DD2D - Matryoshka approach 0

Generated by Doxygen 1.8.3.1

Wed Jun 5 2013 17:20:43

Contents

1	Mair	n Page			1
2	Hier	archica	Index		3
	2.1	Class I	Hierarchy		3
3	Data	Struct	ure Index		5
	3.1	Data S	tructures		5
4	File	Index			7
	4.1	File Lis	st		7
5	Data	Struct	ure Docun	nentation	9
	5.1	Defect	Class Ref	erence	9
		5.1.1	Detailed	Description	10
		5.1.2	Construc	tor & 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 Doo	cumentation	15
			E 1 / 1	200	15

ii CONTENTS

5.2	Disloca	on 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	Disloca	onSource Class Reference	27
	5.3.1	Detailed Description	30
	5.3.2	Constructor & Destructor Documentation	30
		5.3.2.1 DislocationSource	30

CONTENTS

	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 Doo	cumentation	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	nlterations	39
	5.3.4.7	pos	39
	5.3.4.8	rotationMatrix	39
	5.3.4.9	tauCritical	39
Matrix3	33 Class R	eference	39

5.4

iv CONTENTS

	5.4.1	Detailed D	Description	1
	5.4.2	Constructo	or & Destructor Documentation	1
		5.4.2.1	Matrix33	1
		5.4.2.2	Matrix33	1
		5.4.2.3	Matrix33	2
		5.4.2.4	Matrix33	2
	5.4.3	Member F	Function Documentation	2
		5.4.3.1	adjugate	2
		5.4.3.2	getValue	3
		5.4.3.3	operator!	3
		5.4.3.4	operator*	4
		5.4.3.5	operator*	4
		5.4.3.6	operator*	5
		5.4.3.7	operator*=	5
		5.4.3.8	operator*=	5
		5.4.3.9	operator+	3
		5.4.3.10	operator+=	3
		5.4.3.11	operator	3
		5.4.3.12	operator-=	7
		5.4.3.13	operator~	7
		5.4.3.14	setValue	3
		5.4.3.15	transpose	3
	5.4.4	Field Docu	umentation	3
		5.4.4.1	x	3
5.5	Rotatio	nMatrix Cla	ass Reference	Э
	5.5.1	Detailed D	Description)
	5.5.2	Constructo	or & Destructor Documentation	J
		5.5.2.1	RotationMatrix	J
		5.5.2.2	RotationMatrix	1
		5.5.2.3	RotationMatrix	1
	5.5.3	Member F	function Documentation	1
		5.5.3.1	adjugate	1
		5.5.3.2	getValue	2
		5.5.3.3	operator!	2
		5.5.3.4	operator*	3
		5.5.3.5	operator*	3
		5.5.3.6	operator*	4
		5.5.3.7	operator*=	4
		5.5.3.8	operator*=	4
		5.5.3.9	operator+	5

CONTENTS

		5.5.3.10	operator+=	55
		5.5.3.11	operator	55
		5.5.3.12	operator-=	56
		5.5.3.13	operator \sim	56
		5.5.3.14	setValue	57
		5.5.3.15	transpose	57
	5.5.4	Field Doo	cumentation	57
		5.5.4.1	x	57
5.6	SlipPla	ine Class F	Reference	58
	5.6.1	Detailed	Description	60
	5.6.2	Construc	tor & 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

vi CONTENTS

	5.6.4	Field Doo	cumentation	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	$dt \ldots \ldots \ldots \ldots \ldots$	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 Refe	erence	74
	5.7.1	Detailed	Description	76
	5.7.2	Construc	ctor & 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 Doo	cumentation	85
		5.7.4.1	principalStrains	85
		5.7.4.2	shearStrains	85

CONTENTS vii

		5.7.4.3	x
5.8	Stress	Class Refe	erence
	5.8.1	Detailed [Description 8'
	5.8.2	Construct	for & Destructor Documentation
		5.8.2.1	Stress
		5.8.2.2	Stress
		5.8.2.3	Stress
	5.8.3	Member F	Function Documentation
		5.8.3.1	adjugate
		5.8.3.2	getPrincipalStresses
		5.8.3.3	getShearStresses
		5.8.3.4	getValue
		5.8.3.5	operator!
		5.8.3.6	operator*
		5.8.3.7	operator*
		5.8.3.8	operator*
		5.8.3.9	operator*=
		5.8.3.10	operator*=
		5.8.3.11	operator+
		5.8.3.12	operator+=
		5.8.3.13	operator
		5.8.3.14	operator-=
		5.8.3.15	operator~
		5.8.3.16	populateMatrix
		5.8.3.17	rotate
		5.8.3.18	setValue
		5.8.3.19	transpose 9
	5.8.4	Field Doc	umentation
		5.8.4.1	principalStresses
		5.8.4.2	shearStresses
		5.8.4.3	x
5.9	Vector	3d Class R	eference
	5.9.1	Detailed [Description
	5.9.2	Construct	for & Destructor Documentation
		5.9.2.1	Vector3d 96
		5.9.2.2	Vector3d 96
		5.9.2.3	Vector3d 96
	5.9.3	Member F	Function Documentation
		5.9.3.1	getValue
		5.9.3.2	getVector

viii CONTENTS

			5.9.3.3	magnitude	99
			5.9.3.4	normalize	00
			5.9.3.5	operator*	00
			5.9.3.6	operator*	00
			5.9.3.7	operator*=	01
			5.9.3.8	operator+	01
			5.9.3.9	operator+=	01
			5.9.3.10	operator	02
			5.9.3.11	operator-=	02
			5.9.3.12	operator [^]	02
			5.9.3.13	operator^=	03
			5.9.3.14	setValue	03
			5.9.3.15	setVector	03
			5.9.3.16	sum 10	03
		5.9.4	Field Doo	cumentation	04
			5.9.4.1	x	04
6	File	Docume	entation	11	05
•	6.1			Reference	
	0.1	6.1.1		Description	
		6.1.2		efinition Documentation	
		0.1.2	6.1.2.1	PI	
			6.1.2.2	SQRT2	
			6.1.2.3	SQRT3	
			6.1.2.4	SQRT5	
	6.2	defect		eference	
	0.2	6.2.1		Description	
	6.3	_		prence	
	0.0	6.3.1		Description	
	6.4			ile Reference	
		6.4.1		Description	
	6.5	-		Reference	
		6.5.1		Description	
	6.6	disloca		ts.h File Reference	12
		6.6.1		Description	13
		6.6.2		efinition Documentation	14
			6.6.2.1	DEFAULT_BURGERS_0	
			6.6.2.2	DEFAULT_BURGERS_1	
			6.6.2.3	DEFAULT_BURGERS_2	14
			6.6.2.4	DEFAULT_BURGERS_MAGNITUDE	14

CONTENTS

		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	disloca	tionSource	e.cpp File Reference	 115
	6.7.1	Detailed I	Description	 116
6.8	disloca	tionSource	e.h File Reference	 117
	6.8.1	Detailed I	Description	 118
6.9	disloca	tionSource	eDefaults.h File Reference	 118
	6.9.1	Detailed I	Description	 119
	6.9.2	Macro De	efinition Documentation	 120
		6.9.2.1	DEFAULT_NITERATIONS	 120
		6.9.2.2	DEFAULT_TAU_CRITICAL	 120
6.10	main.c	pp File Ref	ference	 120
	6.10.1	Function	Documentation	 121
		6.10.1.1	main	 121
6.11	mainpa	age.dox File	e Reference	 121
6.12	matrix3	33.cpp File	Reference	 122
	6.12.1	Detailed I	Description	 122
6.13	matrix3	33.h File R	deference	 122
	6.13.1	Detailed I	Description	 124
6.14	rotation	Matrix.cpp	p File Reference	 124
	6.14.1	Detailed I	Description	 125
6.15	rotation	nMatrix.h F	File Reference	 125
	6.15.1	Detailed I	Description	 127
6.16	slipPla	ne.cpp File	e Reference	 128
	6.16.1	Detailed I	Description	 128
6.17	slipPla	ne.h File R	Reference	 129
	6.17.1	Detailed I	Description	 130
6.18	slipPla	neDefaults	s.h File Reference	 130
	6.18.1	Detailed I	Description	 132
	6.18.2	Macro De	efinition 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

X CONTENTS

		6.18.2.7	DEFAULT	_SLIPF	PLANE	_NOF	MAL)	/EC1	OR_	_0 .	 	 	 	 	133
		6.18.2.8	DEFAULT	_SLIPF	PLANE	_NOF	MAL)	/EC1	OR_	_1 .	 	 	 	 	133
		6.18.2.9	DEFAULT	_SLIPF	PLANE	_NOF	MAL	/EC1	OR_	_2 .	 	 	 	 	133
		6.18.2.10	DEFAULT	_SLIPF	PLANE	_POS	ITION	1_0			 	 	 	 	133
		6.18.2.11	DEFAULT	_SLIPF	PLANE	_POS	ITION	I_1			 	 	 	 	133
		6.18.2.12	DEFAULT	_SLIPF	PLANE	_POS	ITION	_2			 	 	 	 	133
6.19	strain.c	pp File Re	ference .								 	 	 	 	133
	6.19.1	Detailed I	Description								 	 	 	 	134
6.20	strain.h	File Refer	ence								 	 	 	 	135
	6.20.1	Detailed I	Description								 	 	 	 	136
6.21	stress.c	cpp File Re	eference .								 	 	 	 	136
	6.21.1	Detailed I	Description								 	 	 	 	137
6.22	stress.h	n File Refe	rence								 	 	 	 	138
	6.22.1	Detailed I	Description								 	 	 	 	139
6.23	vector3	d.cpp File	Reference								 	 	 	 	140
	6.23.1	Detailed I	Description								 	 	 	 	140
6.24	vector3	d.h File Re	eference .								 	 	 	 	140
	6.24.1	Detailed I	Description								 	 	 	 	142

Index

142

Chapter 1

Main Page

The files in this program provide a heirarchical 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.

