

# matvec Reference Manual

1.1.1

Generated by Doxygen 1.4.7

Wed Oct 28 12:40:32 2009



# Contents

|          |                                   |          |
|----------|-----------------------------------|----------|
| <b>1</b> | <b>matvec</b>                     | <b>1</b> |
| 1.1      | Copyright . . . . .               | 1        |
| 1.2      | Usage . . . . .                   | 2        |
| <b>2</b> | <b>matvec File Documentation</b>  | <b>5</b> |
| 2.1      | macros.h File Reference . . . . . | 5        |



# Chapter 1

## matvec

The `matvec` package provides C/C++ preprocessor macros which perform matrix and vector operations in three dimensional space. There is support for 4-dimensional "homogeneous coordinates" which provide efficient combined rotation and translation operations.

The underlying macros are independent of the storage implementation. The package provides extractors to handle the most common storage implementations. Custom extractors are simple to construct.

### 1.1 Copyright

Copyright (C) 2006 Smithsonian Astrophysical Observatory

`matvec` 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.

`matvec` 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 Usage

### 1.2.1 Accessing the macros

To access the macros from code, ensure that the header file is included in the compilation unit:

```
#include <matvec/macros.h>
```

### 1.2.2 Matrix and Vector Storage

Matrices may be stored either as flat arrays or as C-style multi-dimensional arrays. Vectors may be flat arrays or structures. The underlying routines which perform the translations and rotations are fed the complete list of matrix and vector elements; they know nothing about how they are stored.

To isolate the caller from that interface, macros are provided which extract the elements from a particular storage type. These macros should be used whenever a macro parameter is listed as an "extractor". The list of extractors is available [here](#).

For example, here's how to rotate and translate a vector stored in a structure using a matrix stored in a flat array:

```
#define MATRIX(mat_arr) MATVEC_MAT3X3_AS_ARRAY(matrix)
#define VECTOR(vector) MATVEC_VEC_AS_STRUCT(vector,x,y,z)

double matrix[9];
struct { double x, y, z } va, ta, ra;
MATVEC_VEC_ROTATE_TRANSLATE(MATRIX(mat_arr),
                             VECTOR(va),
                             VECTOR(ta),
                             VECTOR(ra));
```

(The definitions of `MATRIX` and `VECTOR` are not necessary; they just clean up the code a bit.) Note that this interface allows mixing different vector implementations if necessary.

### 1.2.3 Implementation Subtleties

All of the facilities provided by this package are C/C++ preprocessor macros. Most of the error checking is provided by the compiler. However, there are some subtleties:

- Some macro arguments are used more than once in the macro. If the expression passed as the arguments has side effects (such as an autoincrement operation, or a function call) then those effects will be replicated each time the argument is used in the macro. It is best to be safe and pass only static expressions as arguments.

- Most macros don't care about the type used to store the data (float, double, whatever). However, some macros allocate temporary storage, and thus must know the type.

### 1.2.4 Rotation Matrix

A rotation matrix may represent either the rotation of a point within a coordinate frame, or the rotation of the coordinate frame. The convention used will dictate the interpretation of the result of applying the matrix to a vector – it does not change how the rotation matrix is applied. Thus, the routines which populate a rotation matrix will follow a convention, but the routine which applies the matrix to a vector will work for either convention.

In this package the rotation matrix constructors create matrices which represent rotations of coordinate frames.





## Chapter 2

# matvec File Documentation

### 2.1 macros.h File Reference

#### 2.1.1 Detailed Description

Macros dealing with transformations.

These macros provide operations on matrices, transformation matrices, as well as vector transformations (rotations and translations).

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

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

Include dependency graph for macros.h:

#### Matrix and Vector Element Extractors

These macros extract elements from matrices and vectors. They should be used wherever a macro parameter specifies an "extractor".

