
Definition at line 55 of file [slipPlaneDefaults.h](#).

6.18.2.3 `#define` [DEFAULT_SLIPPLANE_EXTREMITY1_2](#) 0.0

Default value of the position vector of extremity 1 z-coordinate.
Definition at line 60 of file [slipPlaneDefaults.h](#).

6.18.2.4 `#define` [DEFAULT_SLIPPLANE_EXTREMITY2_0](#) 0.0

Default value of the position vector of extremity 2 x-coordinate.
Definition at line 65 of file [slipPlaneDefaults.h](#).

6.18.2.5 #define DEFAULT_SLIPPLANE_EXTREMITY2_1 5.0e-6

Default value of the position vector of extremity 2 y-coordinate.

Definition at line 70 of file slipPlaneDefaults.h.

6.18.2.6 #define DEFAULT_SLIPPLANE_EXTREMITY2_2 0.0

Default value of the position vector of extremity 2 z-coordinate.

Definition at line 75 of file slipPlaneDefaults.h.

6.18.2.7 #define DEFAULT_SLIPPLANE_NORMALVECTOR_0 1.0

Default value of the normal vector x-coordinate.

Definition at line 35 of file slipPlaneDefaults.h.

6.18.2.8 #define DEFAULT_SLIPPLANE_NORMALVECTOR_1 1.0

Default value of the normal vector y-coordinate.

Definition at line 40 of file slipPlaneDefaults.h.

6.18.2.9 #define DEFAULT_SLIPPLANE_NORMALVECTOR_2 1.0

Default value of the normal vector z-coordinate.

Definition at line 45 of file slipPlaneDefaults.h.

6.18.2.10 #define DEFAULT_SLIPPLANE_POSITION_0 0.0

Default value of the position vector x-coordinate.

Definition at line 20 of file slipPlaneDefaults.h.

6.18.2.11 #define DEFAULT_SLIPPLANE_POSITION_1 0.0

Default value of the position vector y-coordinate.

Definition at line 25 of file slipPlaneDefaults.h.

6.18.2.12 #define DEFAULT_SLIPPLANE_POSITION_2 0.0

Default value of the position vector z-coordinate.

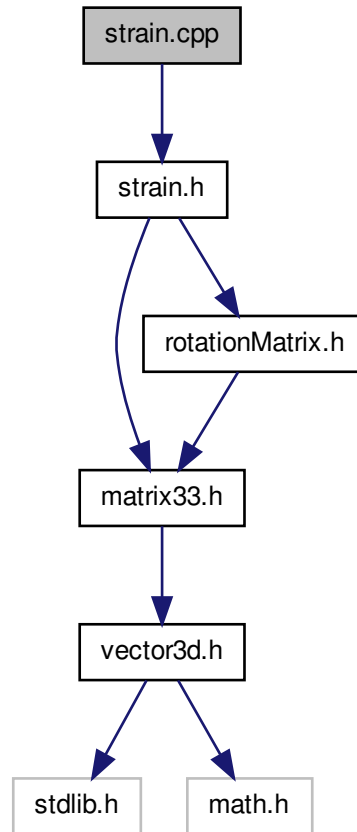
Definition at line 30 of file slipPlaneDefaults.h.

6.19 strain.cpp File Reference

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

```
#include "strain.h"
```

Include dependency graph for strain.cpp:



6.19.1 Detailed Description

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

Author

Adhish Majumdar

Version

1.0

Date

05/06/2013

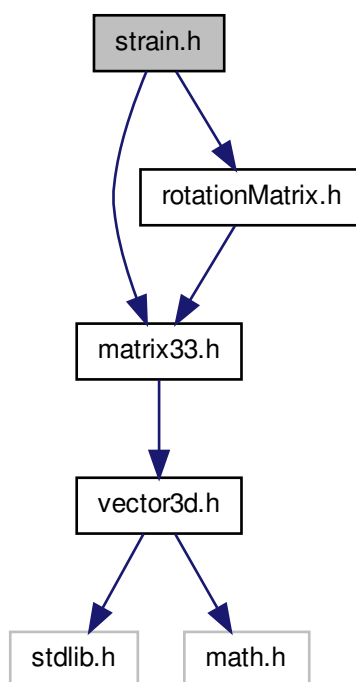
This file defines the member functions of the [Strain](#) class for the strain tensor.