2 Main Page

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

Hierarchical Index

Chapter 3

Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

Class Defect representing a generic defect in a material	9
on Control of the Con	
Dislocation class representing a dislocation in the simulation	15
onSource	
DislocationSource class representing a source of dislocations in the simulation	27
Matrix33 class representing a 3x3 square matrix	39
Matrix Matrix	
RotationMatrix class to represent a rotation matrix	49
SlipPlane class representing a slip plane in the simulation	58
Strain class to represent the strain tensor	74
Stress class to represent the stress tensor	85
Vector3d class representing a single 3-dimensional vector in the simulation	96
	Dislocation class representing a dislocation in the simulation

6 **Data Structure Index**

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

05
07
08
10
11
12
15
17
18
20
22
22
24
25
28
29
30
33
35

8 File Index

stress.cp	op	
	Definition of the member functions if the Stress class	136
stress.h		
	Definition of the Stress class	138
vector3c	d.cpp	
	Definition of member functions and operators of the Vector3d class	140
vector3c	d.h	
	Definition of the Vector3d class	140

Chapter 5

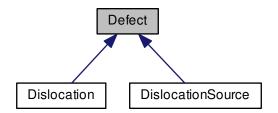
Data Structure Documentation

5.1 Defect Class Reference

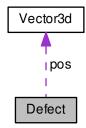
Class Defect representing a generic defect in a material.

#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.
У	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.

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 Defect Class Reference 13

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

а	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.
У	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 fiven 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 }
```

5.1.3.9 void Defect::setX (double x)

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 }
```

5.1.3.10 void Defect::setY (double y)

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 }
```

5.1.3.11 void Defect::setZ (double z)

Sets the Z-coordinate of the defect.

Parameters

```
z 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

р	Position vector of the the point where the stress field is to be calculated.
mu	Shear modulus in Pascals.
nu	Poisson's ratio.

Returns

Stress field value at the position p.

Reimplemented in Dislocation.

Definition at line 135 of file defect.h.

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:

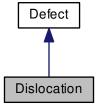
- defect.h
- · defect.cpp

5.2 Dislocation Class Reference

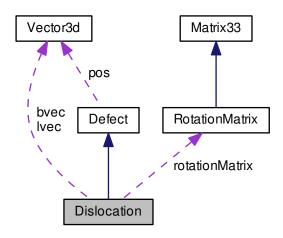
Dislocation class representing a dislocation in the simulation.

```
#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 Ivec

Line vector if 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.

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

burgers	Burgers vector.
line	Line vector.
position	Position of the dislocation.
bm	Magnitude of the Burgers vector in metres.
m	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
      globalSystem[0] = Vector3d (1.0, 0.0, 0.0);
globalSystem[1] = Vector3d (0.0, 1.0, 0.0);
124
125
      globalSystem[2] = Vector3d (0.0, 0.0, 1.0);
126
127
128
       // Vectors of the dislocation co-ordinate system
      localSystem[0] = bvec.normalize ();
localSystem[2] = lvec.normalize ();
localSystem[1] = (lvec ^ bvec).normalize ();
129
130
131
132
133
       // Calculate rotation matrix
134
      this->rotationMatrix = RotationMatrix (globalSystem, localSystem);
135
136
      // Release memory
      delete (globalSystem); globalSystem = NULL;
137
138
      delete (localSystem); localSystem = NULL;
```

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 coordinate 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 tensorm 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

sigma	The stress tensor, expressed in the global co-ordinate system.
tau_crss	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
      \ensuremath{//} Stress in the local co-ordinate system
      Stress sigmaLocal = sigma.rotate(this->rotationMatrix);
212
     Vector3d force:
213
214
      // Check for CRSS condition
215
      if (sigmaLocal.getValue(0,1) >= tau_crss)
          Vector3d force = sigma * ((this->bvec)^(this->lvec));
217
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.

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 min-Distance. This function calculates the ideal time increment for this dislocation to not collide with the defect.

Parameters

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

Returns

The ideal time increment for this dislocation.

Definition at line 232 of file dislocation.cpp.

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

```
247
      if (norm_p01 == 0.0)
248
           // The dislocation is lying on top of the obstacle - so it should not move
249
250
          return (0.0);
2.51
252
      else
253
254
           \ensuremath{//} Find out if the dislocation is approaching the defect or not
255
256
           // Velocities
           Vector3d v01 = v1 - v0;
257
          double norm_v01 = v01.magnitude();
258
           double dotProduct = v01 * p01;
259
260
           double cosine = dotProduct/(norm_v01 * norm_p01);
261
           if (cosine < 0.0)</pre>
262
               // The dislocation is approaching the other defect
return ( (norm_p01 - minDistance)/norm_v01 );
263
264
265
266
          else
267
               // They are diverging
268
               // So any time increment will do
269
               return (1000.0);
270
271
        }
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

```
burgers | Bergers 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

line 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

```
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 }
```

5.2.3.17 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

X	X-coordinate of the defect.
у	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 }
```

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

Sets the position of the defect.

The position of the defect is set to the position vector fiven 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

```
z 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

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

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

р	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.
mu	Shear modulus in Pascals.
nu	Poisson's ratio.

Returns

Stress tensor, expressed in the dislocation's local co-ordinate system.

Definition at line 177 of file dislocation.cpp.

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

Definition at line 32 of file dislocation.h.

```
5.2.4.4 bool Dislocation::mobile [protected]
```

Boolean term indicating mobility.

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

Definition at line 38 of file dislocation.h.

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

Position vector of the defect in 2D space.

Definition at line 26 of file defect.h.

5.2.4.6 RotationMatrix Dislocation::rotationMatrix [protected]

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

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

Definition at line 50 of file dislocation.h.

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

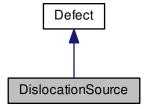
- · dislocation.h
- · dislocation.cpp

5.3 DislocationSource Class Reference

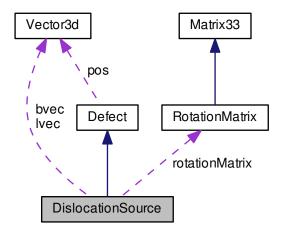
DislocationSource class representing a source of dislocations in the simulation.

#include <dislocationSource.h>

Inheritance diagram for DislocationSource:



Collaboration diagram for DislocationSource:



Public Member Functions

• DislocationSource ()

Default constructor.

- DislocationSource (Vector3d burgers, Vector3d line, Vector3d position, double bm, double tau, int nlter) Constructor that explicitly specifies all parameters.
- void setBurgers (Vector3d burgers)

Sets the Burgers vector of the dislocation.

void setLineVector (Vector3d line)

Sets the line vector of the dislocation.

• void setBurgersMagnitude (double bm)

Set the magnitude of the Burgers vector.

void setTauCritical (double tauC)

Set the critical shear stres for dipole emission.

• void setNumIterations (int nIter)

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 Ivec

Line vector if 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 nlterations

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.

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

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

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

Parameters

burgers	Burgers vector.
line	Line vector.
position	Position of the dislocation.
bm	Magnitude of the Burgers vector in metres.
tau	Critical shear stress value.
nlter	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;
                   = line;
     this->lvec
41
    this->pos
                   = position;
     this->bmag
                   = bm;
    this->tauCritical = tau;
this->nIterations = nIter;
45
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

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

Returns

The length of the dislocation dipole.

Definition at line 167 of file dislocationSource.cpp.

