

MathematicalFunctions

5.0

Generated by Doxygen 1.8.5

Wed Jun 1 2022 12:07:35

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	GSL_SQRT_DBL_MIN	11
5.3.2.2	MAKE_MATH_MESSAGE_CODE	11
6	Namespace Documentation	13
6.1	jeod Namespace Reference	13
6.1.1	Detailed Description	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	16
7.2.2	Constructor & Destructor Documentation	17
7.2.2.1	MathMessages	17
7.2.2.2	MathMessages	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	17
7.3.1	Detailed Description	18
7.3.2	Member Function Documentation	18
7.3.2.1	add	18
7.3.2.2	copy	19
7.3.2.3	cross_matrix	19
7.3.2.4	decr	19
7.3.2.5	identity	19
7.3.2.6	incr	19
7.3.2.7	initialize	20
7.3.2.8	invert	20
7.3.2.9	invert_symmetric	20
7.3.2.10	negate	21
7.3.2.11	negate	21
7.3.2.12	outer_product	21
7.3.2.13	print	21
7.3.2.14	product	21
7.3.2.15	product_left_transpose	22
7.3.2.16	product_right_transpose	22
7.3.2.17	product_transpose_transpose	22
7.3.2.18	scale	23
7.3.2.19	scale	23
7.3.2.20	subtract	23
7.3.2.21	transform_matrix	23
7.3.2.22	transpose	24
7.3.2.23	transpose	24
7.3.2.24	transpose_transform_matrix	24
7.4	jeod::Numerical Class Reference	25
7.4.1	Detailed Description	25
7.4.2	Member Function Documentation	25
7.4.2.1	fabs	25

7.4.2.2	square	25
7.4.2.3	square_incr	26
7.5	jeod::Vector3 Class Reference	26
7.5.1	Detailed Description	27
7.5.2	Member Function Documentation	28
7.5.2.1	copy	28
7.5.2.2	cross	28
7.5.2.3	cross_decr	28
7.5.2.4	cross_incr	28
7.5.2.5	decr	29
7.5.2.6	decr	29
7.5.2.7	diff	29
7.5.2.8	dot	30
7.5.2.9	fill	30
7.5.2.10	incr	30
7.5.2.11	incr	30
7.5.2.12	initialize	31
7.5.2.13	negate	31
7.5.2.14	negate	31
7.5.2.15	normalize	31
7.5.2.16	normalize	32
7.5.2.17	scale	32
7.5.2.18	scale	32
7.5.2.19	scale_decr	33
7.5.2.20	scale_incr	33
7.5.2.21	sum	33
7.5.2.22	sum	33
7.5.2.23	transform	34
7.5.2.24	transform	34
7.5.2.25	transform_decr	34
7.5.2.26	transform_incr	35
7.5.2.27	transform_transpose	35
7.5.2.28	transform_transpose	35
7.5.2.29	transform_transpose_decr	36
7.5.2.30	transform_transpose_incr	36
7.5.2.31	unit	36
7.5.2.32	vmag	36
7.5.2.33	vmagsq	37
7.5.2.34	zero_small	37

8 File Documentation	39
8.1 dm_invert.cc File Reference	39
8.1.1 Detailed Description	39
8.2 dm_invert_symm.cc File Reference	39
8.2.1 Detailed Description	39
8.3 gauss_quadrature.cc File Reference	40
8.3.1 Detailed Description	40
8.4 gauss_quadrature.hh File Reference	40
8.4.1 Detailed Description	40
8.5 math_messages.cc File Reference	40
8.5.1 Detailed Description	41
8.6 math_messages.hh File Reference	41
8.6.1 Detailed Description	41
8.7 matrix3x3.hh File Reference	41
8.7.1 Detailed Description	42
8.8 matrix3x3_inline.hh File Reference	42
8.8.1 Detailed Description	42
8.9 numerical.hh File Reference	42
8.9.1 Detailed Description	42
8.10 numerical_inline.hh File Reference	43
8.10.1 Detailed Description	43
8.11 vector3.hh File Reference	43
8.11.1 Detailed Description	43
8.12 vector3_inline.hh File Reference	43
8.12.1 Detailed Description	44
 Index	 45

