

vm\_math

1.0.7

Generated by Doxygen 1.7.1

Thu Jul 19 2012 17:19:13

## Contents

<b>1</b>	<b>The vm_Math C++ template Library</b>	<b>1</b>
1.1	Copyright . . . . .	1
1.2	Overview . . . . .	1
1.3	vm_VMath common numerical operations . . . . .	2
1.4	vm_V3Math common numerical operations . . . . .	2
1.5	vm_M3math common numerical operations . . . . .	3
<b>2</b>	<b>Module Index</b>	<b>3</b>
2.1	Modules . . . . .	3
<b>3</b>	<b>Directory Hierarchy</b>	<b>4</b>
3.1	Directories . . . . .	4
<b>4</b>	<b>Class Index</b>	<b>4</b>
4.1	Class Hierarchy . . . . .	4
<b>5</b>	<b>Class Index</b>	<b>5</b>
5.1	Class List . . . . .	5
<b>6</b>	<b>Module Documentation</b>	<b>5</b>
6.1	vm_VMath common numerical operations . . . . .	5
6.2	vm_V3Math common numerical operations . . . . .	5
6.3	vm_M3Math common numerical operations . . . . .	5
6.4	Index calculations . . . . .	5
6.4.1	Function Documentation . . . . .	5
6.5	Initialize a matrix. . . . .	6
6.5.1	Function Documentation . . . . .	6
6.6	For a matrix, insert or extract a vector. . . . .	7
6.6.1	Function Documentation . . . . .	7
6.7	Matrix Vector operations. . . . .	9
6.7.1	Function Documentation . . . . .	9
6.8	Matrix Matrix operations. . . . .	10

6.9	IO operations. . . . .	10
6.9.1	Function Documentation . . . . .	10
6.10	Set vector components . . . . .	11
6.10.1	Function Documentation . . . . .	11
6.11	Normalize vectors . . . . .	12
6.11.1	Function Documentation . . . . .	12
6.12	Tests for normality, orthogonality . . . . .	13
6.12.1	Function Documentation . . . . .	13
6.13	Dot and Cross products. . . . .	14
6.13.1	Function Documentation . . . . .	14
6.14	IO operations. . . . .	15
6.14.1	Function Documentation . . . . .	15
6.15	Copy routines . . . . .	16
6.15.1	Function Documentation . . . . .	16
6.16	Set functions . . . . .	17
6.16.1	Function Documentation . . . . .	17
6.17	Componentwise op_eq operations (+=, -=, /=) . . . . .	17
6.17.1	Function Documentation . . . . .	18
6.18	{add,sub,mul,div}: componentwise binary operations. . . . .	20
6.18.1	Function Documentation . . . . .	21
6.19	Linear combinations . . . . .	24
6.19.1	Function Documentation . . . . .	24
6.20	I/O operations. . . . .	25
6.20.1	Function Documentation . . . . .	25
<b>7</b>	<b>Directory Documentation</b>	<b>26</b>
7.1	vm_math/ Directory Reference . . . . .	26
<b>8</b>	<b>Class Documentation</b>	<b>26</b>
8.1	vm_M3Math< T_fp > Class Template Reference . . . . .	26
8.1.1	Detailed Description . . . . .	28
8.2	vm_V3Math< T_fp > Class Template Reference . . . . .	29

8.2.1	Detailed Description . . . . .	30
8.2.2	Member Typedef Documentation . . . . .	30
8.3	vm_VMath< T_fp, N_len > Class Template Reference . . . . .	31
8.3.1	Detailed Description . . . . .	32
8.3.2	Member Typedef Documentation . . . . .	32

## 1 The vm\_Math C++ template Library

### 1.1 Copyright

#### Author

Terry Gaetz

Copyright (C) 2006 Smithsonian Astrophysical Observatory

This file is part of vm\_math

vm\_math is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

vm\_Math is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc. 51 Franklin Street, Fifth Floor Boston, MA 02110-1301, USA

### 1.2 Overview

The vm\_math C++ library is a collection of templatized routines to deal efficiently with low-level floating point vectors, 3-vectors, and 3x3 matrices. It is split up into several sub-packages (see the **Modules** page for more information).

The vm\_math package brings together a number of functions for operating on 3-vectors and 3-matrices, and for creating orthonormal 3-matrices corresponding to proper rotations (i.e., orthonormal matrices preserving the coordinate system parity).

These are implemented as inlined static member functions; the functions are bundled into classes in order to take them out of the global namespace. By making the functions static member functions, they can be called without recourse to a an instantiated object, e.g.

```
double v1[] = { 1.3, 2.4, 7.2, 9.7 };
double v2[] = { 7.2, 1.9, 3.1, 4.1 };
double foo = vm_VMath<double,4>::dot( v2, v1 );
```

The package currently contains 3 classes:

- `vm_VMath<T_fp,N_len>`: common numerical operations on `N_len`-long one-dimensional arrays of floating point type `T_fp`. This class captures operations which do not depend on the explicit 3-vector or 3x3-matrix properties.
- `vm_V3math<T_fp>`: common numerical operations on 3-vectors of floating point type `T_fp`. Publicly derives from `vm_VMath<T_fp,3>`. This class adds in specifically 3-vector functionality such as vector cross products.
- `vm_M3math<T_fp>`: common numerical operations on 3x3 matrices of floating point type `T_fp`. The matrix is assumed to be stored as a contiguous one-dimensional array of 9 `T_fp`'s. Publicly derives from `vm_VMath<T_fp,9>`. This class adds in specifically matrix properties such as multiplication and matrix-vector multiplication.

All members are inlined, so the package consists only of header files.

### 1.3 **vm\_VMath common numerical operations**

on `N_len`-long \* 1 dimensional arrays of `T_fp`

These routines provide the basic low-level support for 1-dimensional arrays of floating point values. The library is a C++ template library, with template parameters `T_fp` and `N_len`. `T_fp` is the floating point type. It must be a simple type (float or double), since it is assumed that the array can be copied as plain ol' data using `memcpy`. `N_len` is the dimension of the vector, assumed to be small, e.g., 9 for a flat representation of a 3x3 matrix. The class has no state information with all functionality implemented as (we hope) inlined functions.

To use these routines, ensure that you include the correct header [file](#):

```
#include <vm_vmath/vm_vmath.h>
```

or

```
#include <vm_vmath/vm_math.h>
```

### 1.4 **vm\_V3Math common numerical operations**

on 3-vectors of floating point values

These routines provide the basic low-level support for 3-vectors of floating point values of type `T_fp` (float or double). The library is a C++ template library, templated on `T_fp`.

