

MathematicalFunctions

5.3

Generated by Doxygen 1.8.14

Contents

1	Module Index	1
1.1	Modules	1
2	Namespace Index	3
2.1	Namespace List	3
3	Data Structure Index	5
3.1	Data Structures	5
4	File Index	7
4.1	File List	7
5	Module Documentation	9
5.1	Models	9
5.1.1	Detailed Description	9
5.2	Utils	10
5.2.1	Detailed Description	10
5.3	Math	11
5.3.1	Detailed Description	11
5.3.2	Macro Definition Documentation	11
5.3.2.1	IDENTITY_3X3	11
6	Namespace Documentation	13
6.1	jeod Namespace Reference	13
6.1.1	Detailed Description	13
6.1.2	Variable Documentation	13
6.1.2.1	SQRT_DBL_MIN	13

7 Data Structure Documentation	15
7.1 jeod::GaussQuadrature Class Reference	15
7.1.1 Detailed Description	15
7.1.2 Field Documentation	15
7.1.2.1 gauss_weights	15
7.1.2.2 gauss_xvalues	16
7.1.2.3 max_order	16
7.2 jeod::MathMessages Class Reference	16
7.2.1 Detailed Description	17
7.2.2 Constructor & Destructor Documentation	17
7.2.2.1 MathMessages() [1/2]	17
7.2.2.2 MathMessages() [2/2]	17
7.2.3 Member Function Documentation	17
7.2.3.1 operator=()	17
7.2.4 Field Documentation	17
7.2.4.1 ill_conditioned	17
7.3 jeod::Matrix3x3 Class Reference	18
7.3.1 Detailed Description	19
7.3.2 Member Function Documentation	19
7.3.2.1 add()	19
7.3.2.2 copy()	19
7.3.2.3 cross_matrix()	20
7.3.2.4 decr()	20
7.3.2.5 identity()	20
7.3.2.6 incr()	21
7.3.2.7 initialize()	21
7.3.2.8 invert()	21
7.3.2.9 invert_symmetric()	22
7.3.2.10 negate() [1/2]	23
7.3.2.11 negate() [2/2]	23

7.3.2.12	<code>outer_product()</code>	23
7.3.2.13	<code>print()</code>	24
7.3.2.14	<code>product()</code>	24
7.3.2.15	<code>product_left_transpose()</code>	24
7.3.2.16	<code>product_right_transpose()</code>	25
7.3.2.17	<code>product_transpose_transpose()</code>	25
7.3.2.18	<code>scale()</code> [1/2]	26
7.3.2.19	<code>scale()</code> [2/2]	26
7.3.2.20	<code>subtract()</code>	27
7.3.2.21	<code>transform_matrix()</code>	27
7.3.2.22	<code>transpose()</code> [1/2]	28
7.3.2.23	<code>transpose()</code> [2/2]	28
7.3.2.24	<code>transpose_transform_matrix()</code>	28
7.4	<code>jeod::Numerical Class Reference</code>	29
7.4.1	Detailed Description	29
7.4.2	Member Function Documentation	29
7.4.2.1	<code>compare_exact()</code>	29
7.4.2.2	<code>fabs()</code>	30
7.4.2.3	<code>square()</code>	30
7.4.2.4	<code>square_incr()</code>	31
7.5	<code>jeod::Vector3 Class Reference</code>	31
7.5.1	Detailed Description	33
7.5.2	Member Function Documentation	33
7.5.2.1	<code>copy()</code>	33
7.5.2.2	<code>cross()</code>	33
7.5.2.3	<code>cross_decr()</code>	34
7.5.2.4	<code>cross_incr()</code>	34
7.5.2.5	<code>decr()</code> [1/2]	35
7.5.2.6	<code>decr()</code> [2/2]	35
7.5.2.7	<code>diff()</code>	36

7.5.2.8	<code>dot()</code>	36
7.5.2.9	<code>fill()</code>	36
7.5.2.10	<code>incr()</code> [1/2]	37
7.5.2.11	<code>incr()</code> [2/2]	37
7.5.2.12	<code>initialize()</code>	38
7.5.2.13	<code>negate()</code> [1/2]	38
7.5.2.14	<code>negate()</code> [2/2]	38
7.5.2.15	<code>normalize()</code> [1/2]	39
7.5.2.16	<code>normalize()</code> [2/2]	39
7.5.2.17	<code>scale()</code> [1/2]	40
7.5.2.18	<code>scale()</code> [2/2]	40
7.5.2.19	<code>scale_decr()</code>	41
7.5.2.20	<code>scale_incr()</code>	41
7.5.2.21	<code>sum()</code> [1/2]	42
7.5.2.22	<code>sum()</code> [2/2]	42
7.5.2.23	<code>transform()</code> [1/2]	42
7.5.2.24	<code>transform()</code> [2/2]	43
7.5.2.25	<code>transform_decr()</code>	43
7.5.2.26	<code>transform_incr()</code>	44
7.5.2.27	<code>transform_transpose()</code> [1/2]	44
7.5.2.28	<code>transform_transpose()</code> [2/2]	45
7.5.2.29	<code>transform_transpose_decr()</code>	45
7.5.2.30	<code>transform_transpose_incr()</code>	46
7.5.2.31	<code>unit()</code>	46
7.5.2.32	<code>vmag()</code>	47
7.5.2.33	<code>vmagsq()</code>	47
7.5.2.34	<code>zero_small()</code>	48

8 File Documentation	49
8.1 dm_invert.cc File Reference	49
8.1.1 Detailed Description	49
8.2 dm_invert_symm.cc File Reference	49
8.2.1 Detailed Description	50
8.3 gauss_quadrature.cc File Reference	50
8.3.1 Detailed Description	50
8.4 gauss_quadrature.hh File Reference	50
8.4.1 Detailed Description	50
8.5 macro_def.hh File Reference	50
8.5.1 Detailed Description	51
8.6 macro_undef.hh File Reference	51
8.6.1 Detailed Description	51
8.7 math_messages.cc File Reference	51
8.7.1 Detailed Description	51
8.7.2 Macro Definition Documentation	51
8.7.2.1 MAKE_MATH_MESSAGE_CODE	52
8.8 math_messages.hh File Reference	52
8.8.1 Detailed Description	52
8.9 matrix3x3.hh File Reference	52
8.9.1 Detailed Description	53
8.10 matrix3x3_inline.hh File Reference	53
8.10.1 Detailed Description	53
8.11 numerical.hh File Reference	53
8.11.1 Detailed Description	53
8.12 numerical_inline.hh File Reference	54
8.12.1 Detailed Description	54
8.13 vector3.hh File Reference	54
8.13.1 Detailed Description	54
8.14 vector3_inline.hh File Reference	55
8.14.1 Detailed Description	55
Index	57

Chapter 1

Module Index

1.1 Modules

Here is a list of all modules:

Models	9
Utils	10
Math	11

Chapter 2

Namespace Index

2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

jeod	Namespace jeod	13
----------------------	--------------------------	--------------------

Chapter 3

Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

jeod::GaussQuadrature	15
jeod::MathMessages Specifies the message IDs used in the math model	16
jeod::Matrix3x3 Provides static methods for operations that involve 3x3 matrices	18
jeod::Numerical Provides miscellaneous numerical functions	29
jeod::Vector3 Provides static methods for operations that involve 3-vectors	31

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

dm_invert.cc	
Define Matrix3x3::invert	49
dm_invert_symm.cc	
Define Matrix3x3::invert_symmetric	49
gauss_quadrature.cc	
Define Gauss Quadrature functionality	50
gauss_quadrature.hh	
Gauss Quadrature implementation	50
macro_def.hh	
Math macro definitions	50
macro_undef.hh	
Math macro undefinitions	51
math_messages.cc	
Implement the class MathMessages	51
math_messages.hh	
Define the class MathMessages	52
matrix3x3.hh	
Matrix math inline functions	52
matrix3x3_inline.hh	
Matrix math inline functions	53
numerical.hh	
Miscellaneous math inline functions	53
numerical_inline.hh	
Vector math inline functions	54
vector3.hh	
Vector math inline functions	54
vector3_inline.hh	
Vector math inline functions	55

Chapter 5

Module Documentation

5.1 Models

Modules

- [Utils](#)

5.1.1 Detailed Description

5.2 Utils

Modules

- [Math](#)

5.2.1 Detailed Description

5.3 Math

Files

- file [gauss_quadrature.hh](#)
Gauss Quadrature implementation.
- file [macro_def.hh](#)
Math macro definitions.
- file [macro_undef.hh](#)
Math macro undefinitions.
- file [math_messages.hh](#)
Define the class MathMessages.
- file [matrix3x3.hh](#)
Matrix math inline functions.
- file [matrix3x3_inline.hh](#)
Matrix math inline functions.
- file [numerical.hh](#)
Miscellaneous math inline functions.
- file [numerical_inline.hh](#)
Vector math inline functions.
- file [vector3.hh](#)
Vector math inline functions.
- file [vector3_inline.hh](#)
Vector math inline functions.
- file [dm_invert.cc](#)
Define Matrix3x3::invert.
- file [dm_invert_symm.cc](#)
Define Matrix3x3::invert_symmetric.
- file [gauss_quadrature.cc](#)
Define Gauss Quadrature functionality.
- file [math_messages.cc](#)
Implement the class MathMessages.

