

# The `glmatrix` Package

Rene Warnking

Version 1.0.0

2026/04/03

## Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Example</b>	<b>2</b>
<b>3</b>	<b>Macro Documentation</b>	<b>3</b>
3.1	matrix . . . . .	3
3.2	scalar . . . . .	16
3.3	vector . . . . .	27
<b>4</b>	<b>Implementation</b>	<b>50</b>
4.1	matrix . . . . .	50
4.2	scalar . . . . .	65
4.3	vector . . . . .	76
<b>5</b>	<b>License</b>	<b>98</b>

## 1 Introduction

The `glmatrix` package provides a small set of macros that allow you to do basic vector and matrix arithmetics in  $\LaTeX$ . It is inspired by the javascript library with the same name (<https://glmatrix.net/>) and provides the same functions and more. Using this package, you can easily do vector calculations like dot and cross product as well as more complex calculations, like vector matrix multiplication.

The package is specifically targeting use cases in computer graphics. Consequently, it provides macros for vector transformations using matrices and the creation of commonly used matrices in computer graphics, like the `lookAt` matrix. In combination, these macros allow you to do many calculations that regularly occur in computer graphics.

## 2 Example

Given the following equation:

$$AB(\vec{v} + \vec{w}) \mid v, w \in \mathbb{R}^3, A, B \in \mathbb{R}^{3 \times 3}$$

The code below recreates the equation with exemplary values and directly calculates the results. Then, the results are used to print a step-by-step solution with the chosen values.

Note:

- While creating a new vector or matrix is done by providing its name as a simple string, accessing its content after creation is done as a command (with backslash)
- This package mainly provides macros for mathematical calculations. These macros assign the calculated result automatically to a variable using the optional parameter (for example `\addVec`). If you wish to specify the name of the variable yourself, you may pass a string for the optional argument.

```
% Initialize two vectors and add them together
\newVec{myVecV}{1,2,3}
\newVec{myVecW}{7,8,9}
\addVec[addVecResult]{\myVecV}{\myVecW}

% Initialize two matrices and multiply them with each other
\newMat{myMatA}{{0.5,2,3}{4,5,6}{7,8,9}}
\newDiagMat{myMatB}{2, 3, 4}
\mulMat[mulMatResult]{\myMatA}{\myMatB}

% Transform the result vector with the created matrix
\transformVec{\mulMatResult}{\addVecResult}

% Print a step-by-step solution
\begin{align}
&\printMat{\myMatA} \printMat{\myMatB} \\
&\left(\printVec{\myVecV} + \printVec{\myVecW}\right) \\
&= \\
&\printMat{\mulMatResult} \\
&\printVec{\addVecResult} \\
&= \\
&\printVec{\transformVecResult}
\end{align}
```

When compiling the above  $\text{\LaTeX}$  code, the following output is created.

$$\begin{pmatrix} 0.5 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} \begin{pmatrix} 2 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 4 \end{pmatrix} \left( \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} + \begin{pmatrix} 7 \\ 8 \\ 9 \end{pmatrix} \right) = \begin{pmatrix} 1 & 6 & 12 \\ 8 & 15 & 24 \\ 14 & 24 & 36 \end{pmatrix} \begin{pmatrix} 8 \\ 10 \\ 12 \end{pmatrix} = \begin{pmatrix} 212 \\ 502 \\ 784 \end{pmatrix}$$

All three classes provide various macros to print the content of a variable for all kinds of use cases. For example, you can print a vector inside a line of text using the `\printVecT`, which prints a transposed version of the vector's contents.

## 3 Macro Documentation

### 3.1 matrix

Command	Arguments
<code>\newMat</code>	{ <i>out</i> }, { <i>scalars</i> }
<code>\newMatFromRow</code>	{ <i>out</i> }, { <i>row</i> }
<code>\newMatFromRowVecs</code>	{ <i>out</i> }, { <i>rows</i> }
<code>\newMatFromColVecs</code>	{ <i>out</i> }, { <i>columns</i> }
<code>\newEmptyMat</code>	{ <i>out</i> }
<code>\newDiagMat</code>	{ <i>out</i> }, { <i>scalars</i> }
<code>\appendRowToMat</code>	{ <i>out</i> }, { <i>matrix</i> }, { <i>vector r</i> }
<code>\newTransMat</code>	{ <i>out</i> }, { <i>vector t</i> }
<code>\newScaleMat</code>	{ <i>out</i> }, { <i>vector s</i> }
<code>\newRotMatTwoD</code>	[ <i>out</i> ], { <i>scalar</i> }
	Default Argument: rotMatResult
<code>\newRotMat</code>	[ <i>out</i> ], { <i>vector</i> }, { <i>scalar</i> }
	Default Argument: rotMatResult
<code>\newLookAtMat</code>	[ <i>out</i> ], { <i>right</i> }, { <i>up</i> }, { <i>dir</i> }, { <i>pos</i> }
	Default Argument: lookAtMatResult
<code>\newProjMat</code>	{ <i>out</i> }
<code>\newProjMatGen</code>	{ <i>out</i> }
<code>\newFrustumMat</code>	[ <i>out</i> ], { <i>fovx</i> }, { <i>fovy</i> }, { <i>near</i> }, { <i>far</i> }
	Default Argument: frustumMatResult
<code>\printMat</code>	[ <i>fraction</i> ], { <i>matrix</i> }
	Default Argument: 0
<code>\printMatDiag</code>	[ <i>fraction</i> ], { <i>matrix</i> }
	Default Argument: 0
<code>\getMatHeight</code>	[ <i>out</i> ], { <i>matrix</i> }
	Default Argument: matHeightResult
<code>\getMatWidth</code>	[ <i>out</i> ], { <i>matrix</i> }
	Default Argument: matWidthResult
<code>\transposeMat</code>	[ <i>out</i> ], { <i>matrix</i> }
	Default Argument: transposeMatResult
<code>\addMat</code>	[ <i>out</i> ], { <i>matrix</i> }, { <i>matrices</i> }
	Default Argument: addMatResult
<code>\subMat</code>	[ <i>out</i> ], { <i>matrix</i> }, { <i>matrices</i> }
	Default Argument: subMatResult
<code>\mulMatScalar</code>	[ <i>out</i> ], { <i>matrix</i> }, { <i>scalar</i> }
	Default Argument: mulMatScalarResult
<code>\mulMatVec</code>	[ <i>out</i> ], { <i>matrix</i> }, { <i>vector</i> }
	Default Argument: mulMatVecResult
<code>\mulMat</code>	[ <i>out</i> ], { <i>matrix M</i> }, { <i>matrix N</i> }
	Default Argument: mulMatResult
<code>\getMatTranslation</code>	[ <i>out</i> ], { <i>matrix</i> }
	Default Argument: matTranslationResult
<code>\detMat</code>	[ <i>out</i> ], { <i>matrix</i> }
	Default Argument: detMatResult

`newMat`  $\{\langle out \rangle\}\{\langle scalars \rangle\}$

$\{\langle out \rangle\}$ : var name to save the result (matrix) to  
 $\{\langle scalars \rangle\}$ : matrix values that are assigned to out

#### Description

Create a new matrix from the given values.

#### Equation

$$o = \mathbf{M} \in \mathbb{R}^{m \times n} = \begin{bmatrix} s_{11} & \dots & s_{1n} \\ \vdots & \ddots & \vdots \\ s_{m1} & \dots & s_{mn} \end{bmatrix}$$

#### Example

```
\newMat{myNewMatrix}{{{0.5,2,3}{4,5,6}{7,8,9}}}
```

$$\backslash\text{myNewMatrix} \mapsto \begin{pmatrix} 0.5 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

```
\newMat{myNewMatrix}{{{0.5,2,3}}}
```

$$\backslash\text{myNewMatrix} \mapsto (0.5 \quad 2 \quad 3)$$

```
\newMat{myNewMatrix}{{{0.5}{4}{7}}}
```

$$\backslash\text{myNewMatrix} \mapsto \begin{pmatrix} 0.5 \\ 4 \\ 7 \end{pmatrix}$$

`newMatFromRow`  $\{\langle out \rangle\}\{\langle row \rangle\}$

$\{\langle out \rangle\}$ : var name to save the result (matrix) to  
 $\{\langle row \rangle\}$ : matrix row that is assigned to out

#### Description

Create a new matrix from the given row.