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	17
jeod::Numerical Provides miscellaneous numerical functions	25
jeod::Vector3 Provides static methods for operations that involve 3-vectors	26

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

dm_invert.cc	
Define Matrix3x3::invert	39
dm_invert_symm.cc	
Define Matrix3x3::invert_symmetric	39
gauss_quadrature.cc	
Define Gauss Quadrature functionality	40
gauss_quadrature.hh	
Gauss Quadrature implementation	40
math_messages.cc	
Implement the class MathMessages	40
math_messages.hh	
Define the class MathMessages	41
matrix3x3.hh	
Matrix math inline functions	41
matrix3x3_inline.hh	
Matrix math inline functions	42
numerical.hh	
Miscellaneous math inline functions	42
numerical_inline.hh	
Vector math inline functions	43
vector3.hh	
Vector math inline functions	43
vector3_inline.hh	
Vector math inline functions	43

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 [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 GSL_SQRT_DBL_MIN 1.4916681462400413e-154`
- `#define MAKE_MATH_MESSAGE_CODE(id) JEOD_MAKE_MESSAGE_CODE(MathMessages, "utils/math/", id)`

5.3.1 Detailed Description

5.3.2 Macro Definition Documentation

5.3.2.1 `#define GSL_SQRT_DBL_MIN 1.4916681462400413e-154`

Definition at line 33 of file `numerical_inline.hh`.

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

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

Definition at line 37 of file `math_messages.cc`.

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.

6.1.1 Detailed Description

Namespace jeod.

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 35 of file gauss_quadrature.hh.

7.1.2 Field Documentation

7.1.2.1 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 39 of file gauss_quadrature.hh.

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

Initial value:

```
=
{ { 0.00000000, 0.00000000, 0.00000000, 0.00000000, 0.00000000, 0.00000000,
    0.00000000, 0.00000000 },
  { 0.00000000, 0.00000000, 0.00000000, 0.00000000, 0.00000000, 0.00000000,
    0.00000000, 0.00000000 },
  { -0.5773503, 0.5773503, 0.00000000, 0.00000000, 0.00000000, 0.00000000,
    0.00000000, 0.00000000 },
  { -0.7745967, 0.00000000, 0.7745967, 0.00000000, 0.00000000, 0.00000000,
    0.00000000, 0.00000000 },
  { -0.8611363, -0.3399810, 0.3399810, 0.8611363, 0.00000000, 0.00000000,
    0.00000000, 0.00000000 },
  { -0.9061798, -0.5384693, 0.00000000, 0.5384693, 0.9061798, 0.00000000,
    0.00000000, 0.00000000 },
  { -0.9324695, -0.6612094, -0.2386192, 0.2386192, 0.6612094, 0.9324695,
    0.00000000, 0.00000000 },
  { -0.9491079, -0.7415312, -0.4058452, 0.00000000, 0.4058452, 0.7415312,
    0.9491079, 0.00000000 },
  { -0.9602899, -0.7966665, -0.5255324, -0.1834346, 0.1834346, 0.5255324,
    0.7966665, 0.9602899 }
}
```

Definition at line 40 of file `gauss_quadrature.hh`.