Namespaces

- [jeod](#)
Namespace jeod.

Macros

- `#define IDENTITY_3X3 {1.0, 0.0, 0.0}, {0.0, 1.0, 0.0}, {0.0, 0.0, 1.0}`

5.3.1 Detailed Description

5.3.2 Macro Definition Documentation

5.3.2.1 IDENTITY_3X3

```
#define IDENTITY_3X3 {1.0, 0.0, 0.0}, {0.0, 1.0, 0.0}, {0.0, 0.0, 1.0}
```

Definition at line 56 of file `macro_def.hh`.

Chapter 6

Namespace Documentation

6.1 jeod Namespace Reference

Namespace jeod.

Data Structures

- class [GaussQuadrature](#)
- class [MathMessages](#)
Specifies the message IDs used in the math model.
- class [Matrix3x3](#)
Provides static methods for operations that involve 3x3 matrices.
- class [Numerical](#)
Provides miscellaneous numerical functions.
- class [Vector3](#)
Provides static methods for operations that involve 3-vectors.

Variables

- static constexpr double [SQRT_DBL_MIN](#) {1.4916681462400413e-154}

6.1.1 Detailed Description

Namespace jeod.

6.1.2 Variable Documentation

6.1.2.1 SQRT_DBL_MIN

```
constexpr double jeod::SQRT_DBL_MIN {1.4916681462400413e-154} [static]
```

Definition at line 66 of file `numerical_inline.hh`.

Referenced by `jeod::Numerical::square()`, and `jeod::Numerical::square_incr()`.

Chapter 7

Data Structure Documentation

7.1 jeod::GaussQuadrature Class Reference

```
#include <gauss_quadrature.hh>
```

Static Public Attributes

- static const int [max_order](#) = 8
- static const double [gauss_weights](#) [[max_order](#)+1][[max_order](#)]
- static const double [gauss_xvalues](#) [[max_order](#)+1][[max_order](#)]

7.1.1 Detailed Description

Definition at line 69 of file gauss_quadrature.hh.

7.1.2 Field Documentation

7.1.2.1 gauss_weights

```
const double jeod::GaussQuadrature::gauss_weights [static]
```

Initial value:

```
= {  
    {0.0000000, 0.0000000, 0.0000000, 0.0000000, 0.0000000, 0.0000000, 0.0000000, 0.0000000},  
    {2.0000000, 0.0000000, 0.0000000, 0.0000000, 0.0000000, 0.0000000, 0.0000000, 0.0000000},  
    {1.0000000, 1.0000000, 0.0000000, 0.0000000, 0.0000000, 0.0000000, 0.0000000, 0.0000000},  
    {0.5555556, 0.8888889, 0.5555556, 0.0000000, 0.0000000, 0.0000000, 0.0000000, 0.0000000},  
    {0.3478548, 0.6521452, 0.6521452, 0.3478548, 0.0000000, 0.0000000, 0.0000000, 0.0000000},  
    {0.2369269, 0.4786287, 0.5688889, 0.4786287, 0.2369269, 0.0000000, 0.0000000, 0.0000000},  
    {0.1713245, 0.3607616, 0.4679139, 0.4679139, 0.3607616, 0.1713245, 0.0000000, 0.0000000},  
    {0.1294850, 0.2797054, 0.3818301, 0.4179592, 0.3818301, 0.2797054, 0.1294850, 0.0000000},  
    {0.1012285, 0.2223810, 0.3137067, 0.3626838, 0.3626838, 0.3137067, 0.2223810, 0.1012285}  
}
```

Definition at line 73 of file gauss_quadrature.hh.

7.1.2.2 gauss_xvalues

```
const double jeod::GaussQuadrature::gauss_xvalues [static]
```

Initial value:

```
= {
  { 0.0000000, 0.0000000, 0.0000000, 0.0000000, 0.0000000, 0.0000000, 0.0000000, 0.0000000},
  { 0.0000000, 0.0000000, 0.0000000, 0.0000000, 0.0000000, 0.0000000, 0.0000000, 0.0000000},
  {-0.5773503, 0.5773503, 0.0000000, 0.0000000, 0.0000000, 0.0000000, 0.0000000, 0.0000000},
  {-0.7745967, 0.0000000, 0.7745967, 0.0000000, 0.0000000, 0.0000000, 0.0000000, 0.0000000},
  {-0.8611363, -0.3399810, 0.3399810, 0.8611363, 0.0000000, 0.0000000, 0.0000000, 0.0000000},
  {-0.9061798, -0.5384693, 0.0000000, 0.5384693, 0.9061798, 0.0000000, 0.0000000, 0.0000000},
  {-0.9324695, -0.6612094, -0.2386192, 0.2386192, 0.6612094, 0.9324695, 0.0000000, 0.0000000},
  {-0.9491079, -0.7415312, -0.4058452, 0.0000000, 0.4058452, 0.7415312, 0.9491079, 0.0000000},
  {-0.9602899, -0.7966665, -0.5255324, -0.1834346, 0.1834346, 0.5255324, 0.7966665, 0.9602899}
}
```

Definition at line 74 of file gauss_quadrature.hh.

7.1.2.3 max_order

```
const int jeod::GaussQuadrature::max_order = 8 [static]
```

Definition at line 72 of file gauss_quadrature.hh.

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

- [gauss_quadrature.hh](#)
- [gauss_quadrature.cc](#)

7.2 jeod::MathMessages Class Reference

Specifies the message IDs used in the math model.

```
#include <math_messages.hh>
```

Public Member Functions

- [MathMessages](#) ()=delete
- [MathMessages](#) (const [MathMessages](#) &)=delete
- [MathMessages](#) & [operator=](#) (const [MathMessages](#) &)=delete

Static Public Attributes

- static const char * [ill_conditioned](#) = "utils/math/" "ill_conditioned"
Error issued when an ill-conditioned matrix is detected.

7.2.1 Detailed Description

Specifies the message IDs used in the math model.

Definition at line 71 of file math_messages.hh.

7.2.2 Constructor & Destructor Documentation

7.2.2.1 MathMessages() [1/2]

```
jeod::MathMessages::MathMessages ( ) [delete]
```

7.2.2.2 MathMessages() [2/2]

```
jeod::MathMessages::MathMessages (
    const MathMessages & ) [delete]
```

7.2.3 Member Function Documentation

7.2.3.1 operator=()

```
MathMessages& jeod::MathMessages::operator= (
    const MathMessages & ) [delete]
```

7.2.4 Field Documentation

7.2.4.1 ill_conditioned

```
char const * jeod::MathMessages::ill_conditioned = "utils/math/" "ill_conditioned" [static]
```

Error issued when an ill-conditioned matrix is detected.

trick_units(-)

Definition at line 78 of file math_messages.hh.

Referenced by jeod::Matrix3x3::invert(), and jeod::Matrix3x3::invert_symmetric().