- `#define MATVEC_MAT3X3_AS_MDARRAY(matrix)`  
*Extract the elements of the upper left 3x3 (sub)matrix from a two-dimensional C array.*
- `#define MATVEC_MAT4X4_AS_MDARRAY(matrix)`  
*Extract the elements of the upper left 4x4 (sub)matrix from a two-dimensional C array.*
- `#define MATVEC_MAT3X3_AS_ARRAY(matrix, ncols)`  
*Extract elements for the upper left 3x3 (sub)matrix from a matrix represented as a flat C array.*

- #define `MATVEC_MAT4X4_AS_ARRAY`(matrix, ncols)  
*Extract elements for the upper left 4x4 matrix from a matrix represented as a flat C array.*
- #define `MATVEC_VEC_AS_ARRAY`(vector, stride)  
*Extract components for a 3 component vector from an array.*
- #define `MATVEC_VEC_AS_STRUCT`(vector, x, y, z)  
*Extract components for a 3 component vector from a structure.*

## Basic Matrix Operations

- #define `MATVEC_MAT3X3_ASSIGN`(dst, src)
- #define `MATVEC_MAT4X4_ASSIGN`(dst, src)
- #define `MATVEC_MAT3X3_SET`(matrix, value)
- #define `MATVEC_MAT4X4_SET`(matrix, value)
- #define `MATVEC_MAT3X3_IDENTITY`(matrix)
- #define `MATVEC_MAT4X4_IDENTITY`(matrix)
- #define `MATVEC_MAT3X3_TRANSPOSE`(res, src)
- #define `MATVEC_MAT4X4_TRANSPOSE`(res, src)
- #define `MATVEC_MAT3X3_MULT`(res, a, b)
- #define `MATVEC_MAT4X4_MULT`(res, a, b)

## Transformation Matrix Operations

- #define `MATVEC_MAT3X3_ROT_X`(matrix, theta)
- #define `MATVEC_MAT3X3_ROT_Y`(matrix, theta)
- #define `MATVEC_MAT3X3_ROT_Z`(matrix, theta)
- #define `MATVEC_MAT3X3_ROT_YMXZ`(matrix, azimuth, elevation, clock)
- #define `MATVEC_MAT3X3_ROT_ZXZ`(matrix, phi, theta, psi)

## Basic Vector Operations

- #define `MATVEC_VEC_ASSIGN`(dst, src)
- #define `MATVEC_VEC_SCALE`(res, vin, fac)
- #define `MATVEC_VEC_ADD`(sum, vca, vecb)
- #define `MATVEC_VEC_SUB`(res, vca, vecb)
- #define `MATVEC_VEC_PROJECT`(res, pos, dir, s)
- #define `MATVEC_VEC_DOTPROD`(v, w)
- #define `MATVEC_VEC_CROSSPROD`(res, v, w)
- #define `MATVEC_VEC_NORM`(vector)

## Vector Transformations

These macros operate on vectors. Depending upon the convention used when constructing the rotation matrices and translation vectors, these operations will either transform the coordinate frames of the vectors and specify the vector in the new coordinate frame, or will transform the vector within the existing coordinate frame. Be sure to understand which convention was used, and don't mix 'em up!

- `#define MATVEC_VEC_ROTATE_TRANSLATE(res, vin, matrix, vtrans)`  
*Apply a rotation followed by a translation to a vector.*
- `#define MATVEC_VEC_TRANSLATE_ROTATE(type, vout, vin, vtrans, matrix)`  
*Apply a translation followed by a rotation to a vector.*

### 2.1.2 Define Documentation

#### 2.1.2.1 `#define MATVEC_MAT3X3_AS_ARRAY(matrix, ncols)`

Extract elements for the upper left 3x3 (sub)matrix from a matrix represented as a flat C array.

##### Parameters:

- matrix*** the flat (one-dimensional) array containing the matrix
- ncols*** the number of columns in the matrix. this parameter is used multiple times, so should not cause any side effects

The macro extracts the upper left 3x3 matrix from a two dimensional matrix represented by a flat (one-dimensional array). The input matrix may be larger than 3x3.