`newMatFromRowVecs`  $\{\langle out \rangle\}\{\langle rows \rangle\}$

$\{\langle out \rangle\}$ : var name to save the result (matrix) to  
 $\{\langle rows \rangle\}$ : the rows that are used to construct the matrix

#### Description

Create a new matrix from the given row vectors.

**Example**

```

\newVec{myRowVecA}{1, 0, 0}
\newVec{myRowVecB}{0, 2, 0}
\newVec{myRowVecC}{0, 0, 3}
\newMatFromRowVecs{myNewMatrix}{\myRowVecA, \myRowVecB,
\myRowVecC}
\myNewMatrix  $\mapsto$   $\begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{pmatrix}$ 

```

`newMatFromColVecs`  $\{\langle out \rangle\}\{\langle columns \rangle\}$   
 $\{\langle out \rangle\}$ : var name to save the result (matrix) to  
 $\{\langle columns \rangle\}$ : the columns that are used to construct the matrix

**Description**

Create a new matrix from the given column vectors.

`newEmptyMat`  $\{\langle out \rangle\}$   
 $\{\langle out \rangle\}$ : var name to save the result (matrix) to

**Description**

Create a new empty matrix. Obacht: It is not necessary to call this to create a new matrix before using any of the arithmetic commands.

**Equation**

$$o = \mathbf{M} = \emptyset$$

**Example**

```

\newEmptyMat{myNewMatrix}
\myNewMatrix  $\mapsto$   $\emptyset$ 

```

`newDiagMat`  $\{\langle out \rangle\}\{\langle scalars \rangle\}$   
 $\{\langle out \rangle\}$ : var name to save the result (matrix) to  
 $\{\langle scalars \rangle\}$ : the values that define the diagonal of the matrix

**Description**

Create a new diagonal matrix from the given vector.

### Equation

$$o = \mathbf{M} \in \mathbb{R}^{3 \times 3} = \begin{bmatrix} s_1 & 0 & 0 \\ 0 & s_2 & 0 \\ 0 & 0 & s_3 \end{bmatrix}$$

### Example

```
\newDiagMat{myNewMatrix}{0.5,2,3}
\myNewMatrix \mapsto \begin{pmatrix} 0.5 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{pmatrix}
```

`appendRowToMat`  $\{\langle out \rangle\} \{\langle matrix \rangle\} \{\langle vector r \rangle\}$

$\{\langle out \rangle\}$ : var name to save the result (matrix) to

$\{\langle matrix \rangle\}$ : the matrix to append the vector to

$\{\langle vector r \rangle\}$ : the row vector that should be appended to the matrix

### Description

Append the given row to the given matrix.

### Equation

$$o = \begin{bmatrix} \mathbf{M} \\ \mathbf{r} \end{bmatrix} \in \mathbb{R}^{m+1 \times n} \mid \mathbf{M} \in \mathbb{R}^{m \times n}, \mathbf{r} \in \mathbb{R}^{1 \times n}$$

`newTransMat`  $\{\langle out \rangle\} \{\langle vector t \rangle\}$

$\{\langle out \rangle\}$ : var name to save the result (matrix) to

$\{\langle vector t \rangle\}$ : the translation vector to construct the matrix from

### Description

Create translation matrix from the given vector.

### Equation

$$o = \mathbf{T} \in \mathbb{R}^{4 \times 4} = \begin{bmatrix} 1 & 0 & 0 & \mathbf{t}_x \\ 0 & 1 & 0 & \mathbf{t}_y \\ 0 & 0 & 1 & \mathbf{t}_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

### Example

```
\newVec{vecTest}{5, 2, 3}
\newTransMat{myNewMatrix}{\vecTest}
\myNewMatrix \mapsto \begin{pmatrix} 1 & 0 & 0 & 5 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 3 \\ 0 & 0 & 0 & 1 \end{pmatrix}
```

`newScaleMat` [`\out`]{`vector s`}

[`\out`]: var name to save the result (matrix) to

[`vector s`]: the scaling vector to construct the matrix from

### Description

Create scaling matrix from the given vector.

### Equation

$$o = \mathbf{S} \in \mathbb{R}^{4 \times 4} = \begin{bmatrix} \mathbf{s}_x & 0 & 0 & 0 \\ 0 & \mathbf{s}_y & 0 & 0 \\ 0 & 0 & \mathbf{s}_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

### Example

```
\newVec{vecTest}{5, 2, 3}
\newScaleMat{myNewMatrix}{\vecTest}
\myNewMatrix \mapsto \begin{pmatrix} 5 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}
```

`newRotMatTwoD` [`\out`]{`scalar`}

[`\out`]: var name to save the result (matrix) to, default: `rotMatResult`

[`scalar`]: the rotation value

### Description

Create rotation matrix from the given scalar (2D).

### Equation

$$o = \mathbf{R} \in \mathbb{R}^{3 \times 3}$$

### Example

```
\newScalarPi{scalarTest}{0.5}
\newRotMatTwoD[myNewMatrix]{\scalarTest}
\myNewMatrix \mapsto \begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}
```

`newRotMat` [`<out>`] {`<vector>`} {`<scalar>`}

[`<out>`]: var name to save the result (matrix) to, default: `rotMatResult`

{`<vector>`}: the axis to rotate around

{`<scalar>`}: the angle in degree to rotate with (counterclockwise)

### Description

Create a rotation matrix using the rodrigues' matrix

### Equation

$$o = \mathbf{R} \in \mathbb{R}^{3 \times 3}$$

### Example

```
\newVec{rotationAxis}{1,1,1}
\newScalarPi{scalarTest}{0.5}
\newRotMat[myNewMatrix]{\rotationAxis}{\scalarTest}
\myNewMatrix \mapsto \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}
```

`newLookAtMat` [`<out>`] {`<right>`} {`<up>`} {`<dir>`} {`<pos>`}

[`<out>`]: var name to save the result (matrix) to, default: `lookAtMatResult`

{`<right>`}: the right vector

{`<up>`}: the up vector

{`<dir>`}: the direction vector

{`<pos>`}: the camera position vector

### Description

Create a new lookAt matrix.

### Equation

$$o = \mathbf{L} \in \mathbb{R}^{4 \times 4} = \begin{bmatrix} \mathbf{r}_x & \mathbf{r}_y & \mathbf{r}_z & -\mathbf{r} \cdot \mathbf{p} \\ \mathbf{u}_x & \mathbf{u}_y & \mathbf{u}_z & -\mathbf{u} \cdot \mathbf{p} \\ -\mathbf{d}_x & -\mathbf{d}_y & -\mathbf{d}_z & \mathbf{d} \cdot \mathbf{p} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



**Example**

```

\newVec{right}{5, 2, 3}
\newVec{up}{0.5, 1.75, 2}
\newVec{dir}{0.5, 2, -3}
\newVec{pos}{5, 2, 3}
\newLookAtMat[myNewMatrix]{\right}{\up}{\dir}{\pos}

\myNewMatrix \mapsto \begin{pmatrix} 5 & 2 & 3 & -38 \\ 0.5 & 1.75 & 2 & -12 \\ -0.5 & -2 & 3 & -2.5 \\ 0 & 0 & 0 & 1 \end{pmatrix}

```

`newProjMat`  $\{\langle out \rangle\}$

$\{\langle out \rangle\}$ : var name to save the result (matrix) to

**Description**

Create a new default projection matrix.

**Equation**

$$o = \mathbf{P} \in \mathbb{R}^{4 \times 4}$$

**Example**

```

\newProjMat{\myNewMatrix}

\myNewMatrix \mapsto \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 0 \end{pmatrix}

```

`newProjMatGen`  $\{\langle out \rangle\}$

$\{\langle out \rangle\}$ : var name to save the result (matrix) to

**Description**

Create a new projection matrix from normal and distance from origin.