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

- [math_messages.hh](#)
- [math_messages.cc](#)

7.3 jeod::Matrix3x3 Class Reference

Provides static methods for operations that involve 3x3 matrices.

```
#include <matrix3x3.hh>
```

Static Public Member Functions

- static void **initialize** (double mat[3][3])
Zero-fill matrix: $mat[i][j] = 0.0$.
- static void **identity** (double mat[3][3])
Construct identity matrix: $mat[i][j] = \delta_{ij}$.
- static void **cross_matrix** (const double vec[3], double cross_mat[3][3])
Construct the skew symmetric cross product matrix: $mat[i][k] = \epsilon_{ijk} vec[j]$, ϵ_{ijk} is the Levi-Cevita symbol.
- static void **outer_product** (const double vec_left[3], const double vec_right[3], double prod[3][3])
*Construct the outer product of two vectors: $mat[i][j] = vec_left[i] * vec_right[j]$.*
- static void **negate** (double mat[3][3])
Negated matrix in-place: $mat[i][j] = -mat[i][j]$.
- static void **transpose** (double mat[3][3])
Transpose matrix in-place: $mat[i][j] = mat[j][i]$.
- static void **scale** (double scalar, double mat[3][3])
*Scale matrix in-place, $mat[i][j] = scalar * mat[i][j]$.*
- static void **incr** (const double addend[3][3], double mat[3][3])
Increment matrix in-place: $mat[i][j] = mat[i][j] + addend[i][j]$.
- static void **decr** (const double subtrahend[3][3], double mat[3][3])
Decrement matrix in-place: $mat[i][j] = mat[i][j] - subtrahend[i][j]$.
- static void **copy** (const double input_mat[3][3], double copy[3][3])
Copy matrix: $copy[i][j] = mat[i][j]$.
- static void **negate** (const double input_mat[3][3], double copy[3][3])
Negate matrix: $copy[i][j] = -mat[i][j]$.
- static void **transpose** (const double input_mat[3][3], double trans[3][3])
Transpose matrix: $copy[i][j] = mat[j][i]$.
- static void **scale** (const double mat[3][3], double scalar, double prod[3][3])
*Scale matrix: $copy[i][j] = scalar * mat[i][j]$.*
- static void **add** (const double augend[3][3], const double addend[3][3], double sum[3][3])
Add matrices: $sum[i][j] = augend[i][j] + addend[i][j]$.
- static void **subtract** (const double minuend[3][3], const double subtrahend[3][3], double diff[3][3])
Subtract matrices: $diff[i][j] = minuend[i][j] - subtrahend[i][j]$.
- static void **product** (const double mat_left[3][3], const double mat_right[3][3], double prod[3][3])
*Compute the matrix product $mat_left * mat_right$: $prod[i][j] = mat_left[i][k] * mat_right[k][j]$.*
- static void **product_left_transpose** (const double mat_left[3][3], const double mat_right[3][3], double prod[3][3])
*Compute the matrix product $mat_left^T * mat_right$: $prod[i][j] = mat_left[k][i] * mat_right[k][j]$.*
- static void **product_right_transpose** (const double mat_left[3][3], const double mat_right[3][3], double prod[3][3])
*Compute the matrix product $mat_left * mat_right^T$: $prod[i][j] = sum_k mat_left[i][k] * mat_right[j][k]$.*
- static void **product_transpose_transpose** (const double mat_left[3][3], const double mat_right[3][3], double prod[3][3])
*Compute the matrix product $mat_left^T * mat_right^T$: $prod[i][j] = sum_k mat_left[k][i] * mat_right[j][k]$.*
- static void **transform_matrix** (const double trans[3][3], const double mat[3][3], double prod[3][3])

- Compute the matrix product $\text{trans} * \text{mat} * \text{trans}^T$ $\text{prod}[i][j] = \text{trans}[i][k] * \text{mat}[k][l] * \text{trans}[j][l]$.

 - static void [transpose_transform_matrix](#) (const double trans[3][3], const double mat[3][3], double prod[3][3])

Compute the matrix product $\text{trans}^T * \text{mat} * \text{trans}$ $\text{prod}[i][j] = \text{trans}[k][i] * \text{mat}[k][l] * \text{trans}[l][j]$.

 - static int [invert](#) (const double matrix[3][3], double inverse[3][3])

Compute the inverse of a 3x3 matrix.

 - static int [invert_symmetric](#) (const double matrix[3][3], double inverse[3][3])

Compute the inverse of a symmetric 3x3 matrix.

 - static void [print](#) (const double mat[3][3])

Print matrix to standard error.

7.3.1 Detailed Description

Provides static methods for operations that involve 3x3 matrices.

Definition at line 71 of file matrix3x3.hh.

7.3.2 Member Function Documentation

7.3.2.1 add()

```
void jeod::Matrix3x3::add (
    const double augend[3][3],
    const double addend[3][3],
    double sum[3][3] ) [inline], [static]
```

Add matrices: $\text{sum}[i][j] = \text{augend}[i][j] + \text{addend}[i][j]$.

Parameters

in	<i>augend</i>	Matrix
in	<i>addend</i>	Matrix
out	<i>sum</i>	Sum

Definition at line 326 of file matrix3x3_inline.hh.

7.3.2.2 copy()

```
void jeod::Matrix3x3::copy (
    const double input_mat[3][3],
    double copy[3][3] ) [inline], [static]
```

Copy matrix: $\text{copy}[i][j] = \text{mat}[i][j]$.

Parameters

in	<i>input_mat</i>	Source matrix
out	<i>copy</i>	Matrix copy

Definition at line 237 of file matrix3x3_inline.hh.

Referenced by `negate()`.

7.3.2.3 cross_matrix()

```
void jeod::Matrix3x3::cross_matrix (
    const double vec[3],
    double cross_mat[3][3] ) [inline], [static]
```

Construct the skew symmetric cross product matrix: $\text{mat}[i][k] = \epsilon_{ijk} \text{vec}[j]$, ϵ_{ijk} is the Levi-Cevita symbol.

Parameters

in	<i>vec</i>	Vector
out	<i>cross_mat</i>	Cross product matrix

Definition at line 101 of file matrix3x3_inline.hh.

7.3.2.4 decr()

```
void jeod::Matrix3x3::decr (
    const double subtrahend[3][3],
    double mat[3][3] ) [inline], [static]
```

Decrement matrix in-place: $\text{mat}[i][j] = \text{mat}[i][j] - \text{subtrahend}[i][j]$.

Parameters

in	<i>subtrahend</i>	Decrement
in, out	<i>mat</i>	Decrement matrix

Definition at line 219 of file matrix3x3_inline.hh.

7.3.2.5 identity()

```
void jeod::Matrix3x3::identity (
    double mat[3][3] ) [inline], [static]
```

Construct identity matrix: $\text{mat}[i][j] = \text{delta_ij}$.

Parameters

out	<i>mat</i>	Identity matrix
-----	------------	-----------------

Definition at line 86 of file matrix3x3_inline.hh.

7.3.2.6 incr()

```
void jeod::Matrix3x3::incr (
    const double addend[3][3],
    double mat[3][3] ) [inline], [static]
```

Increment matrix in-place: $\text{mat}[i][j] = \text{mat}[i][j] + \text{addend}[i][j]$.

Parameters

in	<i>addend</i>	Increment
in, out	<i>mat</i>	Incremented matrix

Definition at line 201 of file matrix3x3_inline.hh.

7.3.2.7 initialize()

```
void jeod::Matrix3x3::initialize (
    double mat[3][3] ) [inline], [static]
```

Zero-fill matrix: $\text{mat}[i][j] = 0.0$.

Parameters

out	<i>mat</i>	Zero-filled matrix
-----	------------	--------------------

Definition at line 73 of file matrix3x3_inline.hh.

7.3.2.8 invert()

```
int jeod::Matrix3x3::invert (
    const double matrix[3][3],
    double inverse[3][3] ) [static]
```

Compute the inverse of a 3x3 matrix.

Assumptions and Limitations

- Input and output matrices are distinct.
- Input matrix is well-conditioned.

Returns

0=success, non-zero=singular

Parameters

in	<i>matrix</i>	Matrix to invert
out	<i>inverse</i>	Inverse

Definition at line 72 of file dm_invert.cc.

References jeod::MathMessages::ill_conditioned.

7.3.2.9 invert_symmetric()

```
int jeod::Matrix3x3::invert_symmetric (
    const double matrix[3][3],
    double inverse[3][3] ) [static]
```

Compute the inverse of a symmetric 3x3 matrix.

Assumptions and Limitations

- Input and output matrices are distinct.
- Input matrix is symmetric.
- Determinate is non-zero.

Returns

0=success, non-zero=singular

Parameters

in	<i>matrix</i>	Symmetric matrix to invert
out	<i>inverse</i>	Inverse

Definition at line 73 of file dm_invert_symm.cc.

References jeod::MathMessages::ill_conditioned.

7.3.2.10 negate() [1/2]

```
void jeod::Matrix3x3::negate (
    double mat[3][3] ) [inline], [static]
```

Negated matrix in-place: $\text{mat}[i][j] = -\text{mat}[i][j]$.

Parameters

in, out	<i>mat</i>	Negated matrix
---------	------------	----------------

Definition at line 142 of file matrix3x3_inline.hh.

7.3.2.11 negate() [2/2]

```
void jeod::Matrix3x3::negate (
    const double input_mat[3][3],
    double copy[3][3] ) [inline], [static]
```

Negate matrix: $\text{copy}[i][j] = -\text{mat}[i][j]$.

Assumptions and Limitations

- Input and output matrices are distinct.

Parameters

in	<i>input_mat</i>	Source matrix
out	<i>copy</i>	Negated matrix

Definition at line 260 of file matrix3x3_inline.hh.

