

DD2D

0

Generated by Doxygen 1.8.3.1

Mon May 27 2013 14:12:58

Contents

1	Main Page	1
2	Hierarchical Index	3
2.1	Class Hierarchy	3
3	Data Structure Index	5
3.1	Data Structures	5
4	File Index	7
4.1	File List	7
5	Data Structure Documentation	9
5.1	Defect Class Reference	9
5.1.1	Detailed Description	10
5.1.2	Constructor & Destructor Documentation	10
5.1.2.1	Defect	10
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	12
5.1.3.6	setPosition	13
5.1.3.7	setPosition	13
5.1.3.8	setX	13
5.1.3.9	setY	13
5.1.3.10	setZ	14
5.1.3.11	stressField	14
5.1.4	Field Documentation	14
5.1.4.1	pos	14
5.2	Dislocation Class Reference	15

5.2.1	Detailed Description	17
5.2.2	Constructor & Destructor Documentation	17
5.2.2.1	Dislocation	17
5.2.2.2	Dislocation	17
5.2.3	Member Function Documentation	18
5.2.3.1	calculateRotationMatrix	18
5.2.3.2	getBurgers	18
5.2.3.3	getLineVector	18
5.2.3.4	getPosition	19
5.2.3.5	getPosition	19
5.2.3.6	getX	19
5.2.3.7	getY	19
5.2.3.8	getZ	20
5.2.3.9	setBurgers	20
5.2.3.10	setLineVector	20
5.2.3.11	setMobile	20
5.2.3.12	setPinned	20
5.2.3.13	setPosition	21
5.2.3.14	setPosition	21
5.2.3.15	setX	21
5.2.3.16	setY	22
5.2.3.17	setZ	22
5.2.3.18	stressField	22
5.2.3.19	stressFieldLocal	23
5.2.4	Field Documentation	23
5.2.4.1	bmag	23
5.2.4.2	bvec	23
5.2.4.3	lvec	23
5.2.4.4	mobile	24
5.2.4.5	pos	24
5.2.4.6	rotationMatrix	24
5.3	DislocationSource Class Reference	24
5.3.1	Detailed Description	26
5.3.2	Constructor & Destructor Documentation	26
5.3.2.1	DislocationSource	26
5.3.2.2	DislocationSource	27
5.3.3	Member Function Documentation	27
5.3.3.1	dipoleNucleationLength	27
5.3.3.2	getPosition	27
5.3.3.3	getPosition	28

5.3.3.4	getX	28
5.3.3.5	getY	28
5.3.3.6	getZ	29
5.3.3.7	setBurgers	29
5.3.3.8	setLineVector	29
5.3.3.9	setPosition	29
5.3.3.10	setPosition	30
5.3.3.11	setX	30
5.3.3.12	setY	30
5.3.3.13	setZ	30
5.3.3.14	stressField	31
5.3.4	Field Documentation	31
5.3.4.1	bmag	31
5.3.4.2	bvec	31
5.3.4.3	lvec	31
5.3.4.4	mobile	31
5.3.4.5	nIterations	32
5.3.4.6	pos	32
5.3.4.7	rotationMatrix	32
5.3.4.8	tauCritical	32
5.4	Matrix33 Class Reference	32
5.4.1	Detailed Description	34
5.4.2	Constructor & Destructor Documentation	34
5.4.2.1	Matrix33	34
5.4.2.2	Matrix33	34
5.4.2.3	Matrix33	35
5.4.2.4	Matrix33	35
5.4.3	Member Function Documentation	35
5.4.3.1	adjugate	35
5.4.3.2	getValue	36
5.4.3.3	operator!	36
5.4.3.4	operator*	37
5.4.3.5	operator*	37
5.4.3.6	operator*	38
5.4.3.7	operator*=operator*="	38
5.4.3.8	operator*=operator*="	38
5.4.3.9	operator+operator+="	39
5.4.3.10	operator+=operator+="	39
5.4.3.11	operator-operator-="	39
5.4.3.12	operator-=operator-="	40

5.4.3.13	<code>operator^</code>	40
5.4.3.14	<code>operator~</code>	41
5.4.3.15	<code>setValue</code>	41
5.4.4	Field Documentation	41
5.4.4.1	<code>x</code>	41
5.5	RotationMatrix Class Reference	42
5.5.1	Detailed Description	43
5.5.2	Constructor & Destructor Documentation	43
5.5.2.1	<code>RotationMatrix</code>	43
5.5.2.2	<code>RotationMatrix</code>	44
5.5.3	Member Function Documentation	44
5.5.3.1	<code>adjugate</code>	44
5.5.3.2	<code>getValue</code>	44
5.5.3.3	<code>operator!</code>	45
5.5.3.4	<code>operator*</code>	45
5.5.3.5	<code>operator*</code>	46
5.5.3.6	<code>operator*</code>	46
5.5.3.7	<code>operator*=<code></code></code>	47
5.5.3.8	<code>operator*=<code></code></code>	47
5.5.3.9	<code>operator+</code>	47
5.5.3.10	<code>operator+=</code>	48
5.5.3.11	<code>operator-</code>	48
5.5.3.12	<code>operator-=</code>	48
5.5.3.13	<code>operator^</code>	49
5.5.3.14	<code>operator~</code>	49
5.5.3.15	<code>setValue</code>	49
5.5.4	Field Documentation	50
5.5.4.1	<code>x</code>	50
5.6	Strain Class Reference	50
5.6.1	Detailed Description	52
5.6.2	Constructor & Destructor Documentation	52
5.6.2.1	<code>Strain</code>	52
5.6.2.2	<code>Strain</code>	52
5.6.3	Member Function Documentation	53
5.6.3.1	<code>adjugate</code>	53
5.6.3.2	<code>getPrincipalStrains</code>	53
5.6.3.3	<code>getShearStrains</code>	54
5.6.3.4	<code>getValue</code>	54
5.6.3.5	<code>operator!</code>	54
5.6.3.6	<code>operator*</code>	55

5.6.3.7	operator*	55
5.6.3.8	operator*	56
5.6.3.9	operator*=	56
5.6.3.10	operator*=	56
5.6.3.11	operator+	57
5.6.3.12	operator+=	57
5.6.3.13	operator-	57
5.6.3.14	operator-=	58
5.6.3.15	operator^	58
5.6.3.16	operator~	59
5.6.3.17	populateMatrix	59
5.6.3.18	rotate	59
5.6.3.19	setValue	60
5.6.4	Field Documentation	60
5.6.4.1	principalStrains	60
5.6.4.2	shearStrains	60
5.6.4.3	x	60
5.7	Stress Class Reference	60
5.7.1	Detailed Description	62
5.7.2	Constructor & Destructor Documentation	62
5.7.2.1	Stress	62
5.7.2.2	Stress	63
5.7.3	Member Function Documentation	63
5.7.3.1	adjugate	63
5.7.3.2	getPrincipalStresses	64
5.7.3.3	getShearStresses	64
5.7.3.4	getValue	64
5.7.3.5	operator!	65
5.7.3.6	operator*	65
5.7.3.7	operator*	66
5.7.3.8	operator*	66
5.7.3.9	operator*=	67
5.7.3.10	operator*=	67
5.7.3.11	operator+	67
5.7.3.12	operator+=	68
5.7.3.13	operator-	68
5.7.3.14	operator-=	68
5.7.3.15	operator^	69
5.7.3.16	operator~	69
5.7.3.17	populateMatrix	69

5.7.3.18	rotate	70
5.7.3.19	setValue	70
5.7.4	Field Documentation	70
5.7.4.1	principalStresses	70
5.7.4.2	shearStresses	70
5.7.4.3	x	71
5.8	Vector3d Class Reference	71
5.8.1	Detailed Description	72
5.8.2	Constructor & Destructor Documentation	72
5.8.2.1	Vector3d	72
5.8.2.2	Vector3d	72
5.8.2.3	Vector3d	73
5.8.3	Member Function Documentation	73
5.8.3.1	getValue	73
5.8.3.2	getVector	73
5.8.3.3	magnitude	74
5.8.3.4	normalize	74
5.8.3.5	operator*	74
5.8.3.6	operator*	75
5.8.3.7	operator*=	75
5.8.3.8	operator+	75
5.8.3.9	operator+=	76
5.8.3.10	operator-	76
5.8.3.11	operator-=	76
5.8.3.12	operator^	77
5.8.3.13	operator^=	77
5.8.3.14	setValue	77
5.8.3.15	setVector	78
5.8.3.16	sum	78
5.8.4	Field Documentation	78
5.8.4.1	x	78
6	File Documentation	79
6.1	constants.h File Reference	79
6.1.1	Detailed Description	80
6.1.2	Macro Definition Documentation	80
6.1.2.1	PI	80
6.1.2.2	SQRT2	80
6.1.2.3	SQRT3	80
6.1.2.4	SQRT5	80

6.2	defect.cpp File Reference	80
6.2.1	Detailed Description	81
6.3	defect.h File Reference	82
6.3.1	Detailed Description	83
6.4	dislocation.cpp File Reference	83
6.4.1	Detailed Description	84
6.5	dislocation.h File Reference	85
6.5.1	Detailed Description	86
6.6	dislocationDefaults.h File Reference	86
6.6.1	Detailed Description	87
6.6.2	Macro Definition Documentation	88
6.6.2.1	DEFAULT_BURGERS_0	88
6.6.2.2	DEFAULT_BURGERS_1	88
6.6.2.3	DEFAULT_BURGERS_2	88
6.6.2.4	DEFAULT_BURGERS_MAGNITUDE	88
6.6.2.5	DEFAULT_LINEVECTOR_0	88
6.6.2.6	DEFAULT_LINEVECTOR_1	88
6.6.2.7	DEFAULT_LINEVECTOR_2	88
6.7	dislocationSource.cpp File Reference	88
6.7.1	Detailed Description	89
6.8	dislocationSource.h File Reference	90
6.8.1	Detailed Description	91
6.9	dislocationSourceDefaults.h File Reference	91
6.9.1	Detailed Description	92
6.9.2	Macro Definition Documentation	92
6.9.2.1	DEFAULT_NITERATIONS	92
6.9.2.2	DEFAULT_TAU_CRITICAL	93
6.10	mainpage.dox File Reference	93
6.11	matrix33.cpp File Reference	93
6.11.1	Detailed Description	93
6.12	matrix33.h File Reference	94
6.12.1	Detailed Description	95
6.13	rotationMatrix.cpp File Reference	96
6.13.1	Detailed Description	96
6.14	rotationMatrix.h File Reference	96
6.14.1	Detailed Description	97
6.15	strain.cpp File Reference	98
6.15.1	Detailed Description	98
6.16	strain.h File Reference	99
6.16.1	Detailed Description	100

6.17 stress.cpp File Reference	100
6.17.1 Detailed Description	101
6.18 stress.h File Reference	102
6.18.1 Detailed Description	103
6.19 vector3d.cpp File Reference	104
6.19.1 Detailed Description	104
6.20 vector3d.h File Reference	104
6.20.1 Detailed Description	106

Index	106
--------------	------------

Chapter 1

Main Page

This is a program to carry out dislocation dynamics simulations in two-dimensions.

A hierarchical data structure tree is created so that various entities in the simulations, such as dislocations, dislocation sources, slip planes, etc., can be managed efficiently. This program will also take advantage of the functionality provided by the C++ STL to manage lists of various objects in the simulation.