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

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 }
```

5.3.3.8 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 nlterations. When the counter variable countlterations 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

burgers | 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

bm | 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

line 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 nlter)

Set the number of iterations before a dipole is emitted.

Parameters

nlter 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

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 }
```

5.3.3.21 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

X	X-coordinate of the defect.
У	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 }
```

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

Sets the position of the defect.

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

Parameters

а	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 stres for dipole emission.

Parameters

```
tauC | 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

```
x 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

```
y 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

```
z 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

р	Position vector of the the point where the stress field is to be calculated.
mu	Shear modulus in Pascals.
nu	Poisson's ratio.

Returns

Stress field value at the position p.

Reimplemented in Dislocation.

Definition at line 135 of file defect.h.

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 nlterations, and this variable countlterations 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::nlterations [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 nlterations.

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:

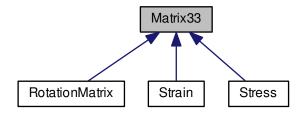
- · dislocationSource.h
- · dislocationSource.cpp

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 \sim () 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 mstrix with data present in corresponding elements of the provided 3x3 array.

Parameters

```
a Pointer to the two-dimensional 3x3 array.
```

Definition at line 35 of file matrix33.cpp.

```
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

a The vector whose dyadic product results in the matrix.

Definition at line 53 of file matrix33.cpp.

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

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

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

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

Parameters

а	First vector.
b	Second vector.

Definition at line 72 of file matrix33.cpp.

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

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

row	Row index of the element.
column	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     return (0.0);
121     return (0.0);
```

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.

```
376
     Matrix33 r; // Result matrix
377
378
     double determinant = ~(*this);
379
     if (determinant == 0.0)
380
381
382
         // The matrix is non-invertible
383
         return (r); // Zero matrix
384
385
     // If we are still here, the matrix is invertible
386
387
     // Transpose
388
389
     Matrix33 tr = this->transpose();
390
     // Find Adjugate matrix
391
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;
      Matrix33 r;
256
257
258
      for (i=0; i<3; i++)</pre>
259
          for (j=0; j<3; j++)</pre>
260
261
               r.setValue(i, j, (this->x[i][j] * p));
2.62
263
264
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    {
```

```
for (j=0; j<3; j++)</pre>
300
             s = 0.0;
301
             for (k=0; k<3; k++)
302
303
                 s += this->x[i][k] * p.getValue(k,j);
304
306
             r.setValue (i, j, s);
307
      }
308
309
     return (r);
310
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 {
     Vector3d r(0.0, 0.0, 0.0);
335
336
     double s;
337
     int i, j;
338
     for (i=0; i<3; i++)</pre>
340
341
         s = 0.0;
342
        for (j=0; j<3; j++)
343
             s += this->x[i][j] * v.getValue(j);
344
345
     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.

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

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

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 {
      int i, j;
216
      Matrix33 r;
217
218
219
      for (i=0; i<3; i++)</pre>
220
          for (j=0; j<3; j++)</pre>
221
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
      for (i=0; i<3; i++)</pre>
238
239
240
          for (j=0; j<3; j++)</pre>
241
            {
242
               this->x[i][j] -= p.getValue(i, j);
243
244
        }
245 }
```

5.4.3.13 double Matrix33::operator \sim () 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
                                       x[1][2]);
                                                                       this-\times[0][1] * ( (this-\times[1][2]*this-\times[2][0]) - (this-\times[1][0]*this-\times
363
                                       x[2][2]));
364
                                    d += this -> x[0][2] * ( (this -> x[1][0] * this -> x[2][1]) - (this -> x[2][0] * this -> x[2][0] *
                                       x[1][1]));
365
                                      return (d);
366
367 }
```

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

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

Definition at line 93 of file matrix33.cpp.

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 viceversa.

Returns

The transpose of the present matrix.

Definition at line 153 of file matrix33.cpp.

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

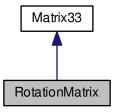
- matrix33.h
- · matrix33.cpp

5.5 RotationMatrix Class Reference

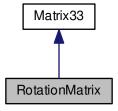
RotationMatrix class to represent a rotation matrix.

#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.

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.

```
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

m 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

unPrimed	Pointer to the array containing the three axes vectors of the unprimed (old) system.
primed	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 {
        Matrix33 adj;
131
132
       adj.setValue(0, 0, ((this->x[1][1]*this->x[2][2]) - (this->x[1][2]*this->x[2][1])));
adj.setValue(0, 1, ((this->x[1][2]*this->x[2][0]) - (this->x[1][0]*this->x[2][2])));
adj.setValue(0, 2, ((this->x[1][0]*this->x[2][1]) - (this->x[1][1]*this->x[2][0])));
133
134
135
136
137
        adj.setValue(1, 0, ((this->x[2][1]*this->x[0][2]) - (this->x[0][1]*this->x[2][2])));
adj.setValue(1, 1, ((this->x[2][2]*this->x[0][0]) - (this->x[2][0]*this->x[0][2])));
138
139
        adj.setValue(1, 2, ((this->x[2][0]*this->x[0][1]) - (this->x[2][1]*this->x[0][0])));
140
       141
142
143
144
145
       return (adj);
```

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

row	Row index of the element.
column	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
     if (determinant == 0.0)
380
381
382
         // The matrix is non-invertible
383
         return (r); // Zero matrix
384
385
     // If we are still here, the matrix is invertible
386
387
     // Transpose
388
389
     Matrix33 tr = this->transpose();
390
     // Find Adjugate matrix
391
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.

```
255
      int i, j;
      Matrix33 r;
256
257
258
      for (i=0; i<3; i++)</pre>
259
260
          for (j=0; j<3; j++)</pre>
261
               r.setValue(i, j, (this->x[i][j] * p));
2.62
263
264
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    {
```

```
for (j=0; j<3; j++)
300
             s = 0.0;
301
             for (k=0; k<3; k++)</pre>
302
303
                  s += this->x[i][k] * p.getValue(k,j);
304
306
             r.setValue (i, j, s);
307
      }
308
309
     return (r);
310
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 {
     Vector3d r(0.0, 0.0, 0.0);
335
336
     double s;
337
     int i, j;
338
     for (i=0; i<3; i++)</pre>
340
341
         s = 0.0;
342
        for (j=0; j<3; j++)
343
             s += this->x[i][j] * v.getValue(j);
344
345
     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.

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

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 {
      int i, j;
216
217
      Matrix33 r;
218
219
      for (i=0; i<3; i++)</pre>
220
221
          for (j=0; j<3; j++)</pre>
222
               r.setValue(i, j, (this->x[i][j] - p.getValue(i, j)));
223
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
      for (i=0; i<3; i++)</pre>
238
239
240
           for (j=0; j<3; j++)</pre>
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
                                       x[1][2]);
                                                                       this-\times[0][1] * ( (this-\times[1][2]*this-\times[2][0]) - (this-\times[1][0]*this-\times
363
                                       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
                                      return (d);
366
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

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

Definition at line 93 of file matrix33.cpp.

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 viceversa.

Returns

The transpose of the present matrix.

Definition at line 153 of file matrix33.cpp.

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:

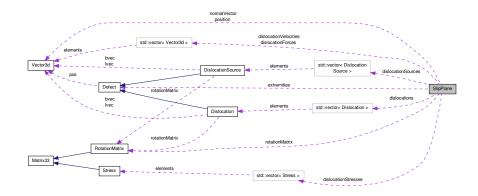
- · rotationMatrix.h
- · rotationMatrix.cpp

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 dislocation-Stresses.

void calculateDislocationForces (double tau_crss)

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

• void calculate Velocities (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^{\wedge} 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.
     Vector3d pos(DEFAULT_SLIPPLANE_POSITION_0,
DEFAULT_SLIPPLANE_POSITION_1,
2.0
                   DEFAULT_SLIPPLANE_POSITION_2);
     Vector3d normal(DEFAULT_SLIPPLANE_NORMALVECTOR_0,
24
                       DEFAULT_SLIPPLANE_NORMALVECTOR_1
2.5
                       DEFAULT_SLIPPLANE_NORMALVECTOR_2);
2.6
     Vector3d ends[2];
     ends[0] = Vector3d(DEFAULT_SLIPPLANE_EXTREMITY1_0,
27
28
                          DEFAULT_SLIPPLANE_EXTREMITY1_1,
                          DEFAULT_SLIPPLANE_EXTREMITY1_2);
29
30
     ends[1] = Vector3d(DEFAULT_SLIPPLANE_EXTREMITY2_0,
                          DEFAULT_SLIPPLANE_EXTREMITY2_1,
DEFAULT_SLIPPLANE_EXTREMITY2_2);
31
32
     std::vector<Dislocation> dislocationList(1, Dislocation());
33
34
     std::vector<DislocationSource> dislocationSourceList(1, DislocationSource());
     *this = SlipPlane(ends, normal, pos, dislocationList, dislocationSourceList);
```

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

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

dislocation- A vector container of type DislocationSource containing the dislocation sources lying on this slip plane.