`matrix` should resolve into an array specification which can be indexed using the standard C bracket operators. For example:

```
double matrix[3*3];
MATVEC_MAT3X3_AS_ARRAY(matrix, 3)

double matrix[4*4];
MATVEC_MAT3X3_AS_ARRAY(matrix, 4)

void func( double *matrixa, double matrixb[] )
{
    MATVEC_MAT3X3_AS_ARRAY(matrixa, 3)
    MATVEC_MAT3X3_AS_ARRAY(matrixb, 3)
}
```

Definition at line 291 of file macros.h.

### 2.1.2.2 **#define MATVEC\_MAT3X3\_AS\_MDARRAY(matrix)**

Extract the elements of the upper left 3x3 (sub)matrix from a two-dimensional C array.

**Parameters:**

*matrix* the variable containing the matrix

*matrix* should resolve into an array specification which can be indexed using the standard C bracket operators. For example:

```
double matrix[3][3];
MATVEC_MAT3X3_AS_MDARRAY(matrix)

void func( double matrix[][3] )
{
    MATVEC_MAT3X3_AS_MDARRAY(matrix)
}
```

Definition at line 226 of file macros.h.

### 2.1.2.3 **#define MATVEC\_MAT3X3\_ASSIGN(dst, src)**

Assign a 3X3 upper-left (sub)matrix

**Parameters:**

*dst* the destination matrix extractor

*src* the source matrix extractor

Assign a 3X3 matrix to another

Example:

```
double mata[3][3];
double *matb = malloc((3*3) * sizeof(double));

MATVEC_MAT3X3_ASSIGN(
    MATVEC_MAT3X3_AS_ARRAY(matb),
    MATVEC_MAT3X3_AS_MDARRAY(mata)
);
```

Definition at line 481 of file macros.h.

### 2.1.2.4 **#define MATVEC\_MAT3X3\_IDENTITY(matrix)**

Set a 3X3 upper-left (sub)matrix to the identity matrix

**Parameters:**

*matrix* the (sub)matrix extractor

Definition at line 554 of file macros.h.

**2.1.2.5 #define MATVEC\_MAT3X3\_MULT(res, a, b)**

Multiply two 3X3 Upper-left (sub)matrices

**Parameters:**

*res* the result (sub)matrix extractor.

*a* the first (sub)matrix extractor

*b* the second (sub)matrix extractor.

Multiply two 3X3 upper-left (sub)matrices, storing the result in another matrix.

Definition at line 782 of file macros.h.

**2.1.2.6 #define MATVEC\_MAT3X3\_ROT\_X(matrix, theta)**

Set a 3X3 matrix to a rotation matrix representing rotation of the *coordinate frame* about the X axis

**Parameters:**

*matrix* result 3x3 (sub)matrix extractor

*theta* rotation angle in radians

For positive  $\theta$ , a rotation about X carries the Y axis toward the Z axis.

$$\text{rotX}(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \theta & \sin \theta \\ 0 & -\sin \theta & \cos \theta \end{bmatrix}$$

Definition at line 832 of file macros.h.

**2.1.2.7 #define MATVEC\_MAT3X3\_ROT\_Y(matrix, theta)**

Set a 3X3 matrix to a rotation matrix representing rotation of the *coordinate frame* about the Y axis

**Parameters:*****matrix*** result 3x3 (sub)matrix extractor***theta*** rotation angle in radians

For positive  $\theta$ , a rotation about Y carries the Z axis toward the X axis.

$$\text{rotY}(\theta) = \begin{bmatrix} \cos \theta & 0 & -\sin \theta \\ 0 & 1 & 0 \\ \sin \theta & 0 & \cos \theta \end{bmatrix}$$

Definition at line 864 of file macros.h.

### 2.1.2.8 #define MATVEC\_MAT3X3\_ROT\_YMXZ(matrix, azimuth, elevation, clock)

Set a 3x3 (sub)matrix to a rotation matrix representing rotation of the *coordinate system* through a modified YXZ Euler angle rotation sequence (YMXZ stands for Y, *minus* X, Z).

This system can be thought of as representing an alt-az system with the horizon in the X-Z plane. The system differs from the standard Euler YXZ system in that the sense of the rotation about the X axis is reversed (Z rotates towards Y) to match the common definition of altitude.