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

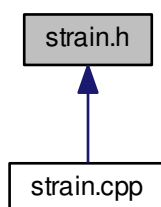
6.20 strain.h File Reference

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.

6.20.1 Detailed Description

Definition of the [Strain](#) class.

Author

Adhish Majumdar

Version

1.0

Date

05/06/2013

This file defines the [Strain](#) class for the strain tensor.

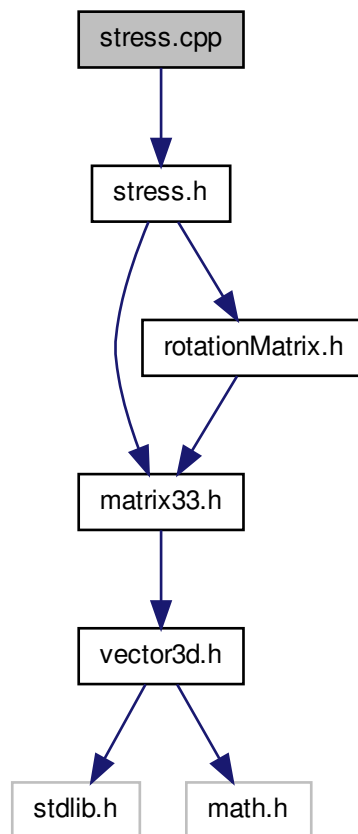
Definition in file [strain.h](#).

6.21 stress.cpp File Reference

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

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

Include dependency graph for stress.cpp:



6.21.1 Detailed Description

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

Author

Adhish Majumdar

Version

1.0

Date

05/06/2013

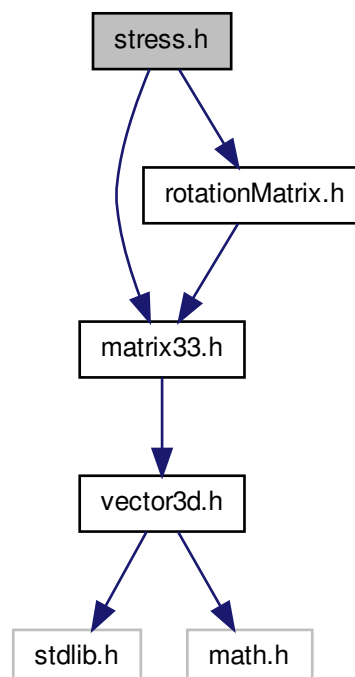
This file defines the member functions of the [Stress](#) class for the stress tensor.