To use these routines, ensure that you include the correct header [file](#):

```
#include <vm_vmath/vm_v3math.h>
```

or

```
#include <vm_vmath/vm_math.h>
```

## 1.5 vm\_M3math common numerical operations

on 3x3 matrices of floating point values

These routines provide the basic low-level support for 3x3 matrices of floating point values of type `T_fp` (float or double). The library is a C++ template library, templated on `T_fp`.

To use these routines, ensure that you include the correct header [file](#):

```
#include <vm_vmath/vm_m3math.h>
```

or

```
#include <vm_vmath/vm_math.h>
```

## 2 Module Index

### 2.1 Modules

Here is a list of all modules:

<b>vm_VMath common numerical operations</b>	<b>5</b>
<b>vm_V3Math common numerical operations</b>	<b>5</b>
<b>vm_M3Math common numerical operations</b>	<b>5</b>
<b>Index calculations</b>	<b>5</b>
<b>Initialize a matrix.</b>	<b>6</b>
<b>For a matrix, insert or extract a vector.</b>	<b>7</b>
<b>Matrix Vector operations.</b>	<b>9</b>

Matrix Matrix operations.	<a href="#">10</a>
IO operations.	<a href="#">10</a>
Set vector components	<a href="#">11</a>
Normalize vectors	<a href="#">12</a>
Tests for normality, orthogonality	<a href="#">13</a>
Dot and Cross products.	<a href="#">14</a>
IO operations.	<a href="#">15</a>
Copy routines	<a href="#">16</a>
Set functions	<a href="#">17</a>
Componentwise op_eq operations (+, -, =, /=)	<a href="#">17</a>
{add,sub,mul,div}: componentwise binary operations.	<a href="#">20</a>
Linear combinations	<a href="#">24</a>
I/O operations.	<a href="#">25</a>

## 3 Directory Hierarchy

### 3.1 Directories

This directory hierarchy is sorted roughly, but not completely, alphabetically:

vm_math	<a href="#">26</a>
---------	--------------------

## 4 Class Index

### 4.1 Class Hierarchy

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

vm_VMath< T_fp, N_len >	<a href="#">31</a>
vm_VMath< T_fp, 3 >	<a href="#">31</a>

<a href="#">vm_V3Math&lt; T_fp &gt;</a>	<a href="#">29</a>
<a href="#">vm_VMath&lt; T_fp, 9 &gt;</a>	<a href="#">31</a>
<a href="#">vm_M3Math&lt; T_fp &gt;</a>	<a href="#">26</a>

## 5 Class Index

### 5.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

<a href="#">vm_M3Math&lt; T_fp &gt;</a>	<a href="#">26</a>
<a href="#">vm_V3Math&lt; T_fp &gt;</a>	<a href="#">29</a>
<a href="#">vm_VMath&lt; T_fp, N_len &gt;</a>	<a href="#">31</a>

## 6 Module Documentation

### 6.1 vm\_VMath common numerical operations

### 6.2 vm\_V3Math common numerical operations

### 6.3 vm\_M3Math common numerical operations

### 6.4 Index calculations

#### Functions

- static int [vm\\_M3Math::at](#) (int i, int j)

#### 6.4.1 Function Documentation

##### 6.4.1.1 `template<class T_fp> int vm_M3Math< T_fp >::at ( int i, int j )` [inline, static, inherited]

Index calculation.

```
Array value m[i][j] is *(m + i*EColStride + j)
```



**Returns**

offset of index set  $i, j$  (used for flat array storage)

**Parameters**

$i$  index

$j$  index

Definition at line 290 of file `vm_m3math.h`.

**6.5 Initialize a matrix.****Functions**

- static void `vm_M3Math::init_by_row` ( $T\_fp$   $m[]$ ,  $T\_fp$  const  $row0[]$ ,  $T\_fp$  const  $row1[]$ ,  $T\_fp$  const  $row2[]$ )
- static void `vm_M3Math::init_by_col` ( $T\_fp$   $m[]$ ,  $T\_fp$  const  $col0[]$ ,  $T\_fp$  const  $col1[]$ ,  $T\_fp$  const  $col2[]$ )
- static void `vm_M3Math::dyad_product` ( $T\_fp$   $m[]$ ,  $T\_fp$  const  $v1[]$ ,  $T\_fp$  const  $v2[]$ )

**6.5.1 Function Documentation**

**6.5.1.1** `template<class T_fp> void vm_M3Math< T_fp >::dyad_product ( T_fp  $m[]$ , T_fp const  $v1[]$ , T_fp const  $v2[]$  ) [inline, static, inherited]`

Initialize a matrix to a dyadic (outer) product of vectors

For each  $i, j$ :  $m[at(i,j)] = v1[i] * v2[j]$

**Parameters**

$m$  matrix (as stored flat 1D array)

$v1$  1st vector

$v2$  2nd vector

Definition at line 361 of file `vm_m3math.h`.

**6.5.1.2** `template<class T_fp> void vm_M3Math< T_fp >::init_by_col ( T_fp  $m[]$ , T_fp const  $col0[]$ , T_fp const  $col1[]$ , T_fp const  $col2[]$  ) [inline, static, inherited]`

Initialize matrix by column; set columns of  $m$  to  $col0, col1, col2$ .

**Parameters**

*m* matrix (as stored flat 1D array)  
*col0* 1st column vector  
*col1* 2nd column vector  
*col2* 3rd column vector

Definition at line 308 of file vm\_m3math.h.

**6.5.1.3** `template<class T_fp > void vm_M3Math< T_fp >::init_by_row ( T_fp m[], T_fp const row0[], T_fp const row1[], T_fp const row2[] ) [inline, static, inherited]`

Initialize matrix by row; set rows of m to row0, row1, row2.

**Parameters**

*m* matrix (as stored flat 1D array)  
*row0* 1st row vector  
*row1* 2nd row vector  
*row2* 3rd row vector

Definition at line 298 of file vm\_m3math.h.

**6.6 For a matrix, insert or extract a vector.****Functions**

- static void [vm\\_M3Math::inject\\_row](#) (T\_fp m[], T\_fp const row[], int whichrow)
- static void [vm\\_M3Math::inject\\_col](#) (T\_fp m[], T\_fp const col[], int whichcol)
- static void [vm\\_M3Math::extract\\_row](#) (T\_fp const m[], T\_fp row[], int whichrow)
- static void [vm\\_M3Math::extract\\_col](#) (T\_fp const m[], T\_fp col[], int whichcol)

**6.6.1 Function Documentation**

**6.6.1.1** `template<class T_fp > void vm_M3Math< T_fp >::extract_col ( T_fp const m[], T_fp col[], int whichcol ) [inline, static, inherited]`

Copy a given column of m to a vector.