References `copy()`.

7.3.2.12 outer_product()

```
void jeod::Matrix3x3::outer_product (
    const double vec_left[3],
    const double vec_right[3],
    double prod[3][3] ) [inline], [static]
```

Construct the outer product of two vectors: $\text{mat}[i][j] = \text{vec_left}[i] * \text{vec_right}[j]$.

Parameters

in	<i>vec_left</i>	Vector
in	<i>vec_right</i>	Vector
out	<i>prod</i>	Outer product matrix

Definition at line 125 of file matrix3x3_inline.hh.

7.3.2.13 print()

```
void jeod::Matrix3x3::print (
    const double mat[3][3] ) [inline], [static]
```

Print matrix to standard error.

Parameters

in	<i>mat</i>	Matrix to print
----	------------	-----------------

Definition at line 540 of file matrix3x3_inline.hh.

7.3.2.14 product()

```
void jeod::Matrix3x3::product (
    const double mat_left[3][3],
    const double mat_right[3][3],
    double prod[3][3] ) [inline], [static]
```

Compute the matrix product $\text{mat_left} * \text{mat_right}$: $\text{prod}[i][j] = \text{mat_left}[i][k] * \text{mat_right}[k][j]$.

Assumptions and Limitations

- Input and output matrices are distinct.

Parameters

in	<i>mat_left</i>	Multiplier
in	<i>mat_right</i>	Multiplicand
out	<i>prod</i>	Product

Definition at line 368 of file matrix3x3_inline.hh.

Referenced by transform_matrix(), and transpose_transform_matrix().

7.3.2.15 product_left_transpose()

```
void jeod::Matrix3x3::product_left_transpose (
    const double mat_left[3][3],
```



```
const double mat_right[3][3],
double prod[3][3] ) [inline], [static]
```

Compute the matrix product $\text{mat_left}^T * \text{mat_right}$: $\text{prod}[i][j] = \text{mat_left}[k][i] * \text{mat_right}[k][j]$.

Assumptions and Limitations

- Input and output matrices are distinct.

Parameters

in	<i>mat_left</i>	Multiplier
in	<i>mat_right</i>	Multiplicand
out	<i>prod</i>	Product

Definition at line 399 of file matrix3x3_inline.hh.

Referenced by `transpose_transform_matrix()`.

7.3.2.16 product_right_transpose()

```
void jeod::Matrix3x3::product_right_transpose (
    const double mat_left[3][3],
    const double mat_right[3][3],
    double prod[3][3] ) [inline], [static]
```

Compute the matrix product $\text{mat_left} * \text{mat_right}^T$: $\text{prod}[i][j] = \sum_k \text{mat_left}[i][k] * \text{mat_right}[j][k]$.

Assumptions and Limitations

- Input and output matrices are distinct.

Parameters

in	<i>mat_left</i>	Multiplier
in	<i>mat_right</i>	Multiplicand
out	<i>prod</i>	Product

Definition at line 432 of file matrix3x3_inline.hh.

Referenced by `transform_matrix()`.

7.3.2.17 product_transpose_transpose()

```
void jeod::Matrix3x3::product_transpose_transpose (
    const double mat_left[3][3],
```

```
const double mat_right[3][3],
double prod[3][3] ) [inline], [static]
```

Compute the matrix product $\text{mat_left}^T * \text{mat_right}^T$: $\text{prod}[i][j] = \sum_k \text{mat_left}[k][i] * \text{mat_right}[j][k]$.

Assumptions and Limitations

- Input and output matrices are distinct.

Parameters

in	<i>mat_left</i>	Multiplier
in	<i>mat_right</i>	Multiplicand
out	<i>prod</i>	Product

Definition at line 465 of file matrix3x3_inline.hh.

7.3.2.18 scale() [1/2]

```
void jeod::Matrix3x3::scale (
    double scalar,
    double mat[3][3] ) [inline], [static]
```

Scale matrix in-place, $\text{mat}[i][j] = \text{scalar} * \text{mat}[i][j]$.

Parameters

in	<i>scalar</i>	Scalar
in, out	<i>mat</i>	Scaled matrix

Definition at line 183 of file matrix3x3_inline.hh.

7.3.2.19 scale() [2/2]

```
void jeod::Matrix3x3::scale (
    const double mat[3][3],
    double scalar,
    double prod[3][3] ) [inline], [static]
```

Scale matrix: $\text{copy}[i][j] = \text{scalar} * \text{mat}[i][j]$.

Parameters

in	<i>mat</i>	Matrix
in	<i>scalar</i>	Scalar
out	<i>prod</i>	Product

Definition at line 307 of file matrix3x3_inline.hh.

7.3.2.20 subtract()

```
void jeod::Matrix3x3::subtract (
    const double minuend[3][3],
    const double subtrahend[3][3],
    double diff[3][3] ) [inline], [static]
```

Subtract matrices: $\text{diff}[i][j] = \text{minuend}[i][j] - \text{subtrahend}[i][j]$.

Parameters

in	<i>minuend</i>	Matrix
in	<i>subtrahend</i>	Matrix
out	<i>diff</i>	Difference

Definition at line 345 of file matrix3x3_inline.hh.

7.3.2.21 transform_matrix()

```
void jeod::Matrix3x3::transform_matrix (
    const double trans[3][3],
    const double mat[3][3],
    double prod[3][3] ) [inline], [static]
```

Compute the matrix product $\text{trans} * \text{mat} * \text{trans}^T$ $\text{prod}[i][j] = \text{trans}[i][k] * \text{mat}[k][l] * \text{trans}[j][l]$.

Assumptions and Limitations

- Input and output matrices are distinct.

Parameters

in	<i>trans</i>	Transformation matrix
in	<i>mat</i>	Matrix to transform
out	<i>prod</i>	Product

Definition at line 510 of file matrix3x3_inline.hh.

References [product\(\)](#), and [product_right_transpose\(\)](#).

7.3.2.22 transpose() [1/2]

```
void jeod::Matrix3x3::transpose (
    double mat[3][3] ) [inline], [static]
```

Transpose matrix in-place: $\text{mat}[i][j] = \text{mat}[j][i]$.

Parameters

in, out	<i>mat</i>	Transposed matrix
---------	------------	-------------------

Definition at line 161 of file matrix3x3_inline.hh.

7.3.2.23 transpose() [2/2]

```
void jeod::Matrix3x3::transpose (
    const double input_mat[3][3],
    double trans[3][3] ) [inline], [static]
```

Transpose matrix: $\text{copy}[i][j] = \text{mat}[j][i]$.

Assumptions and Limitations

- Input and output matrices are distinct.

Parameters

in	<i>input_mat</i>	Source matrix
out	<i>trans</i>	Matrix transpose

Definition at line 283 of file matrix3x3_inline.hh.

7.3.2.24 transpose_transform_matrix()

```
void jeod::Matrix3x3::transpose_transform_matrix (
    const double trans[3][3],
    const double mat[3][3],
    double prod[3][3] ) [inline], [static]
```

Compute the matrix product $\text{trans}^T * \text{mat} * \text{trans}$ $\text{prod}[i][j] = \text{trans}[k][i] * \text{mat}[k][l] * \text{trans}[l][j]$.

Assumptions and Limitations

- Input and output matrices are distinct.

Parameters

in	<i>trans</i>	Transformation matrix
in	<i>mat</i>	Matrix to transform
out	<i>prod</i>	Product

Definition at line 528 of file matrix3x3_inline.hh.

References `product()`, and `product_left_transpose()`.

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

- [matrix3x3.hh](#)
- [matrix3x3_inline.hh](#)
- [dm_invert.cc](#)
- [dm_invert_symm.cc](#)

7.4 jeod::Numerical Class Reference

Provides miscellaneous numerical functions.

```
#include <numerical.hh>
```

Static Public Member Functions

- static double [fabs](#) (double x)
Absolute value.
- static double [square](#) (double value)
Compute the square of a number, protecting against underflow.
- static double [square_incr](#) (double value, double &sum)
Add number squared to accumulator, protecting against underflow.
- static bool [compare_exact](#) (double x, double y)
Compare two doubles for exact equality.

7.4.1 Detailed Description

Provides miscellaneous numerical functions.

Definition at line 66 of file numerical.hh.

7.4.2 Member Function Documentation

7.4.2.1 `compare_exact()`

```
bool jeod::Numerical::compare_exact (
    double x,
    double y ) [inline], [static]
```

Compare two doubles for exact equality.

Returns

whether inputs are exactly the same

Parameters

in	<i>x</i>	Value1
in	<i>y</i>	Value2

Definition at line 123 of file numerical_inline.hh.