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

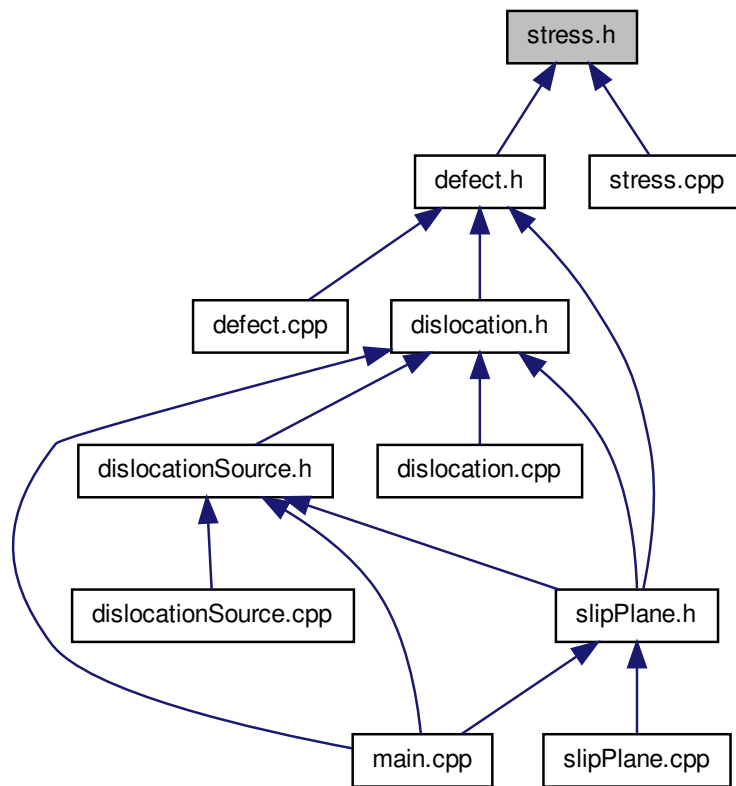
6.22 stress.h File Reference

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.

6.22.1 Detailed Description

Definition of the [Stress](#) class.

Author

Adhish Majumdar

Version

1.0

Date

05/06/2013

This file defines the [Stress](#) class for the stress tensor.

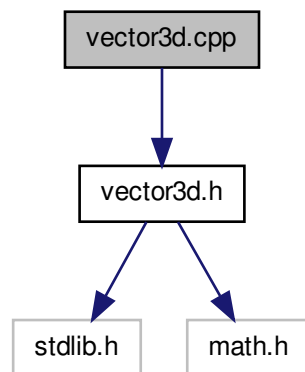
Definition in file [stress.h](#).

6.23 vector3d.cpp File Reference

Definition of member functions and operators of the [Vector3d](#) class.

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

Include dependency graph for vector3d.cpp:



6.23.1 Detailed Description

Definition of member functions and operators of the [Vector3d](#) class.

Author

Adhish Majumdar

Version

1.0

Date

04/06/2013

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](#).

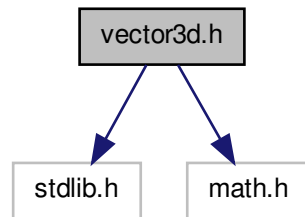
6.24 vector3d.h File Reference

Definition of the [Vector3d](#) class.