**Parameters:*****matrix*** result 3x3 (sub)matrix extractor***azimuth*** The azimuth angle (in radians). This is the rotation angle about the current Y axis. Positive azimuth rotates Z towards X.***elevation*** The elevation angle (in radians). This is the rotation angle about the new X axis. Positive elevation rotates Z towards Y, which is opposite from the standard convention.***clock*** The clock angle (in radians). This is the rotation angle about the new Z axis. Positive clocking rotates X towards Y.

The new coordinate frame is generated via the equivalent of

- rotating the input frame about its Y axis by `azimuth`
- rotating the new frame about its X axis by `-elevation`
- rotating the new frame about its Z axis by `clock`

Let  $\alpha$  be the azimuth angle,  $\lambda$  be the elevation angle, and  $\zeta$  the clocking angle about the Z axis. Then, in detail  $\mathbf{A} = \mathbf{BCD}$  where

$$\begin{aligned}\mathbf{D} &= \begin{bmatrix} \cos \alpha & 0 & -\sin \alpha \\ 0 & 1 & 0 \\ \sin \alpha & 0 & \cos \alpha \end{bmatrix} \\ \mathbf{C} &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \lambda & -\sin \lambda \\ 0 & \sin \lambda & \cos \lambda \end{bmatrix} \\ \mathbf{B} &= \begin{bmatrix} \cos \zeta & \sin \zeta & 0 \\ -\sin \zeta & \cos \zeta & 0 \\ 0 & 0 & 1 \end{bmatrix}\end{aligned}$$

so that

$$\mathbf{A} = \begin{bmatrix} \cos \zeta \cos \alpha - \sin \zeta \sin \lambda \sin \alpha & \sin \zeta \cos \lambda & -\cos \zeta \sin \alpha - \sin \zeta \sin \lambda \cos \alpha \\ -\sin \zeta \cos \alpha - \cos \zeta \sin \lambda \cos \alpha & \cos \zeta \cos \lambda & \sin \zeta \sin \alpha - \cos \zeta \sin \lambda \cos \alpha \\ \cos \lambda \sin \alpha & \sin \lambda & \cos \lambda \cos \alpha \end{bmatrix}$$

In terms of the MATVEC rotation macros, this would be  $\mathbf{A} = \mathbf{BCD}$  where:

```
MATVEC_MAT3X3_ROT_Y(D, azimuth);
MATVEC_MAT3X3_ROT_X(C, -elevation);
MATVEC_MAT3X3_ROT_Z(B, clocking);
```

Definition at line 1003 of file macros.h.

### 2.1.2.9 #define MATVEC\_MAT3X3\_ROT\_Z(matrix, theta)

Set a 3X3 matrix to a rotation matrix representing rotation of the *coordinate frame* about the Z axis

#### Parameters:

**matrix** result 3x3 (sub)matrix extractor

**theta** rotation angle in radians

For positive  $\theta$ , a rotation about Z carries the X axis toward the Y axis.

$$\text{rotZ}(\theta) = \begin{bmatrix} \cos \theta & \sin \theta & 0 \\ -\sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Definition at line 896 of file macros.h.

**2.1.2.10 #define MATVEC\_MAT3X3\_ROT\_ZXZ(matrix, phi, theta, psi)**

Set a 3x3 (sub)matrix to a rotation matrix representing rotation of the *coordinate system* through a ZXZ Euler angle rotation sequence.

**Parameters:**

*matrix* result 3x3 (sub)matrix extractor

*phi* The 1st Euler angle (in radians). This is a rotation about the current Z axis. Positive phi takes Y towards X.

*theta* The 2nd Euler angle (in radians) This is a rotation about the current X axis. Positive theta takes Z towards Y.

*psi* The 3rd Euler angle (in radians) This is a rotation about the current Z axis(again). Positive psi takes Y towards X.

See also [mathworld](#)

Definition at line 1055 of file macros.h.

**2.1.2.11 #define MATVEC\_MAT3X3\_SET(matrix, value)**

Set a 3X3 upper-left (sub)matrix to a given value.