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

Definition at line 38 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>
```

Static Public Attributes

- static char const * `ill_conditioned` = "utils/math/" "ill_conditioned"
Error issued when an ill-conditioned matrix is detected.

Private Member Functions

- `MathMessages` (void)
Not implemented.
- `MathMessages` (const `MathMessages` &)
Not implemented.
- `MathMessages` & `operator=` (const `MathMessages` &)
Not implemented.

7.2.1 Detailed Description

Specifies the message IDs used in the math model.

Definition at line 38 of file `math_messages.hh`.

7.2.2 Constructor & Destructor Documentation

7.2.2.1 jeod::MathMessages::MathMessages (void) [private]

Not implemented.

7.2.2.2 jeod::MathMessages::MathMessages (const MathMessages &) [private]

Not implemented.

7.2.3 Member Function Documentation

7.2.3.1 MathMessages& jeod::MathMessages::operator= (const MathMessages &) [private]

Not implemented.

7.2.4 Field Documentation

7.2.4.1 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 46 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](#) (double const 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](#) (double const vec_left[3], double const 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** (double const addend[3][3], double mat[3][3])
Increment matrix in-place: $mat[i][j] = mat[i][j] + addend[i][j]$.
- static void **decr** (double const subtrahend[3][3], double mat[3][3])
Decrement matrix in-place: $mat[i][j] = mat[i][j] - subtrahend[i][j]$.
- static void **copy** (double const input_mat[3][3], double copy[3][3])
Copy matrix: $copy[i][j] = mat[i][j]$.
- static void **negate** (double const input_mat[3][3], double copy[3][3])
Negate matrix: $copy[i][j] = -mat[i][j]$.
- static void **transpose** (double const input_mat[3][3], double trans[3][3])
Transpose matrix: $copy[i][j] = mat[j][i]$.
- static void **scale** (double const mat[3][3], double scalar, double prod[3][3])
*Scale matrix: $copy[i][j] = scalar * mat[i][j]$.*
- static void **add** (double const augend[3][3], double const addend[3][3], double sum[3][3])
Add matrices: $sum[i][j] = augend[i][j] + addend[i][j]$.
- static void **subtract** (double const minuend[3][3], double const subtrahend[3][3], double diff[3][3])
Subtract matrices: $diff[i][j] = minuend[i][j] - subtrahend[i][j]$.
- static void **product** (double const mat_left[3][3], double const 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** (double const mat_left[3][3], double const 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** (double const mat_left[3][3], double const 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** (double const mat_left[3][3], double const 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** (double const trans[3][3], double const mat[3][3], double prod[3][3])
*Compute the matrix product $trans * mat * trans^T$: $prod[i][j] = trans[i][k] * mat[k][l] * trans[j][l]$.*
- static void **transpose_transform_matrix** (double const trans[3][3], double const mat[3][3], double prod[3][3])
*Compute the matrix product $trans^T * mat * trans$: $prod[i][j] = trans[k][i] * mat[k][l] * trans[l][j]$.*
- static int **invert** (double const matrix[3][3], double inverse[3][3])
Compute the inverse of a 3x3 matrix.
- static int **invert_symmetric** (double const matrix[3][3], double inverse[3][3])
Compute the inverse of a symmetric 3x3 matrix.
- static void **print** (double const 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 38 of file matrix3x3.hh.

7.3.2 Member Function Documentation

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

Add matrices: $sum[i][j] = augend[i][j] + addend[i][j]$.

Parameters

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

Definition at line 378 of file matrix3x3_inline.hh.

7.3.2.2 `void jeod::Matrix3x3::copy (double const 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 264 of file matrix3x3_inline.hh.

7.3.2.3 `void jeod::Matrix3x3::cross_matrix (double const 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 83 of file matrix3x3_inline.hh.

7.3.2.4 `void jeod::Matrix3x3::decr (double const 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 239 of file matrix3x3_inline.hh.

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

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

Parameters

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

Definition at line 62 of file matrix3x3_inline.hh.

7.3.2.6 `void jeod::Matrix3x3::incr (double const 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

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

Definition at line 214 of file `matrix3x3_inline.hh`.

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

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

Parameters

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

Definition at line 44 of file `matrix3x3_inline.hh`.

7.3.2.8 `int jeod::Matrix3x3::invert (double const 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

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

Definition at line 78 of file `dm_invert.cc`.

References `jeod::MathMessages::ill_conditioned`.

7.3.2.9 `int jeod::Matrix3x3::invert_symmetric (double const 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 79 of file dm_invert_symm.cc.

References `jeod::MathMessages::ill_conditioned`.

7.3.2.10 `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 137 of file matrix3x3_inline.hh.

7.3.2.11 `void jeod::Matrix3x3::negate (double const 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 293 of file matrix3x3_inline.hh.

7.3.2.12 `void jeod::Matrix3x3::outer_product (double const vec_left[3], double const 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 113 of file matrix3x3_inline.hh.

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

Print matrix to standard error.

Parameters

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

Definition at line 704 of file matrix3x3_inline.hh.

7.3.2.14 `void jeod::Matrix3x3::product (double const mat_left[3][3], double const 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 436 of file matrix3x3_inline.hh.

Referenced by transform_matrix(), and transpose_transform_matrix().

7.3.2.15 void jeod::Matrix3x3::product_left_transpose (double const *mat_left*[3][3], double const *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 493 of file matrix3x3_inline.hh.

Referenced by transpose_transform_matrix().

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

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

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 550 of file matrix3x3_inline.hh.

Referenced by transform_matrix().

7.3.2.17 void jeod::Matrix3x3::product_transpose_transpose (double const *mat_left*[3][3], double const *mat_right*[3][3], double *prod*[3][3]) [inline],[static]

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

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 607 of file matrix3x3_inline.hh.