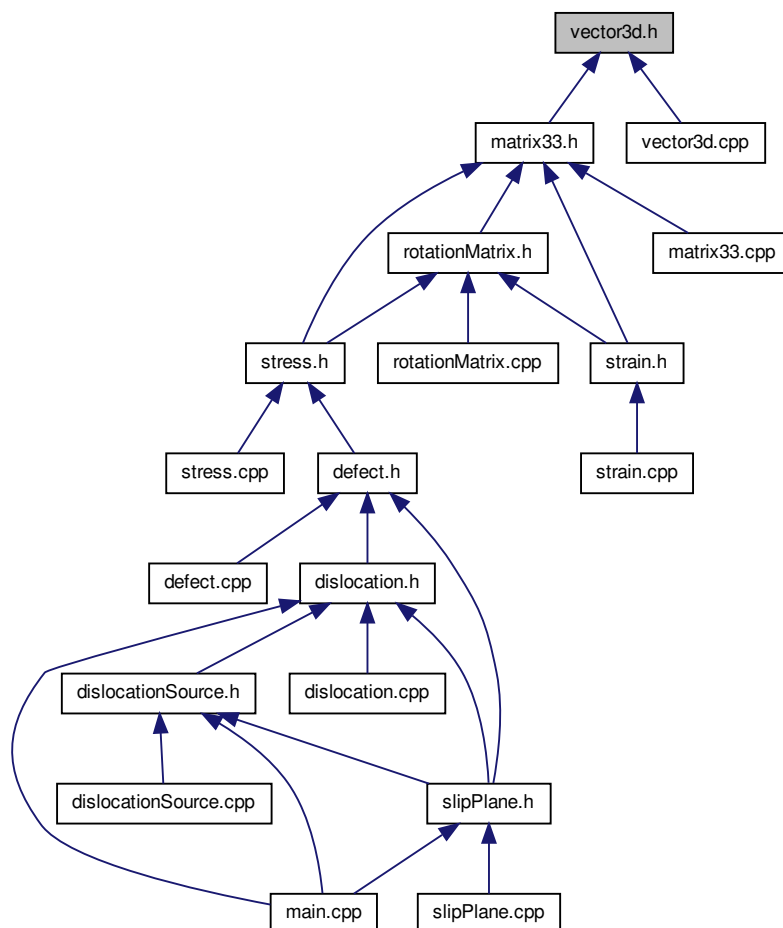
```
#include <stdlib.h>
```

```
#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.

6.24.1 Detailed Description

Definition of the [Vector3d](#) class.

Author

Adhish Majumdar

Version

1.0

Date

04/06/2013

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

Definition in file [vector3d.h](#).

Index

- adjugate
 - Matrix33, [42](#)
 - RotationMatrix, [51](#)
 - Strain, [78](#)
 - Stress, [89](#)
- bmag
 - Dislocation, [26](#)
 - DislocationSource, [38](#)
- bvec
 - Dislocation, [26](#)
 - DislocationSource, [38](#)
- calculateDislocationForces
 - SlipPlane, [61](#)
- calculateDislocationStresses
 - SlipPlane, [61](#)
- calculateRotationMatrix
 - Dislocation, [18](#)
 - SlipPlane, [62](#)
- calculateTimeIncrement
 - SlipPlane, [62](#)
- calculateVelocities
 - SlipPlane, [64](#)
- constants.h, [105](#)
 - PI, [106](#)
 - SQRT2, [106](#)
 - SQRT3, [106](#)
 - SQRT5, [106](#)
- countIterations
 - DislocationSource, [38](#)
- DEFAULT_BURGERS_0
 - dislocationDefaults.h, [114](#)
- DEFAULT_BURGERS_1
 - dislocationDefaults.h, [114](#)
- DEFAULT_BURGERS_2
 - dislocationDefaults.h, [114](#)
- DEFAULT_NITERATIONS
 - dislocationSourceDefaults.h, [120](#)
- DEFAULT_POSITION_0
 - dislocationDefaults.h, [115](#)
- DEFAULT_POSITION_1
 - dislocationDefaults.h, [115](#)
- DEFAULT_POSITION_2
 - dislocationDefaults.h, [115](#)
- 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, [107](#)
- defect.h, [108](#)
- dipoleNucleationLength
 - DislocationSource, [31](#)
- Dislocation, [15](#)
 - bmag, [26](#)
 - bvec, [26](#)
 - calculateRotationMatrix, [18](#)
 - Dislocation, [18](#)
 - forcePeachKoehler, [19](#)
 - getBurgers, [19](#)
 - getLineVector, [19](#)
 - getPosition, [20](#)
 - getX, [20](#)
 - getY, [20](#)
 - getZ, [21](#)
 - idealTimeIncrement, [21](#)
 - isMobile, [22](#)
 - lvec, [26](#)
 - mobile, [26](#)
 - pos, [26](#)
 - rotationMatrix, [26](#)
 - setBurgers, [22](#)
 - setLineVector, [22](#)
 - setMobile, [23](#)
 - setPinned, [23](#)
 - setPosition, [23](#), [24](#)
 - setX, [24](#)
 - setY, [24](#)
 - setZ, [24](#)
 - stressField, [25](#)
 - stressFieldLocal, [25](#)
- dislocation.cpp, [110](#)
- dislocation.h, [111](#)
- dislocationDefaults.h, [112](#)
 - DEFAULT_BURGERS_0, [114](#)
 - DEFAULT_BURGERS_1, [114](#)
 - DEFAULT_BURGERS_2, [114](#)
 - DEFAULT_POSITION_0, [115](#)
 - DEFAULT_POSITION_1, [115](#)
 - DEFAULT_POSITION_2, [115](#)

- dislocationForces
 - SlipPlane, 73
- DislocationSource, 27
 - bmag, 38
 - bvec, 38
 - countIterations, 38
 - dipoleNucleationLength, 31
 - DislocationSource, 30
 - DislocationSource, 30
 - getBurgers, 31