7.4.2.2 fabs()

```
double jeod::Numerical::fabs (
    double x ) [inline], [static]
```

Absolute value.

Returns

Absolute value of *x*

Parameters

in	<i>x</i>	<i>x</i>
----	----------	----------

Definition at line 73 of file numerical_inline.hh.

Referenced by square(), square_incr(), and jeod::Vector3::zero_small().

7.4.2.3 square()

```
double jeod::Numerical::square (
    double value ) [inline], [static]
```

Compute the square of a number, protecting against underflow.

Returns

value^2 or zero if too small

Parameters

in	<i>value</i>	Value
----	--------------	-------

Definition at line 90 of file numerical_inline.hh.

References fabs(), and jeod::SQRT_DBL_MIN.

7.4.2.4 square_incr()

```
double jeod::Numerical::square_incr (
    double value,
    double & sum ) [inline], [static]
```

Add number squared to accumulator, protecting against underflow.

Returns

Accumulated value

Parameters

in	value	Value
in, out	sum	Accumulator

Definition at line 108 of file numerical_inline.hh.

References fabs(), and jeod::SQRT_DBL_MIN.

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

- [numerical.hh](#)
- [numerical_inline.hh](#)

7.5 jeod::Vector3 Class Reference

Provides static methods for operations that involve 3-vectors.

```
#include <vector3.hh>
```

Static Public Member Functions

- static double * [initialize](#) (double vec[3])
Zero-fill vector, vec[i] = 0.0.
- static double * [unit](#) (unsigned int index, double vec[3])
Construct unit vector, vec[i] = delta_ij (delta_ij is the Kronecker delta)
- static double * [fill](#) (double scalar, double vec[3])
Construct a vector from scalar, vec[i] = scalar.
- static double * [zero_small](#) (double limit, double vec[3])
Zero-out small components of a vector, vec[i] = 0 if abs(vec[i]) < limit.
- static double * [copy](#) (const double vec[3], double copy[3])
Copy vector contents, copy[i] = vec[i].
- static double [dot](#) (const double vec2[3], const double vec1[3])
*Compute vector inner product, result = sum_i vec1[i] * vec2[i].*
- static double [vmagsq](#) (const double vec[3])
Compute square of vector magnitude, result = dot(vec,vec), but protects against underflow.
- static double [vmag](#) (const double vec[3])

- Compute vector magnitude, result = sqrt(vmagsq(vec))*

 - static double * **normalize** (double vec[3])

*Make vector a unit vector in-place, vec = vec * 1/vmag(vec)*

 - static double * **normalize** (const double vec[3], double unit_vec[3])

*Construct unit vector, unit_vec = vec * 1/vmag(vec)*

 - static double * **scale** (double scalar, double vec[3])

Scale a vector in-place, vec[i] = scalar.

 - static double * **scale** (const double vec[3], double scalar, double prod[3])

*Scale a vector, prod[i] = vec[i] * scalar.*

 - static double * **negate** (double vec[3])

Negate vector in-place, vec[i] = -vec[i].

 - static double * **negate** (const double vec[3], double copy[3])

Negate vector, copy[i] = -vec[i].

 - static double * **transform** (const double tmat[3][3], const double vec[3], double prod[3])

*Transform a column vector, prod[i] = tmat[i][j]*vec[j].*

 - static double * **transform** (const double tmat[3][3], double vec[3])

*Transform a column vector in-place, vec[i] <- tmat[i][j]*vec[j].*

 - static double * **transform_transpose** (const double tmat[3][3], const double vec[3], double prod[3])

*Transform a column vector with the transpose, prod[i] = tmat[j][i]*vec[j].*

 - static double * **transform_transpose** (const double tmat[3][3], double vec[3])

*Transform a column vector in-place with the transpose, vec[i] <- tmat[j][i]*vec[j].*

 - static double * **incr** (const double addend[3], double vec[3])

Increment a vector, vec[i] += addend[i].

 - static double * **incr** (const double addend1[3], const double addend2[3], double vec[3])

Increment a vector, vec[i] += addend1[i] + addend2[i].

 - static double * **decr** (const double subtrahend[3], double vec[3])

Decrement a vector, vec[i] -= subtrahend[i].

 - static double * **decr** (const double subtrahend1[3], const double subtrahend2[3], double vec[3])

Decrement a vector, vec[i] -= subtrahend1[i] + subtrahend2[i].

 - static double * **sum** (const double addend1[3], const double addend2[3], double vec[3])

Compute the sum of two vectors, vec[i] = addend1[i] + addend2[i].

 - static double * **sum** (const double addend1[3], const double addend2[3], const double addend3[3], double vec[3])

Compute the sum of three vectors, vec[i] = addend1[i] + addend2[i] + addend3[i].

 - static double * **diff** (const double minuend[3], const double subtrahend[3], double vec[3])

Compute the difference between two vectors, diff[i] = minuend[i] - subtrahend[i].

 - static double * **cross** (const double vec_left[3], const double vec_right[3], double prod[3])

*Compute the cross product between two vectors, prod[i] = epsilon_ijk * vec_left[j] * vec_right[k].*

 - static double * **scale_incr** (const double vec[3], double scalar, double prod[3])

*Increment a vector with a scaled vector, prod[i] += scalar*vec[i].*

 - static double * **scale_decr** (const double vec[3], double scalar, double prod[3])

*Decrement a vector with a scaled vector, prod[i] += scalar*vec[i].*

 - static double * **cross_incr** (const double vec_left[3], const double vec_right[3], double prod[3])

*Increment a vector with the the cross product between two vectors, prod[i] += epsilon_ijk * vec_left[j] * vec_right[k].*

 - static double * **cross_decr** (const double vec_left[3], const double vec_right[3], double prod[3])

*Decrement a vector with the the cross product between two vectors, prod[i] -= epsilon_ijk * vec_left[j] * vec_right[k].*

 - static double * **transform_incr** (const double tmat[3][3], const double vec[3], double prod[3])

*Increment a vector with a transformed column vector, prod[i] += tmat[i][j]*vec[j].*

 - static double * **transform_decr** (const double tmat[3][3], const double vec[3], double prod[3])

*Decrement a vector with a transformed column vector, prod[i] += tmat[i][j]*vec[j].*

 - static double * **transform_transpose_incr** (const double tmat[3][3], const double vec[3], double prod[3])

*Increment a vector with a transpose-transformed column vector, prod[i] += tmat[j][i]*vec[j].*

 - static double * **transform_transpose_decr** (const double tmat[3][3], const double vec[3], double prod[3])

*decrement a vector with a transpose-transformed column vector, prod[i] -= tmat[j][i]*vec[j]*

7.5.1 Detailed Description

Provides static methods for operations that involve 3-vectors.

Definition at line 66 of file vector3.hh.

7.5.2 Member Function Documentation

7.5.2.1 copy()

```
double * jeod::Vector3::copy (
    const double vec[3],
    double copy[3] ) [inline], [static]
```

Copy vector contents, $\text{copy}[i] = \text{vec}[i]$.

Returns

Copied vector

Parameters

in	<i>vec</i>	Source vector
out	<i>copy</i>	Copied vector

Definition at line 143 of file vector3_inline.hh.

Referenced by `negate()`, `normalize()`, `transform()`, and `transform_transpose()`.

7.5.2.2 cross()

```
double * jeod::Vector3::cross (
    const double vec_left[3],
    const double vec_right[3],
    double prod[3] ) [inline], [static]
```

Compute the cross product between two vectors, $\text{prod}[i] = \epsilon_{ijk} * \text{vec_left}[j] * \text{vec_right}[k]$.

Returns

Cross product vector

Parameters

in	<i>vec_left</i>	Left vector
in	<i>vec_right</i>	Right vector
out	<i>prod</i>	Cross product vector

Definition at line 484 of file vector3_inline.hh.

7.5.2.3 cross_decr()

```
double * jeod::Vector3::cross_decr (
    const double vec_left[3],
    const double vec_right[3],
    double prod[3] ) [inline], [static]
```

Decrement a vector with the the cross product between two vectors, $prod[i] -= \epsilon_{ijk} * vec_left[j] * vec_right[k]$.

Returns

Decrementated vector

Parameters

in	<i>vec_left</i>	Left vector
in	<i>vec_right</i>	Right vector
in, out	<i>prod</i>	Decrementated vector

Definition at line 552 of file vector3_inline.hh.

7.5.2.4 cross_incr()

```
double * jeod::Vector3::cross_incr (
    const double vec_left[3],
    const double vec_right[3],
    double prod[3] ) [inline], [static]
```

Increment a vector with the the cross product between two vectors, $prod[i] += \epsilon_{ijk} * vec_left[j] * vec_right[k]$.