**Parameters***m* matrix (as stored flat 1D array)*col* vector to receive the copy*whichcol* column index

Definition at line 347 of file vm\_m3math.h.

**6.6.1.2** `template<class T_fp > void vm_M3Math< T_fp >::extract_row ( T_fp const m[], T_fp row[], int whichrow ) [inline, static, inherited]`

Copy a given row of m to a vector.

**Parameters***m* matrix (as stored flat 1D array)*row* vector to receive the copy*whichrow* row index

Definition at line 341 of file vm\_m3math.h.

**6.6.1.3** `template<class T_fp > void vm_M3Math< T_fp >::inject_col ( T_fp m[], T_fp const col[], int whichcol ) [inline, static, inherited]`

Copy a vector to a given column of m.

**Parameters***m* matrix (as stored flat 1D array)*col* vector to be stored*whichcol* column index

Definition at line 327 of file vm\_m3math.h.

**6.6.1.4** `template<class T_fp > void vm_M3Math< T_fp >::inject_row ( T_fp m[], T_fp const row[], int whichrow ) [inline, static, inherited]`

Copy a vector to a given row of m.

**Parameters**

- m* matrix (as stored flat 1D array)
- row* vector to be stored
- whichrow* row index

Definition at line 321 of file vm\_m3math.h.

**6.7 Matrix Vector operations.****Functions**

- static void [vm\\_M3Math::mvmult](#) (T\_fp res[ ], T\_fp const m[ ], T\_fp const v[ ])
- static void [vm\\_M3Math::mtvmult](#) (T\_fp res[ ], T\_fp const m[ ], T\_fp const v[ ])

**6.7.1 Function Documentation**

**6.7.1.1** `template<class T_fp > void vm_M3Math< T_fp >::mtvmult ( T_fp res[ ], T_fp const m[ ], T_fp const v[ ] ) [inline, static, inherited]`

Matrix multiplication of vector v by transpose of matrix m.

result = m\_transpose\_matrix\_multiply\_v.

**Parameters**

- res* resulting matrix (as stored flat 1D array)
- m* matrix to be transposed and multiplied
- v* vector to be multiplied

Definition at line 382 of file vm\_m3math.h.

**6.7.1.2** `template<class T_fp > void vm_M3Math< T_fp >::mvmult ( T_fp res[ ], T_fp const m[ ], T_fp const v[ ] ) [inline, static, inherited]`

Matrix multiplication of vector v by matrix m.

result = m\_matrix\_multiply\_v.

**Parameters**

- res* resulting matrix (as stored flat 1D array)

*m* matrix to be multiplied

*v* vector to be multiplied

Definition at line 373 of file vm\_m3math.h.

## 6.8 Matrix Matrix operations.

### Functions

- static void **vm\_M3Math::mmult** (T\_fp mres[ ], T\_fp const m1[ ], T\_fp const m2[ ])

## 6.9 IO operations.

### Functions

- static std::ostream & **vm\_M3Math::print\_on** (std::ostream &os, T\_fp const m[ ], char const prefix[ ]="", char const postfix[ ]="")
- static void **vm\_M3Math::cprint\_on** (FILE \*of, T\_fp const m[ ], char const prefix[ ]="", char const postfix[ ]="")

### 6.9.1 Function Documentation

**6.9.1.1** `template<class T_fp > void vm_M3Math< T_fp >::cprint_on ( FILE * of, T_fp const m[ ], char const prefix[ ] = "", char const postfix[ ] = "" ) [inline, static, inherited]`

Print a matrix to a FILE\* stream.

#### Parameters

*of* the FILE\*

*m* matrix to be printed

*prefix* optional prefix string

*postfix* optional postfix string

Definition at line 421 of file vm\_m3math.h.

**6.9.1.2** `template<class T_fp > std::ostream & vm_M3Math< T_fp >::print_on ( std::ostream & os, T_fp const m[ ], char const prefix[ ] = "", char const postfix[ ] = "" ) [inline, static, inherited]`

Print a matrix to an ostream.

**Parameters**

- os* the ostream
- m* matrix to be printed
- prefix* optional prefix string
- postfix* optional postfix string

Definition at line 408 of file vm\_m3math.h.

**6.10 Set vector components****Functions**

- static void [vm\\_V3Math::set](#) (T\_fp v[], T\_fp x, T\_fp y, T\_fp z)
- static void [vm\\_V3Math::set](#) (T\_fp v[], T\_fp x)

**6.10.1 Function Documentation**

**6.10.1.1** `template<class T_fp> void vm_V3Math< T_fp >::set ( T_fp v[],  
T_fp x, T_fp y, T_fp z ) [inline, static, inherited]`

Set components of v to x, y, z.

**Parameters**

- v* vector to be set
- x* x component
- y* y component
- z* z component

Definition at line 255 of file vm\_v3math.h.

Referenced by `vm_V3Math< T_fp >::set()`.

**6.10.1.2** `template<class T_fp> void vm_V3Math< T_fp >::set ( T_fp v[],  
T_fp x ) [inline, static, inherited]`

Set all components of v to x.

**Parameters**

- v* vector to be set
- x* value

Reimplemented from `vm_VMath< T_fp, 3 >`.

Definition at line 260 of file `vm_v3math.h`.

References `vm_V3Math< T_fp >::set()`.

## 6.11 Normalize vectors

### Functions

- static `T_fp vm_V3Math::unitize (T_fp v[ ])`
- static `T_fp vm_V3Math::unitize (T_fp vu[ ], T_fp const vi[ ])`

### 6.11.1 Function Documentation

**6.11.1.1** `template<class T_fp > T_fp vm_V3Math< T_fp >::unitize ( T_fp v[ ] ) [inline, static, inherited]`

Normalize a vector v.

#### Returns

return original vector magnitude

#### Parameters

*v* vector to be normalized

Definition at line 282 of file `vm_v3math.h`.

References `vm_VMath< T_fp, 3 >::div_eq()`, and `vm_V3Math< T_fp >::dot()`.

Referenced by `vm_V3Math< T_fp >::unitize()`.

**6.11.1.2** `template<class T_fp > T_fp vm_V3Math< T_fp >::unitize ( T_fp vu[ ], T_fp const vi[ ] ) [inline, static, inherited]`

Normalize a vector vi returning the normalized value in vu.

#### Returns

return original magnitude of vi

#### Parameters

*vi* vector to be normalized

*vu* normalized version of vi

Definition at line 296 of file vm\_v3math.h.

References `vm_VMath< T_fp, 3 >::copy()`, and `vm_V3Math< T_fp >::unitize()`.