**Equation**

$$o = \mathbf{P} \in \mathbb{R}^{4 \times 4}$$

**Example**

```
\newScalar{myDelta}{5}
\newVec{myNormal}{1,1,0}
\newProjMatGen{\myNewMatrix}{\myNormal}{\myDelta}
\myNewMatrix \mapsto \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0.2 & 0.2 & 0.2 & 0 \end{pmatrix}
```

**newFrustumMat** [*out*]{*fovx*}{*fovy*}{*near*}{*far*}

[*out*]: var name to save the result (matrix) to, default: frustumMatResult  
 {*fovx*}: the fovx scalar  
 {*fovy*}: the fovy scalar  
 {*near*}: the position of the near plane (scalar)  
 {*far*}: the position of the far plane (scalar)

**Description**

Create a new frustum matrix.

**Equation**

$$o = \mathbf{F} \in \mathbb{R}^{4 \times 4}$$

**Example**

```
\newScalar{fovx}{90}
\newScalar{fovy}{90}
\newScalar{near}{1}
\newScalar{far}{6}
\newFrustumMat[myNewMatrix]{\fovx}{\fovy}{\nearrow}{\far}
\myNewMatrix \mapsto \begin{pmatrix} 0.6174 & 0 & 0 & 0 \\ 0 & 0.6174 & 0 & 0 \\ 0 & 0 & -1.4 & -2.4 \\ 0 & 0 & -1 & 0 \end{pmatrix}
```

**printMat** [*fraction*]{*matrix*}

[*fraction*]: whether or not the value should be printed as a fraction (val=1), default: 0  
 {*matrix*}: the matrix that should be printed

**Description**

Print the values of this matrix variable.

### Equation

$$\begin{bmatrix} a_{11} & \dots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \dots & a_{mn} \end{bmatrix}$$

### Example

```
\newMat{myNewMatrix}{\{a_{11}, \dots, a_{1n}\}\{a_{21}, \dots, a_{2n}\}\{a_{m1}, \dots, a_{mn}\}}
\printMat{\myNewMatrix} \mapsto \begin{bmatrix} a_{11} & \dots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \dots & a_{mn} \end{bmatrix}
```

`printMatDiag` [*fraction*] {*matrix*}

[*fraction*]: whether or not the value should be printed as a fraction (val=1), default: 0

{*matrix*}: the matrix that should be printed

### Description

Print only the diagonal values of this matrix variable.

### Example

```
\newMat{myNewMatrix}{\{a_{11}, \dots, a_{1n}\}\{a_{21}, \dots, a_{2n}\}\{a_{m1}, \dots, a_{mn}\}}
\printMatDiag{\myNewMatrix} \mapsto \text{diag}(a_{11}, a_{22}, a_{mm})
```

`getMatHeight` [*out*] {*matrix*}

[*out*]: var name to save the result (scalar) to, default: `matHeightResult`

{*matrix*}: the matrix from which to get the height

### Description

Get the height of the given matrix.

### Equation

$$o = \text{rows}(\mathbf{M}) \mid \mathbf{M} \in \mathbb{R}^{m \times n}$$

### Example

```
\newMat{myNewMatrix}{\{a_{11}, \dots, a_{1n}\}\{a_{21}, \dots, a_{2n}\}\{a_{m1}, \dots, a_{mn}\}}
\getMatHeight{myNewMatrix}
\matHeightResult \mapsto m
```

`getMatWidth` [*out*] {*matrix*}

[*out*]: var name to save the result (scalar) to, default: `matWidthResult`

`{<matrix>}`: the matrix from which to get the width

Description
Get the width of the given matrix.

Equation
$o = \text{cols}(\mathbf{M}) \mid \mathbf{M} \in \mathbb{R}^{m \times n}$

Example
<pre>\newMat{myNewMatrix}{\{a_{11}, \dots, a_{1n}\}\{a_{21}, \dots, a_{2n}\}\{a_{m1}, \dots, a_{mn}\}} \getMatWidth{myNewMatrix} \matWidthResult \mapsto n</pre>

`transposeMat` [`<out>`]{`<matrix>`}

[`<out>`]: var name to save the result (matrix) to, default: `transposeMatResult`

`{<matrix>}`: the matrix to transpose

Description
Transpose a given Matrix. The parameter can also be a vector, but the result will be a matrix.

Equation
$o = \mathbf{M}^T$

Example
<pre>\newMat{myNewMatrix}{\{0.5,2,3\}\{4,5,6\}\{7,8,9\}} \transposeMat{\myNewMatrix} \mapsto \begin{bmatrix} 0.5 &amp; 4 &amp; 7 \\ 2 &amp; 5 &amp; 8 \\ 3 &amp; 6 &amp; 9 \end{bmatrix}</pre>

`addMat` [`<out>`]{`<matrix>`}{`<matrices>`}

[`<out>`]: var name to save the result (matrix) to, default: `addMatResult`

`{<matrix>}`: the matrix to add to

`{<matrices>}`: the matrices to add

Description
Add two or more matrices.

Equation
$o = \mathbf{X} = \mathbf{M} + \mathbf{B} \mid \mathbf{M}, \mathbf{M}, \mathbf{B} \in \mathbb{R}^{m \times n}$

### Example

```

\newMat{myNewMatrixA}{\{0.5,2,3\}\{4,5,6\}\{7,8,9\}}
\newMat{myNewMatrixB}{\{0.5,2,3\}\{4,5,6\}\{7,8,9\}}
\addMat{\myNewMatrixA}{\myNewMatrixB}
\addMatResult \mapsto \begin{bmatrix} 1 & 4 & 6 \\ 8 & 10 & 12 \\ 14 & 16 & 18 \end{bmatrix}
\addMat[resultName]{\myNewMatrixA}{\myNewMatrixB}
\resultName \mapsto \begin{bmatrix} 1 & 4 & 6 \\ 8 & 10 & 12 \\ 14 & 16 & 18 \end{bmatrix}

```

**subMat** [*out*]{*matrix*}{*matrices*}

[*out*]: var name to save the result (matrix) to, default: subMatResult

{*matrix*}: the matrix to subtract from

{*matrices*}: the matrices to subtract

### Description

Subtract two or more matrices.

### Equation

$$o = \mathbf{X} = \mathbf{M} - \mathbf{B} \mid \mathbf{X}, \mathbf{M}, \mathbf{B} \in \mathbb{R}^{m \times n}$$

### Example

```

\newMat{myNewMatrixA}{\{0.5,2,3\}\{4,5,6\}\{7,8,9\}}
\newMat{myNewMatrixB}{\{0.25,1,2\}\{3,2,1\}\{3,3,3\}}
\subMat[subMatResult]{\myNewMatrixA}{\myNewMatrixB}
\subMatResult \mapsto \begin{bmatrix} 0.25 & 1 & 1 \\ 1 & 3 & 4 \\ 4 & 5 & 6 \end{bmatrix}

```

**mulMatScalar** [*out*]{*matrix*}{*scalar*}

[*out*]: var name to save the result (matrix) to, default: mulMatScalarResult

{*matrix*}: the matrix to multiply

{*scalar*}: the scalar to multiply with

### Description

Multiply the given matrix with a scalar.

### Equation

$$o = \mathbf{X} = \mathbf{M} * s \mid \mathbf{X}, \mathbf{M} \in \mathbb{R}^{m \times n}, s \in \mathbb{R}$$

### Example

```

\newMat{myNewMatrix}{\{0.5,2,3\}\{4,5,6\}\{7,8,9\}}
\newScalar{myNewScalar}{2}
\mulMatScalar{\myNewMatrix}{\myNewScalar}
\mulMatScalarResult \mapsto \begin{bmatrix} 1 & 4 & 6 \\ 8 & 10 & 12 \\ 14 & 16 & 18 \end{bmatrix}
\mulMatScalar[resultName]{\myNewMatrix}{\myNewScalar}
\resultName \mapsto \begin{bmatrix} 1 & 4 & 6 \\ 8 & 10 & 12 \\ 14 & 16 & 18 \end{bmatrix}

```

**mulMatVec** [*out*]{*matrix*}{*vector*}

[*out*]: var name to save the result (vector) to, default: mulMatVecResult  
 {*matrix*}: the matrix to transform the vector with  
 {*vector*}: the vector to transform

### Description

Multiply the given matrix with a vector (for transformation).