7.3.2.18 `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 189 of file matrix3x3_inline.hh.

7.3.2.19 `void jeod::Matrix3x3::scale (double const 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 351 of file matrix3x3_inline.hh.

7.3.2.20 `void jeod::Matrix3x3::subtract (double const minuend[3][3], double const 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 405 of file matrix3x3_inline.hh.

7.3.2.21 `void jeod::Matrix3x3::transform_matrix (double const trans[3][3], double const 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 664 of file matrix3x3_inline.hh.

References product(), and product_right_transpose().

7.3.2.22 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 162 of file matrix3x3_inline.hh.

7.3.2.23 void jeod::Matrix3x3::transpose (double const *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 323 of file matrix3x3_inline.hh.

7.3.2.24 void jeod::Matrix3x3::transpose_transform_matrix (double const *trans*[3][3], double const *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 687 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 undeflow.
- static double [square_incr](#) (double value, double &sum)
Add number squared to accumulator, protecting against undeflow.

7.4.1 Detailed Description

Provides miscellaneous numerical functions.

Definition at line 32 of file numerical.hh.

7.4.2 Member Function Documentation

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

Absolute value.

Returns

Absolute value of x

Parameters

in	x	x
----	---	---

Definition at line 45 of file numerical_inline.hh.

Referenced by [square\(\)](#), [square_incr\(\)](#), and [jeod::Vector3::zero_small\(\)](#).

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

Compute the square of a number, protecting against undeflow.

Returns

value^2 or zero if too small

Parameters

in	value	Value
----	-------	-------

Definition at line 58 of file numerical_inline.hh.

References [fabs\(\)](#), and [GSL_SQRT_DBL_MIN](#).

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

Add number squared to accumulator, protecting against underflow.

Returns

Accumulated value

Parameters

<code>in</code>	<code>value</code>	Value
<code>in, out</code>	<code>sum</code>	Accumulator

Definition at line 78 of file `numerical_inline.hh`.

References `fabs()`, and `GSL_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, $vec[i] = \delta_{ij}$ (δ_{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](#) (double const vec[3], double copy[3])
Copy vector contents, $copy[i] = vec[i]$.
- static double [dot](#) (double const vec2[3], double const vec1[3])
*Compute vector inner product, $result = \sum_i vec1[i] * vec2[i]$.*
- static double [vmagsq](#) (double const vec[3])
Compute square of vector magnitude, $result = dot(vec, vec)$, but protects against underflow.
- static double [vmag](#) (double const 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](#) (double const 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](#) (double const 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** (double const vec[3], double copy[3])
Negate vector, $copy[i] = -vec[i]$.
- static double * **transform** (double const tmat[3][3], double const vec[3], double prod[3])
*Transform a column vector, $prod[i] = tmat[i][j]*vec[j]$.*
- static double * **transform** (double const tmat[3][3], double vec[3])
*Transform a column vector in-place, $vec[i] <- tmat[i][j]*vec[j]$.*
- static double * **transform_transpose** (double const tmat[3][3], double const vec[3], double prod[3])
*Transform a column vector with the transpose, $prod[i] = tmat[j][i]*vec[j]$.*
- static double * **transform_transpose** (double const 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** (double const addend[3], double vec[3])
Increment a vector, $vec[i] += addend[i]$.
- static double * **incr** (double const addend1[3], double const addend2[3], double vec[3])
Increment a vector, $vec[i] += addend1[i] + addend2[i]$.
- static double * **decr** (double const subtrahend[3], double vec[3])
Decrement a vector, $vec[i] -= subtrahend[i]$.
- static double * **decr** (double const subtrahend1[3], double const subtrahend2[3], double vec[3])
Decrement a vector, $vec[i] -= subtrahend1[i] + subtrahend2[i]$.
- static double * **sum** (double const addend1[3], double const addend2[3], double vec[3])
Compute the sum of two vectors, $vec[i] = addend1[i] + addend2[i]$.
- static double * **sum** (double const addend1[3], double const addend2[3], double const addend3[3], double vec[3])
Compute the sum of three vectors, $vec[i] = addend1[i] + addend2[i] + addend3[i]$.
- static double * **diff** (double const minuend[3], double const subtrahend[3], double vec[3])
Compute the difference between two vectors, $diff[i] = minuend[i] - subtrahend[i]$.
- static double * **cross** (double const vec_left[3], double const 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** (double const vec[3], double scalar, double prod[3])
*Increment a vector with a scaled vector, $prod[i] += scalar*vec[i]$.*
- static double * **scale_decr** (double const vec[3], double scalar, double prod[3])
*Decrement a vector with a scaled vector, $prod[i] += scalar*vec[i]$.*
- static double * **cross_incr** (double const vec_left[3], double const 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** (double const vec_left[3], double const 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** (double const tmat[3][3], double const vec[3], double prod[3])
*Increment a vector with a transformed column vector, $prod[i] += tmat[i][j]*vec[j]$.*
- static double * **transform_decr** (double const tmat[3][3], double const 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** (double const tmat[3][3], double const 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** (double const tmat[3][3], double const 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 33 of file vector3.hh.

7.5.2 Member Function Documentation

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

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

Returns

Copied vector

Parameters

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

Definition at line 124 of file `vector3_inline.hh`.

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

7.5.2.2 `double * jeod::Vector3::cross (double const vec_left[3], double const 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

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

Definition at line 568 of file `vector3_inline.hh`.

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

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

Returns

Decrementated vector

Parameters

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

Definition at line 656 of file `vector3_inline.hh`.

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

Increment a vector with the 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
in, out	<i>prod</i>	Cross product vector

Definition at line 634 of file vector3_inline.hh.