Definition at line 48 of file slipPlane.cpp.

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

```
tau_crss | 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
      s = this->dislocationStresses.begin();
351
352
      f = this->dislocationForces.begin();
353
354
       for (d = this->dislocations.begin(); d!=this->dislocations.end(); d++)
355
           *f = d->forcePeachKoehler (*s, tau_crss);
356
357
          s++:
358
           f++;
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 dislocation-Stresses.

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

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

Definition at line 313 of file slipPlane.cpp.

```
314
315
       std::vector<Dislocation>::iterator d1; // Iterator for each dislocation
      std::vector<Dislocation>::iterator d2; // Nested iterator
316
317
                                               // Iterator for the Stress
       std::vector<Stress>::iterator s;
318
319
                                               // 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
        }
```

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.

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

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

minDistance	Minimum distance of approach between dislocations having Burgers vectors of the same sign.
minDt	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
       // Get the number of dislocations
int nDisl = this->dislocations.size();
418
419
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_p01;
427
428
       // Velocity vectors
429
       Vector3d v0, v1;
430
       double norm_v01;
431
                       // Counter for the loop
432
       int i;
433
       double t1, t2;
434
       double dtMin; // Minimum time increment
435
436
       // For the first dislocation, the time increment has to be calculated
437
       \ensuremath{//} for approach to both a dislocation and the slip plane extremity.
       // Time for slip plane extremity
438
       t1 = this->dislocations[0].idealTimeIncrement(this->
439
      dislocationVelocities[0],
440
                                                          minDistance,
441
                                                          this->extremities[0],
442
                                                          Vector3d(0.0, 0.0, 0.0));
       t2 = this->dislocations[0].idealTimeIncrement(this->
443
      dislocationVelocities[0],
444
                                                          minDistance,
445
                                                          this->dislocations[1],
446
                                                          this->dislocationVelocities[1]);
447
       \ensuremath{//} Choose the smaller of the two
       timeIncrement[0] = t1 < t2 ? t1:t2;
448
       if (timeIncrement[0] < minDt)</pre>
449
450
451
            // This dislocation should not move in this iteration because it might collide with the next defect
452
            timeIncrement[0] = minDt;
453
            this->dislocationVelocities[0] = Vector3d(0.0, 0.0, 0.0);
454
            // The other defect is a slip plane extremity // This dislocation will not move any more
455
456
457
            this->dislocations[0].setPinned();
458
459
460
       for (i=1; i<(nDisl-1); i++)</pre>
461
           t1 = this->dislocations[i].idealTimeIncrement(this->
462
      dislocationVelocities[i],
463
464
                                                               this->dislocations[i-1],
465
                                                              this->dislocationVelocities[i-1])
466
            t2 = this->dislocations[i].idealTimeIncrement(this->
      dislocationVelocities[i],
467
                                                              minDistance,
468
                                                               this->dislocations[i+1],
469
                                                              this->dislocationVelocities[i+1])
470
            timeIncrement[i] = t1 < t2 ? t1:t2;</pre>
471
472
            if (timeIncrement[i] < minDt)</pre>
473
474
                // This dislocation should not move in this iteration because it might collide with the next
       defect
475
                timeIncrement[i] = minDt;
476
                this->dislocationVelocities[i] = Vector3d(0.0, 0.0, 0.0);
```

```
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.
       \ensuremath{//} Time for slip plane extremity
482
       i=nDisl-1;
483
484
       t1 = this->dislocations[i].idealTimeIncrement(this->
      dislocationVelocities[i],
                                                        minDistance,
185
486
                                                        this->extremities[1]
                                                        Vector3d(0.0, 0.0, 0.0));
487
       t2 = this->dislocations[i].idealTimeIncrement(this->
488
      dislocationVelocities[i],
489
490
                                                        this->dislocations[i-1],
491
                                                       this->dislocationVelocities[i-1]);
       // Choose the smaller of the two
492
493
       timeIncrement[i] = t1 < t2 ? t1:t2;</pre>
494
495
       if (timeIncrement[i] < minDt)</pre>
496
497
           // This dislocation should not move in this iteration because it might collide with the next defect
498
           timeIncrement[i] = minDt;
           this->dislocationVelocities[i] = Vector3d(0.0, 0.0, 0.0);
499
500
501
           // The other defect is a slip plane extremity
           // This dislocation will not move any more
502
503
           this->dislocations[i].setPinned();
504
505
506
       dtMin = 1000;
507
       for (i=0; i<nDisl; i++)</pre>
508
509
           if (timeIncrement[i] < dtMin)</pre>
510
               dtMin = timeIncrement[i];
511
512
513
      this->dt = dtMin;
515
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
      std::vector<Dislocation>::iterator d; // Iterator for dislocations
369
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);
               norm_v = v->magnitude();
386
387
               if (norm v > 0.0)
388
389
390
                   // Project the velocity on to the slip plane line
391
                   p0 = this->extremities[0].getPosition();
                   p1 = this->extremities[1].getPosition();
p01 = p1 - p0;
392
393
                   norm_p01 = p01.magnitude();
394
395
396
                   cosine = ((*v) * p01)/(norm_v * norm_p01);
```

```
(*v) *= cosine;
398
399
400
           else
401
           {
               *v = Vector3d(0.0, 0.0, 0.0);
402
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

pos	Position vector of the point whose distance is to be calculated.
n	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^{\uparrow} th extremity of the slip plane.

Definition at line 547 of file slipPlane.cpp.

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 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 {
      Vector3d axis;
244
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;
        Vector3d *e2 = new Vector3d;
256
2.57
258
        *e1 = this->extremities[0].getPosition();
259
        *e2 = this->extremities[1].getPosition();
        axis = ((*e2) - (*e1));
260
261
       delete(e1); e1 = NULL;
delete(e2); e2 = NULL;
262
263
      }
264
265
266
      <u>if</u> (i==1)
267
      {
       // Y-axis = Z x X
268
269
       axis = this->getAxis(2) ^ this->getAxis(0);
270
271
      return ( axis.normalize() );
```

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	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.
d	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	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.
dSource	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
     {
       *dSource = this->dislocationSources[i];
       return (true);
203
204
     else
2.0.5
     {
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 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

points	STL vector container with position vectors (Vector3d) of points at which the stress field is to be
	calculated.
appliedStress	The externally applied stress (in the global co-ordinate system).
mu	Shear modulus of the material in Pa.
nu	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
      std::vector<Stress>::iterator s = stressVector.begin();
609
610
611
      // Temporary variable for stress
612
613
     while (p != points.end())
614
615
         sTemp = appliedStress;
616
          // Iterator for the dislocations
617
618
          std::vector<Dislocation>::iterator d = this->dislocations.begin();
619
          while (d != this->dislocations.end())
620
              sTemp += d->stressField (*p, mu, nu);
621
622
             d++;
           }
623
624
         *s = sTemp;
625
626
         s++;
62.7
         p++;
62.8
629
     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

points	STL vector container with position vectors (Vector3d) of points at which the stress field is to be
	calculated.
appliedStress	The externally applied stress (in the global co-ordinate system).
mu	Shear modulus of the material in Pa.
nu	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
     while (p != points.end())
656
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
              sTemp += d->stressField (*p, mu, nu);
663
664
              d++;
```

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.

```
523
      std::vector<Dislocation>::iterator d;
524
      std::vector<Vector3d>::iterator v;
525
      Vector3d p;
526
     d = this->dislocations.begin();
52.7
528
     v = this->dislocationVelocities.begin();
529
530
      while (d != this->dislocations.end())
531
          p = d->getPosition();
532
         p += (*v) * (this->dt);
d->setPosition(p);
533
534
535
536
537
          v++;
       }
538
539 1
```

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

Set the list of dislocations of the slip plane.

Parameters

dislocationList 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

dislocation- 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

ends 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

```
normal | 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

pos 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;
570
      for (i=0; i<nDisl-1; i++)</pre>
571
572
          for (j=i+1; j<nDisl; j++)</pre>
573
574
              pi = this->dislocations[i].getPosition();
575
              di = this->distanceFromExtremity(pi, 0);
576
577
              pj = this->dislocations[j].getPosition();
              dj = this->distanceFromExtremity(pj, 0);
578
579
              if (dj < di)</pre>
580
581
582
                  // Swap the two
583
                  temp = this->dislocations[i];
                  this->dislocations[i] = this->dislocations[j];
584
                  this->dislocations[j] = temp;
585
586
            }
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 Peah-Koehler force experienced by each dislocation. They are calculated in each iteration by thefunction 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.

```
\textbf{5.6.4.5} \quad \textbf{std::vector} < \textbf{Vector3d} > \textbf{SlipPlane::dislocationVelocities} \quad \texttt{[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

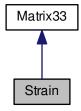
5.7 Strain Class Reference

Strain class to represent the strain tensor.

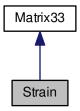
#include <strain.h>

5.7 Strain Class Reference 75

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.

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.

5.7 Strain Class Reference 77

```
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 contstructed using them.