### Equation

$$o = \mathbf{M} \cdot \mathbf{v} \mid \mathbf{M} \in \mathbb{R}^{m \times n}, \mathbf{v} \in \mathbb{R}^n$$

### Example

```

\newVec{myNewVec}{0.5, 1, 2}
\newMat{myNewMatrix}{\{0.5, 1, 2\}\{4, 5, 6\}\{7, 8, 9\}}
\mulMatVec{\myNewMatrix}{\myNewVec}
\mulMatVecResult \mapsto \begin{pmatrix} 5.25 \\ 19 \\ 29.5 \end{pmatrix}
\mulMatVec[resultName]{\myNewMatrix}{\myNewVec}
\resultName \mapsto \begin{pmatrix} 5.25 \\ 19 \\ 29.5 \end{pmatrix}

```

### Warning

When matrix and vector have a dimension mismatch

**mulMat** [*out*]{*matrix M*}{*matrix N*}

[*out*]: var name to save the result (matrix) to, default: mulMatResult  
 {*matrix M*}: the matrix to multiply  
 {*matrix N*}: the matrix to multiply with

### Description

Multiply the given matrix with another matrix.

### Equation

$$O = \mathbf{X} = \mathbf{MN} \mid \mathbf{M} \in \mathbb{R}^{m \times n}, \mathbf{N} \in \mathbb{R}^{n \times p}, \mathbf{X} \in \mathbb{R}^{m \times p}$$

### Example

```
\newMat{myNewMatrixA}{{{0.5,2,3}{4,5,6}{7,8,9}}}
\newMat{myNewMatrixB}{{{0.25,1,2}{3,2,1}{3,3,3}}}
\mulMat[mulMatResult]{\myNewMatrixA}{\myNewMatrixB}
\mulMatResult \mapsto \begin{bmatrix} 15.125 & 13.5 & 12 \\ 34 & 32 & 31 \\ 52.75 & 50 & 49 \end{bmatrix}
```

### Warning

When the matrices have a dimension mismatch

`getMatTranslation` [*out*]{*matrix*}

[*out*]: var name to save the result (vector) to, default: `matTranslationResult`  
 {*matrix*}: the matrix to extract the translation from

### Description

Get the translation component from the given matrix.

### Equation

$$o = \mathbf{v} = \mathit{trans}(\mathbf{M}) \mid \mathbf{M} \in \mathbb{R}^{m \times n}, \mathbf{v} \in \mathbb{R}^{m-1}$$

### Example

```
\newMat{myNewMatrix}{{{0.5,2,3}{4,5,6}{7,8,9}}}
\getMatTranslation[matTranslationResult]{\myNewMatrix}
\matTranslationResult \mapsto \begin{pmatrix} 3 \\ 6 \end{pmatrix}
```

`detMat` [*out*]{*matrix*}

[*out*]: var name to save the result (scalar) to, default: `detMatResult`  
 {*matrix*}: the matrix to calculate the determinate for

### Description

Get the determinate of the given matrix.

### Equation

$$o = \det(\mathbf{M}) \mid \mathbf{M} \in \mathbb{R}^{m \times n}$$

### Example

```
\newMat{myNewMatrix}{\{0.5,2,3\}\{4,5,6\}\{7,8,9\}}
\detMat[detMatResult]{\myNewMatrix}
\detMatResult \mapsto 1.5
```

## 3.2 scalar

Command	Arguments
<code>\newScalar</code>	$\{\langle out \rangle\}, \{\langle scalar \rangle\}$
<code>\newScalarPi</code>	$\{\langle out \rangle\}, \{\langle scalar \rangle\}$
<code>\newScalarFraction</code>	$\{\langle out \rangle\}, \{\langle scalar n \rangle\}, \{\langle scalar d \rangle\}$
<code>\randomScalar</code>	$\{\langle out \rangle\}, \{\langle lower bound \rangle\}, \{\langle upper bound \rangle\}$
<code>\copyScalar</code>	$[\langle out \rangle], \{\langle scalar \rangle\}$ Default Argument: scalarCopyResult
<code>\setScalar</code>	$\{\langle out \rangle\}, \{\langle scalar \rangle\}$
<code>\zeroScalar</code>	$\{\langle out \rangle\}$
<code>\printScalar</code>	$[\langle fraction \rangle], \{\langle scalar \rangle\}$ Default Argument: 0
<code>\printScalarAsPi</code>	$[\langle fraction \rangle], \{\langle scalar \rangle\}$ Default Argument: 0
<code>\printPiScalarInDegree</code>	$[\langle fraction \rangle], \{\langle scalar \rangle\}$ Default Argument: 0
<code>\printScalarRounded</code>	$[\langle digits \rangle], \{\langle scalar \rangle\}$ Default Argument: 4
<code>\roundScalar</code>	$[\langle digits \rangle], \{\langle scalar \rangle\}$ Default Argument: 4
<code>\ceilScalar</code>	$[\langle out \rangle], \{\langle scalar \rangle\}$ Default Argument: scalarCeilResult
<code>\floorScalar</code>	$[\langle out \rangle], \{\langle scalar \rangle\}$ Default Argument: scalarFloorResult
<code>\clipScalar</code>	$[\langle out \rangle], \{\langle scalar \rangle\}$ Default Argument: clipScalarResult
<code>\radianToDegree</code>	$[\langle out \rangle], \{\langle scalar \rangle\}$ Default Argument: radianToDegreeResult
<code>\toRadian</code>	$[\langle out \rangle], \{\langle scalar \rangle\}$ Default Argument: degreeToRadianResult
<code>\addScalar</code>	$[\langle out \rangle], \{\langle scalar s \rangle\}, \{\langle scalar t \rangle\}$ Default Argument: scalarAddResult
<code>\subScalar</code>	$[\langle out \rangle], \{\langle scalar s \rangle\}, \{\langle scalar t \rangle\}$ Default Argument: scalarSubResult
<code>\mulScalar</code>	$[\langle out \rangle], \{\langle scalar s \rangle\}, \{\langle scalar t \rangle\}$ Default Argument: scalarMulResult



<code>\divScalar</code>	$[\langle out \rangle], \{\langle scalar\ s \rangle\}, \{\langle scalar\ t \rangle\}$ Default Argument: scalarDivResult
<code>\powScalar</code>	$[\langle out \rangle], \{\langle scalar\ s \rangle\}, \{\langle scalar\ t \rangle\}$ Default Argument: scalarPowResult
<code>\negateScalar</code>	$[\langle out \rangle], \{\langle scalar \rangle\}$ Default Argument: scalarNegateResult
<code>\arccosScalar</code>	$[\langle out \rangle], \{\langle scalar \rangle\}$ Default Argument: scalarArccosResult
<code>\mulAndAddScalar</code>	$[\langle out \rangle], \{\langle scalar\ s \rangle\}, \{\langle scalar\ t \rangle\}, \{\langle scalar\ u \rangle\}$ Default Argument: scalarMulAndAddResult
<code>\lerpScalar</code>	$[\langle out \rangle], \{\langle scalar\ i \rangle\}, \{\langle scalar\ j \rangle\}, \{\langle scalar\ t \rangle\}$ Default Argument: lerpScalarResult
<code>\minScalar</code>	$[\langle out \rangle], \{\langle scalar\ s \rangle\}, \{\langle scalar\ t \rangle\}$ Default Argument: minScalarResult
<code>\maxScalar</code>	$[\langle out \rangle], \{\langle scalar\ s \rangle\}, \{\langle scalar\ t \rangle\}$ Default Argument: maxScalarResult
<code>\betweenScalar</code>	$[\langle out \rangle], \{\langle scalar \rangle\}, \{\langle lower\ bound \rangle\}, \{\langle upper\ bound \rangle\}$ Default Argument: betweenScalarResult
<code>\forEachScalar</code>	$\{\langle input \rangle\}, \{\langle macro \rangle\}, \{\langle args \rangle\}$

`newScalar`  $\{\langle out \rangle\}\{\langle scalar \rangle\}$

$\{\langle out \rangle\}$ : var name to save the result (scalar) to  
 $\{\langle scalar \rangle\}$ : scalar value that is assigned to out

#### Description

Create a new scalar and assign the given value.