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

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

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

Defect	9
Dislocation	15
DislocationSource	24
Matrix33	32
RotationMatrix	42
Strain	50
Stress	60
Vector3d	71

Chapter 3

Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

Defect	Class Defect representing a generic defect in a material	9
Dislocation	Dislocation class representing a dislocation in the simulation	15
DislocationSource	DislocationSource class representing a source of dislocations in the simulation	24
Matrix33	Matrix33 class representing a 3x3 square matrix	32
RotationMatrix	RotationMatrix class to represent a rotation matrix	42
Strain	Strain class to represent the strain tensor	50
Stress	Stress class to represent the stress tensor	60
Vector3d	Vector3d class representing a single 3-dimensional vector in the simulation	71

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

constants.h	Definition of constants used in the program	79
defect.cpp	Definition of member functions of the Defect class	80
defect.h	Definition of the Defect class	82
dislocation.cpp	Definition of constructors and member functions of the Dislocation class	83
dislocation.h	Definition of the Dislocation class	85
dislocationDefaults.h	Definition of certain default values for members of the Dislocation class	86
dislocationSource.cpp	Definition of the member functions of the DislocationSource class	88
dislocationSource.h	Definition of the DislocationSource class	90
dislocationSourceDefaults.h	Definition of certain default values for members of the DislocationSource class	91
matrix33.cpp	Definition of the member functions and operators of the Matrix33 class	93
matrix33.h	Definition of the Matrix33 class	94
rotationMatrix.cpp	Definition of the RotationMatrix class member functions	96
rotationMatrix.h	Definition of the RotationMatrix class	96
strain.cpp	Definition of the member functions if the Strain class	98
strain.h	Definition of the Strain class	99
stress.cpp	Definition of the member functions if the Stress class	100
stress.h	Definition of the Stress class	102
vector3d.cpp	Definition of member functions and operators of the Vector3d class	104
vector3d.h	Definition of the Vector3d class	104

Chapter 5

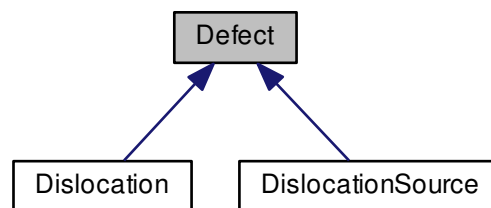
Data Structure Documentation

5.1 Defect Class Reference

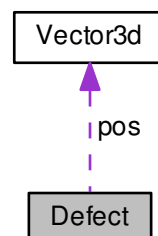
Class [Defect](#) representing a generic defect in a material.

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

Protected Attributes

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

5.1.1 Detailed Description

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

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

Definition at line 20 of file defect.h.

5.1.2 Constructor & Destructor Documentation

5.1.2.1 [Defect::Defect](#) ()

Default constructor.

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

Definition at line 17 of file defect.cpp.

```

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

```

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

Constructor specifying the position.

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

Parameters

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

Definition at line 32 of file defect.cpp.

```

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

```

5.1.2.3 Defect::Defect (double * p)

Constructor specifying the position.

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

Parameters

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

Definition at line 44 of file defect.cpp.

```

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

```

5.1.3 Member Function Documentation

5.1.3.1 double * Defect::getPosition ()

Returns in an array the position.

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

Returns

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

Definition at line 109 of file defect.cpp.

```

110 {
111     return (this->pos.getVector ());
112 }

```

5.1.3.2 void Defect::getPosition (double * a)

Returns the array position in a pre-allocated array.

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

Parameters

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

Definition at line 119 of file defect.cpp.

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

5.1.3.3 double Defect::getX ()

Returns the X-coordinate of the defect.

Returns

X-coordinate of the defect.

Definition at line 128 of file defect.cpp.

```
129 {  
130   return (this->getValue (0));  
131 }
```

5.1.3.4 double Defect::getY ()

Returns the Y-coordinate of the defect.

Returns

Y-coordinate of the defect.

Definition at line 137 of file defect.cpp.

```
138 {  
139   return (this->pos.getValue (1));  
140 }
```

5.1.3.5 double Defect::getZ ()

Returns the Z-coordinate of the defect.

Returns

Z-coordinate of the defect.

Definition at line 146 of file defect.cpp.

```
147 {  
148   return (this->pos.getValue (2));  
149 }
```

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

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

Definition at line 56 of file defect.cpp.

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

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

Sets the position of the defect.

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

Parameters

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

Definition at line 69 of file defect.cpp.

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

5.1.3.8 void Defect::setX (double x)

Sets the X-coordinate of the defect.

Parameters

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

Definition at line 80 of file defect.cpp.

```
81 {  
82     this->pos.setValue (0, x);  
83 }
```

5.1.3.9 void Defect::setY (double y)

Sets the Y-coordinate of the defect.

Parameters

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

Definition at line 89 of file defect.cpp.

```
90 {
91     this->pos.setValue (1, y);
92 }
```

5.1.3.10 void Defect::setZ (double z)

Sets the Z-coordinate of the defect.

Parameters

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

Definition at line 98 of file defect.cpp.

```
99 {
100     this->pos.setValue (2, z);
101 }
```

5.1.3.11 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

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

Returns

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

Reimplemented in [Dislocation](#).

Definition at line 121 of file defect.h.

```
122 {
123     // This virtual function returns a zero matrix.
124     // Inheriting classes will have functions implementing this in their own way
125     // They will override this behaviour.
126     Stress s;
127     return (s);
128 }
```

5.1.4 Field Documentation**5.1.4.1 Vector3d Defect::pos [protected]**

Position vector of the defect in 2D space.

Definition at line 26 of file defect.h.

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

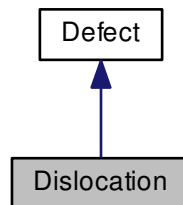
- [defect.h](#)
- [defect.cpp](#)

5.2 Dislocation Class Reference

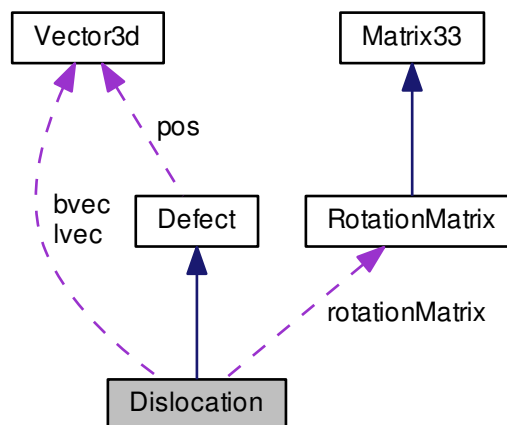
[Dislocation](#) class representing a dislocation in the simulation.

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

Inheritance diagram for Dislocation:



Collaboration diagram for Dislocation:



Public Member Functions

- [Dislocation](#) ()

Default constructor.

- [Dislocation](#) ([Vector3d](#) burgers, [Vector3d](#) line, [Vector3d](#) position, double bm, bool m)

Constructor that explicitly specifies all parameters.

- void [setBurgers](#) ([Vector3d](#) burgers)

Sets the Burgers vector of the dislocation.

- void [setLineVector](#) ([Vector3d](#) line)

Sets the line vector of the dislocation.

- void [setMobile](#) ()

Sets the dislocation as mobile.

- void [setPinned](#) ()

Sets the dislocation as pinned.

- [Vector3d](#) [getBurgers](#) ()

Gets the Burgers vector of the dislocation.

- [Vector3d](#) [getLineVector](#) ()

Gets the line vector of the dislocation.

- void [calculateRotationMatrix](#) ()

Calculate the rotation matrix.

- [Stress](#) [stressField](#) ([Vector3d](#) p, double mu, double nu)

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

- [Stress](#) [stressFieldLocal](#) ([Vector3d](#) p, double mu, double nu)

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

- void [setPosition](#) (double *a)

Sets the position of the defect.

- void [setPosition](#) (double x, double y, double z)

Sets the position of the defect.

- void [setX](#) (double x)

Sets the X-coordinate of the defect.

- void [setY](#) (double y)

Sets the Y-coordinate of the defect.

- void [setZ](#) (double z)

Sets the Z-coordinate of the defect.

- double * [getPosition](#) ()

Returns in an array the position.

- void [getPosition](#) (double *a)

Returns the array position in a pre-allocated array.

- double [getX](#) ()

Returns the X-coordinate of the defect.

- double [getY](#) ()

Returns the Y-coordinate of the defect.

- double [getZ](#) ()

Returns the Z-coordinate of the defect.

Protected Attributes

- [Vector3d](#) [bvec](#)

Burgers vector of the dislocation.

- [Vector3d](#) [lvec](#)

Line vector of the dislocation.

- bool [mobile](#)

Boolean term indicating mobility.

- double [bmag](#)
Magnitude of the Burgers vector in metres.
- [RotationMatrix](#) [rotationMatrix](#)
The rotation matrix for rotating from the global to the local co-ordinate system and vice-versa.
- [Vector3d](#) [pos](#)
Position vector of the defect in 2D space.

5.2.1 Detailed Description

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

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

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

5.2.2 Constructor & Destructor Documentation

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

Default constructor.

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

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

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

```

22 {
23     this->setPosition ( 0.0, 0.0, 0.0 );
24     this->setBurgers ( Vector3d ( DEFAULT_BURGERS_0,
25                                DEFAULT_BURGERS_1, DEFAULT_BURGERS_2 ) );
26     this->setLineVector ( Vector3d ( DEFAULT_LINEVECTOR_0,
27                                   DEFAULT_LINEVECTOR_1, DEFAULT_LINEVECTOR_2 ) );
28     this->bmag = DEFAULT_BURGERS_MAGNITUDE;
29     this->mobile = true;
30     this->calculateRotationMatrix ();
31 }
```

5.2.2.2 [Dislocation::Dislocation \(Vector3d burgers, Vector3d line, Vector3d position, double bm, bool m \)](#)

Constructor that explicitly specifies all parameters.

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

Parameters

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

Definition at line 40 of file [dislocation.cpp](#).

```

41 {
42     this->bvec = burgers;
```

```

43  this->lvec   = line;
44  this->pos    = position;
45  this->mobile = m;
46  this->bmag   = bm;
47  this->calculateRotationMatrix ();
48  }

```

5.2.3 Member Function Documentation

5.2.3.1 void Dislocation::calculateRotationMatrix ()

Calculate the roation matrix.

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

Definition at line 109 of file dislocation.cpp.

```

110 {
111     Vector3d globalSystem[3];    // Global co-ordinate systems
112     Vector3d localSystem[3];    // Dislocation co-ordinate system
113
114     // Vectors of the global co-ordinate system
115     globalSystem[0] = Vector3d (1.0, 0.0, 0.0);
116     globalSystem[1] = Vector3d (0.0, 1.0, 0.0);
117     globalSystem[2] = Vector3d (0.0, 0.0, 1.0);
118
119     // Vectors of the dislocation co-ordinate system
120     localSystem[0] = bvec.normalize ();
121     localSystem[2] = lvec.normalize ();
122     localSystem[1] = (lvec ^ bvec).normalize ();
123
124     // Calculate rotation matrix
125     this->rotationMatrix = RotationMatrix (globalSystem, localSystem);
126 }

```

5.2.3.2 Vector3d Dislocation::getBurgers ()

Gets the Burgers vector of the dislocation.

Returns

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

Definition at line 90 of file dislocation.cpp.

```

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

```

5.2.3.3 Vector3d Dislocation::getLineVector ()

Gets the line vector of the dislocation.

Returns

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

Definition at line 99 of file dislocation.cpp.

```

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

```

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

Returns in an array the position.

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

Returns

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

Definition at line 109 of file defect.cpp.

```
110 {
111     return (this->pos.getVector ());
112 }
```

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

Returns the array position in a pre-allocated array.

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

Parameters

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

Definition at line 119 of file defect.cpp.

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

5.2.3.6 `double Defect::getX ()` [inherited]

Returns the X-coordinate of the defect.

Returns

X-coordinate of the defect.

Definition at line 128 of file defect.cpp.

```
129 {
130     return (this->getValue (0));
131 }
```

5.2.3.7 `double Defect::getY ()` [inherited]

Returns the Y-coordinate of the defect.

Returns

Y-coordinate of the defect.

Definition at line 137 of file defect.cpp.

```
138 {
139     return (this->pos.getValue (1));
140 }
```

5.2.3.8 double Defect::getZ () [inherited]

Returns the Z-coordinate of the defect.

Returns

Z-coordinate of the defect.

Definition at line 146 of file defect.cpp.

```
147 {
148     return (this->pos.getValue (2));
149 }
```

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

Sets the Burgers vector of the dislocation.

Parameters

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

Definition at line 54 of file dislocation.cpp.

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

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

Sets the line vector of the dislocation.

Parameters

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

Definition at line 62 of file dislocation.cpp.

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

5.2.3.11 void Dislocation::setMobile ()

Sets the dislocation as mobile.

Sets the flag mobile to true.

Definition at line 71 of file dislocation.cpp.

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

5.2.3.12 void Dislocation::setPinned ()

Sets the dislocation as pinned.

Sets the flag mobile to false.

Definition at line 80 of file dislocation.cpp.

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

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

Sets the position of the defect.

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

Parameters

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

Definition at line 56 of file defect.cpp.

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

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

Sets the position of the defect.

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

Parameters

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

Definition at line 69 of file defect.cpp.

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

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

Sets the X-coordinate of the defect.

Parameters

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

Definition at line 80 of file defect.cpp.

```
81 {
82     this->pos.setValue (0, x);
83 }
```

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

Sets the Y-coordinate of the defect.

Parameters

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

Definition at line 89 of file defect.cpp.

```
90 {
91     this->pos.setValue (1, y);
92 }
```

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

Sets the Z-coordinate of the defect.