 - getBurgersMag, 31
 - getIterationCount, 32
 - getLineVector, 32
 - getNumIterations, 32
 - getPosition, 32, 33
 - getTauCritical, 33
 - getX, 33
 - getY, 33
 - getZ, 34
 - ifEmitDipole, 34
 - incrementIterationCount, 34
 - lvec, 38
 - mobile, 39
 - nIterations, 39
 - pos, 39
 - resetIterationCounter, 34
 - rotationMatrix, 39
 - setBurgers, 35
 - setBurgersMagnitude, 35
 - setLineVector, 35
 - setNumIterations, 35
 - setPosition, 36
 - setTauCritical, 37
 - setX, 37
 - setY, 37
 - setZ, 37
 - stressField, 38
 - tauCritical, 39
- dislocationSource.cpp, 115
- dislocationSource.h, 117
- dislocationSourceDefaults.h, 118
- dislocationSources
 - SlipPlane, 73
- dislocationStresses
 - SlipPlane, 73
- dislocationVelocities
 - SlipPlane, 73
- dislocations
 - SlipPlane, 73
- distanceFromExtremity
 - SlipPlane, 65
- dt
 - SlipPlane, 73
- extremities
 - SlipPlane, 74
- forcePeachKoehler
 - Dislocation, 19
- getAxis
 - SlipPlane, 65
- getBurgers
 - Dislocation, 19
 - DislocationSource, 31
- getBurgersMag
 - DislocationSource, 31
- getDislocation
 - SlipPlane, 66
- getDislocationList
 - SlipPlane, 66
- getDislocationSource
 - SlipPlane, 66
- getDislocationSourceList
 - SlipPlane, 67
- getExtremity
 - SlipPlane, 67
- getIterationCount
 - DislocationSource, 32
- getLineVector
 - Dislocation, 19
 - DislocationSource, 32
- getNormal
 - SlipPlane, 68
- getNumDislocationSources
 - SlipPlane, 68
- getNumDislocations
 - SlipPlane, 68
- getNumIterations
 - DislocationSource, 32
- getPosition
 - Defect, 11, 12
 - Dislocation, 20
 - DislocationSource, 32, 33
 - SlipPlane, 68
- getPrincipalStrains
 - Strain, 78
- getPrincipalStresses
 - Stress, 89
- getRotationMatrix
 - SlipPlane, 69
- getShearStrains
 - Strain, 78
- getShearStresses
 - Stress, 89
- getSlipPlaneStress_global
 - SlipPlane, 69
- getSlipPlaneStress_local
 - SlipPlane, 70
- getTauCritical
 - DislocationSource, 33
- getValue
 - Matrix33, 43
 - RotationMatrix, 52
 - Strain, 79
 - Stress, 90
 - Vector3d, 98
- getVector

- Vector3d, 99
- getX
 - Defect, 12
 - Dislocation, 20
 - DislocationSource, 33
- getY
 - Defect, 12
 - Dislocation, 20
 - DislocationSource, 33
- getZ
 - Defect, 12
 - Dislocation, 21
 - DislocationSource, 34
- idealTimeIncrement
 - Dislocation, 21
- ifEmitDipole
 - DislocationSource, 34
- incrementIterationCount
 - DislocationSource, 34
- isMobile
 - Dislocation, 22
- lvec
 - Dislocation, 26
 - DislocationSource, 38
- magnitude
 - Vector3d, 99
- main
 - main.cpp, 121
- main.cpp, 120
 - main, 121
- mainpage.dox, 121
- Matrix33, 39
 - adjugate, 42
 - getValue, 43
 - Matrix33, 41, 42
 - operator*, 44, 45
 - operator*=: 45
 - operator~, 47