Returns

Cross product vector

Parameters

in	<i>vec_left</i>	Left vector
in	<i>vec_right</i>	Right vector
in, out	<i>prod</i>	Cross product vector

Definition at line 535 of file vector3_inline.hh.

7.5.2.5 `decr()` [1/2]

```
double * jeod::Vector3::decr (
    const double subtrahend[3],
    double vec[3] ) [inline], [static]
```

Decrement a vector, $\text{vec}[i] -= \text{subtrahend}[i]$.

Returns

Decrementated vector

Parameters

<i>in</i>	<i>subtrahend</i>	Decrement
<i>in, out</i>	<i>vec</i>	Vector

Definition at line 398 of file vector3_inline.hh.

7.5.2.6 `decr()` [2/2]

```
double * jeod::Vector3::decr (
    const double subtrahend1[3],
    const double subtrahend2[3],
    double vec[3] ) [inline], [static]
```

Decrement a vector, $\text{vec}[i] -= \text{subtrahend1}[i] + \text{subtrahend2}[i]$.

Returns

Decrementated vector

Parameters

<i>in</i>	<i>subtrahend1</i>	Decrement
<i>in</i>	<i>subtrahend2</i>	Decrement
<i>in, out</i>	<i>vec</i>	Vector

Definition at line 415 of file vector3_inline.hh.

7.5.2.7 diff()

```
double * jeod::Vector3::diff (
    const double minuend[3],
    const double subtrahend[3],
    double vec[3] ) [inline], [static]
```

Compute the difference between two vectors, $\text{diff}[i] = \text{minuend}[i] - \text{subtrahend}[i]$.

Returns

Difference vector

Parameters

in	<i>minuend</i>	Minuend
in	<i>subtrahend</i>	Subtrahend
out	<i>vec</i>	Difference vector

Definition at line 467 of file vector3_inline.hh.

7.5.2.8 dot()

```
double jeod::Vector3::dot (
    const double vec2[3],
    const double vec1[3] ) [inline], [static]
```

Compute vector inner product, $\text{result} = \sum_i \text{vec1}[i] * \text{vec2}[i]$.

Returns

Inner product

Parameters

in	<i>vec2</i>	Vector 2
in	<i>vec1</i>	Vector 1

Definition at line 159 of file vector3_inline.hh.

7.5.2.9 fill()

```
double * jeod::Vector3::fill (
    double scalar,
    double vec[3] ) [inline], [static]
```

Construct a vector from scalar, $\text{vec}[i] = \text{scalar}$.

Returns

Filled vector

Parameters

in	<i>scalar</i>	Scalar
out	<i>vec</i>	Filled vector

Definition at line 103 of file vector3_inline.hh.

7.5.2.10 incr() [1/2]

```
double * jeod::Vector3::incr (
    const double addend[3],
    double vec[3] ) [inline], [static]
```

Increment a vector, `vec[i] += addend[i]`.

Returns

Incremented vector

Parameters

in	<i>addend</i>	Increment
in, out	<i>vec</i>	Vector

Definition at line 365 of file vector3_inline.hh.

7.5.2.11 incr() [2/2]

```
double * jeod::Vector3::incr (
    const double addend1[3],
    const double addend2[3],
    double vec[3] ) [inline], [static]
```

Increment a vector, `vec[i] += addend1[i] + addend2[i]`.

Returns

Incremented vector

Parameters

in	<i>addend1</i>	Increment
in	<i>addend2</i>	Increment
in, out	<i>vec</i>	Vector

Definition at line 382 of file vector3_inline.hh.

7.5.2.12 initialize()

```
double * jeod::Vector3::initialize (
    double vec[3] ) [inline], [static]
```

Zero-fill vector, $\text{vec}[i] = 0.0$.

Returns

Zero-filled vector

Parameters

out	vec	Zero-filled vector
-----	-----	--------------------

Definition at line 76 of file vector3_inline.hh.

Referenced by normalize().

7.5.2.13 negate() [1/2]

```
double * jeod::Vector3::negate (
    double vec[3] ) [inline], [static]
```

Negate vector in-place, $\text{vec}[i] = -\text{vec}[i]$.

Returns

Negated vector

Parameters

in, out	vec	Vector
---------	-----	--------

Definition at line 261 of file vector3_inline.hh.

7.5.2.14 negate() [2/2]

```
double * jeod::Vector3::negate (
    const double vec[3],
    double copy[3] ) [inline], [static]
```

Negate vector, $\text{copy}[i] = -\text{vec}[i]$.

Returns

Negated vector

Parameters

in	<i>vec</i>	Source vector
out	<i>copy</i>	Negated vector

Definition at line 277 of file vector3_inline.hh.

References `copy()`.

7.5.2.15 normalize() [1/2]

```
double * jeod::Vector3::normalize (
    double vec[3] ) [inline], [static]
```

Make vector a unit vector in-place, $\text{vec} = \text{vec} * 1/\text{vmag}(\text{vec})$

Returns

Normalized vector

Parameters

in, out	<i>vec</i>	Vector
---------	------------	--------

Definition at line 192 of file vector3_inline.hh.

References `initialize()`, `scale()`, and `vmag()`.

Referenced by `normalize()`.

7.5.2.16 normalize() [2/2]

```
double * jeod::Vector3::normalize (
    const double vec[3],
    double unit_vec[3] ) [inline], [static]
```

Construct unit vector, $\text{unit_vec} = \text{vec} * 1/\text{vmag}(\text{vec})$

Returns

Unit vector

Parameters

in	<i>vec</i>	Vector
out	<i>unit_vec</i>	Unit vector

Definition at line 215 of file vector3_inline.hh.

References `copy()`, and `normalize()`.

7.5.2.17 scale() [1/2]

```
double * jeod::Vector3::scale (
    double scalar,
    double vec[3] ) [inline], [static]
```

Scale a vector in-place, `vec[i] = scalar`.

Returns

Scaled vector

Parameters

in	<i>scalar</i>	Scalar
in, out	<i>vec</i>	Scaled vector

Definition at line 229 of file vector3_inline.hh.

Referenced by `normalize()`.

7.5.2.18 scale() [2/2]

```
double * jeod::Vector3::scale (
    const double vec[3],
    double scalar,
    double prod[3] ) [inline], [static]
```

Scale a vector, `prod[i] = vec[i] * scalar`.

Returns

Scaled vector

Parameters

in	<i>vec</i>	Source vector
in	<i>scalar</i>	Scalar
out	<i>prod</i>	Scaled vector

Definition at line 246 of file vector3_inline.hh.

7.5.2.19 scale_decr()

```
double * jeod::Vector3::scale_decr (
    const double vec[3],
    double scalar,
    double prod[3] ) [inline], [static]
```

Decrement a vector with a scaled vector, $\text{prod}[i] += \text{scalar} * \text{vec}[i]$.

Returns

Decrementated vector

Parameters

in	<i>vec</i>	Source vector
in	<i>scalar</i>	Scalar
in, out	<i>prod</i>	Decrementated vector

Definition at line 518 of file vector3_inline.hh.

7.5.2.20 scale_incr()

```
double * jeod::Vector3::scale_incr (
    const double vec[3],
    double scalar,
    double prod[3] ) [inline], [static]
```

Increment a vector with a scaled vector, $\text{prod}[i] += \text{scalar} * \text{vec}[i]$.

Returns

Incremented vector

Parameters

in	<i>vec</i>	Source vector
in	<i>scalar</i>	Scalar
in, out	<i>prod</i>	Incremented vector

Definition at line 501 of file vector3_inline.hh.

7.5.2.21 `sum()` [1/2]

```
double * jeod::Vector3::sum (
    const double addend1[3],
    const double addend2[3],
    double vec[3] ) [inline], [static]
```

Compute the sum of two vectors, $vec[i] = addend1[i] + addend2[i]$.

Returns

Sum vector

Parameters

in	<i>addend1</i>	Addend
in	<i>addend2</i>	Addend
out	<i>vec</i>	Sum vector

Definition at line 432 of file `vector3_inline.hh`.

7.5.2.22 `sum()` [2/2]

```
double * jeod::Vector3::sum (
    const double addend1[3],
    const double addend2[3],
    const double addend3[3],
    double vec[3] ) [inline], [static]
```

Compute the sum of three vectors, $vec[i] = addend1[i] + addend2[i] + addend3[i]$.

Returns

Sum vector

Parameters

in	<i>addend1</i>	Addend
in	<i>addend2</i>	Addend
in	<i>addend3</i>	Addend
out	<i>vec</i>	Sum vector

Definition at line 450 of file `vector3_inline.hh`.