**Parameters:**

*matrix* the (sub)matrix extractor

*value* the value to which the matrix entries are to be set. this is used multiple times, so should not have any side-effects.

Definition at line 518 of file macros.h.

**2.1.2.12 #define MATVEC\_MAT3X3\_TRANSPOSE(res, src)**

Transpose a 3X3 upper-left (sub)matrix

**Parameters:**

*res* the result (sub)matrix extractor.

*src* the input (sub)matrix extractor

Transpose a 3X3 upper-left (sub)matrix, storing the results in another matrix.

Definition at line 666 of file macros.h.



### 2.1.2.13 #define MATVEC\_MAT4X4\_AS\_ARRAY(matrix, ncols)

Extract elements for the upper left 4x4 matrix from a matrix represented as a flat C array.

**Parameters:**

- matrix* the flat (one-dimensional) array containing the matrix
- ncols* the number of columns in the matrix. this parameter is used multiple times, so should not cause any side effects

The macro extracts the upper left 4x4 matrix from a two dimensional matrix represented by a flat (one-dimensional array). The input matrix may be larger than 4x4.

*matrix* should resolve into an array specification which can be indexed using the standard C bracket operators. For example:

```
double matrixa[4*4];
MATVEC_MAT4X4_AS_ARRAY(matrixa, 4)

double matrixb[5*5];
MATVEC_MAT4X4_AS_ARRAY(matrixb, 5)

void func( double *matrixa, double matrixb[] )
{
    MATVEC_MAT4X4_AS_ARRAY(matrixa, 4)
    MATVEC_MAT4X4_AS_ARRAY(matrixb, 5)
}
```

Definition at line 329 of file macros.h.

### 2.1.2.14 #define MATVEC\_MAT4X4\_AS\_MDARRAY(matrix)

Extract the elements of the upper left 4x4 (sub)matrix from a two-dimensional C array.

**Parameters:**

- matrix* the variable containing the matrix

*matrix* should resolve into an array specification which can be indexed using the standard C bracket operators. For example:

```
double matrix[4][4];
MATVEC_MAT4X4_AS_MDARRAY(matrix)

void func( double matrix[][4] )
{
    MATVEC_MAT4X4_AS_MDARRAY(matrix)
}
```

Definition at line 251 of file macros.h.

**2.1.2.15 #define MATVEC\_MAT4X4\_ASSIGN(dst, src)**

Assign a 4X4 upper-left (sub)matrix

**Parameters:**

*dst* the destination matrix extractor

*src* the source matrix extractor

Example:

```
double mata[4][4];
double *matb = malloc((4*4) * sizeof(double));

MATVEC_MAT4X4_COPY (
    MATVEC_MAT4X4_AS_ARRAY (matb),
    MATVEC_MAT4X4_AS_MDARRAY (mata) );
```

Definition at line 503 of file macros.h.

**2.1.2.16 #define MATVEC\_MAT4X4\_IDENTITY(matrix)**

Set a 4X4 upper-left (sub)matrix to the identity matrix

**Parameters:**

*matrix* the (sub)matrix extractor

Definition at line 570 of file macros.h.

**2.1.2.17 #define MATVEC\_MAT4X4\_MULT(res, a, b)**

Multiply two 4X4 Upper-left (sub)matrices

**Parameters:**

*res* the output (sub)matrix extractor.

*a* the first (sub)matrix extractor

*b* the second (sub)matrix extractor.

Multiply two 4X4 upper-left (sub)matrices, storing the result in another matrix.

Definition at line 797 of file macros.h.

### 2.1.2.18 #define MATVEC\_MAT4X4\_SET(matrix, value)

Set a 4X4 upper-left (sub)matrix to a given value.

**Parameters:**

*matrix* the (sub)matrix extractor

*value* the value to which the matrix entries are to be set. this is used multiple times, so should not have any side-effects.

Definition at line 537 of file macros.h.

### 2.1.2.19 #define MATVEC\_MAT4X4\_TRANSPOSE(res, src)

Transpose a 4X4 upper-left (sub)matrix

**Parameters:**

*res* the result (sub)matrix extractor

*dst* the output (sub)matrix extractor.