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

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

Returns

Decrementated vector

Parameters

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

Definition at line 457 of file vector3_inline.hh.

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

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

Returns

Decrementated vector

Parameters

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

Definition at line 478 of file vector3_inline.hh.

7.5.2.7 `double * jeod::Vector3::diff (double const minuend[3], double const 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 546 of file vector3_inline.hh.

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

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

Returns

Inner product

Parameters

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

Definition at line 144 of file `vector3_inline.hh`.

7.5.2.9 `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

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

Definition at line 78 of file `vector3_inline.hh`.

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

Increment a vector, $\text{vec}[i] += \text{addend}[i]$.

Returns

Incremented vector

Parameters

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

Definition at line 415 of file `vector3_inline.hh`.

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

Increment a vector, $\text{vec}[i] += \text{addend1}[i] + \text{addend2}[i]$.

Returns

Incremented vector

Parameters

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

Definition at line 436 of file vector3_inline.hh.

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

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

Returns

Zero-filled vector

Parameters

out	<i>vec</i>	Zero-filled vector
-----	------------	--------------------

Definition at line 44 of file vector3_inline.hh.

Referenced by `normalize()`.

7.5.2.13 `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	<i>vec</i>	Vector
---------	------------	--------

Definition at line 272 of file vector3_inline.hh.

7.5.2.14 `double * jeod::Vector3::negate (double const 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 291 of file vector3_inline.hh.

References `copy()`.

7.5.2.15 `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

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

Definition at line 189 of file vector3_inline.hh.

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

Referenced by normalize().

7.5.2.16 `double * jeod::Vector3::normalize (double const 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

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

Definition at line 213 of file vector3_inline.hh.

References copy(), and normalize().

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

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

Returns

Scaled vector

Parameters

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

Definition at line 231 of file vector3_inline.hh.

Referenced by normalize().

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

Scale a vector, $\text{prod}[i] = \text{vec}[i] * \text{scalar}$.

Returns

Scaled vector

Parameters

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

Definition at line 252 of file vector3_inline.hh.

7.5.2.19 `double * jeod::Vector3::scale_decr (double const 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

Decrement vector

Parameters

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

Definition at line 612 of file vector3_inline.hh.

7.5.2.20 `double * jeod::Vector3::scale_incr (double const 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 590 of file vector3_inline.hh.

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

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

Returns

Sum vector

Parameters

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

Definition at line 500 of file vector3_inline.hh.

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

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

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 523 of file vector3_inline.hh.

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

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

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 312 of file vector3_inline.hh.

Referenced by transform().

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

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

Returns

Transformed vector

Parameters

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

Definition at line 342 of file vector3_inline.hh.

References copy(), and transform().

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

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

Returns

Decrementated vector

Parameters

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

Definition at line 709 of file vector3_inline.hh.

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

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

Definition at line 678 of file vector3_inline.hh.

7.5.2.27 `double * jeod::Vector3::transform_transpose (double const tmat[3][3], double const 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 364 of file vector3_inline.hh.

Referenced by transform_transpose().

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

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

Returns

Transformed vector

Parameters

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

Definition at line 394 of file vector3_inline.hh.

References copy(), and transform_transpose().

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

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

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 771 of file vector3_inline.hh.

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

<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 740 of file vector3_inline.hh.

7.5.2.31 `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

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

Definition at line 60 of file vector3_inline.hh.

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

Compute vector magnitude, $result = \sqrt{vmagsq(vec)}$

Returns

Vector magnitude

Parameters

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

Definition at line 175 of file vector3_inline.hh.

References vmagsq().

Referenced by normalize().

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

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

Returns

Inner product

Parameters

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

Definition at line 161 of file vector3_inline.hh.

Referenced by vmag().

7.5.2.34 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 95 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 "utils/message/include/message_handler.hh"
#include "../include/matrix3x3.hh"
#include "../include/math_messages.hh"
```