## 6.12 Tests for normality, orthogonality

### Functions

- static int `vm_V3Math::is_unit_vector` (`T_fp` const `v[ ]`, `T_fp` const `tol`)
- static int `vm_V3Math::are_orthogonal` (`T_fp` const `v[ ]`, `T_fp` const `other[ ]`, `T_fp` const `tol`)
- static int `vm_V3Math::are_orthonormal` (`T_fp` const `v[ ]`, `T_fp` const `other[ ]`, `T_fp` const `tol`)

### 6.12.1 Function Documentation

**6.12.1.1** `template<class T_fp> int vm_V3Math< T_fp >::are_orthogonal ( T_fp const v[], T_fp const other[], T_fp const tol ) [inline, static, inherited]`

Test whether a `v` is within tolerance `tol` of orthogonality to other vector

#### Returns

1 if `v` and `other` are orthogonal within tolerance `tol`, 0 otherwise.

#### Parameters

*v* vector to be checked

*other* other vector

*tol* tolerance

Definition at line 311 of file vm\_v3math.h.

References `vm_V3Math< T_fp >::dot()`.

Referenced by `vm_V3Math< T_fp >::are_orthonormal()`.

**6.12.1.2** `template<class T_fp> int vm_V3Math< T_fp >::are_orthonormal ( T_fp const v[], T_fp const other[], T_fp const tol ) [inline, static, inherited]`

Test whether `v` is within tolerance `tol` of orthonormality with other vector



**Returns**

1 if *v* and *other* are orthonormal within tolerance *tol*, 0 otherwise.

**Parameters**

*v* vector to be checked

*other* other vector

*tol* tolerance

Definition at line 320 of file `vm_v3math.h`.

References `vm_V3Math< T_fp >::are_orthogonal()`, and `vm_V3Math< T_fp >::is_unit_vector()`.

```
6.12.1.3 template<class T_fp > int vm_V3Math< T_fp >::is_unit_vector
      ( T_fp const v[], T_fp const tol ) [inline, static,
      inherited]
```

Test whether a vector is within tolerance *tol* of a unit vector

**Returns**

1 if *v* is within tolerance *tol* of being a unit vector, 0 otherwise.

**Parameters**

*v* vector to be checked

*tol* tolerance

Definition at line 304 of file `vm_v3math.h`.

References `vm_V3Math< T_fp >::dot()`.

Referenced by `vm_V3Math< T_fp >::are_orthonormal()`.

**6.13 Dot and Cross products.****Functions**

- static `T_fp vm_V3Math::dot` (`T_fp const v1[]`, `T_fp const v2[]`)
- static void `vm_V3Math::cross` (`T_fp prod[]`, `T_fp const v1[]`, `T_fp const v2[]`)

### 6.13.1 Function Documentation

**6.13.1.1** `template<class T_fp> void vm_V3Math< T_fp >::cross ( T_fp  
prod[], T_fp const v1[], T_fp const v2[] ) [inline, static,  
inherited]`

Cross (wedge) product of two vectors

#### Returns

cross (wedge) product of v1 with v2.

#### Parameters

*prod* cross (wedge) product of v1 with v2.

*v1* 1st vector

*v2* 2nd vector

Definition at line 272 of file vm\_v3math.h.

**6.13.1.2** `template<class T_fp> T_fp vm_V3Math< T_fp >::dot ( T_fp const  
v1[], T_fp const v2[] ) [inline, static, inherited]`

Dot product of two vectors

#### Returns

dot (scalar) product of v1 with v2.

#### Parameters

*v1* 1st vector

*v2* 2nd vector

Definition at line 265 of file vm\_v3math.h.

Referenced by `vm_V3Math< T_fp >::are_orthogonal()`, `vm_V3Math< T_fp >::is_unit_vector()`, and `vm_V3Math< T_fp >::unitize()`.

## 6.14 IO operations.

### Functions

- static `std::ostream & vm_V3Math::print_on (std::ostream &os, T_fp const v[], char const prefix[]="", char const postfix[]="")`
- static `void vm_V3Math::cprint_on (FILE *of, T_fp const v[], char const prefix[]="", char const postfix[]="")`

### 6.14.1 Function Documentation

**6.14.1.1** `template<class T_fp > void vm_V3Math< T_fp >::cprint_on ( FILE  
* of, T_fp const v[], char const prefix[] = "", char const postfix[]  
= "" ) [inline, static, inherited]`

Print a vector to a FILE\* stream.

#### Parameters

*of* the FILE\*  
*v* vector to be printed  
*prefix* optional prefix string  
*postfix* optional postfix string

Definition at line 340 of file vm\_v3math.h.

**6.14.1.2** `template<class T_fp > std::ostream & vm_V3Math< T_fp  
>::print_on ( std::ostream & os, T_fp const v[], char const  
prefix[] = "", char const postfix[] = "" ) [inline, static,  
inherited]`

Print a vector to an ostream.

#### Parameters

*os* the ostream  
*v* vector to be printed  
*prefix* optional prefix string  
*postfix* optional postfix string

Definition at line 331 of file vm\_v3math.h.

## 6.15 Copy routines

### Functions

- static void [vm\\_VMath::copy](#) (T\_fp *v*[], T\_fp const *cv*[])

### 6.15.1 Function Documentation

**6.15.1.1** `template<class T_fp, int N_len> void vm_VMath< T_fp, N_len >::copy ( T_fp v[], T_fp const cv[] ) [inline, static, inherited]`

Normalize a vector v.

#### Parameters

- v* destination vector
- cv* source vector

REQUIREMENT: \*v and \*cv each has a length of at least N\_len contiguous T\_fps and is appropriately aligned for T\_fps.

Definition at line 385 of file vm\_vmath.h.

## 6.16 Set functions

### Functions

- static void [vm\\_VMath::set](#) (T\_fp v[], T\_fp r)

### 6.16.1 Function Documentation

**6.16.1.1** `template<class T_fp, int N_len> void vm_VMath< T_fp, N_len >::set ( T_fp v[], T_fp r ) [inline, static, inherited]`

Set all elements of v to r.

#### Parameters

- v* vector to be set
- r* value

Reimplemented in [vm\\_V3Math< T\\_fp >](#).

Definition at line 393 of file vm\_vmath.h.