Parameters

principal	Pointer to the array containing principal strains.
shear	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

```
m Matrix33 variable containing the full strain tensor.
```

Definition at line 53 of file strain.cpp.

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

```
this->principalStrains [0] = m.getValue(0,0);
         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);
82
     else
83
         // The matrix is asymmetrical
84
         // A zero matrix will be returned
85
         for (i=0; i<3; i++)</pre>
86
88
             this->principalStrains[i] = 0.0;
89
             this->shearStrains[i] = 0.0;
90
      }
91
92
    this->populateMatrix ();
```

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
      adj.setValue(0, 0, ((this->x[1][1]*this->x[2][2]) - (this->x[1][2]*this->x[2][1])));
adj.setValue(0, 1, ((this->x[1][2]*this->x[2][0]) - (this->x[1][0]*this->x[2][2])));
133
134
      adj.setValue(0, 2, ((this->x[1][0]*this->x[2][1]) - (this->x[1][1]*this->x[2][0])));
135
136
      137
138
139
140
141
      {\tt 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])));
      adj.setValue(2, 2, ((this->x[0][0]*this->x[1][1]) - (this->x[1][0]*this->x[0][1])));\\
143
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.

5.7 Strain Class Reference 79

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.

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

row	Row index of the element.
column	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 {
```

```
Matrix33 r; // Result matrix
377
378
     double determinant = ~(*this);
379
380
     if (determinant == 0.0)
381
         // The matrix is non-invertible
382
383
         return (r);
384
385
     // If we are still here, the matrix is invertible
386
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 {
      int i, j;
Matrix33 r;
255
256
257
258
      for (i=0; i<3; i++)</pre>
260
         for (j=0; j<3; j++)
261
262
               r.setValue(i, j, (this->x[i][j] * p));
263
       }
264
265
      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++)
```

5.7 Strain Class Reference 81

```
301
            s = 0.0;
             for (k=0; k<3; k++)
302
303
                 s += this->x[i][k] * p.getValue(k,j);
304
305
            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 {
     Vector3d r(0.0, 0.0, 0.0);
335
336
     double s;
337
     int i, j;
339
     for (i=0; i<3; i++)</pre>
340
         s = 0.0:
341
         for (j=0; j<3; j++)
342
343
             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.

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 {
      int i, j;
Matrix33 r;
177
178
179
180
      for (i=0; i<3; i++)</pre>
181
          for (j=0; j<3; j++)</pre>
182
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.

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.

5.7 Strain Class Reference 83

Returns

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

Definition at line 214 of file matrix33.cpp.

```
215 {
      int i, j;
216
      Matrix33 r;
217
218
219
      for (i=0; i<3; i++)</pre>
220
221
          for (j=0; j<3; j++)</pre>
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
      for (i=0; i<3; i++)</pre>
238
239
240
           for (j=0; j<3; j++)</pre>
241
             {
242
               this->x[i][j] -= p.getValue(i, j);
243
244
245 }
```

5.7.3.15 double Matrix33::operator \sim () 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
                                       x[1][2]);
                                                                       this-\times[0][1] * ( (this-\times[1][2]*this-\times[2][0]) - (this-\times[1][0]*this-\times
363
                                       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
                                      return (d);
366
367 }
```

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

```
alpha Rotation matrix.
```

Returns

Rotated strain tensor.

Definition at line 141 of file strain.cpp.

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

row	Row index of the element.
column	Column index of the element.
value	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)
```

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 viceversa.

Returns

The transpose of the present matrix.

Definition at line 153 of file matrix33.cpp.

```
154 {
      Matrix33 tr;
156
      int i, j;
157
      for (i=0; i<3; i++)</pre>
158
159
          for (j=0; j<3; j++)</pre>
160
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:

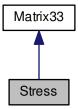
- strain.h
- strain.cpp

5.8 Stress Class Reference

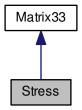
Stress class to represent the stress tensor.

#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 contstructed using them.

Parameters

principal	Pointer to the array containing principal stresses.
shear	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
5    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

```
m Matrix33 variable containing the full stress tensor.
```

Definition at line 53 of file stress.cpp.

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

```
this->principalStresses [0] = m.getValue(0,0);
         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);
82
     else
83
         // The matrix is asymmetrical
84
         // A zero matrix will be returned
85
         for (i=0; i<3; i++)</pre>
86
88
             this->principalStresses[i] = 0.0;
89
             this->shearStresses[i] = 0.0;
90
      }
91
92
    this->populateMatrix ();
```

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
      adj.setValue(0, 0, ((this->x[1][1]*this->x[2][2]) - (this->x[1][2]*this->x[2][1])));
adj.setValue(0, 1, ((this->x[1][2]*this->x[2][0]) - (this->x[1][0]*this->x[2][2])));
133
134
      adj.setValue(0, 2, ((this->x[1][0]*this->x[2][1]) - (this->x[1][1]*this->x[2][0])));
135
136
      137
138
139
140
141
      {\tt 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])));
      adj.setValue(2, 2, ((this->x[0][0]*this->x[1][1]) - (this->x[1][0]*this->x[0][1])));\\
143
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.

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.

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

row	Row index of the element.
column	Column index of the element.

Returns

Value of the element located at the given position.

Definition at line 111 of file matrix33.cpp.

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 {
```

```
Matrix33 r; // Result matrix
377
378
     double determinant = ~(*this);
379
380
     if (determinant == 0.0)
381
         // The matrix is non-invertible
382
383
         return (r);
384
385
     // If we are still here, the matrix is invertible
386
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;
      Matrix33 r;
256
257
258
      for (i=0; i<3; i++)</pre>
260
          for (j=0; j<3; j++)</pre>
261
262
               r.setValue(i, j, (this->x[i][j] * p));
263
       }
264
265
      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++)
```

```
301
             s = 0.0;
             for (k=0; k<3; k++)
302
303
                 s += this->x[i][k] * p.getValue(k,j);
304
305
             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 {
     Vector3d r(0.0, 0.0, 0.0);
335
336
     double s;
337
     int i, j;
339
     for (i=0; i<3; i++)</pre>
340
         s = 0.0:
341
         for (j=0; j<3; j++)</pre>
342
343
              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 {
      int i, j;
Matrix33 r;
177
178
179
180
      for (i=0; i<3; i++)</pre>
181
          for (j=0; j<3; j++)</pre>
182
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 {
      int i, j;
216
217
      Matrix33 r;
218
219
      for (i=0; i<3; i++)</pre>
220
221
           for (j=0; j<3; j++)</pre>
222
               r.setValue(i, j, (this->x[i][j] - p.getValue(i, j)));
223
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
      for (i=0; i<3; i++)</pre>
238
239
240
           for (j=0; j<3; j++)</pre>
241
             {
242
               this->x[i][j] -= p.getValue(i, j);
243
244
245 }
```

5.8.3.15 double Matrix33::operator \sim () 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
                                       x[1][2]);
                                                                       this-\times[0][1] * ( (this-\times[1][2]*this-\times[2][0]) - (this-\times[1][0]*this-\times
363
                                       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
                                      return (d);
366
367 }
```

5.8 Stress Class Reference 95

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

```
alpha Rotation matrix.
```

Returns

Rotated stress tensor.

Definition at line 142 of file stress.cpp.

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

row	Row index of the element.
column	Column index of the element.
value	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)
```

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 viceversa.

Returns

The transpose of the present matrix.

Definition at line 153 of file matrix33.cpp.

```
154 {
      Matrix33 tr;
155
      int i, j;
156
157
158
      for (i=0; i<3; i++)</pre>
           for (j=0; j<3; j++)</pre>
160
161
               tr.setValue (i, j, this->x[j][i]);
162
163
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*= (const double &)

Operator for reflexive scaling of the vector by a scalar.

• double operator* (const Vector3d &) const

Operator for the scalar product of two vectors.

Vector3d operator[∧] (const Vector3d &) const

Operator for the vector product of two vectors.

void operator[^]= (const Vector3d &)

Operator for reflexive vector product of two vectors.

Protected Attributes

double x [3]

The elements of the vector.

5.9.1 Detailed Description

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

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

Definition at line 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

a 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

a1	Value of the first element of the vector.
a2	Value of the second element of the vector.
а3	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;
```

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

```
index 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.

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.

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 {
     Vector3d r(0.0, 0.0, 0.0);
241
242
      int i:
243
244
      for (i=0; i<3; i++)</pre>
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.

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.

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.

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.

5.9.3.13 void Vector3d::operator $^{\wedge}$ = (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.

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

index	Index of the element whose value is to be set.
value	Value that is to be given to the element.

Definition at line 56 of file vector3d.cpp.

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 if the vector to values in the array pointed to by the argument a.

Parameters

```
a 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.