#### Example

```
\newScalar{myNewScalar}{3}
\myNewScalar  $\mapsto$  3
```

`newScalarPi`  $\{\langle out \rangle\}\{\langle scalar \rangle\}$

$\{\langle out \rangle\}$ : var name to save the result (scalar) to  
 $\{\langle scalar \rangle\}$ : scalar value that is multiplied with pi and assigned to out

#### Description

Create a new scalar and assign the given value as a multiple of pi.

#### Example

```
\newScalarPi{myNewScalar}{3}
\myNewScalar  $\mapsto$  9.42
```

`newScalarFraction`  $\{\langle out \rangle\}\{\langle scalar\ n \rangle\}\{\langle scalar\ d \rangle\}$   
 $\{\langle out \rangle\}$ : var name to save the result (scalar) to  
 $\{\langle scalar\ n \rangle\}$ : nominator  
 $\{\langle scalar\ d \rangle\}$ : denominator

#### Description

Create a scalar from a given fraction

#### Example

```
\newScalarFraction{myScalarFraction}{3}{10}
\myScalarFraction  $\mapsto \frac{3}{10}$ 
```

`randomScalar`  $\{\langle out \rangle\}\{\langle lower\ bound \rangle\}\{\langle upper\ bound \rangle\}$   
 $\{\langle out \rangle\}$ : var name to save the result (scalar) to  
 $\{\langle lower\ bound \rangle\}$ : scalar that limits the possible values at the bottom  
 $\{\langle upper\ bound \rangle\}$ : scalar that limits the possible values at the top

#### Description

Create a new scalar variable with a random value between lower and upper bound. The value is obtained by creating a random value between 0 and 1 and then interpolating between lower and upper bound.

#### Example

```
\randomScalar{myRandomScalar}{3}{10}
\myRandomScalar  $\mapsto 5$ 
```

`copyScalar`  $[\langle out \rangle]\{\langle scalar \rangle\}$   
 $[\langle out \rangle]$ : var name to save the result (scalar) to, default: `scalarCopyResult`  
 $\{\langle scalar \rangle\}$ : scalar which value is assigned to out

#### Description

Copy the value of one scalar variable to another.

#### Example

```
\newScalar{myNewScalar}{3}
\copyScalar{myCopyScalar}{\myNewScalar}
\myCopyScalar  $\mapsto 3$ 
```

`setScalar`  $\{\langle out \rangle\}\{\langle scalar \rangle\}$   
 $\{\langle out \rangle\}$ : var name to save the result (scalar) to  
 $\{\langle scalar \rangle\}$ : scalar value that is assigned to out

### Description

Set the value of a given scalar.

### Example

```
\newScalar{myNewScalar}{3}
\setScalar{myNewScalar}{4}
\myNewScalar ↦ 4
```

**zeroScalar**  $\{\langle out \rangle\}$   
 $\{\langle out \rangle\}$ : var name to save the result (scalar) to

### Description

Set or create a scalar with value 0.

### Example

```
\zeroScalar{myZeroScalar}
\myZeroScalar ↦ 0
```

**printScalar**  $[\langle fraction \rangle]\{\langle scalar \rangle\}$   
 $[\langle fraction \rangle]$ : whether or not the value should be printed as a fraction (val=1), default: 0  
 $\{\langle scalar \rangle\}$ : the scalar that should be printed

### Description

Print the value of the given scalar.

### Example

```
\newScalar{myNewScalar}{0.5}
\printScalar[1]{\myNewScalar} ↦  $\frac{1}{2}$ 
```

**printScalarAsPi**  $[\langle fraction \rangle]\{\langle scalar \rangle\}$   
 $[\langle fraction \rangle]$ : whether or not the value should be printed as a fraction (val=1), default: 0  
 $\{\langle scalar \rangle\}$ : the scalar that should be printed

### Description

Print the value of the given scalar as a multiple of Pi.

### Equation

$$\left\{ \frac{s}{\pi} \right\} \pi$$

### Example

```
\newScalarPi{myNewScalar}{0.5}
\printScalarAsPi[1]{\myNewScalar} \mapsto \frac{1}{2}\pi
```

**printPiScalarInDegree** [*<fraction>*]{*<scalar>*}

[*<fraction>*]: whether or not the value should be printed as a fraction (val=1), default: 0

{*<scalar>*}: the value that should be printed

### Description

Print the value of the given scalar in degree. This macro does a conversion and therefore expects the input to be in radian. A degree sign is automatically added.

### Equation

$$\left\{ \frac{s*180}{\pi} \right\}^\circ$$

### Example

```
\newScalarPi{myNewScalar}{0.5}
\printPiScalarInDegree[1]{\myNewScalar} \mapsto 90^\circ
```

**printScalarRounded** [*<digits>*]{*<scalar>*}

[*<digits>*]: number of digits after comma, default: 4

{*<scalar>*}: the value that should be printed

### Description

Print the rounded value of the given scalar.

### Example

```
\newScalar{myNewScalar}{0.5}
\printScalarRounded{\myNewScalar} \mapsto 1
```

**roundScalar** [*<digits>*]{*<scalar>*}

[*<digits>*]: number of digits after comma, default: 4

{*<scalar>*}: the value that should be rounded

### Description

Round the value of the given scalar and save it.

### Example

```
\newScalar{myNewScalar}{0.5}
\roundScalar{\myNewScalar}
\myNewScalar \mapsto 1
```

**ceilScalar** [*out*]{*scalar*}

[*out*]: var name to save the result (scalar) to, default: scalarCeilResult  
 {*scalar*}: the scalar that should be ceiled

### Description

Ceil the value of the given scalar.

### Example

```
\newScalar{myNewScalar}{0.5}
\ceilScalar{\myNewScalar}
\myNewScalar \mapsto 1
```

**floorScalar** [*out*]{*scalar*}

[*out*]: var name to save the result (scalar) to, default: scalarFloorResult  
 {*scalar*}: the scalar that should be floored

### Description

Floor the value of the given scalar.

### Example

```
\newScalar{myNewScalar}{0.5}
\floorScalar{\myNewScalar}
\myNewScalar \mapsto 0
```

**clipScalar** [*out*]{*scalar*}

[*out*]: var name to save the result (scalar) to, default: clipScalarResult  
 {*scalar*}: the scalar from which the zeros should be removed

### Description

Remove all trailing zeros from the value of the given scalar.

### Example

```
\newScalar{myNewScalar}{5.0000}
\clipScalar{\myNewScalar}
\myNewScalar \mapsto 5
```

**radianToDegree** [*out*]{*scalar*}

`[\langle out \rangle]`: var name to save the result (scalar) to, default: `radianToDegreeResult`  
`\{\langle scalar \rangle\}`: the scalar which to convert

**Description**

Convert the value of the given scalar to degree.

**Example**

```
\newScalarPi{myNewScalar}{0.5}
\radianToDegree{\myNewScalar}
\radianToDegreeResult \mapsto 90
```

`toRadian` `[\langle out \rangle]\{\langle scalar \rangle\}`

`[\langle out \rangle]`: var name to save the result (scalar) to, default: `degreeToRadianResult`  
`\{\langle scalar \rangle\}`: the scalar which to convert

**Description**

Convert the value of the given scalar to radian.

**Example**

```
\newScalarPi{myNewScalar}{180}
\toRadian{\myNewScalar}
\degreeToRadianResult \mapsto 1.5707
```

`addScalar` `[\langle out \rangle]\{\langle scalar s \rangle\}\{\langle scalar t \rangle\}`

`[\langle out \rangle]`: var name to save the result (scalar) to, default: `scalarAddResult`  
`\{\langle scalar s \rangle\}`: the first scalar of the sum  
`\{\langle scalar t \rangle\}`: the second scalar of the sum

**Description**

Add the values of two scalars.

**Equation**

$$o = s + t$$

**Example**

```
\newScalar{myNewScalarA}{1}
\newScalar{myNewScalarB}{2}
\addScalar{\myNewScalarA}{\myNewScalarB}
\scalarAddResult \mapsto 3
```

`subScalar` `[\langle out \rangle]\{\langle scalar s \rangle\}\{\langle scalar t \rangle\}`

`[\langle out \rangle]`: var name to save the result (scalar) to, default: `scalarSubResult`

{*scalar s*}: the scalar to subtract from  
 {*scalar t*}: the scalar to subtract

#### Description

Subtract the values of two scalars.