## 6.17 Componentwise op\_eq operations (+=, -=, /=)

### Functions

- static void [vm\\_VMath::add\\_eq](#) (T\_fp v[], T\_fp const cv[])

- static void `vm_VMath::sub_eq` (T\_fp v[], T\_fp const cv[])
- static void `vm_VMath::mul_eq` (T\_fp v[], T\_fp const cv[])
- static void `vm_VMath::div_eq` (T\_fp v[], T\_fp const cv[])
- static void `vm_VMath::add_eq` (T\_fp v[], T\_fp r)
- static void `vm_VMath::sub_eq` (T\_fp v[], T\_fp r)
- static void `vm_VMath::mul_eq` (T\_fp v[], T\_fp r)
- static void `vm_VMath::div_eq` (T\_fp v[], T\_fp r)
- static void `vm_VMath::negate` (T\_fp v[])

### 6.17.1 Function Documentation

**6.17.1.1** `template<class T_fp, int N_len> static void vm_VMath< T_fp, N_len >::add_eq ( T_fp v[], T_fp const cv[] ) [inline, static, inherited]`

component-wise  $v[n] += cv[n]$

#### Parameters

- v* LHS vector
- cv* RHS vector

**6.17.1.2** `template<class T_fp, int N_len> static void vm_VMath< T_fp, N_len >::add_eq ( T_fp v[], T_fp r ) [inline, static, inherited]`

component-wise  $v[n] += r$

#### Parameters

- v* LHS vector
- r* RHS T\_fp

**6.17.1.3** `template<class T_fp, int N_len> static void vm_VMath< T_fp, N_len >::div_eq ( T_fp v[], T_fp r ) [inline, static, inherited]`

component-wise  $v[n] /= r$

#### Parameters

- v* LHS vector
- r* RHS T\_fp

**6.17.1.4** `template<class T_fp, int N_len> static void vm_VMath< T_fp, N_len  
>::div_eq ( T_fp v[], T_fp const cv[] ) [inline, static,  
inherited]`

component-wise  $v[n] /= cv[n]$

#### Parameters

*v* LHS vector  
*cv* RHS vector

**6.17.1.5** `template<class T_fp, int N_len> static void vm_VMath< T_fp,  
N_len >::mul_eq ( T_fp v[], T_fp r ) [inline, static,  
inherited]`

component-wise  $v[n] *= r$

#### Parameters

*v* LHS vector  
*r* RHS  $T_{fp}$

**6.17.1.6** `template<class T_fp, int N_len> static void vm_VMath< T_fp, N_len  
>::mul_eq ( T_fp v[], T_fp const cv[] ) [inline, static,  
inherited]`

component-wise  $v[n] *= cv[n]$

#### Parameters

*v* LHS vector  
*cv* RHS vector

**6.17.1.7** `template<class T_fp, int N_len> int N_len void vm_VMath< T_fp,  
N_len >::negate ( T_fp v[] ) [inline, static, inherited]`

component-wise negation of *v*

#### Parameters

*v* vector

Definition at line 414 of file `vm_vmath.h`.

**6.17.1.8** `template<class T_fp, int N_len> static void vm_VMath< T_fp, N_len  
>::sub_eq ( T_fp v[], T_fp const cv[] ) [inline, static,  
inherited]`

component-wise  $v[n] -= cv[n]$

#### Parameters

*v* LHS vector  
*cv* RHS vector

**6.17.1.9** `template<class T_fp, int N_len> static void vm_VMath< T_fp,  
N_len >::sub_eq ( T_fp v[], T_fp r ) [inline, static,  
inherited]`

component-wise  $v[n] -= r$

#### Parameters

*v* LHS vector  
*r* RHS T\_fp

## 6.18 {add,sub,mul,div}: componentwise binary operations.

### Functions

- static void `vm_VMath::add` (T\_fp v[], T\_fp const cv1[], T\_fp const cv2[] )
- static void `vm_VMath::sub` (T\_fp v[], T\_fp const cv1[], T\_fp const cv2[] )
- static void `vm_VMath::mul` (T\_fp v[], T\_fp const cv1[], T\_fp const cv2[] )
- static void `vm_VMath::div` (T\_fp v[], T\_fp const cv1[], T\_fp const cv2[] )
- static void `vm_VMath::add` (T\_fp v[], T\_fp const cv[], T\_fp r)
- static void `vm_VMath::sub` (T\_fp v[], T\_fp const cv[], T\_fp r)
- static void `vm_VMath::mul` (T\_fp v[], T\_fp const cv[], T\_fp r)
- static void `vm_VMath::div` (T\_fp v[], T\_fp const cv[], T\_fp r)
- static void `vm_VMath::add` (T\_fp v[], T\_fp r, T\_fp const cv[] )
- static void `vm_VMath::sub` (T\_fp v[], T\_fp r, T\_fp const cv[] )
- static void `vm_VMath::mul` (T\_fp v[], T\_fp r, T\_fp const cv[] )
- static void `vm_VMath::div` (T\_fp v[], T\_fp r, T\_fp const cv[] )

### 6.18.1 Function Documentation

**6.18.1.1** `template<class T_fp, int N_len> static void vm_VMath< T_fp, N_len >::add ( T_fp v[], T_fp const cv1[], T_fp const cv2[] )`  
`[inline, static, inherited]`

component-wise  $v[n] = cv1[n] + cv2[n]$

#### Parameters

*v* result vector  
*cv1* 1st input vector  
*cv2* 2nd input vector

**6.18.1.2** `template<class T_fp, int N_len> static void vm_VMath< T_fp, N_len >::add ( T_fp v[], T_fp const cv[], T_fp r )` `[inline, static, inherited]`

component-wise  $v[n] = cv[n] + r$

#### Parameters

*v* result vector  
*cv* input vector  
*r* input T\_fp

**6.18.1.3** `template<class T_fp, int N_len> static void vm_VMath< T_fp, N_len >::add ( T_fp v[], T_fp r, T_fp const cv[] )` `[inline, static, inherited]`

component-wise  $v[n] = r + cv[n]$

#### Parameters

*v* result vector  
*r* input T\_fp  
*cv* input vector



**6.18.1.4** `template<class T_fp, int N_len> static void vm_VMath< T_fp, N_len  
>::div ( T_fp v[], T_fp const cv[], T_fp r ) [inline, static,  
inherited]`

component-wise  $v[n] = cv[n] / r$

#### Parameters

*v* result vector  
*cv* input vector  
*r* input T\_fp

**6.18.1.5** `template<class T_fp, int N_len> static void vm_VMath< T_fp,  
N_len >::div ( T_fp v[], T_fp const cv1[], T_fp const cv2[] )  
[inline, static, inherited]`

component-wise  $v[n] = cv1[n] / cv2[n]$

#### Parameters

*v* result vector  
*cv1* 1st input vector  
*cv2* 2nd input vector

**6.18.1.6** `template<class T_fp, int N_len> static void vm_VMath< T_fp, N_len  
>::div ( T_fp v[], T_fp r, T_fp const cv[] ) [inline, static,  
inherited]`