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

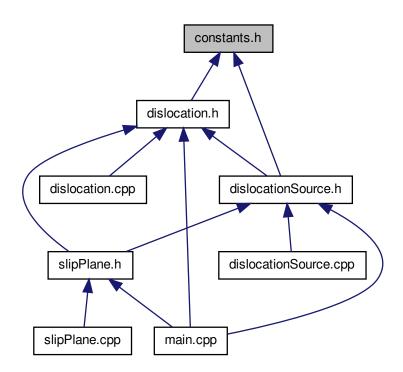
Chapter 6

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

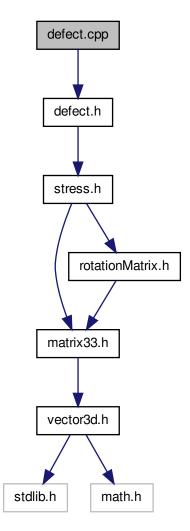
The square root of 5.

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

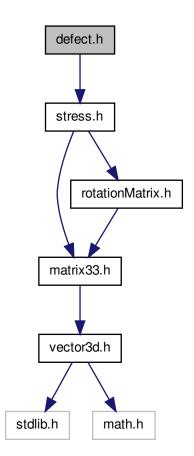
04/06/2013

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

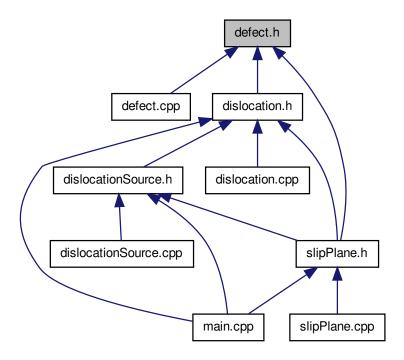
Definition of the **Defect** class.

#include "stress.h"
Include dependency graph for defect.h:



6.3 defect.h File Reference

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.

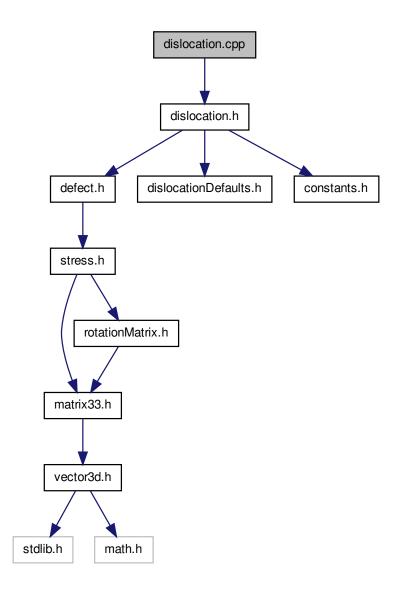
Definition in file defect.h.

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.

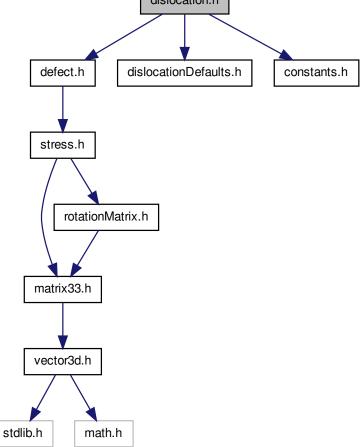
Definition in file dislocation.cpp.

6.5 dislocation.h File Reference

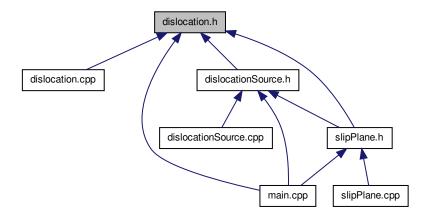
Definition of the Dislocation class.

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

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

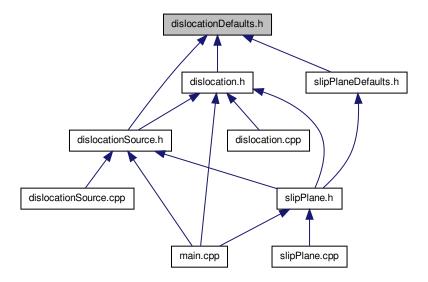
This file defines the Dislocation class representing a dislocation in the simulation. This class inherits from the Defect class.

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.		

6.6.2.10 #define DEFAULT_POSITION_2 0.0

Default value of the position vector z-coordinate.

Definition at line 20 of file dislocationDefaults.h.

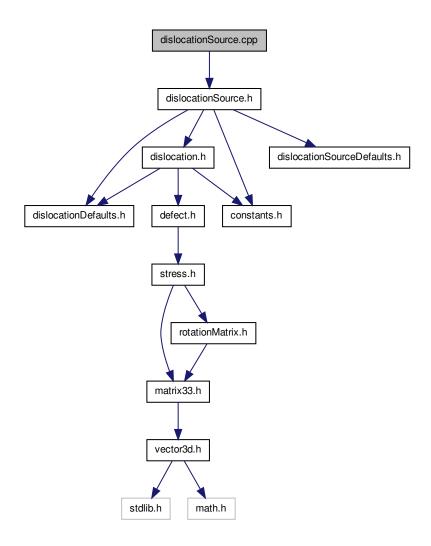
Definition at line 24 of file dislocationDefaults.h.

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

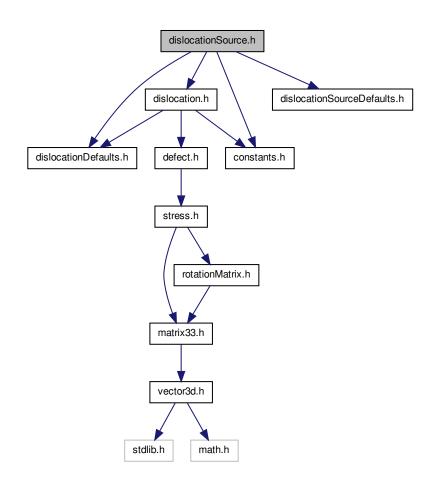
This file defines the member functions of the DislocationSource class representing a source of dislocations in the simulation. This class inherits from the Defect class. This object is basically the representation of a Frank-Read source emitting dislocation dipoles. When the dislocation source experiences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

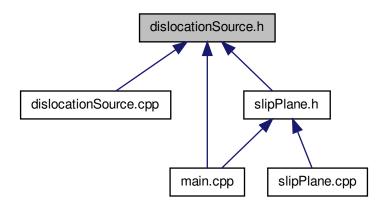
6.8 dislocationSource.h File Reference

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

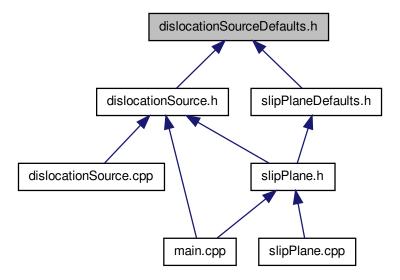
This file defines the DislocationSource class representing a source of dislocations in the simulation. This class inherits from the Defect class. This object is basically the representation of a Frank-Read source emitting dislocation dipoles. When the dislocation source experiences a shear stress greater 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.

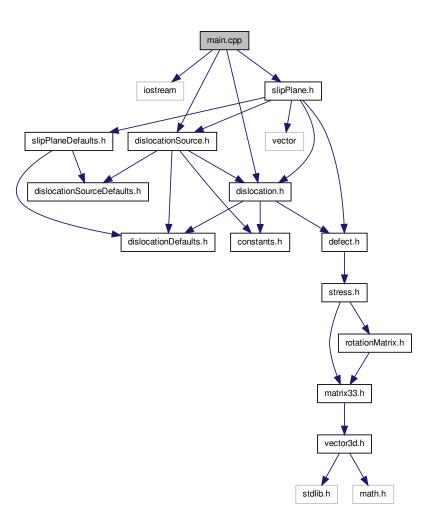
Default value of the critical shear stress for a dislocation source to emit a dipole. The number is expressed in Pa.

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.

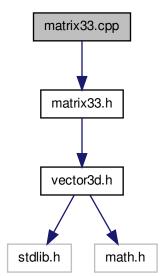
```
8 {
9     return (0);
10 }
```

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

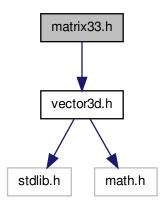
This file defines the member functions and operators of the Matrix33 class representing a 3x3 matrix in the simulation

Definition in file matrix33.cpp.