Parameters

z	Z-coordinate of the defect.
---	-----------------------------

Definition at line 98 of file defect.cpp.

```
99 {
100     this->pos.setValue (2, z);
101 }
```

5.2.3.18 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

p	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 137 of file dislocation.cpp.

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



```

151
152     return (sGlobal);
153 }

```

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

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

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

Parameters

p	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.
μ	Shear modulus in Pascals.
ν	Poisson's ratio.

Returns

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

Definition at line 163 of file dislocation.cpp.

```

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

```

5.2.4 Field Documentation

5.2.4.1 double Dislocation::bmag [protected]

Magnitude of the Burgers vector in metres.

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

Definition at line 44 of file dislocation.h.

5.2.4.2 Vector3d Dislocation::bvec [protected]

Burgers vector of the dislocation.

Definition at line 27 of file dislocation.h.

5.2.4.3 Vector3d Dislocation::lvec [protected]

Line vector if the dislocation.

Definition at line 32 of file dislocation.h.

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

Boolean term indicating mobility.

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

Definition at line 38 of file dislocation.h.

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

Position vector of the defect in 2D space.

Definition at line 26 of file defect.h.

5.2.4.6 `RotationMatrix Dislocation::rotationMatrix` `[protected]`

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

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

Definition at line 50 of file dislocation.h.

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

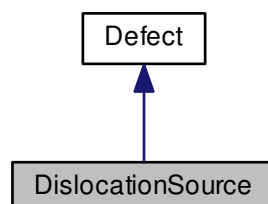
- [dislocation.h](#)
- [dislocation.cpp](#)

5.3 DislocationSource Class Reference

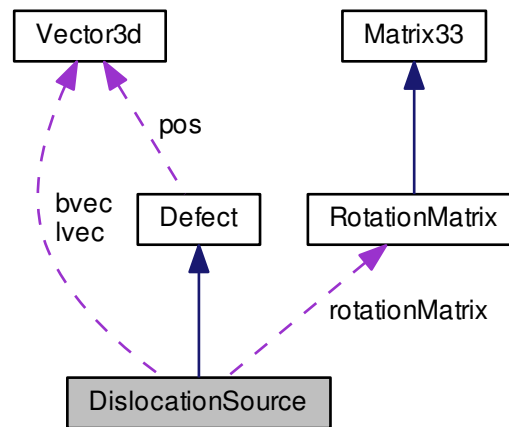
[DislocationSource](#) class representing a source of dislocations in the simulation.

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

Inheritance diagram for DislocationSource:



Collaboration diagram for DislocationSource:



Public Member Functions

- `DislocationSource ()`
Default constructor.
- `DislocationSource (Vector3d burgers, Vector3d line, Vector3d position, double bm, double tau, int nlter)`
Constructor that explicitly specifies all parameters.
- `void setBurgers (Vector3d burgers)`
Sets the Burgers vector of the dislocation.
- `void setLineVector (Vector3d line)`
Sets the line vector of the dislocation.
- `double dipoleNucleationLength (double tau, double mu, double nu)`
The nucleation length of the 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 setX (double x)`
Sets the X-coordinate of the defect.
- `void setY (double y)`
Sets the Y-coordinate of the defect.
- `void setZ (double z)`
Sets the Z-coordinate of the defect.
- `double * getPosition ()`
Returns in an array the position.
- `void getPosition (double *a)`
Returns the array position in a pre-allocated array.
- `double getX ()`
Returns the X-coordinate of the defect.
- `double getY ()`

- Returns the Y-coordinate of the defect.*
- double `getZ ()`
Returns the Z-coordinate of the defect.
- virtual `Stress stressField (Vector3d p, double mu, double nu)`
Virtual function for calculating the stress field.

Protected Attributes

- `Vector3d bvec`
Burgers vector of the dislocation.
- `Vector3d lvec`
Line vector of the dislocation.
- bool `mobile`
Boolean term indicating mobility.
- double `bmag`
Magnitude of the Burgers vector in metres.
- `RotationMatrix rotationMatrix`
The rotation matrix for rotating from the global to the local co-ordinate system and vice-versa.
- double `tauCritical`
Critical stress for the emission of a dislocation dipole.
- int `nIterations`
Number of iterations before a dipole is emitted.
- `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 21 of file `dislocationSource.h`.

5.3.2 Constructor & Destructor Documentation

5.3.2.1 `DislocationSource::DislocationSource ()`

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. Tau critical: Default value set in the defaults file. nIterations: Default value set in the defaults file.

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

```

23 {
24     this->setPosition ( 0.0, 0.0, 0.0 );
25     this->setBurgers ( Vector3d ( DEFAULT_BURGERS_0,
26                                DEFAULT_BURGERS_1, DEFAULT_BURGERS_2 ) );
27     this->setLineVector ( Vector3d ( DEFAULT_LINEVECTOR_0,
28                                DEFAULT_LINEVECTOR_1, DEFAULT_LINEVECTOR_2 ) );
29     this->bmag = DEFAULT_BURGERS_MAGNITUDE;
30     this->tauCritical = DEFAULT_TAU_CRITICAL;
31     this->nIterations = DEFAULT_NITERATIONS;
32 }
```

5.3.2.2 DislocationSource::DislocationSource (Vector3d *burgers*, Vector3d *line*, Vector3d *position*, double *bm*, double *tau*, int *nIter*)

Constructor that explicitly specifies all parameters.

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

Parameters

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

Definition at line 42 of file dislocationSource.cpp.

```

43 {
44     this->bvec    = burgers;
45     this->lvec    = line;
46     this->pos     = position;
47     this->bmag    = bm;
48     this->tauCritical = tau;
49     this->nIterations = nIter;
50 }
```

5.3.3 Member Function Documentation

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

The nucleation length of the dipole.

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

Parameters

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

Returns

The length of the dislocation dipole.

Definition at line 78 of file dislocationSource.cpp.

```

79 {
80     double L = 0.0;
81
82     if (tau >= tauCritical)
83     {
84         L = (mu * this->bmag) / ( 2.0 * PI * (1.0 - nu) * this->tauCritical );
85     }
86
87     return (L);
88 }
```

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

Returns in an array the position.

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

Returns

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

Definition at line 109 of file defect.cpp.

```
110 {
111     return (this->pos.getVector ());
112 }
```

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

Returns the array position in a pre-allocated array.

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

Parameters

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

Definition at line 119 of file defect.cpp.

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

5.3.3.4 double Defect::getX () [inherited]

Returns the X-coordinate of the defect.

Returns

X-coordinate of the defect.

Definition at line 128 of file defect.cpp.

```
129 {
130     return (this->getValue (0));
131 }
```

5.3.3.5 double Defect::getY () [inherited]

Returns the Y-coordinate of the defect.

Returns

Y-coordinate of the defect.

Definition at line 137 of file defect.cpp.

```
138 {
139     return (this->pos.getValue (1));
140 }
```

5.3.3.6 double Defect::getZ () [inherited]

Returns the Z-coordinate of the defect.

Returns

Z-coordinate of the defect.

Definition at line 146 of file defect.cpp.

```
147 {
148     return (this->pos.getValue (2));
149 }
```

5.3.3.7 void DislocationSource::setBurgers (Vector3d burgers)

Sets the Burgers vector of the dislocation.

Parameters

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

Definition at line 56 of file dislocationSource.cpp.

```
57 {
58     this->bvec = burgers;
59 }
```

5.3.3.8 void DislocationSource::setLineVector (Vector3d line)

Sets the line vector of the dislocation.

Parameters

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

Definition at line 65 of file dislocationSource.cpp.

```
66 {
67     this->lvec = line;
68 }
```

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

Sets the position of the defect.

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

Parameters

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

Definition at line 56 of file defect.cpp.

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

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

Sets the position of the defect.

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

Parameters

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

Definition at line 69 of file defect.cpp.

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

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

Sets the X-coordinate of the defect.

Parameters

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

Definition at line 80 of file defect.cpp.

```
81 {  
82     this->pos.setValue (0, x);  
83 }
```

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

Sets the Y-coordinate of the defect.

Parameters

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

Definition at line 89 of file defect.cpp.

```
90 {  
91     this->pos.setValue (1, y);  
92 }
```

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

Sets the Z-coordinate of the defect.

Parameters

z	Z-coordinate of the defect.
---	-----------------------------

Definition at line 98 of file defect.cpp.


```

99 {
100     this->pos.setValue (2, z);
101 }

```

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

Virtual function for calculating the stress field.

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

Parameters

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

Returns

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

Reimplemented in [Dislocation](#).

Definition at line 121 of file defect.h.

```

122 {
123     // This virtual function returns a zero matrix.
124     // Inheriting classes will have functions implementing this in their own way
125     // They will override this behaviour.
126     Stress s;
127     return (s);
128 }

```

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 44 of file dislocationSource.h.

5.3.4.2 Vector3d DislocationSource::bvec [protected]

Burgers vector of the dislocation.

Definition at line 27 of file dislocationSource.h.

5.3.4.3 Vector3d DislocationSource::lvec [protected]

Line vector if the dislocation.

Definition at line 32 of file dislocationSource.h.

5.3.4.4 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 38 of file dislocationSource.h.

5.3.4.5 `int DislocationSource::nIterations` [protected]

Number of iterations before a dipole is emitted.

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

Definition at line 62 of file dislocationSource.h.

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

Position vector of the defect in 2D space.

Definition at line 26 of file defect.h.

5.3.4.7 `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 50 of file dislocationSource.h.

5.3.4.8 `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 56 of file dislocationSource.h.

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

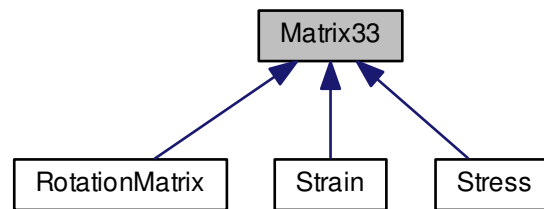
- [dislocationSource.h](#)
- [dislocationSource.cpp](#)

5.4 Matrix33 Class Reference

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

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

Inheritance diagram for Matrix33:



Public Member Functions

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

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

Protected Attributes

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

5.4.1 Detailed Description

`Matrix33` class representing a 3x3 square matrix.

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

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

5.4.2 Constructor & Destructor Documentation

5.4.2.1 `Matrix33::Matrix33 ()`

Default constructor.

Initializes the matrix with all elements equal to 0.0.

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

```

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

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

Constructor with the values provided in a 3x3 matrix.

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

Parameters

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

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

```

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

```

44     }
45 }
46 }

```

5.4.2.3 Matrix33::Matrix33 (Vector3d a)

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

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

Parameters

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

Definition at line 53 of file matrix33.cpp.

```

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

```

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

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

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

Parameters

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

Definition at line 72 of file matrix33.cpp.

```

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

```

5.4.3 Member Function Documentation

5.4.3.1 Matrix33 Matrix33::adjugate ()

Returns the adjugate matrix of the present matrix.

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

Returns

The adjugate matrix of the present matrix.

Definition at line 129 of file matrix33.cpp.

```

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

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

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

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

Parameters

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

Returns

Value of the element located at the given position.

Definition at line 111 of file matrix33.cpp.

```

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

5.4.3.3 Matrix33 Matrix33::operator! () const

Inverse.

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

Returns

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

Definition at line 370 of file matrix33.cpp.

```

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

```

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

Operator for scaling the matrix by a scalar.

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

Returns

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

Definition at line 233 of file matrix33.cpp.

```

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

```

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

Operator for the multiplication of two matrices.

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

Returns

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

Definition at line 271 of file matrix33.cpp.

```

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

```

```

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

```

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

Operator for the multiplication of a matrix with a vector.

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

Returns

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

Definition at line 311 of file matrix33.cpp.

```

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

```

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

Operator for reflexive scaling of the matrix by a scalar.

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

Definition at line 253 of file matrix33.cpp.

```

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

```

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

Operator for reflexive multiplication of two matrices.

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

Definition at line 295 of file matrix33.cpp.


```

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

```

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

Operator for addition of two matrices.

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

Returns

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

Definition at line 155 of file matrix33.cpp.

```

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

```

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

Operator for reflexive addition of two matrices.

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

Definition at line 175 of file matrix33.cpp.

```

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

```

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

Operator for the subtraction of two matrices.

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

Returns

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

Definition at line 194 of file matrix33.cpp.

```

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

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

Operator for reflexive subtraction of two matrices.

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

Definition at line 214 of file matrix33.cpp.

```

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

5.4.3.13 Matrix33 Matrix33::operator^ () const

Transpose.

Performs the transpose of the current matrix.