component-wise  $v[n] = r / cv[n]$

#### Parameters

*v* result vector  
*r* input T\_fp  
*cv* input vector

**6.18.1.7** `template<class T_fp, int N_len> static void vm_VMath< T_fp,  
N_len >::mul ( T_fp v[], T_fp const cv[], T_fp r ) [inline,  
static, inherited]`

component-wise  $v[n] = cv[n] * r$

**Parameters**

*v* result vector  
*cv* input vector  
*r* input T\_fp

**6.18.1.8** `template<class T_fp, int N_len> static void vm_VMath< T_fp,  
 N_len >::mul ( T_fp v[], T_fp r, T_fp const cv[] ) [inline,  
 static, inherited]`

component-wise  $v[n] = r * cv[n]$

**Parameters**

*v* result vector  
*r* input T\_fp  
*cv* input vector

**6.18.1.9** `template<class T_fp, int N_len> static void vm_VMath< T_fp,  
 N_len >::mul ( T_fp v[], T_fp const cv1[], T_fp const cv2[] )  
 [inline, static, inherited]`

component-wise  $v[n] = cv1[n] * cv2[n]$

**Parameters**

*v* result vector  
*cv1* 1st input vector  
*cv2* 2nd input vector

**6.18.1.10** `template<class T_fp, int N_len> static void vm_VMath< T_fp, N_len  
 >::sub ( T_fp v[], T_fp const cv[], T_fp r ) [inline,  
 static, inherited]`

component-wise  $v[n] = cv[n] - r$

**Parameters**

*v* result vector  
*cv* input vector  
*r* input T\_fp

**6.18.1.11** `template<class T_fp, int N_len> static void vm_VMath< T_fp, N_len >::sub ( T_fp v[], T_fp r, T_fp const cv[] ) [inline, static, inherited]`

component-wise  $v[n] = r - cv[n]$

#### Parameters

*v* result vector

*r* input T\_fp

*cv* input vector

**6.18.1.12** `template<class T_fp, int N_len> static void vm_VMath< T_fp, N_len >::sub ( T_fp v[], T_fp const cv1[], T_fp const cv2[] ) [inline, static, inherited]`

component-wise  $v[n] = cv1[n] - cv2[n]$

#### Parameters

*v* result vector

*cv1* 1st input vector

*cv2* 2nd input vector

## 6.19 Linear combinations

### Functions

- static void [vm\\_VMath::lincomb](#) (T\_fp res[], T\_fp c1, T\_fp const v1[], T\_fp c2, T\_fp const v2[])

#### 6.19.1 Function Documentation

**6.19.1.1** `template<class T_fp, int N_len> void vm_VMath< T_fp, N_len >::lincomb ( T_fp res[], T_fp c1, T_fp const v1[], T_fp c2, T_fp const v2[] ) [inline, static, inherited]`

form linear combination:  $res = c1 * v1 + c2 * v2$ .

For each *i*:  $res[i] = c1 * v1[i] + c2 * v2[i]$

**Parameters**

*res* result vector  
*c1* 1st T\_fp  
*v1* 1st vector  
*c2* 2nd T\_fp  
*v2* 2nd vector

Definition at line 441 of file vm\_vmath.h.

**6.20 I/O operations.****Functions**

- static std::ostream & [vm\\_VMath::print\\_on](#) (std::ostream &os, T\_fp const v[], int by, char const prefix[]="", char const postfix[]="")
- static void [vm\\_VMath::cprint\\_on](#) (FILE \*of, T\_fp const v[], int by, char const prefix[]="", char const postfix[]="")

**6.20.1 Function Documentation**

**6.20.1.1** `template<class T_fp, int N_len> void vm_VMath< T_fp, N_len >::cprint_on ( FILE * of, T_fp const v[], int by, char const prefix[] = "", char const postfix[] = "" ) [inline, static, inherited]`

Print a vector to a FILE\* stream.

**Parameters**

*of* the FILE\*  
*v* vector to be printed  
*by* stride by this many  
*prefix* optional prefix string  
*postfix* optional postfix string

Definition at line 471 of file vm\_vmath.h.

**6.20.1.2** `template<class T_fp, int N_len> std::ostream & vm_VMath< T_fp, N_len >::print_on ( std::ostream & os, T_fp const v[], int by, char const prefix[] = "", char const postfix[] = "" ) [inline, static, inherited]`

Print a vector to an ostream.

### Parameters

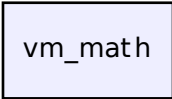
- os* the ostream
- v* vector to be printed
- by* stride by this many
- prefix* optional prefix string
- postfix* optional postfix string

Definition at line 451 of file vm\_vmath.h.

## 7 Directory Documentation

### 7.1 vm\_math/ Directory Reference

Directory dependency graph for vm\_math/:



```
graph TD; vm_math[vm_math];
```

### Files

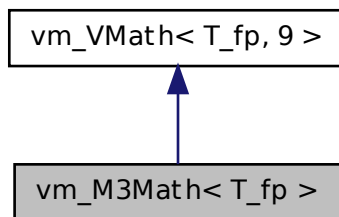
- file **vm\_m3math.cc**
- file **vm\_m3math.h**
- file **vm\_math.h**
- file **vm\_v3math.h**
- file **vm\_vmath.h**

## 8 Class Documentation

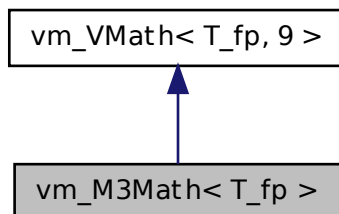
### 8.1 vm\_M3Math< T\_fp > Class Template Reference

```
#include <vm_math/vm_m3math.h>
```

Inheritance diagram for `vm_M3Math< T_fp >`:



Collaboration diagram for `vm_M3Math< T_fp >`:



### Public Types

- typedef `T_fp` **value\_type**

### Static Public Member Functions

- static int `at` (int i, int j)
- static void `init_by_row` (`T_fp` m[], `T_fp` const row0[], `T_fp` const row1[], `T_fp` const row2[])