Transpose a 4X4 upper-left (sub)matrix, storing the results in another matrix.

Definition at line 680 of file macros.h.

### 2.1.2.20 #define MATVEC\_VEC\_ADD(sum, veca, vecb)

Add two vectors and store the result in another vector. The result vector may be one of the input vectors

**Parameters:**

*sum* the result vector extractor

*veca* the first vector extractor

*vecb* the second vector extractor

Definition at line 1229 of file macros.h.

### 2.1.2.21 #define MATVEC\_VEC\_AS\_ARRAY(vector, stride)

Extract components for a 3 component vector from an array.

**Parameters:**

*vector* a pointer to the first element in the vector

*stride* the number of array elements between vector elements. for typical uses this is 1.

Definition at line 347 of file macros.h.

#### 2.1.2.22 **#define MATVEC\_VEC\_AS\_STRUCT(vector, x, y, z)**

Extract components for a 3 component vector from a structure.

##### **Parameters:**

- vector* the structure containing the vector. this is **not** a pointer to the structure. for pointers, pass *\*ptr*
- x* the name of the x component structure element
- y* the name of the y component structure element
- z* the name of the z component structure element

For example,

```
struct { double x, y, z } vector, *vecptr;  
MATVEC_VEC_AS_STRUCT(vector, x, y, z)  
MATVEC_VEC_AS_STRUCT(*vecptr, x, y, z)
```

Definition at line 370 of file macros.h.

#### 2.1.2.23 **#define MATVEC\_VEC\_ASSIGN(dst, src)**

Assign a vector to another.

##### **Parameters:**

- dst* the destination vector extractor
- src* the source vector extractor

Definition at line 1128 of file macros.h.

#### 2.1.2.24 **#define MATVEC\_VEC\_CROSSPROD(res, v, w)**

This macro returns the dot product of two vectors.

##### **Parameters:**

- res* result vector extractor

*v* 1st vector extractor  
*w* 2nd vector extractor

Definition at line 1402 of file macros.h.

#### 2.1.2.25 **#define MATVEC\_VEC\_DOTPROD(v, w)**

This macro returns the dot product of two vectors.

##### **Parameters:**

*v* 1st vector extractor  
*w* 2nd vector extractor

Definition at line 1360 of file macros.h.

#### 2.1.2.26 **#define MATVEC\_VEC\_NORM(vector)**

Normalize a vector

##### **Parameters:**

*vector* the vector extractor

Definition at line 1435 of file macros.h.

#### 2.1.2.27 **#define MATVEC\_VEC\_PROJECT(res, pos, dir, s)**

Project a point along a vector.

##### **Parameters:**

*res* the result vector extractor  
*pos* the point position vector extractor  
*dir* the direction vector extractor  
*s* the fractional distance along the direction vector to project.

The result vector may be the same as the position vector.

Definition at line 1330 of file macros.h.

### 2.1.2.28 `#define MATVEC_VEC_ROTATE_TRANSLATE(res, vin, matrix, vtrans)`

Apply a rotation followed by a translation to a vector.

#### Parameters:

*res* the result vector extractor  
*vin* the initial vector extractor  
*matrix* the rotation matrix extractor  
*vtrans* the translation vector extractor

The parameters may be used more than once, so ensure that if they are expressions that there are no side effects.

Example:

The convention is to use homogenous coordinates and the transformation is stored as a 4x4 C multidimensional array. In this case, the rotation matrix is stored in the upper 3x3 submatrix and the translation vector is stored in the right-most column. The input and output vectors are stored in 3 element C arrays.

```
double transform[4][4];
double vin[3], res[3];

MATVEC_VEC_ROTATE_TRANSLATE (
    MATVEC_VEC_AS_ARRAY (res, 1),
    MATVEC_VEC_AS_ARRAY (vin, 1),
    MATVEC_MAT3X3_AS_MDARRAY (transform),
    MATVEC_VEC_AS_ARRAY (&transform[0][3], 4)
);
```

Example:

The rotation matrix is stored in a 3x3 C multidimensional array and the translation vector is stored in a 3 element C array. The input and output vectors are stored in 3 element C arrays.

```
double transform[3][3];
double vtrans[3], vin[3], vout[3];

MATVEC_VEC_ROTATE_TRANSLATE (
    MATVEC_VEC_AS_ARRAY (res, 1),
    MATVEC_VEC_AS_ARRAY (vin, 1),
    MATVEC_MAT3X3_AS_MDARRAY (transform),
    MATVEC_VEC_AS_ARRAY (vtrans, 1)
);
```

Definition at line 1573 of file macros.h.