Returns

A new matrix with the transpose of the current matrix.

Definition at line 333 of file matrix33.cpp.

```

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

5.4.3.14 double Matrix33::operator~ () const

Determinant.

Calculates the determinant of the current matrix.

Returns

Returns the determinant of the current matrix.

Definition at line 354 of file matrix33.cpp.

```

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

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

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

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

Parameters

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

Definition at line 93 of file matrix33.cpp.

```

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

5.4.4 Field Documentation**5.4.4.1 double Matrix33::x[3][3] [protected]**

Array containing the elements of the matrix.

Definition at line 26 of file matrix33.h.

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

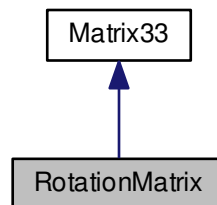
- [matrix33.h](#)
- [matrix33.cpp](#)

5.5 RotationMatrix Class Reference

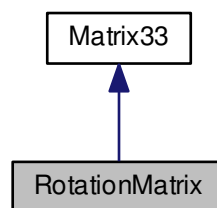
[RotationMatrix](#) class to represent a rotation matrix.

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

Inheritance diagram for [RotationMatrix](#):



Collaboration diagram for [RotationMatrix](#):



Public Member Functions

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

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

Protected Attributes

- double [x](#) [3][3]
Array containing the elements of the matrix.

5.5.1 Detailed Description

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

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

Definition at line 19 of file [rotationMatrix.h](#).

5.5.2 Constructor & Destructor Documentation

5.5.2.1 [RotationMatrix::RotationMatrix](#) ()

Default constructor.

Initializes the rotation matrix with a unit matrix.

Definition at line 14 of file [rotationMatrix.cpp](#).

```

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

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

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

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

Parameters

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

Definition at line 36 of file rotationMatrix.cpp.

```

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

5.5.3 Member Function Documentation

5.5.3.1 Matrix33 Matrix33::adjugate () [inherited]

Returns the adjugate matrix of the present matrix.

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

Returns

The adjugate matrix of the present matrix.

Definition at line 129 of file matrix33.cpp.

```

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

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

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

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

Parameters

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

Returns

Value of the element located at the given position.

Definition at line 111 of file matrix33.cpp.

```

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

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

Inverse.

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

Returns

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

Definition at line 370 of file matrix33.cpp.

```

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

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

Operator for scaling the matrix by a scalar.

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

Returns

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

Definition at line 233 of file matrix33.cpp.

```

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

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

Operator for the multiplication of two matrices.

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

Returns

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

Definition at line 271 of file matrix33.cpp.

```

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

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

Operator for the multiplication of a matrix with a vector.

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

Returns

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

Definition at line 311 of file matrix33.cpp.

```

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



```

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

```

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

Operator for reflexive scaling of the matrix by a scalar.

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

Definition at line 253 of file matrix33.cpp.

```

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

```

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

Operator for reflexive multiplication of two matrices.

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

Definition at line 295 of file matrix33.cpp.

```

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

```

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

Operator for addition of two matrices.

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

Returns

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

Definition at line 155 of file matrix33.cpp.

```

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

```

```

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

```

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

Operator for reflexive addition of two matrices.

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

Definition at line 175 of file matrix33.cpp.

```

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

```

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

Operator for the subtraction of two matrices.

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

Returns

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

Definition at line 194 of file matrix33.cpp.

```

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

```

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

Operator for reflexive subtraction of two matrices.

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

Definition at line 214 of file matrix33.cpp.

```

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

```

```

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

```

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

Transpose.

Performs the transpose of the current matrix.

Returns

A new matrix with the transpose of the current matrix.

Definition at line 333 of file matrix33.cpp.

```

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

```

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

Determinant.

Calculates the determinant of the current matrix.

Returns

Returns the determinant of the current matrix.

Definition at line 354 of file matrix33.cpp.

```

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

```

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

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

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

Parameters

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

Definition at line 93 of file matrix33.cpp.

```

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

5.5.4 Field Documentation

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

Array containing the elements of the matrix.

Definition at line 26 of file matrix33.h.

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

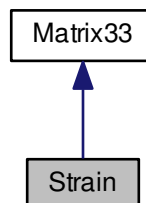
- [rotationMatrix.h](#)
- [rotationMatrix.cpp](#)

5.6 Strain Class Reference

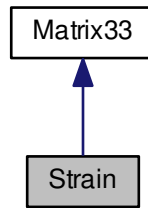
[Strain](#) class to represent the strain tensor.

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

Inheritance diagram for Strain:



Collaboration diagram for Strain:



Public Member Functions

- [Strain](#) ()
Default constructor.
- [Strain](#) (double *principal, double *shear)
Constructor specifying the principal and shear strains.
- void [populateMatrix](#) ()
Construct the strain tensor from the principal and shear strains.
- double * [getPrincipalStrains](#) ()
Get the principal strains.
- double * [getShearStrains](#) ()
Get the shear strains.
- [Strain rotate](#) ([RotationMatrix](#) alpha)
Rotate the strain tensor from one coordinate system to another.
- void [setValue](#) (int row, int column, double value)
Function to set the value of an element indicated by its position.
- double [getValue](#) (int row, int column)
Returns the value of the element located by the row and column indices provided.
- [Matrix33 adjugate](#) ()
Returns the adjugate matrix of the present matrix.
- [Matrix33 operator+](#) (const [Matrix33](#) &) const
Operator for addition of two matrices.
- void [operator+=](#) (const [Matrix33](#) &)
Operator for reflexive addition of two matrices.
- [Matrix33 operator-](#) (const [Matrix33](#) &) const
Operator for the subtraction of two matrices.
- void [operator-=](#) (const [Matrix33](#) &)
Operator for reflexive subtraction of two matrices.
- [Matrix33 operator*](#) (const double &) const
Operator for scaling the matrix by a scalar.
- [Matrix33 operator*](#) (const [Matrix33](#) &) const
Operator for the multiplication of two matrices.
- [Vector3d operator*](#) (const [Vector3d](#) &) const
Operator for the multiplication of a matrix with a vector.
- void [operator*=](#) (const double &)

Operator for reflexive scaling of the matrix by a scalar.

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

Operator for reflexive multiplication of two matrices.

- `Matrix33 operator^` () const

Transpose.

- double `operator~` () const

Determinant.

- `Matrix33 operator!` () const

Inverse.

Protected Attributes

- double `principalStrains` [3]
- double `shearStrains` [3]
- double `x` [3][3]

Array containing the elements of the matrix.

5.6.1 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`.

5.6.2 Constructor & Destructor Documentation

5.6.2.1 `Strain::Strain ()`

Default constructor.

Initializes the strain tensor with zeros.

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

```

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

5.6.2.2 `Strain::Strain (double * principal, double * shear)`

Constructor specifying the principal and shear strains.

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

Parameters

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

Definition at line 35 of file strain.cpp.

```

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

```

5.6.3 Member Function Documentation

5.6.3.1 Matrix33 Matrix33::adjugate () [inherited]

Returns the adjugate matrix of the present matrix.

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

Returns

The adjugate matrix of the present matrix.

Definition at line 129 of file matrix33.cpp.

```

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

```

5.6.3.2 double * Strain::getPrincipalStrains ()

Get the principal strains.

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

Returns

3-member array with the principal strains.

Definition at line 68 of file strain.cpp.

```

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

```

5.6.3.3 double * Strain::getShearStrains ()

Get the shear strains.

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

Returns

3-member array with the shear strains.

Definition at line 86 of file strain.cpp.

```

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

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

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

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

Parameters

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

Returns

Value of the element located at the given position.

Definition at line 111 of file matrix33.cpp.

```

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

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

Inverse.

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

Returns

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

Definition at line 370 of file matrix33.cpp.

```

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

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

Operator for scaling the matrix by a scalar.

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

Returns

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

Definition at line 233 of file matrix33.cpp.

```

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

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

Operator for the multiplication of two matrices.

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

Returns

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

Definition at line 271 of file matrix33.cpp.

```

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

```

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

Operator for the multiplication of a matrix with a vector.

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

Returns

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

Definition at line 311 of file matrix33.cpp.

```

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

```

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

Operator for reflexive scaling of the matrix by a scalar.

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

Definition at line 253 of file matrix33.cpp.

```

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

```

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

Operator for reflexive multiplication of two matrices.

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

Definition at line 295 of file matrix33.cpp.

```

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

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

Operator for addition of two matrices.

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

Returns

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

Definition at line 155 of file matrix33.cpp.

```

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

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

Operator for reflexive addition of two matrices.

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

Definition at line 175 of file matrix33.cpp.

```

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

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

Operator for the subtraction of two matrices.

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

Returns

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

Definition at line 194 of file matrix33.cpp.

```

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

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

Operator for reflexive subtraction of two matrices.

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

Definition at line 214 of file matrix33.cpp.

```

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

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

Transpose.

Performs the transpose of the current matrix.

Returns

A new matrix with the transpose of the current matrix.

Definition at line 333 of file matrix33.cpp.

```

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

5.6.3.16 `double Matrix33::operator~() const` [inherited]

Determinant.

Calculates the determinant of the current matrix.

Returns

Returns the determinant of the current matrix.

Definition at line 354 of file matrix33.cpp.

```

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

5.6.3.17 `void Strain::populateMatrix()`

Construct the strain tensor from the principal and shear strains.

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

Definition at line 52 of file strain.cpp.

```

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

5.6.3.18 `Strain Strain::rotate(RotationMatrix alpha)`

Rotate the strain tensor from one coordinate system to another.

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

Parameters

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

Returns

Rotated strain tensor.

Definition at line 105 of file strain.cpp.

```

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

```

112     return (sNew);
113 }

```

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

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

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

Parameters

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

Definition at line 93 of file matrix33.cpp.

```

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

```

5.6.4 Field Documentation

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

The three principal strains: s11, s22, s33.

Definition at line 27 of file strain.h.

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

The three shear strains: s12, s13, s23,

Definition at line 31 of file strain.h.

5.6.4.3 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:

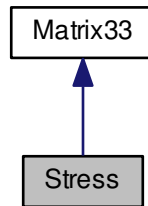
- [strain.h](#)
- [strain.cpp](#)

5.7 Stress Class Reference

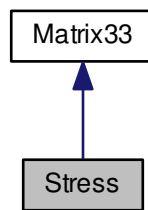
[Stress](#) class to represent the stress tensor.

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

Inheritance diagram for Stress:



Collaboration diagram for Stress:



Public Member Functions

- [Stress](#) ()
Default constructor.
- [Stress](#) (double *principal, double *shear)
Constructor specifying the principal and shear stresses.
- void [populateMatrix](#) ()
Construct the stress tensor from the principal and shear stresses.
- double * [getPrincipalStresses](#) ()
Get the principal stresses.
- double * [getShearStresses](#) ()
Get the shear stresses.
- [Stress rotate](#) ([RotationMatrix](#) alpha)
Rotate the stress tensor from one coordinate system to another.
- void [setValue](#) (int row, int column, double value)
Function to set the value of an element indicated by its position.
- double [getValue](#) (int row, int column)
Returns the value of the element located by the row and column indices provided.
- [Matrix33 adjugate](#) ()
Returns the adjugate matrix of the present matrix.

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

Protected Attributes

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

5.7.1 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`.

5.7.2 Constructor & Destructor Documentation

5.7.2.1 `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 }

```

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

Constructor specifying the principal and shear stresses.

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

Parameters

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

Definition at line 35 of file stress.cpp.

```

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

```

5.7.3 Member Function Documentation

5.7.3.1 Matrix33 Matrix33::adjugate () [inherited]

Returns the adjugate matrix of the present matrix.

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

Returns

The adjugate matrix of the present matrix.

Definition at line 129 of file matrix33.cpp.

```

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

```

5.7.3.2 `double * Stress::getPrincipalStresses ()`

Get the principal stresses.

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

Returns

3-member array with the principal stresses.

Definition at line 68 of file stress.cpp.

```

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

5.7.3.3 `double * Stress::getShearStresses ()`

Get the shear stresses.

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

Returns

3-member array with the shear stresses.

Definition at line 86 of file stress.cpp.

```

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

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

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

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

Parameters

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

Returns

Value of the element located at the given position.

Definition at line 111 of file matrix33.cpp.

```

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

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

Inverse.

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

Returns

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

Definition at line 370 of file matrix33.cpp.

```

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

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

Operator for scaling the matrix by a scalar.

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

Returns

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

Definition at line 233 of file matrix33.cpp.

```

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

```

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

Operator for the multiplication of two matrices.

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

Returns

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

Definition at line 271 of file matrix33.cpp.