- static void `init_by_col` (`T_fp m[ ]`, `T_fp const col0[ ]`, `T_fp const col1[ ]`, `T_fp const col2[ ]`)
- static void `dyad_product` (`T_fp m[ ]`, `T_fp const v1[ ]`, `T_fp const v2[ ]`)
- static void `inject_row` (`T_fp m[ ]`, `T_fp const row[ ]`, `int whichrow`)
- static void `inject_col` (`T_fp m[ ]`, `T_fp const col[ ]`, `int whichcol`)
- static void `extract_row` (`T_fp const m[ ]`, `T_fp row[ ]`, `int whichrow`)
- static void `extract_col` (`T_fp const m[ ]`, `T_fp col[ ]`, `int whichcol`)
- static void `mvmult` (`T_fp res[ ]`, `T_fp const m[ ]`, `T_fp const v[ ]`)
- static void `mtvmult` (`T_fp res[ ]`, `T_fp const m[ ]`, `T_fp const v[ ]`)
- static void `mmult` (`T_fp mres[ ]`, `T_fp const m1[ ]`, `T_fp const m2[ ]`)
- static `std::ostream & print_on` (`std::ostream &os`, `T_fp const m[ ]`, `char const prefix[ ]=""`, `char const postfix[ ]=""`)
- static void `cprint_on` (`FILE *of`, `T_fp const m[ ]`, `char const prefix[ ]=""`, `char const postfix[ ]=""`)

### 8.1.1 Detailed Description

**template<class T\_fp> class `vm_M3Math< T_fp >`**

A template class providing common numerical operations on 3x3-matrices of `T_fp`'s (floating point type). The matrix is assumed to be stored as a \* contiguous one-dimensional array of 9 `T_fp`'s, properly aligned for type `T_fp`.

The class is a simple class to handle common numerical operations on 3x3-matrices of floating point `T_fps`.

Unless otherwise noted, the operations are component by component, e.g.,

```
mul_eq(m1,m2)
```

results in

```
m1[i][j] += m2[i][j], where i = 0,1,2 and j = 0,1,2.
```

`vm_M3Math` has only static member functions; there are no data members.

Where possible, the static member functions are inlined.

Definition at line 77 of file `vm_m3math.h`.

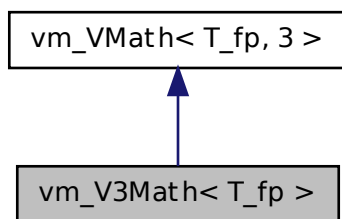
The documentation for this class was generated from the following file:

- `vm_m3math.h`

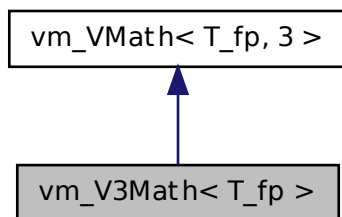
## 8.2 vm\_V3Math< T\_fp > Class Template Reference

```
#include <vm_math/vm_v3math.h>
```

Inheritance diagram for vm\_V3Math< T\_fp >:



Collaboration diagram for vm\_V3Math< T\_fp >:



### Public Types

- typedef T\_fp [value\\_type](#)



### Static Public Member Functions

- static void `set` (`T_fp` v[ ], `T_fp` x, `T_fp` y, `T_fp` z)
- static void `set` (`T_fp` v[ ], `T_fp` x)
- static `T_fp` `unitize` (`T_fp` v[ ])
- static `T_fp` `unitize` (`T_fp` vu[ ], `T_fp` const vi[ ])
- static int `is_unit_vector` (`T_fp` const v[ ], `T_fp` const tol)
- static int `are_orthogonal` (`T_fp` const v[ ], `T_fp` const other[ ], `T_fp` const tol)
- static int `are_orthonormal` (`T_fp` const v[ ], `T_fp` const other[ ], `T_fp` const tol)
- static `T_fp` `dot` (`T_fp` const v1[ ], `T_fp` const v2[ ])
- static void `cross` (`T_fp` prod[ ], `T_fp` const v1[ ], `T_fp` const v2[ ])
- static std::ostream & `print_on` (std::ostream &os, `T_fp` const v[ ], char const prefix[ ]="", char const postfix[ ]="")
- static void `cprint_on` (FILE \*of, `T_fp` const v[ ], char const prefix[ ]="", char const postfix[ ]="")

#### 8.2.1 Detailed Description

**template<class `T_fp`> class `vm_V3Math< T_fp >`**

A template class providing common numerical operations on 3-vectors of `T_fp`'s (floating point type).

Unless otherwise noted, the operations are component by component, e.g.,

```
vm_V3Mathlt<float>::div_eq(v1, v2)
```

corresponds to

```
v1[i] /= v2[i], where i = 0,1,2.
```

`vm_V3Math` has only static member functions; there are no data members.

Where possible, the static member functions are inlined.

Definition at line 69 of file `vm_v3math.h`.

#### 8.2.2 Member Typedef Documentation

##### 8.2.2.1 `template<class T_fp> vm_V3Math< T_fp >::value_type`

A typedef for the floating point type;

Reimplemented from `vm_VMath< T_fp, 3 >`.

Definition at line 83 of file `vm_v3math.h`.

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

- `vm_v3math.h`

### 8.3 `vm_VMath< T_fp, N_len >` Class Template Reference

```
#include <vm_math/vm_vmath.h>
```

#### Public Types

- `typedef T_fp value_type`

#### Static Public Member Functions

- static void `copy` (`T_fp v[ ]`, `T_fp const cv[ ]`)
- static void `set` (`T_fp v[ ]`, `T_fp r`)
- static void `add_eq` (`T_fp v[ ]`, `T_fp const cv[ ]`)
- static void `sub_eq` (`T_fp v[ ]`, `T_fp const cv[ ]`)
- static void `mul_eq` (`T_fp v[ ]`, `T_fp const cv[ ]`)
- static void `div_eq` (`T_fp v[ ]`, `T_fp const cv[ ]`)
- static void `add_eq` (`T_fp v[ ]`, `T_fp r`)
- static void `sub_eq` (`T_fp v[ ]`, `T_fp r`)
- static void `mul_eq` (`T_fp v[ ]`, `T_fp r`)
- static void `div_eq` (`T_fp v[ ]`, `T_fp r`)
- static void `negate` (`T_fp v[ ]`)
- static void `add` (`T_fp v[ ]`, `T_fp const cv1[ ]`, `T_fp const cv2[ ]`)
- static void `sub` (`T_fp v[ ]`, `T_fp const cv1[ ]`, `T_fp const cv2[ ]`)
- static void `mul` (`T_fp v[ ]`, `T_fp const cv1[ ]`, `T_fp const cv2[ ]`)
- static void `div` (`T_fp v[ ]`, `T_fp const cv1[ ]`, `T_fp const cv2[ ]`)
- static void `add` (`T_fp v[ ]`, `T_fp const cv[ ]`, `T_fp r`)
- static void `sub` (`T_fp v[ ]`, `T_fp const cv[ ]`, `T_fp r`)
- static void `mul` (`T_fp v[ ]`, `T_fp const cv[ ]`, `T_fp r`)
- static void `div` (`T_fp v[ ]`, `T_fp const cv[ ]`, `T_fp r`)
- static void `add` (`T_fp v[ ]`, `T_fp r`, `T_fp const cv[ ]`)
- static void `sub` (`T_fp v[ ]`, `T_fp r`, `T_fp const cv[ ]`)
- static void `mul` (`T_fp v[ ]`, `T_fp r`, `T_fp const cv[ ]`)
- static void `div` (`T_fp v[ ]`, `T_fp r`, `T_fp const cv[ ]`)
- static void `lincomb` (`T_fp res[ ]`, `T_fp c1`, `T_fp const v1[ ]`, `T_fp c2`, `T_fp const v2[ ]`)