**2.1.2.29 #define MATVEC\_VEC\_SCALE(res, vin, fac)**

Scale a vector. The result vector may be the input vectors

**Parameters:**

- res* the result vector extractor
- vin* the input vector extractor
- fac* the factor with which the vector will be multiplied

Definition at line 1178 of file macros.h.

**2.1.2.30 #define MATVEC\_VEC\_SUB(res, veca, vecb)**

Subtract two vectors. The result vector may be one of the input vectors.

**Parameters:**

- res* the result vector extractor
- veca* the first vector extractor
- vecb* the second vector extractor

Definition at line 1277 of file macros.h.

**2.1.2.31 #define MATVEC\_VEC\_TRANSLATE\_ROTATE(type, vout, vin, vtrans, matrix)**

Apply a translation followed by a rotation to a vector.

**Parameters:**

- type* the C type of a vector component (e.g. double)
- vout* the output vector extractor
- vin* the input vector extractor
- vtrans* the translation vector extractor
- matrix* the rotation matrix extractor

The parameters may be used more than once, so ensure that if they are expressions that there are no side effects.

Example:

The convention is to use homogenous coordinates and the transformation is stored as a 4x4 C multidimensional array. In this case, the rotation matrix is stored in the upper 3x3 submatrix and the translation vector is stored in the right-most column. The input and output vectors are stored in 3 element C arrays.

```
double transform[4][4];
double vin[3], vout[3];

MATVEC_VEC_TRANSLATE_ROTATE( double,
                             MATVEC_VEC_AS_ARRAY(vout,1),
                             MATVEC_VEC_AS_ARRAY(vin,1),
                             MATVEC_VEC_AS_ARRAY(&transform[0][3], 4),
                             MATVEC_MAT3X3_AS_MDARRAY(transform)
                             );
```

**Example:**

The rotation matrix is stored in a 3x3 C multidimensional array and the translation vector is stored in a 3 element C array. The input and output vectors are stored in 3 element C arrays.

```
double transform[3][3];
double vtrans[3], vin[3], vout[3];

MATVEC_VEC_TRANSLATE_ROTATE( double,
                             MATVEC_VEC_AS_ARRAY(vout,1),
                             MATVEC_VEC_AS_ARRAY(vin,1),
                             MATVEC_VEC_AS_ARRAY(vtrans,1),
                             MATVEC_MAT3X3_AS_MDARRAY(transform)
                             );
```

Definition at line 1699 of file macros.h.