```

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

```

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

Operator for the multiplication of a matrix with a vector.

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

Returns

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

Definition at line 311 of file matrix33.cpp.

```

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

```

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

Operator for reflexive scaling of the matrix by a scalar.

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

Definition at line 253 of file matrix33.cpp.

```

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

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

Operator for reflexive multiplication of two matrices.

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

Definition at line 295 of file matrix33.cpp.

```

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

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

Operator for addition of two matrices.

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

Returns

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

Definition at line 155 of file matrix33.cpp.

```

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

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

Operator for reflexive addition of two matrices.

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

Definition at line 175 of file matrix33.cpp.

```

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

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

Operator for the subtraction of two matrices.

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

Returns

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

Definition at line 194 of file matrix33.cpp.

```

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

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

Operator for reflexive subtraction of two matrices.

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

Definition at line 214 of file matrix33.cpp.

```

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

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

Transpose.

Performs the transpose of the current matrix.

Returns

A new matrix with the transpose of the current matrix.

Definition at line 333 of file matrix33.cpp.

```

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

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

Determinant.

Calculates the determinant of the current matrix.

Returns

Returns the determinant of the current matrix.

Definition at line 354 of file matrix33.cpp.

```

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

5.7.3.17 void Stress::populateMatrix()

Construct the stress tensor from the principal and shear stresses.

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

Definition at line 52 of file stress.cpp.

```

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

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

Rotate the stress tensor from one coordinate system to another.

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

Parameters

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

Returns

Rotated stress tensor.

Definition at line 105 of file stress.cpp.

```

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

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

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

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

Parameters

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

Definition at line 93 of file matrix33.cpp.

```

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

5.7.4 Field Documentation

5.7.4.1 double Stress::principalStresses[3] [protected]

The three principal stresses: s11, s22, s33.

Definition at line 27 of file stress.h.

5.7.4.2 double Stress::shearStresses[3] [protected]

The three shear stresses: s12, s13, s23,

Definition at line 31 of file stress.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:

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

5.8 Vector3d Class Reference

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

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

Public Member Functions

- [Vector3d](#) ()
Default constructor.
- [Vector3d](#) (double *a)
Constructor with values provided in an array.
- [Vector3d](#) (double a1, double a2, double a3)
Constructor with values provided explicitly.
- void [setValue](#) (int index, double value)
Function to set the value of an element of the vector.
- void [setVector](#) (double *a)
Function to set the value of the entire vector using an array.
- double [getValue](#) (int index)
Function to get the value of an element of the vector.
- double * [getVector](#) ()
Function to get the values of the elements of the vector in an array.
- double [sum](#) ()
Computes the sum of the elements of the vector.
- double [magnitude](#) ()
Computes the magnitude of the vector.
- [Vector3d](#) [normalize](#) ()
Returns the vector normalized to be a unit vector.
- [Vector3d](#) [operator+](#) (const [Vector3d](#) &) const
Operator for addition of two vectors.
- void [operator+=](#) (const [Vector3d](#) &)
Operator for reflexive addition of two vectors.
- [Vector3d](#) [operator-](#) (const [Vector3d](#) &) const
Operator for the subtraction of two vectors.
- void [operator-=](#) (const [Vector3d](#) &)
Operator for reflexive subtraction of two vectors.
- [Vector3d](#) [operator*](#) (const double &) const
Operator for scaling the vector by a scalar.
- void [operator*=" \(const double &\)](#)

Operator for reflexive scaling of the vector by a scalar.

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

Operator for the scalar product of two vectors.

- `Vector3d operator^` (const `Vector3d` &) const

Operator for the vector product of two vectors.

- void `operator^=` (const `Vector3d` &)

Operator for reflexive vector product of two vectors.

Protected Attributes

- double `x` [3]

The elements of the vector.

5.8.1 Detailed Description

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

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

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

5.8.2 Constructor & Destructor Documentation

5.8.2.1 `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 }
```

5.8.2.2 `Vector3d::Vector3d (double * a)`

Constructor with values provided in an array.

Initializes the vector with the values provided in the array.

Parameters

<code>a</code>	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 }
```

5.8.2.3 Vector3d::Vector3d (double *a1*, double *a2*, double *a3*)

Constructor with values provided explicitly.

Initializes the vector with the three values provided as arguments.

Parameters

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

Definition at line 42 of file vector3d.cpp.

```

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

5.8.3 Member Function Documentation**5.8.3.1 double Vector3d::getValue (int *index*)**

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

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

Parameters

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

Returns

The value of the element of the vector at the position

Definition at line 83 of file vector3d.cpp.

```

84 {
85     if (index>=0 && index<3)
86     {
87         return (this->x[index]);
88     }
89     else
90     {
91         return (0);
92     }
93 }
```

5.8.3.2 double * Vector3d::getVector ()

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

The vector is returned in an array.

Returns

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

Definition at line 100 of file vector3d.cpp.

```

101 {
```

```

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

```

5.8.3.3 double Vector3d::magnitude ()

Computes the magnitude of the vector.

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

Returns

The magnitude of the vector.

Definition at line 134 of file vector3d.cpp.

```

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

```

5.8.3.4 Vector3d Vector3d::normalize ()

Returns the vector normalized to be a unit vector.

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

Returns

Normalized vector.

Definition at line 152 of file vector3d.cpp.

```

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

```

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

Operator for scaling the vector by a scalar.

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

Returns

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

Definition at line 239 of file vector3d.cpp.

```

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

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

Operator for the scalar product of two vectors.

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

Returns

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

Definition at line 271 of file vector3d.cpp.

```

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

5.8.3.7 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 }
```

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

Operator for addition of two vectors.

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

Returns

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

Definition at line 173 of file vector3d.cpp.

```

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

```

5.8.3.9 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     for (i=0; i<3; i++)
194     {
195         this->x[i] += p.x[i];
196     }
197 }

```

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

Operator for the subtraction of two vectors.

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

Returns

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

Definition at line 206 of file vector3d.cpp.

```

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

```

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

Operator for reflexive subtraction of two vectors.

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

Definition at line 223 of file vector3d.cpp.

```

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

```

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

Operator for the vector product of two vectors.

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

Returns

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

Definition at line 289 of file vector3d.cpp.

```

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

```

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

Operator for reflexive vector product of two vectors.

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

Definition at line 304 of file vector3d.cpp.

```

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

```

5.8.3.14 void Vector3d::setValue (int index, double value)

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

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

Parameters

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

Definition at line 56 of file vector3d.cpp.

```

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

```

5.8.3.15 void Vector3d::setVector (double * a)

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

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

Parameters

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 }

```

5.8.3.16 double Vector3d::sum ()

Computes the sum of the elements of the vector.

Sums the elements of the vector and returns the result.

Returns

The sum of the elements of the vector.

Definition at line 116 of file vector3d.cpp.

```

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

```

5.8.4 Field Documentation

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

The elements of the vector.

Definition at line 26 of file vector3d.h.

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

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

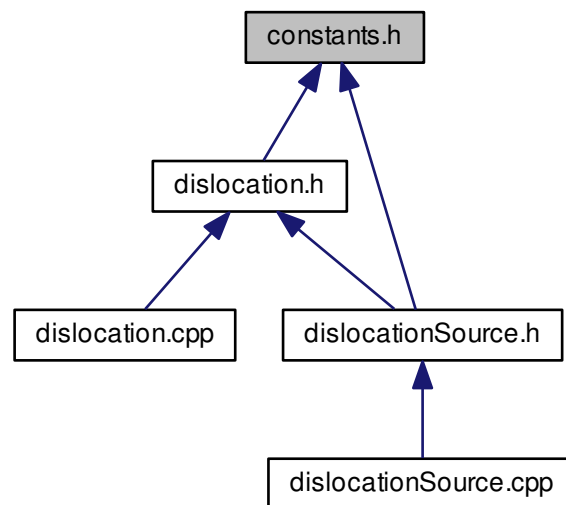
Chapter 6

File Documentation

6.1 constants.h File Reference

Definition of constants used in the program.

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



Macros

- #define **PI** 3.141592654
The irrational number pi.
- #define **SQRT2** 1.414213562
The square root of 2.
- #define **SQRT3** 1.732050808
The square root of 3.
- #define **SQRT5** 2.236067978
The square root of 5.

6.1.1 Detailed Description

Definition of constants used in the program.

Author

Adhish Majumdar

Version

0.0

Date

26/04/2013

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

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

6.1.2 Macro Definition Documentation

6.1.2.1 `#define PI 3.141592654`

The irrational number pi.

Definition at line 16 of file constants.h.

6.1.2.2 `#define SQRT2 1.414213562`

The square root of 2.

Definition at line 21 of file constants.h.

6.1.2.3 `#define SQRT3 1.732050808`

The square root of 3.

Definition at line 26 of file constants.h.

6.1.2.4 `#define SQRT5 2.236067978`

The square root of 5.

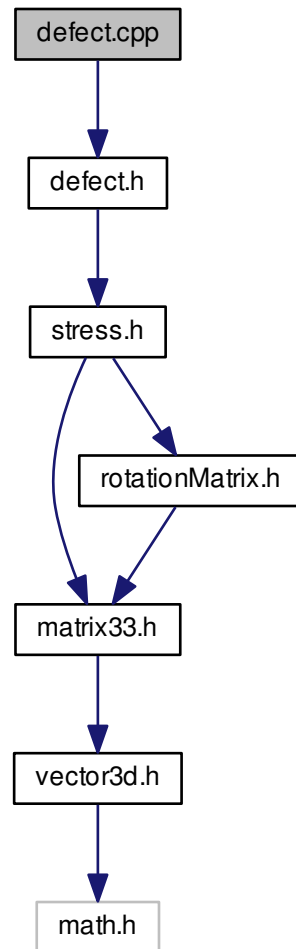
Definition at line 31 of file constants.h.

6.2 defect.cpp File Reference

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

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

Include dependency graph for defect.cpp:



6.2.1 Detailed Description

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

Author

Adhish Majumdar

Version

0.0

Date

22/04/2013

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

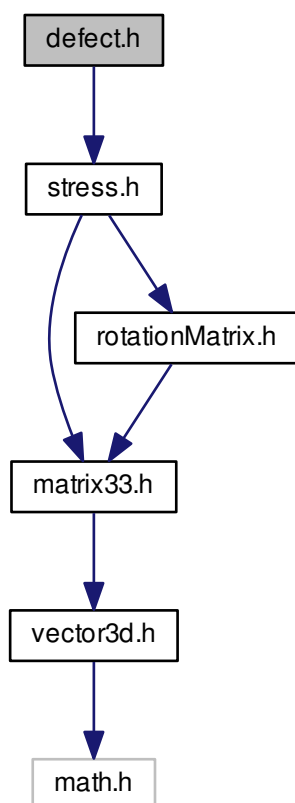
Definition in file [defect.cpp](#).

6.3 defect.h File Reference

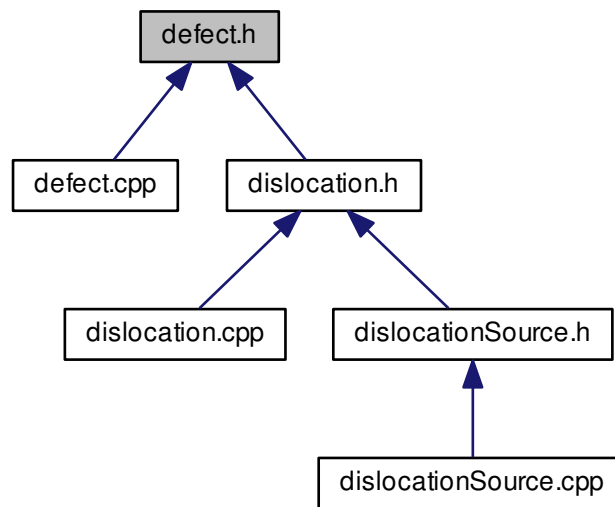
Definition of the [Defect](#) class.

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

Include dependency graph for defect.h:



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



Data Structures

- class [Defect](#)

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

6.3.1 Detailed Description

Definition of the [Defect](#) class.