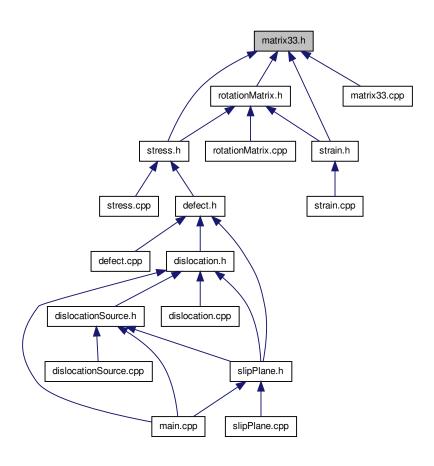
6.13 matrix33.h File Reference

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:



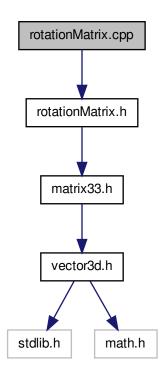
124 **File Documentation Data Structures** • class Matrix33 Matrix33 class representing a 3x3 square matrix. 6.13.1 Detailed Description Definition of the Matrix33 class. **Author** Adhish Majumdar Version 1.0 Date 04/06/2013 This file defines the Matrix33 class representing a 3x3 matrix in the simulation.

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

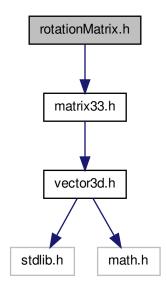
25/04/2013

This file defines member functions of the RotationMatrix class for carrying out 3D rotations and axes transformations. Definition in file rotationMatrix.cpp.

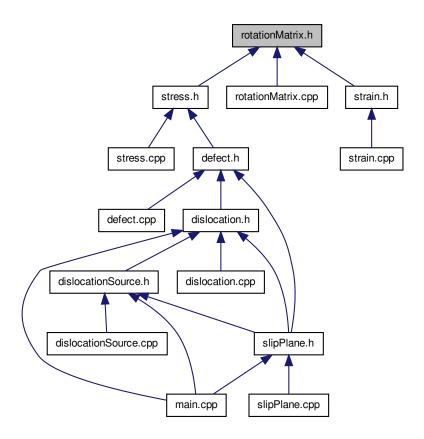
6.15 rotationMatrix.h File Reference

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

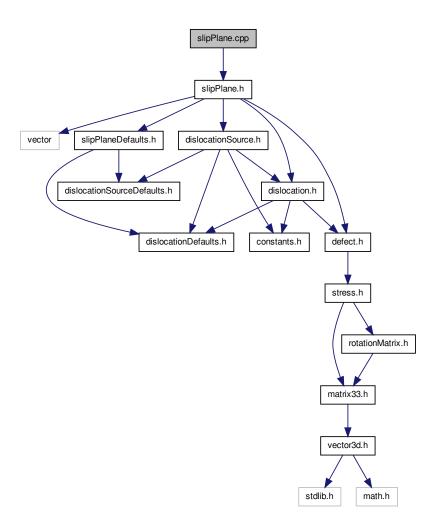
25/04/2013

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

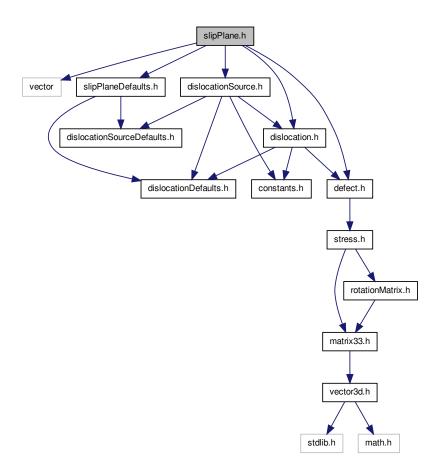
This file defines the member functions of the SlipPlane class.

Definition in file slipPlane.cpp.

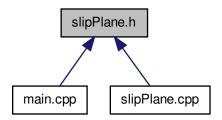
6.17 slipPlane.h File Reference

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.

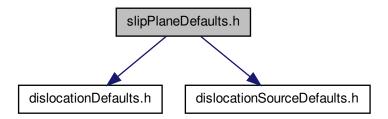
Definition in file slipPlane.h.

6.18 slipPlaneDefaults.h File Reference

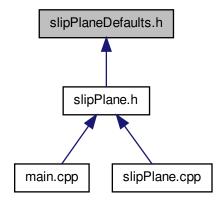
Definition of certain default values for members of the SlipPlane class.

```
#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

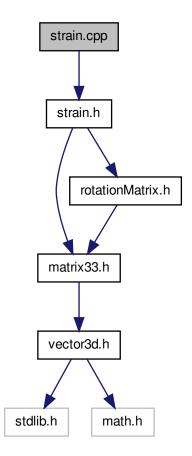
Default value of the position vector z-coordinate.

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 if 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.

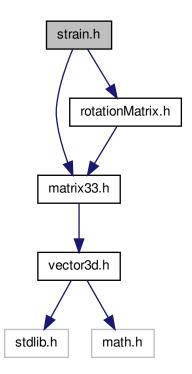
Definition in file strain.cpp.

6.20 strain.h File Reference

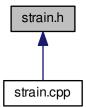
6.20 strain.h File Reference

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

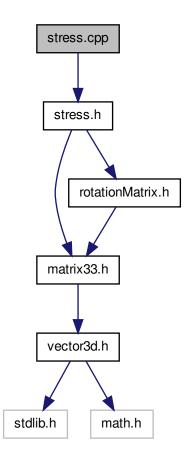
• class Strain

136 File Documentation 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 if the Stress class.

Author

Adhish Majumdar

Version

1.0

Date

05/06/2013

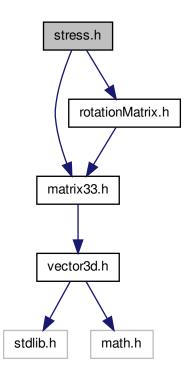
This file defines the member functions of the Stress class for the stress tensor.

Definition in file stress.cpp.

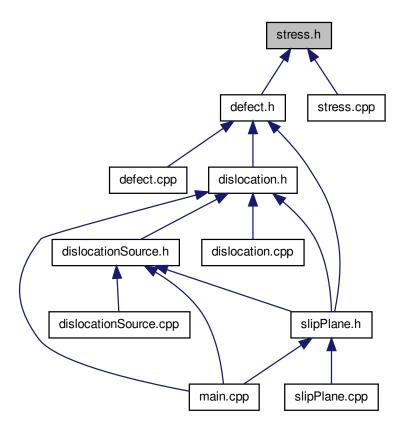
6.22 stress.h File Reference

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

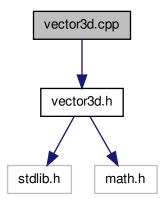
This file defines the Stress class for the stress tensor.

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.

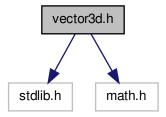
Definition in file vector3d.cpp.

6.24 vector3d.h File Reference

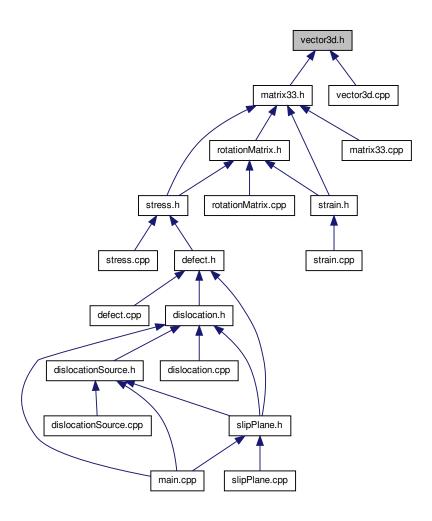
Definition of the Vector3d class.