7.5.2.23 `transform()` [1/2]

```
double * jeod::Vector3::transform (
    const double tmat[3][3],
```

```
const double vec[3],
double prod[3] ) [inline], [static]
```

Transform a column vector, $\text{prod}[i] = \text{tmat}[i][j] * \text{vec}[j]$.

Returns

Transformed vector

Parameters

in	<i>tmat</i>	Transformation matrix
in	<i>vec</i>	Source vector
out	<i>prod</i>	Transformed vector

Definition at line 294 of file vector3_inline.hh.

Referenced by transform().

7.5.2.24 transform() [2/2]

```
double * jeod::Vector3::transform (
    const double tmat[3][3],
    double vec[3] ) [inline], [static]
```

Transform a column vector in-place, $\text{vec}[i] <- \text{tmat}[i][j] * \text{vec}[j]$.

Returns

Transformed vector

Parameters

in	<i>tmat</i>	Transformation matrix
in, out	<i>vec</i>	Transformed vector

Definition at line 312 of file vector3_inline.hh.

References copy(), and transform().

7.5.2.25 transform_decr()

```
double * jeod::Vector3::transform_decr (
    const double tmat[3][3],
    const double vec[3],
    double prod[3] ) [inline], [static]
```

Decrement a vector with a transformed column vector, $\text{prod}[i] += \text{tmat}[i][j] * \text{vec}[j]$.

Returns

Decrementated vector

Parameters

<i>in</i>	<i>tmat</i>	Transformation matrix
<i>in</i>	<i>vec</i>	Source vector
<i>in, out</i>	<i>prod</i>	Decrementated vector

Definition at line 588 of file vector3_inline.hh.

7.5.2.26 transform_incr()

```
double * jeod::Vector3::transform_incr (
    const double tmat[3][3],
    const double vec[3],
    double prod[3] ) [inline], [static]
```

Increment a vector with a transformed column vector, $prod[i] += tmat[i][j]*vec[j]$.

Returns

Incremented vector

Parameters

<i>in</i>	<i>tmat</i>	Transformation matrix
<i>in</i>	<i>vec</i>	Source vector
<i>in, out</i>	<i>prod</i>	Incremented vector

Definition at line 569 of file vector3_inline.hh.

7.5.2.27 transform_transpose() [1/2]

```
double * jeod::Vector3::transform_transpose (
    const double tmat[3][3],
    const double vec[3],
    double prod[3] ) [inline], [static]
```

Transform a column vector with the transpose, $prod[i] = tmat[j][i]*vec[j]$.

Returns

Transformed vector

Parameters

in	<i>tmat</i>	Transformation matrix
in	<i>vec</i>	Source vector
out	<i>prod</i>	Transformed vector

Definition at line 330 of file vector3_inline.hh.

Referenced by transform_transpose().

7.5.2.28 transform_transpose() [2/2]

```
double * jeod::Vector3::transform_transpose (
    const double tmat[3][3],
    double vec[3] ) [inline], [static]
```

Transform a column vector in-place with the transpose, $vec[i] \leftarrow tmat[j][i] * vec[j]$.

Returns

Transformed vector

Parameters

in	<i>tmat</i>	Transformation matrix
in, out	<i>vec</i>	Transformed vector

Definition at line 348 of file vector3_inline.hh.

References copy(), and transform_transpose().

7.5.2.29 transform_transpose_decr()

```
double * jeod::Vector3::transform_transpose_decr (
    const double tmat[3][3],
    const double vec[3],
    double prod[3] ) [inline], [static]
```

decrement a vector with a transpose-transformed column vector, $prod[i] -= tmat[j][i] * vec[j]$

Returns

Decrement vector

Parameters

in	<i>tmat</i>	Transformation matrix
in	<i>vec</i>	Source vector
in, out	<i>prod</i>	Decrementated vector

Definition at line 626 of file vector3_inline.hh.

7.5.2.30 transform_transpose_incr()

```
double * jeod::Vector3::transform_transpose_incr (
    const double tmat[3][3],
    const double vec[3],
    double prod[3] ) [inline], [static]
```

Increment a vector with a transpose-transformed column vector, $prod[i] += tmat[j][i]*vec[j]$.

Returns

Incremented vector

Parameters

in	<i>tmat</i>	Transformation matrix
in	<i>vec</i>	Source vector
in, out	<i>prod</i>	Incremented vector

Definition at line 607 of file vector3_inline.hh.

7.5.2.31 unit()

```
double * jeod::Vector3::unit (
    unsigned int index,
    double vec[3] ) [inline], [static]
```

Construct unit vector, $vec[i] = \delta_{ij}$ (δ_{ij} is the Kronecker delta)

Returns

Unit vector

Parameters

in	<i>index</i>	Unit index: 0,1,2= $\hat{x}, \hat{y}, \hat{z}$
out	<i>vec</i>	Unit vector

Definition at line 89 of file vector3_inline.hh.

7.5.2.32 vmag()

```
double jeod::Vector3::vmag (
    const double vec[3] ) [inline], [static]
```

Compute vector magnitude, result = sqrt(vmagsq(vec))

Returns

Vector magnitude

Parameters

in	vec	Vector
----	-----	--------

Definition at line 181 of file vector3_inline.hh.

References vmagsq().

Referenced by normalize().

7.5.2.33 vmagsq()

```
double jeod::Vector3::vmagsq (
    const double vec[3] ) [inline], [static]
```

Compute square of vector magnitude, result = dot(vec,vec), but protects against underflow.

Returns

Inner product

Parameters

in	vec	Vector
----	-----	--------

Definition at line 170 of file vector3_inline.hh.

Referenced by vmag().

7.5.2.34 zero_small()

```
double * jeod::Vector3::zero_small (
    double limit,
    double vec[3] ) [inline], [static]
```

Zero-out small components of a vector, $\text{vec}[i] = 0$ if $\text{abs}(\text{vec}[i]) < \text{limit}$.

Returns

Truncated vector

Parameters

<i>in</i>	<i>limit</i>	Limit
<i>in, out</i>	<i>vec</i>	Truncated vector

Definition at line 116 of file vector3_inline.hh.

References `jeod::Numerical::fabs()`.

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

- [vector3.hh](#)
- [vector3_inline.hh](#)

Chapter 8

File Documentation

8.1 dm_invert.cc File Reference

Define Matrix3x3::invert.

```
#include <cmath>
#include "utils/message/include/message_handler.hh"
#include "../include/math_messages.hh"
#include "../include/matrix3x3.hh"
```

Namespaces

- [jeod](#)
Namespace jeod.

8.1.1 Detailed Description

Define Matrix3x3::invert.

8.2 dm_invert_symm.cc File Reference

Define Matrix3x3::invert_symmetric.

```
#include <cmath>
#include "utils/message/include/message_handler.hh"
#include "../include/math_messages.hh"
#include "../include/matrix3x3.hh"
```

Namespaces

- [jeod](#)
Namespace jeod.

8.2.1 Detailed Description

Define Matrix3x3::invert_symmetric.

8.3 gauss_quadrature.cc File Reference

Define Gauss Quadrature functionality.

```
#include "../include/gauss_quadrature.hh"
```

Namespaces

- [jeod](#)
Namespace jeod.

8.3.1 Detailed Description

Define Gauss Quadrature functionality.

8.4 gauss_quadrature.hh File Reference

Gauss Quadrature implementation.

Data Structures

- class [jeod::GaussQuadrature](#)

Namespaces

- [jeod](#)
Namespace jeod.

8.4.1 Detailed Description

Gauss Quadrature implementation.

8.5 macro_def.hh File Reference

Math macro definitions.

Macros

- `#define IDENTITY_3X3` {1.0, 0.0, 0.0}, {0.0, 1.0, 0.0}, {0.0, 0.0, 1.0}

8.5.1 Detailed Description

Math macro definitions.

8.6 macro_undef.hh File Reference

Math macro undefinitions.

8.6.1 Detailed Description

Math macro undefinitions.

8.7 math_messages.cc File Reference

Implement the class MathMessages.

```
#include "utils/message/include/make_message_code.hh"
#include "../include/math_messages.hh"
```

Namespaces

- `jeod`
Namespace jeod.

Macros

- `#define MAKE_MATH_MESSAGE_CODE(id)` JEOD_MAKE_MESSAGE_CODE(MathMessages, "utils/math/", id)

8.7.1 Detailed Description

Implement the class MathMessages.

8.7.2 Macro Definition Documentation

8.7.2.1 MAKE_MATH_MESSAGE_CODE

```
#define MAKE_MATH_MESSAGE_CODE(  
    id ) JEOD_MAKE_MESSAGE_CODE(MathMessages, "utils/math/", id)
```

Definition at line 38 of file math_messages.cc.