Author

Adhish Majumdar

Version

0.0

Date

22/04/2013

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

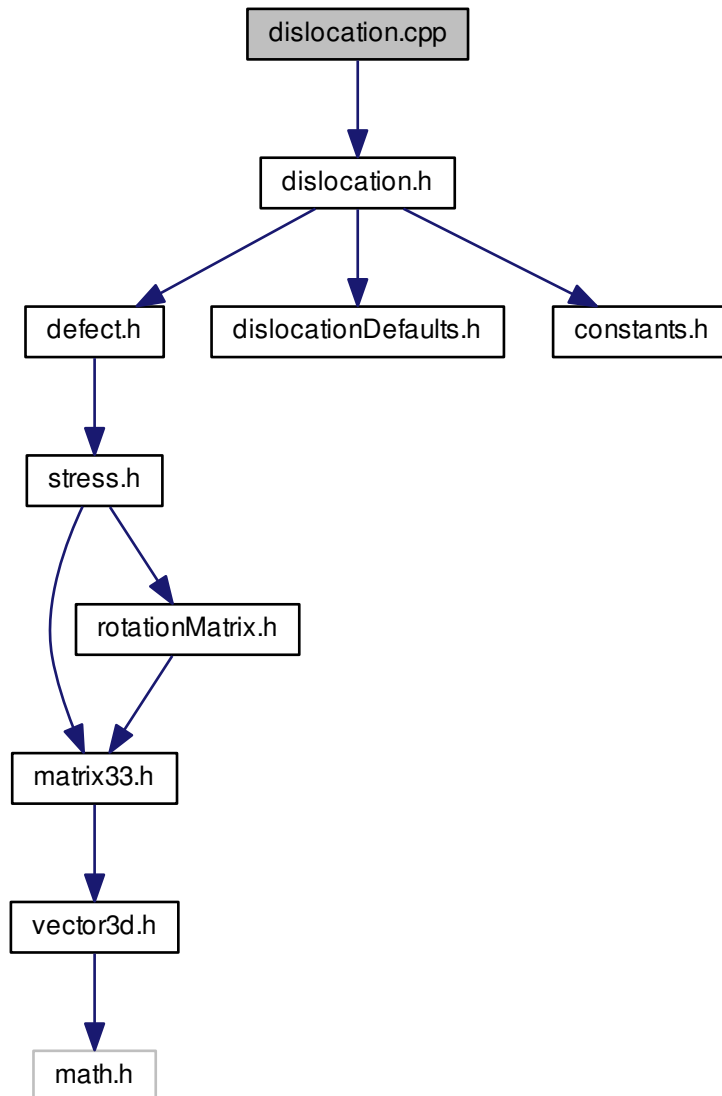
Definition in file [defect.h](#).

6.4 dislocation.cpp File Reference

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

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

Include dependency graph for dislocation.cpp:



6.4.1 Detailed Description

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

Author

Adhish Majumdar

Version

0.0

Date

29/04/2013

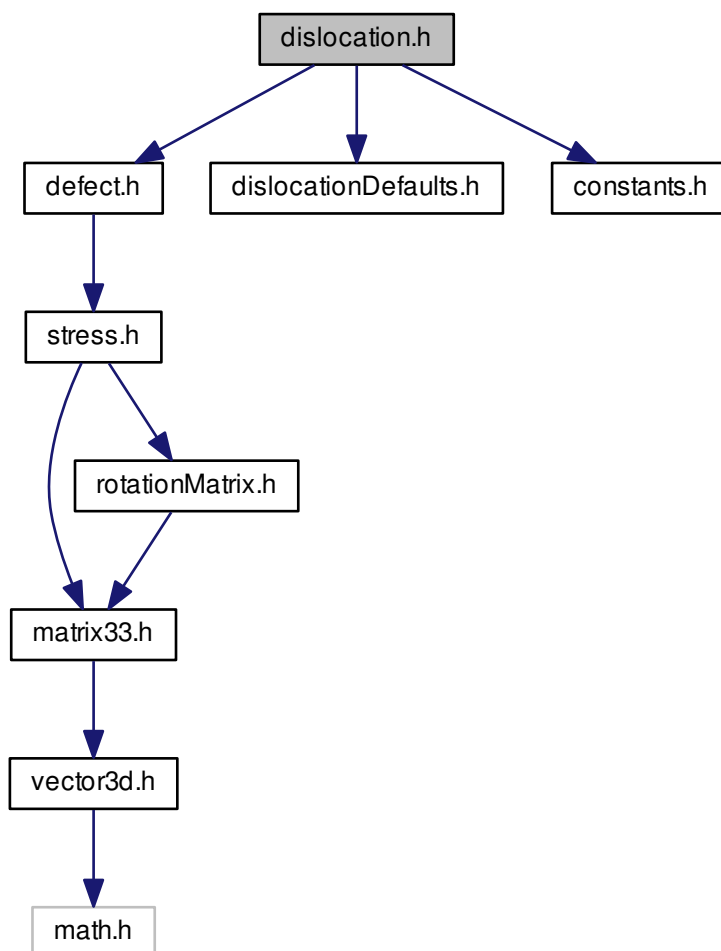
This file defines the constructors and member functions of the [Dislocation](#) class. This class inherits from the [Defect](#) class.

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

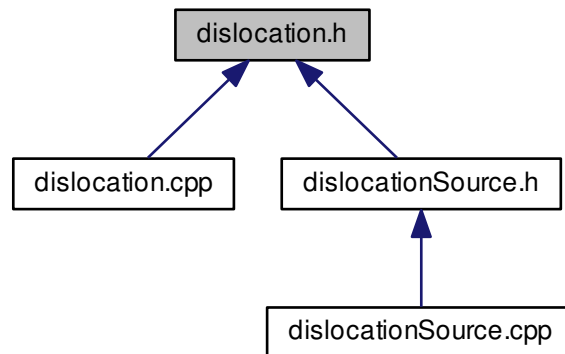
6.5 dislocation.h File Reference

Definition of the [Dislocation](#) class.

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



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



Data Structures

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

6.5.1 Detailed Description

Definition of the [Dislocation](#) class.

Author

Adhish Majumdar

Version

0.0

Date

29/04/2013

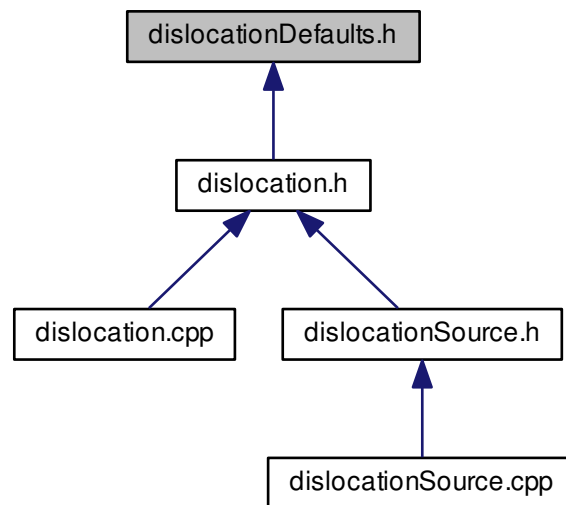
This file defines the [Dislocation](#) class representing a dislocation in the simulation. This class inherits from the [Defect](#) class.

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

6.6 dislocationDefaults.h File Reference

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

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



Macros

- `#define` [DEFAULT_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 21 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 25 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 29 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 16 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 34 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 38 of file [dislocationDefaults.h](#).

6.6.2.7 `#define DEFAULT_LINEVECTOR_2 -2.0`

Default value of the line vector z-coordinate.

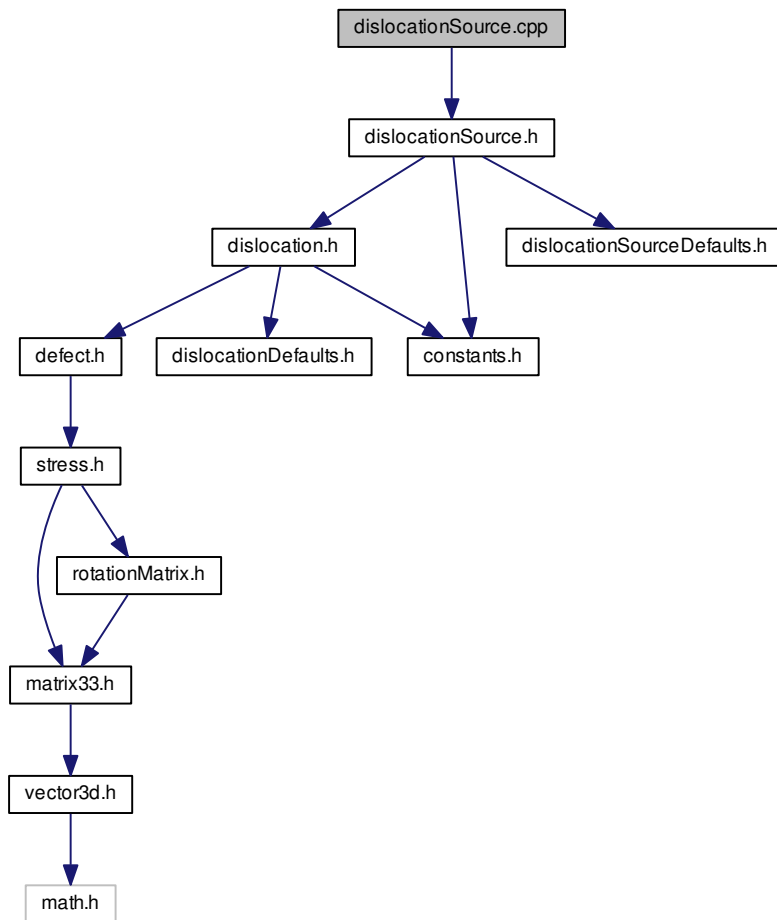
Definition at line 42 of file [dislocationDefaults.h](#).

6.7 [dislocationSource.cpp](#) File Reference

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

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

Include dependency graph for dislocationSource.cpp:



6.7.1 Detailed Description

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

Author

Adhish Majumdar

Version

0.0

Date

27/05/2013

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

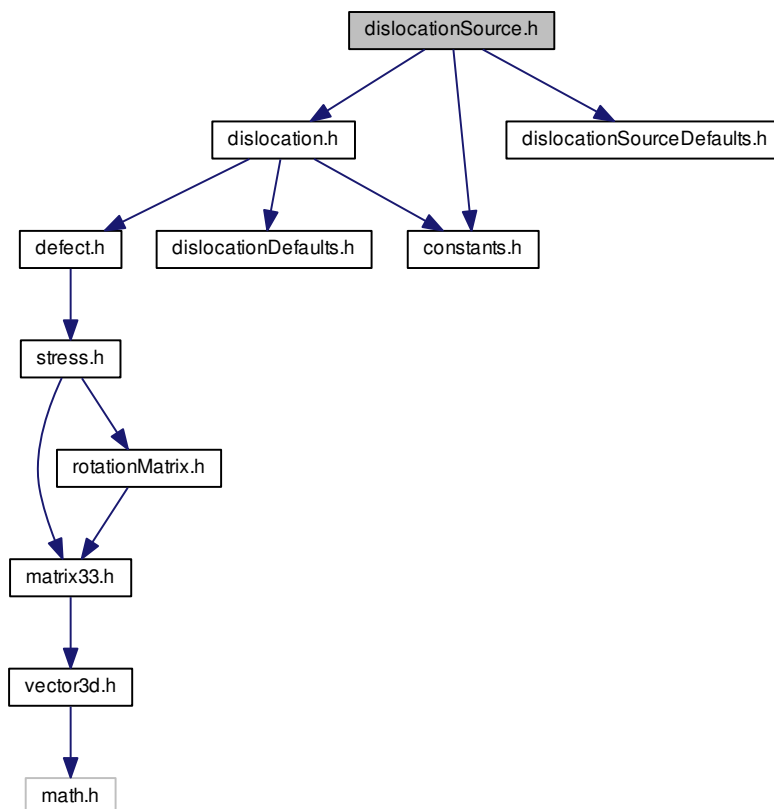
value for a certain amount of time (or number of iterations), it emits a dislocation dipole with a length that is a function of the applied stress.

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

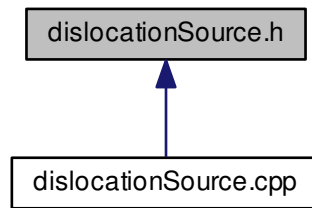
6.8 dislocationSource.h File Reference

Definition of the [DislocationSource](#) class.

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



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



Data Structures

- class [DislocationSource](#)

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

6.8.1 Detailed Description

Definition of the [DislocationSource](#) class.