# Index

macros.h, [5](#)  
    MATVEC\_MAT3X3\_AS\_ARRAY,  
        [7](#)  
    MATVEC\_MAT3X3\_AS\_-  
        MDARRAY, [7](#)  
    MATVEC\_MAT3X3\_ASSIGN, [8](#)  
    MATVEC\_MAT3X3\_IDENTITY, [8](#)  
    MATVEC\_MAT3X3\_MULT, [9](#)  
    MATVEC\_MAT3X3\_ROT\_X, [9](#)  
    MATVEC\_MAT3X3\_ROT\_Y, [9](#)  
    MATVEC\_MAT3X3\_ROT\_YMXZ,  
        [10](#)  
    MATVEC\_MAT3X3\_ROT\_Z, [11](#)  
    MATVEC\_MAT3X3\_ROT\_ZXZ, [11](#)  
    MATVEC\_MAT3X3\_SET, [12](#)  
    MATVEC\_MAT3X3\_-  
        TRANPOSE, [12](#)  
    MATVEC\_MAT4X4\_AS\_ARRAY,  
        [12](#)  
    MATVEC\_MAT4X4\_AS\_-  
        MDARRAY, [13](#)  
    MATVEC\_MAT4X4\_ASSIGN, [13](#)  
    MATVEC\_MAT4X4\_IDENTITY,  
        [14](#)  
    MATVEC\_MAT4X4\_MULT, [14](#)  
    MATVEC\_MAT4X4\_SET, [14](#)  
    MATVEC\_MAT4X4\_-  
        TRANPOSE, [15](#)  
    MATVEC\_VEC\_ADD, [15](#)  
    MATVEC\_VEC\_AS\_ARRAY, [15](#)  
    MATVEC\_VEC\_AS\_STRUCT, [16](#)  
    MATVEC\_VEC\_ASSIGN, [16](#)  
    MATVEC\_VEC\_CROSSPROD, [16](#)  
    MATVEC\_VEC\_DOTPROD, [17](#)  
    MATVEC\_VEC\_NORM, [17](#)  
    MATVEC\_VEC\_PROJECT, [17](#)  
    MATVEC\_VEC\_ROTATE\_-  
        TRANSLATE, [17](#)  
    MATVEC\_VEC\_SCALE, [18](#)  
    MATVEC\_VEC\_SUB, [19](#)  
    MATVEC\_VEC\_TRANSLATE\_-  
        ROTATE, [19](#)  
MATVEC\_MAT3X3\_AS\_ARRAY  
    macros.h, [7](#)  
MATVEC\_MAT3X3\_AS\_MDARRAY  
    macros.h, [7](#)  
MATVEC\_MAT3X3\_ASSIGN  
    macros.h, [8](#)  
MATVEC\_MAT3X3\_IDENTITY  
    macros.h, [8](#)  
MATVEC\_MAT3X3\_MULT  
    macros.h, [9](#)  
MATVEC\_MAT3X3\_ROT\_X  
    macros.h, [9](#)  
MATVEC\_MAT3X3\_ROT\_Y  
    macros.h, [9](#)  
MATVEC\_MAT3X3\_ROT\_YMXZ  
    macros.h, [10](#)  
MATVEC\_MAT3X3\_ROT\_Z  
    macros.h, [11](#)  
MATVEC\_MAT3X3\_ROT\_ZXZ  
    macros.h, [11](#)  
MATVEC\_MAT3X3\_SET  
    macros.h, [12](#)  
MATVEC\_MAT3X3\_TRANPOSE  
    macros.h, [12](#)  
MATVEC\_MAT4X4\_AS\_ARRAY  
    macros.h, [12](#)  
MATVEC\_MAT4X4\_AS\_MDARRAY  
    macros.h, [13](#)  
MATVEC\_MAT4X4\_ASSIGN  
    macros.h, [13](#)  
MATVEC\_MAT4X4\_IDENTITY

macros.h, [14](#)  
MATVEC\_MAT4X4\_MULT  
macros.h, [14](#)  
MATVEC\_MAT4X4\_SET  
macros.h, [14](#)  
MATVEC\_MAT4X4\_TRANSPOSE  
macros.h, [15](#)  
MATVEC\_VEC\_ADD  
macros.h, [15](#)  
MATVEC\_VEC\_AS\_ARRAY  
macros.h, [15](#)  
MATVEC\_VEC\_AS\_STRUCT  
macros.h, [16](#)  
MATVEC\_VEC\_ASSIGN  
macros.h, [16](#)  
MATVEC\_VEC\_CROSSPROD  
macros.h, [16](#)  
MATVEC\_VEC\_DOTPROD  
macros.h, [17](#)  
MATVEC\_VEC\_NORM  
macros.h, [17](#)  
MATVEC\_VEC\_PROJECT  
macros.h, [17](#)  
MATVEC\_VEC\_ROTATE\_-  
TRANSLATE  
macros.h, [17](#)  
MATVEC\_VEC\_SCALE  
macros.h, [18](#)  
MATVEC\_VEC\_SUB  
macros.h, [19](#)  
MATVEC\_VEC\_TRANSLATE\_-  
ROTATE  
macros.h, [19](#)