#### Equation

$$o = s - t$$

#### Example

```
\newScalar{myNewScalarA}{1}
\newScalar{myNewScalarB}{2}
\subScalar{myNewScalarA}{myNewScalarB}
\scalarSubResult \mapsto -1
```

**mulScalar** [*out*]{*scalar s*}{*scalar t*}

[*out*]: var name to save the result (scalar) to, default: scalarMulResult  
 {*scalar s*}: the scalar to multiply  
 {*scalar t*}: the scalar with which to multiply

#### Description

Multiply the values of two scalars.

#### Equation

$$o = s * t$$

#### Example

```
\newScalar{myNewScalarA}{2}
\newScalar{myNewScalarB}{3}
\mulScalar{myNewScalarA}{myNewScalarB}
\scalarMulResult \mapsto 6
```

**divScalar** [*out*]{*scalar s*}{*scalar t*}

[*out*]: var name to save the result (scalar) to, default: scalarDivResult  
 {*scalar s*}: the scalar to divide  
 {*scalar t*}: the scalar to divide with

#### Description

Divide the values of two scalars.

### Equation

$$o = \frac{s}{t}$$

### Example

```
\newScalar{myNewScalarA}{6}
\newScalar{myNewScalarB}{3}
\divScalar{\myNewScalarA}{\myNewScalarB}
\scalarDivResult \mapsto 2
```

### Warning

On division by zero

`powScalar` [*out*]{*scalar s*}{*scalar t*}

[*out*]: var name to save the result (scalar) to, default: `scalarPowResult`

{*scalar s*}: the scalar to raise

{*scalar t*}: the scalar with which to raise

### Description

Raise the values of one scalar by another.

### Equation

$$o = s^t$$

### Example

```
\newScalar{myNewScalarA}{2}
\newScalar{myNewScalarB}{3}
\powScalar{\myNewScalarA}{\myNewScalarB}
\scalarPowResult \mapsto 8
```

`negateScalar` [*out*]{*scalar*}

[*out*]: var name to save the result (scalar) to, default: `scalarNegateResult`

{*scalar*}: the scalar to negate

### Description

Negate the value of the given scalar.

### Equation

$$o = -s$$



### Example

```
\newScalar{myNewScalar}{2}
\negateScalar{\myNewScalar}
\scalarNegateResult \mapsto -2
```

`arccosScalar` [ $\langle out \rangle$ ]{ $\langle scalar \rangle$ }

[ $\langle out \rangle$ ]: var name to save the result (scalar) to, default: `scalarArccosResult`  
 { $\langle scalar \rangle$ }: the scalar to calculate the arccos for

### Description

Get the arccos of the value of the given scalar.

### Equation

$$o = \arccos(s)$$

### Example

```
\newScalar{myNewScalar}{TODO}
\arccosScalar{\myNewScalar}
\scalarArccosResult \mapsto TODO
```

`mulAndAddScalar` [ $\langle out \rangle$ ]{ $\langle scalar s \rangle$ }{ $\langle scalar t \rangle$ }{ $\langle scalar u \rangle$ }

[ $\langle out \rangle$ ]: var name to save the result (scalar) to, default: `scalarMulAndAddResult`  
 { $\langle scalar s \rangle$ }: the scalar to add to  
 { $\langle scalar t \rangle$ }: the scalar that is multiplied and then added  
 { $\langle scalar u \rangle$ }: the scalar that is the multiplier

### Description

Adds two scalars after multiplying the second one by a scalar.

### Equation

$$o = s + t * u$$

### Example

```
\newScalar{myNewScalarA}{2}
\newScalar{myNewScalarB}{3}
\mulAndAddScalar{\myNewScalarA}{\myNewScalarB}{\myNewScalarB}
\scalarMulAndAddResult \mapsto 11
```

`lerpScalar` [ $\langle out \rangle$ ]{ $\langle scalar i \rangle$ }{ $\langle scalar j \rangle$ }{ $\langle scalar t \rangle$ }

[ $\langle out \rangle$ ]: var name to save the result (scalar) to, default: `lerpScalarResult`  
 { $\langle scalar i \rangle$ }: the start scalar

$\{\langle scalar j \rangle\}$ : the end scalar  
 $\{\langle scalar t \rangle\}$ : interpolation amount range [0-1]

**Description**

Linearly interpolate between values of two scalars.

**Equation**

$$o = (1 - t) * i + t * j$$

**Example**

```
\newScalar{myNewScalarA}{2}
\newScalar{myNewScalarB}{3}
\newScalar{myNewScalarT}{0.5}
\lerpScalar{\myNewScalarA}{\myNewScalarB}{\myNewScalarT}
\lerpScalarResult ↦ 2.5
```

**minScalar** [ $\langle out \rangle$ ]{ $\langle scalar s \rangle$ }{ $\langle scalar t \rangle$ }  
 $\langle out \rangle$ : var name to save the result (scalar) to, default: minScalarResult  
 $\{\langle scalar s \rangle\}$ : the first scalar  
 $\{\langle scalar t \rangle\}$ : the second scalar

**Description**

Return the minimum of two scalar values.

**Equation**

$$o = s \text{ falls } s < t \text{ sonst } t$$

**Example**

```
\newScalar{myNewScalarA}{2}
\newScalar{myNewScalarB}{3}
\minScalar{\myNewScalarA}{\myNewScalarB}
\minScalarResult ↦ 2
```

**maxScalar** [ $\langle out \rangle$ ]{ $\langle scalar s \rangle$ }{ $\langle scalar t \rangle$ }  
 $\langle out \rangle$ : var name to save the result (scalar) to, default: maxScalarResult  
 $\{\langle scalar s \rangle\}$ : first scalar value  
 $\{\langle scalar t \rangle\}$ : second scalar value

**Description**

Return the maximum of two scalar values.

**Equation**

$o = s$  falls  $s > t$  sonst  $t$

**Example**

```
\newScalar{myNewScalarA}{2}
\newScalar{myNewScalarB}{3}
\maxScalar{\myNewScalarA}{\myNewScalarB}
\maxScalarResult ↦ 3
\maxScalar[resultName]{\myNewScalarA}{\myNewScalarB}
\resultName ↦ 3
```

**betweenScalar** [*out*]{*scalar*}{*lower bound*}{*upper bound*}

[*out*]: var name to save the result (scalar) to (0 or 1), default: betweenScalar-Result

{*scalar*}: the scalar to test

{*lower bound*}: scalar needs to be greater then this for this macro to return 1

{*upper bound*}: scalar needs to be smaller than this for this macro to return 1

**Description**

Return whether a scalar lies between two other values.

**Example**

```
\newScalar{myNewScalarA}{2}
\newScalar{myNewScalarB}{3}
\newScalar{myNewScalarC}{4}
\betweenScalar{\myNewScalarA}{\myNewScalarB}{\myNewScalarC}
\betweenScalarResult ↦ 1
```

**forEachScalar** {*input*}{*macro*}{*args*}

{*input*}: list of commands (var names) that should be altered

{*macro*}: the chosen function

{*args*}: params for function

**Description**

Apply the given function to a list of variables.