 - operator+, 46
 - operator+=, 46
 - operator-, 46
 - operator-=, 47
 - setValue, 47
 - transpose, 48
 - x, 48
- matrix33.cpp, 122
- matrix33.h, 122
- mobile
 - Dislocation, 26
 - DislocationSource, 39
- moveDislocations
 - SlipPlane, 71
- nIterations
 - DislocationSource, 39
- normalVector
 - SlipPlane, 74
- normalize
 - Vector3d, 99
- operator*
 - Matrix33, 44, 45
 - RotationMatrix, 53, 54
 - Strain, 80, 81
 - Stress, 91, 92
 - Vector3d, 100
- operator*=
 - Matrix33, 45
 - RotationMatrix, 54
 - Strain, 81
 - Stress, 92
 - Vector3d, 101
- operator~
 - Matrix33, 47
 - RotationMatrix, 56
 - Strain, 83
 - Stress, 94
- operator^
 - Vector3d, 102
- operator^=
 - Vector3d, 102
- operator+
 - Matrix33, 46
 - RotationMatrix, 55
 - Strain, 82
 - Stress, 93
 - Vector3d, 101
- operator+=
 - Matrix33, 46
 - RotationMatrix, 55
 - Strain, 82
 - Stress, 93
 - Vector3d, 101
- operator-
 - Matrix33, 46
 - RotationMatrix, 55
 - Strain, 82
 - Stress, 93
 - Vector3d, 101
- operator-=
 - Matrix33, 47
 - RotationMatrix, 56
 - Strain, 83
 - Stress, 94
 - Vector3d, 102
- PI
 - constants.h, 106
- populateMatrix
 - Strain, 83
 - Stress, 94
- pos
 - Defect, 15
 - Dislocation, 26
 - DislocationSource, 39

- position
 - SlipPlane, 74
- principalStrains
 - Strain, 85
- principalStresses
 - Stress, 96
- resetIterationCounter
 - DislocationSource, 34
- rotate
 - Strain, 84
 - Stress, 95
- RotationMatrix, 49
 - adjugate, 51
 - getValue, 52
 - operator*, 53, 54
 - operator*-, 54
 - operator~, 56
 - operator+, 55
 - operator+-, 55
 - operator-, 55
 - operator=, 56
 - RotationMatrix, 50, 51
 - RotationMatrix, 50, 51
 - setValue, 56
 - transpose, 57
 - x, 57
- rotationMatrix
 - Dislocation, 26
 - DislocationSource, 39
 - SlipPlane, 74
- rotationMatrix.cpp, 124
- rotationMatrix.h, 125
- SQRT2
 - constants.h, 106
- SQRT3
 - constants.h, 106
- SQRT5
 - constants.h, 106
- setBurgers
 - Dislocation, 22
 - DislocationSource, 35
- setBurgersMagnitude
 - DislocationSource, 35
- setDislocationList
 - SlipPlane, 71
- setDislocationSourceList
 - SlipPlane, 71
- setExtremities
 - SlipPlane, 71
- setLineVector
 - Dislocation, 22
 - DislocationSource, 35
- setMobile
 - Dislocation, 23
- setNormal
 - SlipPlane, 72
- setNumIterations
 - DislocationSource, 35
- setPinned
 - Dislocation, 23
- setPosition
 - Defect, 13
 - Dislocation, 23, 24
 - DislocationSource, 36
 - SlipPlane, 72
- setTauCritical
 - DislocationSource, 37
- setValue
 - Matrix33, 47
 - RotationMatrix, 56
 - Strain, 84
 - Stress, 95
 - Vector3d, 103
- setVector
 - Vector3d, 103
- setX
 - Defect, 14
 - Dislocation, 24
 - DislocationSource, 37
- setY
 - Defect, 14
 - Dislocation, 24