8.8 math_messages.hh File Reference

Define the class MathMessages.

Data Structures

- class [jeod::MathMessages](#)
Specifies the message IDs used in the math model.

Namespaces

- [jeod](#)
Namespace jeod.

8.8.1 Detailed Description

Define the class MathMessages.

8.9 matrix3x3.hh File Reference

Matrix math inline functions.

```
#include "matrix3x3_inline.hh"
```

Data Structures

- class [jeod::Matrix3x3](#)
Provides static methods for operations that involve 3x3 matrices.

Namespaces

- [jeod](#)
Namespace jeod.

8.9.1 Detailed Description

Matrix math inline functions.

8.10 matrix3x3_inline.hh File Reference

Matrix math inline functions.

```
#include <cstdio>
#include "matrix3x3.hh"
```

Namespaces

- [jeod](#)

Namespace jeod.

8.10.1 Detailed Description

Matrix math inline functions.

8.11 numerical.hh File Reference

Miscellaneous math inline functions.

```
#include "numerical_inline.hh"
```

Data Structures

- class [jeod::Numerical](#)

Provides miscellaneous numerical functions.

Namespaces

- [jeod](#)

Namespace jeod.

8.11.1 Detailed Description

Miscellaneous math inline functions.

8.12 numerical_inline.hh File Reference

Vector math inline functions.

```
#include "numerical.hh"
```

Namespaces

- [jeod](#)
Namespace jeod.

Variables

- static constexpr double [jeod::SQRT_DBL_MIN](#) {1.4916681462400413e-154}

8.12.1 Detailed Description

Vector math inline functions.

8.13 vector3.hh File Reference

Vector math inline functions.

```
#include "vector3_inline.hh"
```

Data Structures

- class [jeod::Vector3](#)
Provides static methods for operations that involve 3-vectors.

Namespaces

- [jeod](#)
Namespace jeod.

8.13.1 Detailed Description

Vector math inline functions.

8.14 vector3_inline.hh File Reference

Vector math inline functions.

```
#include <cmath>
#include "numerical.hh"
#include "vector3.hh"
```

Namespaces

- [jeod](#)

Namespace jeod.

8.14.1 Detailed Description

Vector math inline functions.

Index

- add
 - jeod::Matrix3x3, [19](#)
- compare_exact
 - jeod::Numerical, [29](#)
- copy
 - jeod::Matrix3x3, [19](#)
 - jeod::Vector3, [33](#)
- cross
 - jeod::Vector3, [33](#)
- cross_decr
 - jeod::Vector3, [34](#)
- cross_incr
 - jeod::Vector3, [34](#)
- cross_matrix
 - jeod::Matrix3x3, [20](#)
- decr
 - jeod::Matrix3x3, [20](#)
 - jeod::Vector3, [35](#)
- diff
 - jeod::Vector3, [35](#)
- dm_invert.cc, [49](#)
- dm_invert_symm.cc, [49](#)
- dot
 - jeod::Vector3, [36](#)
- fabs
 - jeod::Numerical, [30](#)
- fill
 - jeod::Vector3, [36](#)
- gauss_quadrature.cc, [50](#)
- gauss_quadrature.hh, [50](#)
- gauss_weights
 - jeod::GaussQuadrature, [15](#)
- gauss_xvalues
 - jeod::GaussQuadrature, [15](#)
- IDENTITY_3X3
 - Math, [11](#)
- identity
 - jeod::Matrix3x3, [20](#)
- ill_conditioned
 - jeod::MathMessages, [17](#)
- incr
 - jeod::Matrix3x3, [21](#)
 - jeod::Vector3, [37](#)
- initialize
 - jeod::Matrix3x3, [21](#)
 - jeod::Vector3, [38](#)
- invert
 - jeod::Matrix3x3, [21](#)
- invert_symmetric
 - jeod::Matrix3x3, [22](#)
- jeod, [13](#)
 - SQRT_DBL_MIN, [13](#)
- jeod::GaussQuadrature, [15](#)
 - gauss_weights, [15](#)
 - gauss_xvalues, [15](#)
 - max_order, [16](#)
- jeod::MathMessages, [16](#)
 - ill_conditioned, [17](#)
 - MathMessages, [17](#)
 - operator=, [17](#)
- jeod::Matrix3x3, [18](#)
 - add, [19](#)
 - copy, [19](#)
 - cross_matrix, [20](#)
 - decr, [20](#)
 - identity, [20](#)
 - incr, [21](#)
 - initialize, [21](#)
 - invert, [21](#)
 - invert_symmetric, [22](#)
 - negate, [22](#), [23](#)
 - outer_product, [23](#)
 - print, [24](#)
 - product, [24](#)
 - product_left_transpose, [24](#)
 - product_right_transpose, [25](#)
 - product_transpose_transpose, [25](#)
 - scale, [26](#)
 - subtract, [27](#)
 - transform_matrix, [27](#)
 - transpose, [27](#), [28](#)
 - transpose_transform_matrix, [28](#)
- jeod::Numerical, [29](#)
 - compare_exact, [29](#)
 - fabs, [30](#)
 - square, [30](#)
 - square_incr, [30](#)
- jeod::Vector3, [31](#)
 - copy, [33](#)
 - cross, [33](#)
 - cross_decr, [34](#)
 - cross_incr, [34](#)
 - decr, [35](#)
 - diff, [35](#)
 - dot, [36](#)

- fill, [36](#)
- incr, [37](#)
- initialize, [38](#)
- negate, [38](#)
- normalize, [39](#)
- scale, [40](#)
- scale_decr, [41](#)
- scale_incr, [41](#)
- sum, [41](#), [42](#)
- transform, [42](#), [43](#)
- transform_decr, [43](#)
- transform_incr, [44](#)
- transform_transpose, [44](#), [45](#)
- transform_transpose_decr, [45](#)
- transform_transpose_incr, [46](#)
- unit, [46](#)
- vmag, [47](#)
- vmagsq, [47](#)
- zero_small, [47](#)

MAKE_MATH_MESSAGE_CODE

- math_messages.cc, [51](#)

macro_def.hh, [50](#)

macro_undef.hh, [51](#)

Math, [11](#)

- IDENTITY_3X3, [11](#)

math_messages.cc, [51](#)

- MAKE_MATH_MESSAGE_CODE, [51](#)

math_messages.hh, [52](#)

MathMessages

- jeod::MathMessages, [17](#)

matrix3x3.hh, [52](#)

matrix3x3_inline.hh, [53](#)

max_order

- jeod::GaussQuadrature, [16](#)

Models, [9](#)

negate

- jeod::Matrix3x3, [22](#), [23](#)
- jeod::Vector3, [38](#)

normalize

- jeod::Vector3, [39](#)

numerical.hh, [53](#)

numerical_inline.hh, [54](#)

operator=

- jeod::MathMessages, [17](#)

outer_product

- jeod::Matrix3x3, [23](#)

print

- jeod::Matrix3x3, [24](#)

product

- jeod::Matrix3x3, [24](#)

product_left_transpose

- jeod::Matrix3x3, [24](#)

product_right_transpose

- jeod::Matrix3x3, [25](#)

product_transpose_transpose

- jeod::Matrix3x3, [25](#)

SQRT_DBL_MIN

- jeod, [13](#)

scale

- jeod::Matrix3x3, [26](#)
- jeod::Vector3, [40](#)

scale_decr

- jeod::Vector3, [41](#)

scale_incr

- jeod::Vector3, [41](#)

square

- jeod::Numerical, [30](#)

square_incr

- jeod::Numerical, [30](#)

subtract

- jeod::Matrix3x3, [27](#)

sum

- jeod::Vector3, [41](#), [42](#)

transform

- jeod::Vector3, [42](#), [43](#)

transform_decr

- jeod::Vector3, [43](#)

transform_incr

- jeod::Vector3, [44](#)

transform_matrix

- jeod::Matrix3x3, [27](#)

transform_transpose

- jeod::Vector3, [44](#), [45](#)

transform_transpose_decr

- jeod::Vector3, [45](#)

transform_transpose_incr

- jeod::Vector3, [46](#)

transpose

- jeod::Matrix3x3, [27](#), [28](#)

transpose_transform_matrix

- jeod::Matrix3x3, [28](#)

unit

- jeod::Vector3, [46](#)

Utils, [10](#)

vector3.hh, [54](#)

vector3_inline.hh, [55](#)

vmag

- jeod::Vector3, [47](#)

vmagsq

- jeod::Vector3, [47](#)

zero_small

- jeod::Vector3, [47](#)