### 3.3 vector

Command	Arguments
<code>\newVec</code>	{ <i>out</i> }, { <i>values</i> }
<code>\newVecPi</code>	{ <i>out</i> }, { <i>values</i> }
<code>\randomVec</code>	{ <i>out</i> }, { <i>dimensions</i> }, { <i>lower bound</i> }, { <i>upper bound</i> }

<code>\copyVec</code>	$[\langle out \rangle], \{\langle vector \rangle\}$ Default Argument: copyVecResult
<code>\zeroVec</code>	$[\langle dims \rangle], \{\langle out \rangle\}$ Default Argument: 3
<code>\appendToVec</code>	$\{\langle out \rangle\}, \{\langle vector \rangle\}, \{\langle scalar \rangle\}$
<code>\printVec</code>	$[\langle fraction \rangle], \{\langle vector \rangle\}$ Default Argument: 0
<code>\printVecT</code>	$[\langle fraction \rangle], \{\langle vector \rangle\}$ Default Argument: 0
<code>\printVecAsPoint</code>	$[\langle fraction \rangle], \{\langle vector \rangle\}$ Default Argument: 0
<code>\printVecAsPi</code>	$[\langle fraction \rangle], \{\langle vector \rangle\}$ Default Argument: 0
<code>\printVecAsPiT</code>	$[\langle fraction \rangle], \{\langle vector \rangle\}$ Default Argument: 0
<code>\printVecContent</code>	$\{\langle vector \rangle\}$
<code>\printVecContentXY</code>	$\{\langle vector \rangle\}$
<code>\printVecContentXYZ</code>	$\{\langle vector \rangle\}$
<code>\printVecContentXYZW</code>	$\{\langle vector \rangle\}$
<code>\getVecHeight</code>	$[\langle out \rangle], \{\langle vector \rangle\}$ Default Argument: vecHeightResult
<code>\printScalarPComp</code>	$[\langle fraction \rangle], \{\langle vector v \rangle\}, \{\langle vector w \rangle\}$ Default Argument: 0
<code>\printCrossPComp</code>	$[\langle fraction \rangle], \{\langle vector v \rangle\}, \{\langle vector w \rangle\}$ Default Argument: 0
<code>\roundVec</code>	$[\langle out \rangle], \{\langle vector \rangle\}, \{\langle digits \rangle\}$ Default Argument: roundVecResult
<code>\ceilVec</code>	$[\langle out \rangle], \{\langle vector \rangle\}$ Default Argument: ceilVecResult
<code>\floorVec</code>	$[\langle out \rangle], \{\langle vector \rangle\}$ Default Argument: floorVecResult
<code>\dehomogenVec</code>	$[\langle out \rangle], \{\langle vector \rangle\}$ Default Argument: dehomogenVecResult
<code>\addVec</code>	$[\langle out \rangle], \{\langle vector \rangle\}, \{\langle vectors \rangle\}$ Default Argument: addVecResult
<code>\subVec</code>	$[\langle out \rangle], \{\langle vector \rangle\}, \{\langle vectors \rangle\}$ Default Argument: subVecResult
<code>\mulVec</code>	$[\langle out \rangle], \{\langle vector \rangle\}, \{\langle vectors \rangle\}$ Default Argument: mulVecResult
<code>\divVec</code>	$[\langle out \rangle], \{\langle vector \rangle\}, \{\langle vectors \rangle\}$ Default Argument: divVecResult
<code>\scaleVec</code>	$[\langle out \rangle], \{\langle vector \rangle\}, \{\langle scalar \rangle\}$ Default Argument: scaleVecResult
<code>\divVecScalar</code>	$[\langle out \rangle], \{\langle vector \rangle\}, \{\langle scalar \rangle\}$ Default Argument: divVecScalarResult
<code>\negateVec</code>	$[\langle out \rangle], \{\langle vector \rangle\}$ Default Argument: negateVecResult
<code>\scaleAndAddVec</code>	$[\langle out \rangle], \{\langle vector v \rangle\}, \{\langle vector w \rangle\}, \{\langle scalar \rangle\}$ Default Argument: scaleAndAddVecResult

<code>\lerpVec</code>	$[\langle out \rangle], \{ \langle vector v \rangle \}, \{ \langle vector w \rangle \}, \{ \langle scalar t \rangle \}$ Default Argument: lerpVecResult
<code>\slerpVec</code>	$[\langle out \rangle], \{ \langle vector v \rangle \}, \{ \langle vector w \rangle \}, \{ \langle scalar t \rangle \}$ Default Argument: slerpVecResult
<code>\dotVec</code>	$[\langle out \rangle], \{ \langle vector v \rangle \}, \{ \langle vector w \rangle \}$ Default Argument: dotVecResult
<code>\lenVec</code>	$[\langle out \rangle], \{ \langle vector \rangle \}$ Default Argument: lenVecResult
<code>\sqrLenVec</code>	$[\langle out \rangle], \{ \langle vector \rangle \}$ Default Argument: sqrLenVecResult
<code>\normalizeVec</code>	$[\langle out \rangle], \{ \langle vector \rangle \}$ Default Argument: normalizeVecResult
<code>\crossVec</code>	$[\langle out \rangle], \{ \langle vector v \rangle \}, \{ \langle vector w \rangle \}$ Default Argument: crossVecResult
<code>\crossVecTwoD</code>	$[\langle out \rangle], \{ \langle vector v \rangle \}, \{ \langle vector w \rangle \}$ Default Argument: crossVecResult
<code>\angleVec</code>	$[\langle out \rangle], \{ \langle vector v \rangle \}, \{ \langle vector w \rangle \}$ Default Argument: angleVecResult
<code>\distVec</code>	$[\langle out \rangle], \{ \langle vector v \rangle \}, \{ \langle vector w \rangle \}$ Default Argument: distVecResult
<code>\sqrDistVec</code>	$[\langle out \rangle], \{ \langle vector v \rangle \}, \{ \langle vector w \rangle \}$ Default Argument: sqrDistVecResult
<code>\transformVec</code>	$[\langle out \rangle], \{ \langle matrix \rangle \}, \{ \langle vector \rangle \}$ Default Argument: transformVecResult
<code>\tensorVec</code>	$[\langle out \rangle], \{ \langle vector \rangle \}$ Default Argument: tensorVecResult
<code>\minVec</code>	$[\langle out \rangle], \{ \langle vector v \rangle \}, \{ \langle vector w \rangle \}$ Default Argument: minVecResult
<code>\maxVec</code>	$[\langle out \rangle], \{ \langle vector v \rangle \}, \{ \langle vector w \rangle \}$ Default Argument: maxVecResult
<code>\solveVecEQL</code>	$[\langle out \rangle], \{ \langle vector v \rangle \}, \{ \langle vector w \rangle \}$ Default Argument: solveVecEQLResult

`newVec`  $\{ \langle out \rangle \} \{ \langle values \rangle \}$

$\{ \langle out \rangle \}$ : var name to save the result (vector) to

$\{ \langle values \rangle \}$ : values that are assigned to out

#### Description

Create a new vector and assign the given values.

#### Example

```
\newVec{myNewVec}{2, 3, 4}
```

```
\myNewVec  $\mapsto \begin{pmatrix} 2 \\ 3 \\ 4 \end{pmatrix}$ 
```

`newVecPi`  $\{\langle out \rangle\}\{\langle values \rangle\}$

$\{\langle out \rangle\}$ : var name to save the result (vector) to

$\{\langle values \rangle\}$ : values that are multiplied with pi and assigned to out

#### Description

Create a new vector and assign the given values as a multiple of pi.

#### Example

`\newVec{myNewVec}{0.5, 1, 2}`

`\myNewVec`  $\mapsto \begin{pmatrix} \frac{1}{2}\pi \\ \pi \\ 2\pi \end{pmatrix}$

`randomVec`  $\{\langle out \rangle\}\{\langle dimensions \rangle\}\{\langle lower bound \rangle\}\{\langle upper bound \rangle\}$

$\{\langle out \rangle\}$ : var name to save the result (vector) to

$\{\langle dimensions \rangle\}$ : scalar that defines the number of dimensions out should have

$\{\langle lower bound \rangle\}$ : scalar that limits the possible values at the bottom

$\{\langle upper bound \rangle\}$ : scalar that limits the possible values at the top

#### Description

Create a new vector and assign random values between lower and upper bound to its components. The value is obtained by creating a random value between 0 and 1 and then interpolating between lower and upper bound.

#### Example

`\randomVec{myNewVec}{4}{3}{10}`

`\myNewVec`  $\mapsto (5, 4, 8, 4)$

`copyVec`  $[\langle out \rangle]\{\langle vector \rangle\}$

$[\langle out \rangle]$ : var name to save the result (vector) to, default: `copyVecResult`

$\{\langle vector \rangle\}$ : vector which values are assigned to out

#### Description

Copy values from one vector variable to another.

#### Example

`\newVec{myNewVec}{0.5, 1, 2}`

`\copyVec[copyVecResult]{\myNewVec}`

`\copyVecResult`  $\mapsto \begin{pmatrix} 0.5 \\ 1 \\ 2 \end{pmatrix}$

`zeroVec`  $[\langle dims \rangle]\{\langle out \rangle\}$

`[\langle dims \rangle]`: number of dimensions (rows), default: 3  
`\{\langle out \rangle\}`: var name to save the result (vector) to

#### Description

Create or override a vector with all its components being 0. Obacht: It is not necessary to call this to create a new vector. This is just for convenience when requiring a vector with initialized with zero.