Namespaces

- [jeod](#)

Namespace jeod.

8.1.1 Detailed Description

Define Matrix3x3::invert.

Definition in file [dm_invert.cc](#).

8.2 dm_invert_symm.cc File Reference

Define Matrix3x3::invert_symmetric.

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

Namespaces

- [jeod](#)

Namespace jeod.

8.2.1 Detailed Description

Define Matrix3x3::invert_symmetric.

Definition in file [dm_invert_symm.cc](#).

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.

Definition in file [gauss_quadrature.cc](#).

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.

Definition in file [gauss_quadrature.hh](#).

8.5 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)`

8.5.1 Detailed Description

Implement the class MathMessages.

Definition in file [math_messages.cc](#).

8.6 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.

8.6.1 Detailed Description

Define the class MathMessages.

Definition in file [math_messages.hh](#).

8.7 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.

8.7.1 Detailed Description

Matrix math inline functions.

Definition in file [matrix3x3.hh](#).

8.8 matrix3x3_inline.hh File Reference

Matrix math inline functions.

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

Namespaces

- [jeod](#)

Namespace jeod.

8.8.1 Detailed Description

Matrix math inline functions.

Definition in file [matrix3x3_inline.hh](#).

8.9 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.

8.9.1 Detailed Description

Miscellaneous math inline functions.

Definition in file [numerical.hh](#).

8.10 numerical_inline.hh File Reference

Vector math inline functions.

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

Namespaces

- [jeod](#)

Namespace jeod.

Macros

- #define [GSL_SQRT_DBL_MIN](#) 1.4916681462400413e-154

8.10.1 Detailed Description

Vector math inline functions.

Definition in file [numerical_inline.hh](#).

8.11 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.

8.11.1 Detailed Description

Vector math inline functions.

Definition in file [vector3.hh](#).

8.12 vector3_inline.hh File Reference

Vector math inline functions.

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

Namespaces

- [jeod](#)

Namespace jeod.

8.12.1 Detailed Description

Vector math inline functions.

Definition in file [vector3_inline.hh](#).