Author

Adhish Majumdar

Version

0.0

Date

27/05/2013

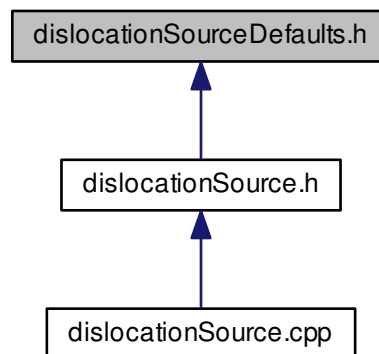
This file defines the [DislocationSource](#) class representing a source of dislocations in the simulation. This class inherits from the [Defect](#) class. This object is basically the representation of a Frank-Read source emitting dislocation dipoles. When the dislocation source experiences a shear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

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

6.9 dislocationSourceDefaults.h File Reference

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

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



Macros

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

6.9.1 Detailed Description

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

Author

Adhish Majumdar

Version

0.0

Date

02/05/2013

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

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

6.9.2 Macro Definition Documentation

6.9.2.1 `#define` [DEFAULT_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.

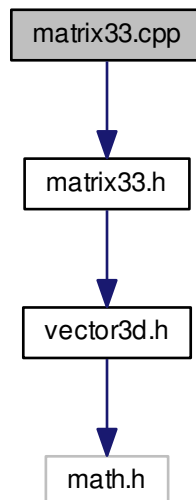
6.10 mainpage.dox File Reference

6.11 matrix33.cpp File Reference

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

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

Include dependency graph for matrix33.cpp:



6.11.1 Detailed Description

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

Author

Adhish Majumdar

Version

0.0

Date

22/04/2013

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

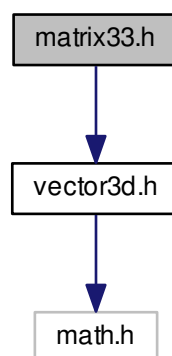
Definition in file [matrix33.cpp](#).

6.12 matrix33.h File Reference

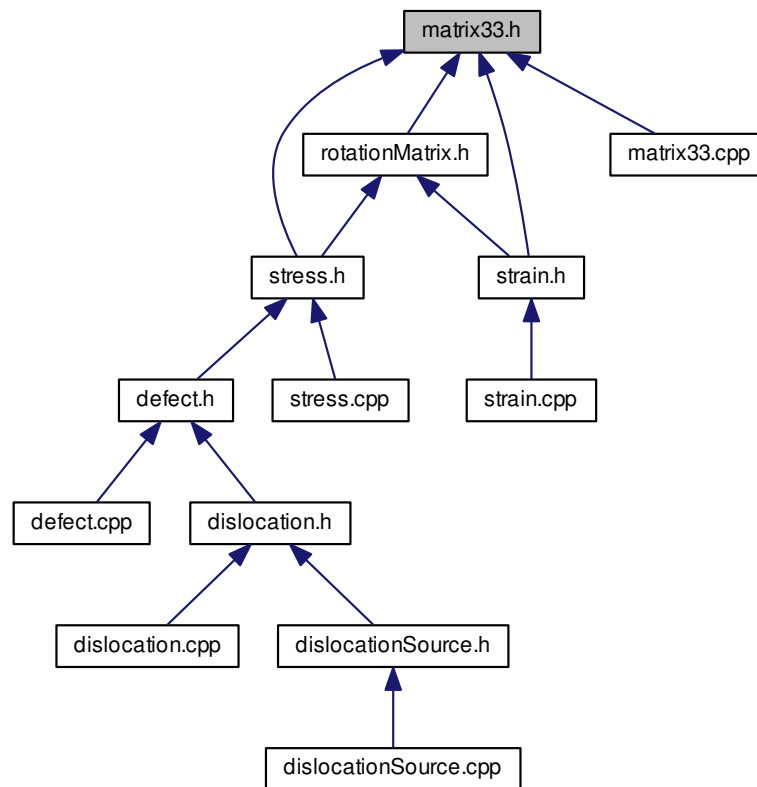
Definition of the [Matrix33](#) class.

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

Include dependency graph for matrix33.h:



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



Data Structures

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

6.12.1 Detailed Description

Definition of the [Matrix33](#) class.

Author

Adhish Majumdar

Version

0.0

Date

22/04/2013

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

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

6.13 rotationMatrix.cpp File Reference

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

6.13.1 Detailed Description

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

Author

Adhish Majumdar

Version

0.0

Date

25/04/2013

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

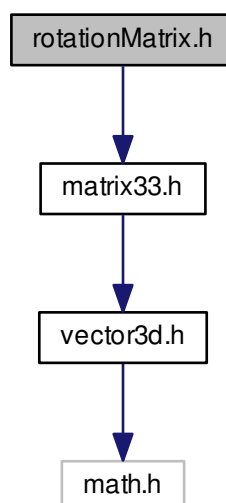
Definition in file [rotationMatrix.cpp](#).

6.14 rotationMatrix.h File Reference

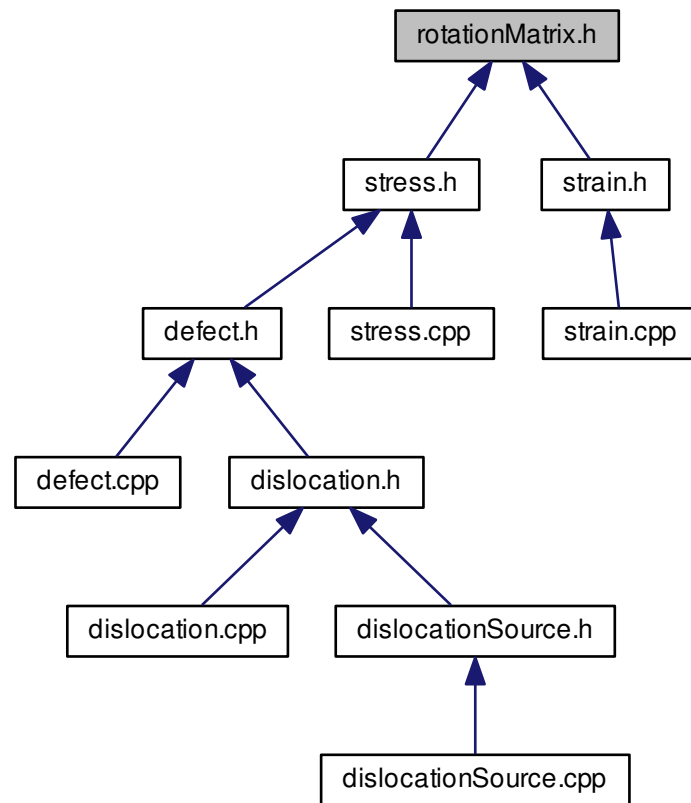
Definition of the [RotationMatrix](#) class.

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

Include dependency graph for rotationMatrix.h:



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



Data Structures

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

6.14.1 Detailed Description

Definition of the [RotationMatrix](#) class.

Author

Adhish Majumdar

Version

0.0

Date

25/04/2013

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

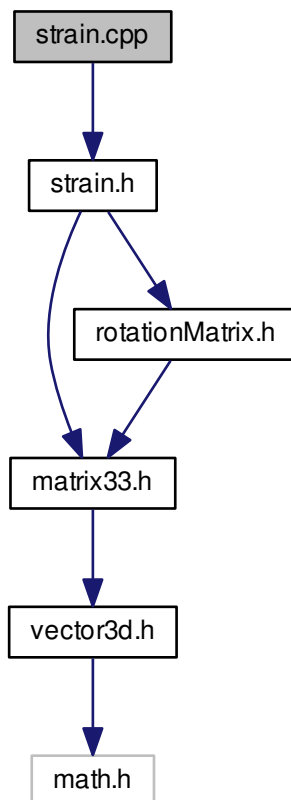
Definition in file [rotationMatrix.h](#).

6.15 strain.cpp File Reference

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

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

Include dependency graph for strain.cpp:



6.15.1 Detailed Description

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

Author

Adhish Majumdar

Version

0.0

Date

22/04/2013

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

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

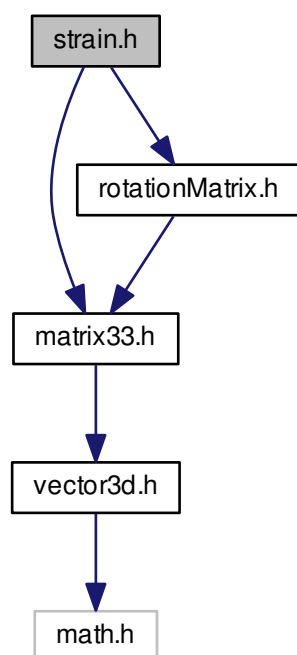
6.16 strain.h File Reference

Definition of the [Strain](#) class.

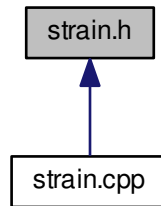
```
#include "matrix33.h"
```

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

Include dependency graph for strain.h:



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



Data Structures

- class [Strain](#)

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

6.16.1 Detailed Description

Definition of the [Strain](#) class.

Author

Adhish Majumdar

Version

0.0

Date

25/04/2013

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

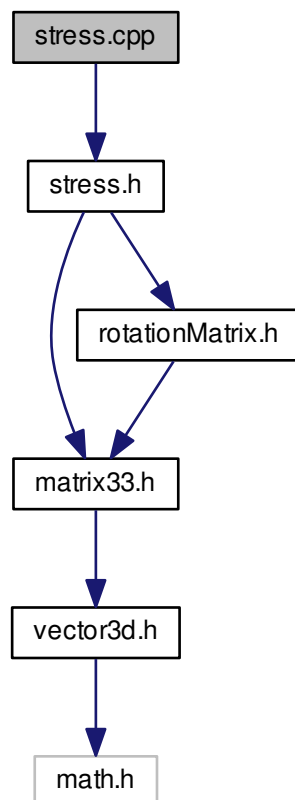
Definition in file [strain.h](#).

6.17 stress.cpp File Reference

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

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

Include dependency graph for stress.cpp:



6.17.1 Detailed Description

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

Author

Adhish Majumdar

Version

0.0

Date

22/04/2013

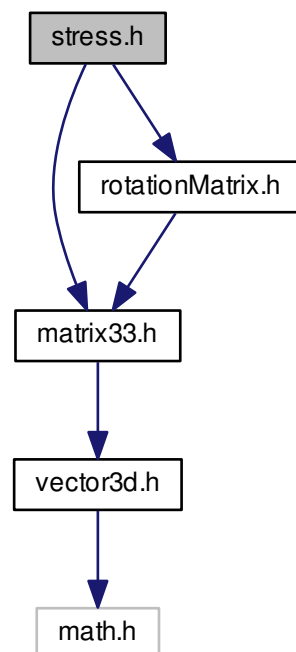
This file defines the member functions of the [Stress](#) class for the stress tensor.

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

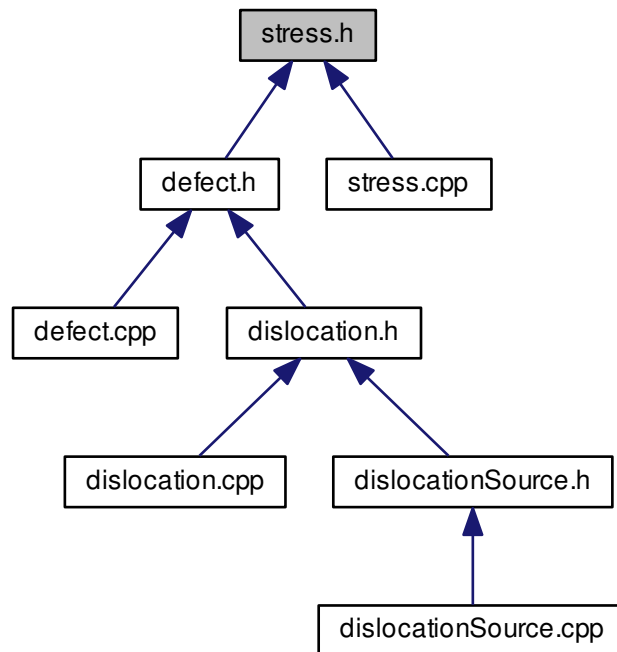
6.18 stress.h File Reference

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.

6.18.1 Detailed Description

Definition of the [Stress](#) class.

Author

Adhish Majumdar

Version

0.0

Date

22/04/2013

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

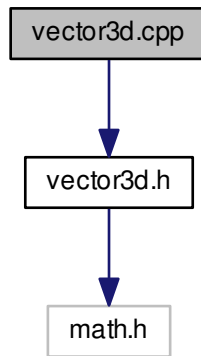
Definition in file [stress.h](#).

6.19 vector3d.cpp File Reference

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

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

Include dependency graph for vector3d.cpp:



6.19.1 Detailed Description

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

Author

Adhish Majumdar

Version

0.0

Date

29/04/2013

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

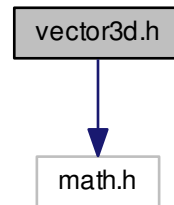
Definition in file [vector3d.cpp](#).