 - DislocationSource, 37
- setZ
 - Defect, 14
 - Dislocation, 24
 - DislocationSource, 37
- shearStrains
 - Strain, 85
- shearStresses
 - Stress, 96
- SlipPlane, 58
 - calculateDislocationForces, 61
 - calculateDislocationStresses, 61
 - calculateRotationMatrix, 62
 - calculateTimeIncrement, 62
 - calculateVelocities, 64
 - dislocationForces, 73
 - dislocationSources, 73
 - dislocationStresses, 73
 - dislocationVelocities, 73
 - dislocations, 73
 - distanceFromExtremity, 65
 - dt, 73
 - extremities, 74
 - getAxis, 65
 - getDislocation, 66
 - getDislocationList, 66
 - getDislocationSource, 66
 - getDislocationSourceList, 67
 - getExtremity, 67
 - getNormal, 68
 - getNumDislocationSources, 68
 - getNumDislocations, 68
 - getPosition, 68

- getRotationMatrix, 69
- getSlipPlaneStress_global, 69
- getSlipPlaneStress_local, 70
- moveDislocations, 71
- normalVector, 74
- position, 74
- rotationMatrix, 74
- setDislocationList, 71
- setDislocationSourceList, 71
- setExtremities, 71
- setNormal, 72
- setPosition, 72
- SlipPlane, 60
- SlipPlane, 60
- sortDislocations, 72
- slipPlane.cpp, 128
- slipPlane.h, 129
- slipPlaneDefaults.h, 130
- sortDislocations
 - SlipPlane, 72
- Strain, 74
 - adjugate, 78
 - getPrincipalStrains, 78
 - getShearStrains, 78
 - getValue, 79
 - operator*, 80, 81
 - operator*=: 81
 - operator~, 83
 - operator+, 82
 - operator+=, 82
 - operator-, 82
 - operator-=, 83
 - populateMatrix, 83
 - principalStrains, 85
 - rotate, 84
 - setValue, 84
 - shearStrains, 85
 - Strain, 76, 77
 - transpose, 85
 - x, 85
- strain.cpp, 133
- strain.h, 135
- Stress, 85
 - adjugate, 89
 - getPrincipalStresses, 89
 - getShearStresses, 89
 - getValue, 90
 - operator*, 91, 92
 - operator*=: 92
 - operator~, 94
 - operator+, 93
 - operator+=, 93
 - operator-, 93
 - operator-=, 94
 - populateMatrix, 94
 - principalStresses, 96
 - rotate, 95
 - setValue, 95
 - shearStresses, 96
 - Stress, 87, 88
 - transpose, 96
 - x, 96
- stress.cpp, 136
- stress.h, 138
- stressField
 - Defect, 14
 - Dislocation, 25
 - DislocationSource, 38
- stressFieldLocal
 - Dislocation, 25
- sum
 - Vector3d, 103
- tauCritical
 - DislocationSource, 39
- transpose
 - Matrix33, 48
 - RotationMatrix, 57
 - Strain, 85
 - Stress, 96
- Vector3d, 96
 - getValue, 98
 - getVector, 99
 - magnitude, 99
 - normalize, 99
 - operator*, 100
 - operator*=: 101
 - operator^, 102
 - operator^=: 102
 - operator+, 101
 - operator+=, 101
 - operator-, 101
 - operator-=, 102
 - setValue, 103
 - setVector, 103
 - sum, 103
 - Vector3d, 98
 - x, 104
- vector3d.cpp, 140
- vector3d.h, 140
- x
 - Matrix33, 48
 - RotationMatrix, 57
 - Strain, 85
 - Stress, 96
 - Vector3d, 104