Index

add
 jeod::Matrix3x3, 18

copy
 jeod::Matrix3x3, 19
 jeod::Vector3, 28

cross
 jeod::Vector3, 28

cross_decr
 jeod::Vector3, 28

cross_incr
 jeod::Vector3, 28

cross_matrix
 jeod::Matrix3x3, 19

decr
 jeod::Matrix3x3, 19
 jeod::Vector3, 29

diff
 jeod::Vector3, 29

dm_invert.cc, 39

dm_invert_symm.cc, 39

dot
 jeod::Vector3, 29

fabs
 jeod::Numerical, 25

fill
 jeod::Vector3, 30

GSL_SQRT_DBL_MIN
 Math, 11

gauss_quadrature.cc, 40

gauss_quadrature.hh, 40

gauss_weights
 jeod::GaussQuadrature, 15

gauss_xvalues
 jeod::GaussQuadrature, 15

identity
 jeod::Matrix3x3, 19

ill_conditioned
 jeod::MathMessages, 17

incr
 jeod::Matrix3x3, 19
 jeod::Vector3, 30

initialize
 jeod::Matrix3x3, 20
 jeod::Vector3, 31

invert
 jeod::Matrix3x3, 20

invert_symmetric
 jeod::Matrix3x3, 20

jeod, 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, 17
 add, 18
 copy, 19
 cross_matrix, 19
 decr, 19
 identity, 19
 incr, 19
 initialize, 20
 invert, 20
 invert_symmetric, 20
 negate, 21
 outer_product, 21
 print, 21
 product, 21
 product_left_transpose, 22
 product_right_transpose, 22
 product_transpose_transpose, 22
 scale, 23
 subtract, 23
 transform_matrix, 23
 transpose, 24
 transpose_transform_matrix, 24

jeod::Numerical, 25
 fabs, 25
 square, 25
 square_incr, 25

jeod::Vector3, 26
 copy, 28
 cross, 28
 cross_decr, 28
 cross_incr, 28
 decr, 29
 diff, 29
 dot, 29
 fill, 30
 incr, 30
 initialize, 31
 negate, 31

- normalize, [31](#), [32](#)
- scale, [32](#)
- scale_decr, [32](#)
- scale_incr, [33](#)
- sum, [33](#)
- transform, [34](#)
- transform_decr, [34](#)
- transform_incr, [35](#)
- transform_transpose, [35](#)
- transform_transpose_decr, [35](#)
- transform_transpose_incr, [36](#)
- unit, [36](#)
- vmag, [36](#)
- vmagsq, [37](#)
- zero_small, [37](#)

Math, [11](#)

- GSL_SQRT_DBL_MIN, [11](#)

math_messages.cc, [40](#)

math_messages.hh, [41](#)

MathMessages

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

matrix3x3.hh, [41](#)

matrix3x3_inline.hh, [42](#)

max_order

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

Models, [9](#)

negate

- jeod::Matrix3x3, [21](#)
- jeod::Vector3, [31](#)

normalize

- jeod::Vector3, [31](#), [32](#)

numerical.hh, [42](#)

numerical_inline.hh, [43](#)

operator=

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

outer_product

- jeod::Matrix3x3, [21](#)

print

- jeod::Matrix3x3, [21](#)

product

- jeod::Matrix3x3, [21](#)

product_left_transpose

- jeod::Matrix3x3, [22](#)

product_right_transpose

- jeod::Matrix3x3, [22](#)

product_transpose_transpose

- jeod::Matrix3x3, [22](#)

scale

- jeod::Matrix3x3, [23](#)
- jeod::Vector3, [32](#)

scale_decr

- jeod::Vector3, [32](#)

scale_incr

- jeod::Vector3, [33](#)

square

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

square_incr

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

subtract

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

sum

- jeod::Vector3, [33](#)

transform

- jeod::Vector3, [34](#)

transform_decr

- jeod::Vector3, [34](#)

transform_incr

- jeod::Vector3, [35](#)

transform_matrix

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

transform_transpose

- jeod::Vector3, [35](#)

transform_transpose_decr

- jeod::Vector3, [35](#)

transform_transpose_incr

- jeod::Vector3, [36](#)

transpose

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

transpose_transform_matrix

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

unit

- jeod::Vector3, [36](#)

Utils, [10](#)

vector3.hh, [43](#)

vector3_inline.hh, [43](#)

vmag

- jeod::Vector3, [36](#)

vmagsq

- jeod::Vector3, [37](#)

zero_small

- jeod::Vector3, [37](#)