```
#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	getY, 12
Matrix33, 42	getZ, 12
RotationMatrix, 51	pos, 15
Strain, 78	setPosition, 13
Stress, 89	setX, 14
,	setY, 14
bmag	setZ, 14
Dislocation, 26	stressField, 14
DislocationSource, 38	defect.cpp, 107
bvec	defect.h, 108
Dislocation, 26	dipoleNucleationLength
DislocationSource, 38	DislocationSource, 31
	Dislocation, 15
calculateDislocationForces	bmag, 26
SlipPlane, 61	bvec, 26
calculateDislocationStresses	calculateRotationMatrix, 18
SlipPlane, 61	Dislocation, 18
calculateRotationMatrix	forcePeachKoehler, 19
Dislocation, 18	getBurgers, 19
SlipPlane, 62	getLineVector, 19
calculateTimeIncrement	getPosition, 20
SlipPlane, 62	getX, 20
calculateVelocities	getY, 20
SlipPlane, 64	getZ, 21
constants.h, 105	idealTimeIncrement, 21
PI, 106	isMobile, 22
SQRT2, 106	lvec, 26
SQRT3, 106	mobile, 26
SQRT5, 106	pos, 26
countIterations	rotationMatrix, 26
DislocationSource, 38	setBurgers, 22
DEEALUT BURGERO O	setLineVector, 22
DEFAULT_BURGERS_0	setMobile, 23
dislocationDefaults.h, 114	setPinned, 23
DEFAULT_BURGERS_1	setPosition, 23, 24
dislocationDefaults.h, 114	setX, 24
DEFAULT_BURGERS_2	setX, 24
dislocationDefaults.h, 114	setZ, 24
DEFAULT_NITERATIONS	stressField, 25
dislocationSourceDefaults.h, 120 DEFAULT POSITION 0	stressFieldLocal, 25
dislocationDefaults.h, 115	dislocation.cpp, 110
DEFAULT_POSITION_1	dislocation.h, 111
dislocationDefaults.h, 115	dislocationDefaults.h, 112
DEFAULT_POSITION_2	DEFAULT_BURGERS_0, 114
dislocationDefaults.h, 115	DEFAULT_BURGERS_1, 114
Defect, 9	DEFAULT BURGERS 2, 114
Defect, 9 Defect, 11	DEFAULT POSITION 0, 115
getPosition, 11, 12	DEFAULT_POSITION_1, 115
getX, 12	DEFAULT_POSITION_1, 115
you, ie	DEI AULI_I UUITIUN_2, 113

dislocationForces	getAxis
SlipPlane, 73	SlipPlane, 65
DislocationSource, 27	getBurgers
bmag, 38	Dislocation, 19
bvec, 38	DislocationSource, 31
countIterations, 38	getBurgersMag
dipoleNucleationLength, 31	DislocationSource, 31
DislocationSource, 30	getDislocation
DislocationSource, 30	SlipPlane, 66
getBurgers, 31	getDislocationList
getBurgersMag, 31	SlipPlane, 66
getIterationCount, 32	getDislocationSource
getLineVector, 32	SlipPlane, 66
getNumIterations, 32 getPosition, 32, 33	getDislocationSourceList
getTauCritical, 33	SlipPlane, 67
getX, 33	getExtremity
getY, 33	SlipPlane, 67
getZ, 34	getIterationCount
ifEmitDipole, 34	DislocationSource, 32
incrementIterationCount, 34	getLineVector
lvec, 38	Dislocation, 19
mobile, 39	DislocationSource, 32
nlterations, 39	getNormal
pos, 39	SlipPlane, 68
resetIterationCounter, 34	getNumDislocationSources
rotationMatrix, 39	SlipPlane, 68
setBurgers, 35	getNumDislocations
setBurgersMagnitude, 35	SlipPlane, 68
setLineVector, 35	getNumIterations
setNumIterations, 35	DislocationSource, 32
setPosition, 36	getPosition
setTauCritical, 37	Defect, 11, 12
setX, 37	Dislocation, 20
setY, 37	DislocationSource, 32, 33
setZ, 37	SlipPlane, 68
stressField, 38	getPrincipalStrains
tauCritical, 39	Strain, 78
dislocationSource.cpp, 115	getPrincipalStresses
dislocationSource.h, 117	Stress, 89
dislocationSourceDefaults.h, 118	getRotationMatrix
dislocationSources	SlipPlane, 69 getShearStrains
SlipPlane, 73	Strain, 78
dislocationStresses	getShearStresses
SlipPlane, 73	•
dislocationVelocities	Stress, 89 getSlipPlaneStress_global
SlipPlane, 73	
dislocations	SlipPlane, 69 getSlipPlaneStress_local
SlipPlane, 73	SlipPlane, 70
distanceFromExtremity	getTauCritical
SlipPlane, 65	
dt	DislocationSource, 33
SlipPlane, 73	getValue Matrix33, 43
extremities 7.4	RotationMatrix, 52 Strain, 79
SlipPlane, 74	Stress, 90
forcePeachKoehler	Vector3d, 98
Dislocation, 19	getVector
Diologation, 10	90.100.01

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

position	DislocationSource, 35
SlipPlane, 74	setPinned
principalStrains	Dislocation, 23
Strain, 85	setPosition
principalStresses	Defect, 13
Stress, 96	Dislocation, 23, 24
C. 666, CC	DislocationSource, 36
resetIterationCounter	SlipPlane, 72
DislocationSource, 34	setTauCritical
rotate	
Strain, 84	DislocationSource, 37
Stress, 95	setValue
RotationMatrix, 49	Matrix33, 47
adjugate, 51	RotationMatrix, 56
getValue, 52	Strain, 84
operator*, 53, 54	Stress, 95
•	Vector3d, 103
operator*=, 54	setVector
operator∼, 56	Vector3d, 103
operator+, 55	setX
operator+=, 55	Defect, 14
operator-, 55	Dislocation, 24
operator-=, 56	DislocationSource, 37
RotationMatrix, 50, 51	setY
RotationMatrix, 50, 51	Defect, 14
setValue, 56	Dislocation, 24
transpose, 57	
x, 57	DislocationSource, 37
rotationMatrix	setZ
Dislocation, 26	Defect, 14
DislocationSource, 39	Dislocation, 24
SlipPlane, 74	DislocationSource, 37
rotationMatrix.cpp, 124	shearStrains
rotationMatrix.h, 125	Strain, 85
Totationiviatrix.ii, 125	shearStresses
SQRT2	Stress, 96
constants.h, 106	SlipPlane, 58
SQRT3	calculateDislocationForces, 61
constants.h, 106	calculateDislocationStresses, 61
SQRT5	calculateRotationMatrix, 62
	calculateTimeIncrement, 62
constants.h, 106	calculateVelocities, 64
setBurgers	dislocationForces, 73
Dislocation, 22	dislocationSources, 73
DislocationSource, 35	dislocationStresses, 73
setBurgersMagnitude	dislocation Velocities, 73
DislocationSource, 35	
setDislocationList	dislocations, 73
SlipPlane, 71	distanceFromExtremity, 65
setDislocationSourceList	dt, 73
SlipPlane, 71	extremities, 74
setExtremities	getAxis, 65
SlipPlane, 71	getDislocation, 66
setLineVector	getDislocationList, 66
Dislocation, 22	getDislocationSource, 66
DislocationSource, 35	getDislocationSourceList, 67
setMobile	getExtremity, 67
Dislocation, 23	getNormal, 68
setNormal	getNumDislocationSources, 68
SlipPlane, 72	getNumDislocations, 68
setNumIterations	getPosition, 68
SS. ISIMORATOR	900 000000

getRotationMatrix, 69	shearStresses, 96
getSlipPlaneStress_global, 69	Stress, 87, 88
getSlipPlaneStress_local, 70	transpose, 96
moveDislocations, 71	x, 96
normalVector, 74	stress.cpp, 136
	stress.h, 138
position, 74	
rotationMatrix, 74	stressField
setDislocationList, 71	Defect, 14
setDislocationSourceList, 71	Dislocation, 25
setExtremities, 71	DislocationSource, 38
setNormal, 72	stressFieldLocal
setPosition, 72	Dislocation, 25
SlipPlane, 60	sum
SlipPlane, 60	Vector3d, 103
sortDislocations, 72	
	tauCritical
slipPlane.cpp, 128	DislocationSource, 39
slipPlane.h, 129	transpose
slipPlaneDefaults.h, 130	Matrix33, 48
sortDislocations	RotationMatrix, 57
SlipPlane, 72	,
Strain, 74	Strain, 85
adjugate, 78	Stress, 96
getPrincipalStrains, 78	
getShearStrains, 78	Vector3d, 96
getValue, 79	getValue, 98
-	getVector, 99
operator*, 80, 81	magnitude, 99
operator*=, 81	normalize, 99
operator \sim , 83	operator*, 100
operator+, 82	operator*=, 101
operator+=, 82	operator [^] , 102
operator-, 82	operator $^{\wedge}$ =, 102
operator-=, 83	operator+, 101
populateMatrix, 83	•
principalStrains, 85	operator+=, 101
rotate, 84	operator-, 101
	operator-=, 102
setValue, 84	setValue, 103
shearStrains, 85	setVector, 103
Strain, 76, 77	sum, 103
transpose, 85	Vector3d, 98
x, 85	x, 104
strain.cpp, 133	vector3d.cpp, 140
strain.h, 135	vector3d.h, 140
Stress, 85	70010100111, 770
adjugate, 89	X
getPrincipalStresses, 89	Matrix33, 48
	RotationMatrix, 57
getShearStresses, 89	
getValue, 90	Strain, 85
operator∗, 91, 92	Stress, 96
operator*=, 92	Vector3d, 104
operator \sim , 94	
operator+, 93	
operator+=, 93	
operator-, 93	
operator-=, 94	
populateMatrix, 94	
principalStresses, 96	
·	
rotate, 95	
setValue, 95	