- static `std::ostream & print_on` (`std::ostream &os`, `T_fp const v[ ]`, `int by`, `char const prefix[ ]=""`, `char const postfix[ ]=""`)
- static void `cprint_on` (`FILE *of`, `T_fp const v[ ]`, `int by`, `char const prefix[ ]=""`, `char const postfix[ ]=""`)

### 8.3.1 Detailed Description

**template<class T\_fp, int N\_len> class vm\_VMath< T\_fp, N\_len >**

A template class providing common numerical operations on `N_len`-long 1 dimensional arrays of `T_fp`.

`T_fp` is a floating point type.

`N_len` is the length of the vector.

The array data are assumed to be stored as a contiguous one-dimensional array of `N_len` `T_fp`'s, properly aligned for type `T_fp`.

Unless otherwise noted, the operations are component by component, e.g.,

```
vm_VMath<float,4>::mul(prod, v1, v2)
```

corresponds to

```
prod[i] = v1[i] * v2[i], where i = 0,1,2,3.
```

`vm_VMath` has only static member functions; there are no data members.

Where possible, the static member functions are inlined.

Definition at line 78 of file `vm_vmath.h`.

### 8.3.2 Member Typedef Documentation

**8.3.2.1 template<class T\_fp, int N\_len> vm\_VMath< T\_fp, N\_len >::value\_type**

a typedef for the floating point type;

Reimplemented in `vm_V3Math< T_fp >`.

Definition at line 91 of file `vm_vmath.h`.

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

- `vm_vmath.h`

## Index

- {add,sub,mul,div}: componentwise binary operations., [20](#)
- add
  - ndvec\_mathop, [21](#)
- add\_eq
  - ndvec\_opeq, [18](#)
- are\_orthogonal
  - vec\_test\_norm, [13](#)
- are\_orthonormal
  - vec\_test\_norm, [13](#)
- at
  - mat\_index\_calculations, [5](#)
- Componentwise op\_eq operations (+, -, =, !=, /=), [17](#)
- copy
  - ndvec\_copy, [16](#)
- Copy routines, [16](#)
- cprint\_on
  - mat\_io, [10](#)
  - ndvec\_io, [25](#)
  - vec\_io, [15](#)
- cross
  - vec\_dot\_cross, [14](#)
- div
  - ndvec\_mathop, [21](#), [22](#)
- div\_eq
  - ndvec\_opeq, [18](#)
- dot
  - vec\_dot\_cross, [15](#)
- Dot and Cross products., [14](#)
- dyad\_product
  - mat\_init, [6](#)
- extract\_col
  - mat\_insert\_extract, [7](#)
- extract\_row
  - mat\_insert\_extract, [7](#)
- For a matrix, insert or extract a vector., [7](#)
- I/O operations., [25](#)
- Index calculations, [5](#)
- init\_by\_col
  - mat\_init, [6](#)
- init\_by\_row
  - mat\_init, [6](#)
- Initialize a matrix., [6](#)
- inject\_col
  - mat\_insert\_extract, [8](#)
- inject\_row
  - mat\_insert\_extract, [8](#)
- IO operations., [10](#), [15](#)
- is\_unit\_vector
  - vec\_test\_norm, [14](#)
- lincomb
  - ndvec\_lincomb, [24](#)
- Linear combinations, [24](#)
- mat\_index\_calculations
  - at, [5](#)
- mat\_init
  - dyad\_product, [6](#)
  - init\_by\_col, [6](#)
  - init\_by\_row, [6](#)
- mat\_insert\_extract
  - extract\_col, [7](#)
  - extract\_row, [7](#)
  - inject\_col, [8](#)
  - inject\_row, [8](#)
- mat\_io
  - cprint\_on, [10](#)
  - print\_on, [10](#)
- mat\_vec\_ops
  - mtvmult, [9](#)
  - mvmult, [9](#)
- Matrix Matrix operations., [10](#)
- Matrix Vector operations., [9](#)
- mtvmult
  - mat\_vec\_ops, [9](#)
- mul
  - ndvec\_mathop, [22](#), [23](#)
- mul\_eq
  - ndvec\_opeq, [19](#)

- mvmult
  - mat\_vec\_ops, 9
- ndvec\_copy
  - copy, 16
- ndvec\_io
  - cprint\_on, 25
  - print\_on, 25
- ndvec\_lincomb
  - lincomb, 24
- ndvec\_mathop
  - add, 21
  - div, 21, 22
  - mul, 22, 23
  - sub, 23, 24
- ndvec\_opeq
  - add\_eq, 18
  - div\_eq, 18
  - mul\_eq, 19
  - negate, 19
  - sub\_eq, 19, 20
- ndvec\_setfuncs
  - set, 17
- negate
  - ndvec\_opeq, 19
- Normalize vectors, 12
- print\_on
  - mat\_io, 10
  - ndvec\_io, 25
  - vec\_io, 16
- set
  - ndvec\_setfuncs, 17
  - vec\_set\_comp, 11
- Set functions, 17
- Set vector components, 11
- sub
  - ndvec\_mathop, 23, 24
- sub\_eq
  - ndvec\_opeq, 19, 20
- Tests for normality, orthogonality, 13
- unitize
  - vec\_norm, 12
- value\_type
  - vm\_V3Math, 30
  - vm\_VMath, 32
- vec\_dot\_cross
  - cross, 14
  - dot, 15
- vec\_io
  - cprint\_on, 15
  - print\_on, 16
- vec\_norm
  - unitize, 12
- vec\_set\_comp
  - set, 11
- vec\_test\_norm
  - are\_orthogonal, 13
  - are\_orthonormal, 13
  - is\_unit\_vector, 14
- vm\_M3Math, 26
- vm\_M3Math common numerical operations, 5
- vm\_math/ Directory Reference, 26
- vm\_V3Math, 29
  - value\_type, 30
- vm\_V3Math common numerical operations, 5
- vm\_VMath, 31
  - value\_type, 32
- vm\_VMath common numerical operations, 5