6.20 vector3d.h File Reference

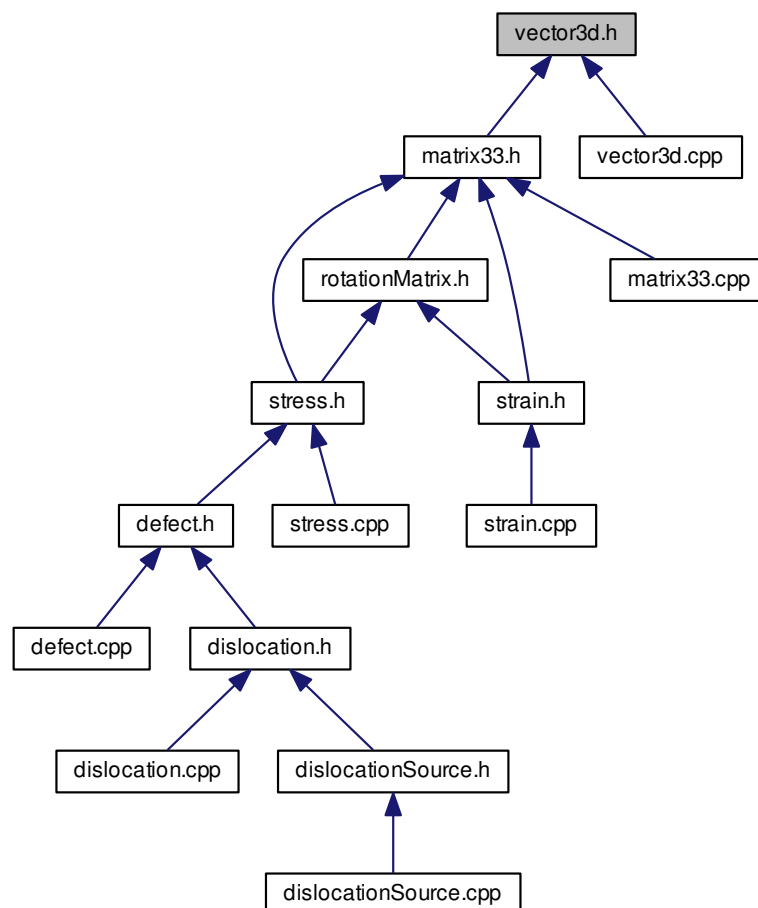
Definition of the [Vector3d](#) class.

```
#include <math.h>
```

Include dependency graph for vector3d.h:



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



Data Structures

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

6.20.1 Detailed Description

Definition of the [Vector3d](#) class.

Author

Adhish Majumdar

Version

0.0

Date

29/04/2013

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

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

Index

- adjugate
 - Matrix33, 35
 - RotationMatrix, 43
 - Strain, 52
 - Stress, 62
- bmag
 - Dislocation, 23
 - DislocationSource, 31
- bvec
 - Dislocation, 23
 - DislocationSource, 31
- calculateRotationMatrix
 - Dislocation, 18
- constants.h, 79
 - PI, 80
 - SQRT2, 80
 - SQRT3, 80
 - SQRT5, 80
- DEFAULT_BURGERS_0
 - dislocationDefaults.h, 88
- DEFAULT_BURGERS_1
 - dislocationDefaults.h, 88
- DEFAULT_BURGERS_2
 - dislocationDefaults.h, 88
- DEFAULT_NITERATIONS
 - dislocationSourceDefaults.h, 92
- Defect, 9
 - Defect, 10, 11
 - getPosition, 11
 - getX, 12
 - getY, 12
 - getZ, 12
 - pos, 14
 - setPosition, 12, 13
 - setX, 13
 - setY, 13
 - setZ, 14
 - stressField, 14
- defect.cpp, 80
- defect.h, 82
- Dislocation, 15
 - bmag, 23
 - bvec, 23
 - calculateRotationMatrix, 18
 - Dislocation, 17
 - getBurgers, 18
 - getLineVector, 18
 - getPosition, 18, 19
 - getX, 19
 - getY, 19
 - getZ, 19
 - lvec, 23
 - mobile, 24
 - pos, 24
 - rotationMatrix, 24
 - setBurgers, 20
 - setLineVector, 20
 - setMobile, 20
 - setPinned, 20
 - setPosition, 21
 - setX, 21
 - setY, 21
 - setZ, 22
 - stressField, 22
 - stressFieldLocal, 23
- dislocation.cpp, 83
- dislocation.h, 85
- dislocationDefaults.h, 86
 - DEFAULT_BURGERS_0, 88
 - DEFAULT_BURGERS_1, 88
 - DEFAULT_BURGERS_2, 88
- DislocationSource, 24
 - bmag, 31
 - bvec, 31
 - DislocationSource, 26
 - DislocationSource, 26
 - getPosition, 27
 - getX, 27
 - getY, 28
 - getZ, 28
 - lvec, 31
 - mobile, 31
 - nIterations, 31
 - pos, 31
 - rotationMatrix, 31
 - setBurgers, 28
 - setLineVector, 28
 - setPosition, 29
 - setX, 29
 - setY, 30
 - setZ, 30
 - stressField, 30
 - tauCritical, 32
- dislocationSource.cpp, 88
- dislocationSource.h, 90
- dislocationSourceDefaults.h, 91

- getBurgers
 - Dislocation, [18](#)
- getLineVector
 - Dislocation, [18](#)
- getPosition
 - Defect, [11](#)
 - Dislocation, [18](#), [19](#)
 - DislocationSource, [27](#)
- getPrincipalStrains
 - Strain, [53](#)
- getPrincipalStresses
 - Stress, [63](#)
- getShearStrains
 - Strain, [53](#)
- getShearStresses
 - Stress, [63](#)
- getValue
 - Matrix33, [35](#)
 - RotationMatrix, [44](#)
 - Strain, [53](#)
 - Stress, [63](#)
 - Vector3d, [72](#)
- getVector
 - Vector3d, [73](#)
- getX
 - Defect, [12](#)
 - Dislocation, [19](#)
 - DislocationSource, [27](#)
- getY
 - Defect, [12](#)
 - Dislocation, [19](#)
 - DislocationSource, [28](#)
- getZ
 - Defect, [12](#)
 - Dislocation, [19](#)
 - DislocationSource, [28](#)
- Ivec
 - Dislocation, [23](#)
 - DislocationSource, [31](#)
- magnitude
 - Vector3d, [73](#)
- mainpage.dox, [93](#)
- Matrix33, [32](#)
 - adjugate, [35](#)
 - getValue, [35](#)
 - Matrix33, [33](#), [34](#)
 - operator*, [36](#), [37](#)
 - operator*=[37](#), [38](#)
 - operator~, [40](#)
 - operator^, [39](#)
 - operator+, [38](#)
 - operator+=, [38](#)
 - operator-, [39](#)
 - operator-=, [39](#)
 - setValue, [40](#)
 - x, [41](#)
- matrix33.cpp, [93](#)
- matrix33.h, [94](#)
- mobile
 - Dislocation, [24](#)
 - DislocationSource, [31](#)
- nIterations
 - DislocationSource, [31](#)
- normalize
 - Vector3d, [73](#)
- operator*
 - Matrix33, [36](#), [37](#)
 - RotationMatrix, [45](#), [46](#)
 - Strain, [54](#), [55](#)
 - Stress, [64](#), [65](#)
 - Vector3d, [74](#)
- operator*=
 - Matrix33, [37](#), [38](#)
 - RotationMatrix, [46](#)
 - Strain, [55](#), [56](#)
 - Stress, [66](#)
 - Vector3d, [74](#)
- operator~
 - Matrix33, [40](#)
 - RotationMatrix, [48](#)
 - Strain, [58](#)
 - Stress, [68](#)
- operator^
 - Matrix33, [39](#)
 - RotationMatrix, [48](#)
 - Strain, [57](#)
 - Stress, [68](#)
 - Vector3d, [76](#)
- operator^=
 - Vector3d, [76](#)
- operator+
 - Matrix33, [38](#)
 - RotationMatrix, [47](#)
 - Strain, [56](#)
 - Stress, [66](#)
 - Vector3d, [75](#)
- operator+=
 - Matrix33, [38](#)
 - RotationMatrix, [47](#)
 - Strain, [56](#)
 - Stress, [67](#)
 - Vector3d, [75](#)
- operator-
 - Matrix33, [39](#)
 - RotationMatrix, [47](#)
 - Strain, [57](#)
 - Stress, [67](#)
 - Vector3d, [75](#)
- operator-=
 - Matrix33, [39](#)
 - RotationMatrix, [48](#)
 - Strain, [57](#)
 - Stress, [67](#)
 - Vector3d, [76](#)

- PI
 - constants.h, 80
- populateMatrix
 - Strain, 58
 - Stress, 68
- pos
 - Defect, 14
 - Dislocation, 24
 - DislocationSource, 31
- principalStrains
 - Strain, 59
- principalStresses
 - Stress, 70
- rotate
 - Strain, 58
 - Stress, 69
- RotationMatrix, 41
 - adjugate, 43
 - getValue, 44
 - operator*, 45, 46
 - operator*=: 46
 - operator~, 48
 - operator^, 48
 - operator+, 47
 - operator+=, 47
 - operator-, 47
 - operator-=, 48
 - RotationMatrix, 43
 - RotationMatrix, 43
 - setValue, 49
 - x, 49
- rotationMatrix
 - Dislocation, 24
 - DislocationSource, 31
- rotationMatrix.cpp, 96
- rotationMatrix.h, 96
- SQRT2
 - constants.h, 80
- SQRT3
 - constants.h, 80
- SQRT5
 - constants.h, 80
- setBurgers
 - Dislocation, 20
 - DislocationSource, 28
- setLineVector
 - Dislocation, 20
 - DislocationSource, 28
- setMobile
 - Dislocation, 20
- setPinned
 - Dislocation, 20
- setPosition
 - Defect, 12, 13
 - Dislocation, 21
 - DislocationSource, 29
- setValue
 - Matrix33, 40
 - RotationMatrix, 49
 - Strain, 59
 - Stress, 69
 - Vector3d, 77
- setVector
 - Vector3d, 77
- setX
 - Defect, 13
 - Dislocation, 21
 - DislocationSource, 29
- setY
 - Defect, 13
 - Dislocation, 21
 - DislocationSource, 30
- setZ
 - Defect, 14
 - Dislocation, 22
 - DislocationSource, 30
- shearStrains
 - Strain, 59
- shearStresses
 - Stress, 70
- Strain, 50
 - adjugate, 52
 - getPrincipalStrains, 53
 - getShearStrains, 53
 - getValue, 53
 - operator*, 54, 55
 - operator*=: 55, 56
 - operator~, 58
 - operator^, 57
 - operator+, 56
 - operator+=, 56
 - operator-, 57
 - operator-=, 57
 - populateMatrix, 58
 - principalStrains, 59
 - rotate, 58
 - setValue, 59
 - shearStrains, 59
 - Strain, 51, 52
 - x, 60
- strain.cpp, 98
- strain.h, 99
- Stress, 60
 - adjugate, 62
 - getPrincipalStresses, 63
 - getShearStresses, 63
 - getValue, 63
 - operator*, 64, 65
 - operator*=: 66
 - operator~, 68
 - operator^, 68
 - operator+, 66
 - operator+=, 67
 - operator-, 67
 - operator-=, 67

- populateMatrix, 68
- principalStresses, 70
- rotate, 69
- setValue, 69
- shearStresses, 70
- Stress, 62
- x, 70
- stress.cpp, 100
- stress.h, 102
- stressField
 - Defect, 14
 - Dislocation, 22
 - DislocationSource, 30
- stressFieldLocal
 - Dislocation, 23
- sum
 - Vector3d, 77
- tauCritical
 - DislocationSource, 32
- Vector3d, 70
 - getValue, 72
 - getVector, 73
 - magnitude, 73
 - normalize, 73
 - operator*, 74
 - operator*=: 74
 - operator[^], 76
 - operator[^]=: 76
 - operator+, 75
 - operator+=, 75
 - operator-, 75
 - operator-=, 76
 - setValue, 77
 - setVector, 77
 - sum, 77
 - Vector3d, 71, 72
 - x, 78
- vector3d.cpp, 104
- vector3d.h, 104
- x
 - Matrix33, 41
 - RotationMatrix, 49
 - Strain, 60
 - Stress, 70
 - Vector3d, 78