#### Example

```
\zeroVec[3]{zeroVecResult}
\zeroVecResult \mapsto \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}
```

`appendToVec \{\langle out \rangle\}\{\langle vector \rangle\}\{\langle scalar \rangle\}`  
`\{\langle out \rangle\}`: var name to save the result (vector) to  
`\{\langle vector \rangle\}`: the currently present values  
`\{\langle scalar \rangle\}`: the scalar to add to vector

#### Description

Append a dimension with the given value to an already defined vector.

#### Example

```
\newVec{myNewVec}{0.5, 1}
\appendToVec{myNewVec}\{myNewVec\}{-10}
\myNewVec \mapsto \begin{pmatrix} 0.5 \\ 1 \\ -10 \end{pmatrix}
```

`printVec [\langle fraction \rangle]\{\langle vector \rangle\}`  
`[\langle fraction \rangle]`: whether or not the value should be printed as a fraction (val=1), default: 0  
`\{\langle vector \rangle\}`: the vector that should be printed

#### Description

Print the values of the given vector in vector notation.

#### Equation

$$\begin{pmatrix} v_1 \\ \dots \\ v_n \end{pmatrix}$$

**Example**

```
\newVec{myNewVec}{1, 2, 3}
\printVec{\myNewVec} \mapsto \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}
```

`printVecT` [*fraction*] {*vector*}

[*fraction*]: whether or not the value should be printed as a fraction (val=1), default: 0

{*vector*}: the vector that should be printed

**Description**

Print the values of the given vector in transposed notation.

**Equation**

$$(\mathbf{v}_1, \dots, \mathbf{v}_n)^T$$

**Example**

```
\newVec{myNewVec}{1, 2, 3}
\printVecT{\myNewVec} \mapsto (1, 2, 3)^T
```

`printVecAsPoint` [*fraction*] {*vector*}

[*fraction*]: whether or not the value should be printed as a fraction (val=1), default: 0

{*vector*}: the vector that should be printed

**Description**

Print the values of the given vector in point notation.

**Equation**

$$\begin{bmatrix} \mathbf{v}_1 \\ \dots \\ \mathbf{v}_n \end{bmatrix}$$

**Example**

```
\newVec{myNewVec}{1, 2, 3}
\printVecAsPoint{\myNewVec} \mapsto \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}
```

`printVecAsPi` [*fraction*] {*vector*}



`[\langle fraction \rangle]`: whether or not the value should be printed as a fraction (val=1), default: 0

`\{\langle vector \rangle\}`: the vector that should be printed

**Description**

Print the values of the given vector as a multiple of  $\pi$  in vector notation.

**Equation**

$$\begin{pmatrix} \{\frac{v_1}{\pi}\} \\ \dots \\ \{\frac{v_n}{\pi}\} \end{pmatrix} \pi$$

**Example**

$$\begin{aligned} &\backslash\text{newVec}\{\text{myNewVec}\}\{0.5, 1, 2\} \\ &\backslash\text{printVecAsPi}\{\backslash\text{myNewVec}\} \mapsto \begin{pmatrix} 0.5\pi \\ \pi \\ 2\pi \end{pmatrix} \end{aligned}$$

`printVecAsPiT` `[\langle fraction \rangle]` `\{\langle vector \rangle\}`

`[\langle fraction \rangle]`: whether or not the value should be printed as a fraction (val=1), default: 0

`\{\langle vector \rangle\}`: the vector that should be printed

**Description**

Print the values of the given vector as a multiple of Pi in transposed notation.

**Equation**

$$\left(\{\frac{v_1}{\pi}\}, \dots, \{\frac{v_n}{\pi}\}\right)^T \pi$$

**Example**

$$\begin{aligned} &\backslash\text{newVec}\{\text{myNewVec}\}\{0.5, 1, 2\} \\ &\backslash\text{printVecAsPi}\{\backslash\text{myNewVec}\} \mapsto (0.5\pi, \pi, 2\pi)^T \end{aligned}$$

`printVecContent` `\{\langle vector \rangle\}`

`\{\langle vector \rangle\}`: the vector which values should be printed

**Description**

Print the values of the given vector comma separated.

**Equation**

$$\mathbf{v}_1, \dots, \mathbf{v}_n$$

`printVecContentXY`  $\{\langle vector \rangle\}$   
 $\{\langle vector \rangle\}$ : the vector which values should be printed

**Description**

Returns the x and y component of the given vector.

**Equation**

$$\mathbf{v}_1, \mathbf{v}_2$$

`printVecContentXYZ`  $\{\langle vector \rangle\}$   
 $\{\langle vector \rangle\}$ : the vector which values should be printed

**Description**

Returns the x, y and z component of the given vector.

**Equation**

$$\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3$$

`printVecContentXYZW`  $\{\langle vector \rangle\}$   
 $\{\langle vector \rangle\}$ : the vector which values should be printed

**Description**

Returns the x, y, z and w component of the given vector

**Equation**

$$\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3, \mathbf{v}_4$$

`getVecHeight`  $[\langle out \rangle] \{\langle vector \rangle\}$   
 $[\langle out \rangle]$ : var name to save the result (scalar) to, default: `vecHeightResult`  
 $\{\langle vector \rangle\}$ : the vector of which to return the height

**Description**

Returns the number of components that given vector has (number of dimensions).

**Equation**

$$o = \dim(\mathbf{v})$$

**Example**

```
\newVec{myNewVec}{0.5, 1, 2}
\getVecHeight{\myNewVec}
\vecHeightResult \mapsto 3
```

**printScalarPComp** [*fraction*]{*vector v*}{*vector w*}

[*fraction*]: whether or not the value should be printed as a fraction (val=1), default: 0  
 {*vector v*}: the first vector  
 {*vector w*}: the second vector

**Description**

Print the intermediate components of the scalarproduct calculation.

**Equation**

$$\mathbf{v}_1 * \mathbf{w}_1 + \dots + \mathbf{v}_n * \mathbf{w}_n$$

**Example**

```
\newVec{myNewVecA}{0.5, 1, 2}
\newVec{myNewVecB}{0.5, 1, 2}
\printScalarPComp{\myNewVecA}{\myNewVecB} \mapsto 0.5 * 0.5 + 1 * 1 + 2 * 2
```

**Warning**

When both vectors do not have the same length

**printCrossPComp** [*fraction*]{*vector v*}{*vector w*}

[*fraction*]: whether or not the value should be printed as a fraction (val=1), default: 0  
 {*vector v*}: the first vector  
 {*vector w*}: the second vector

**Description**

Print the intermediate components of the crossproduct calculation.

**Equation**

$$\begin{pmatrix} \mathbf{v}_2 * \mathbf{w}_3 - \mathbf{v}_3 * \mathbf{w}_2 \\ \mathbf{v}_3 * \mathbf{w}_1 - \mathbf{v}_1 * \mathbf{w}_3 \\ \mathbf{v}_1 * \mathbf{w}_2 - \mathbf{w}_2 * \mathbf{w}_1 \end{pmatrix}$$

### Example

```
\newVec{myNewVecA}{0.5, 1, 2}
\newVec{myNewVecB}{3, 4, 5}

\printScalarPComp{\myNewVecA}{\myNewVecB} \mapsto \begin{pmatrix} 1 * 5 - 2 * 4 \\ 2 * 3 - 0.5 * 5 \\ 0.5 * 4 - 1 * 3 \end{pmatrix}
```

### Warning

When one of both vectors does not have length 3

`roundVec` [*out*]{*vector*}{*digits*}

[*out*]: var name to save the result (vector) to, default: roundVecResult  
 {*vector*}: the vector that should be rounded  
 {*digits*}: number of digits after comma

### Description

Round the values of the given vector.

### Equation

$$\mathbf{v} = \begin{pmatrix} \lfloor d_1 \rfloor \\ \dots \\ \lfloor d_n \rfloor \end{pmatrix}$$

### Example

```
\newVec{myNewVec}{0.5, 1, 2}
\roundVec[roundVecResult]{\myNewVec}{0}

\roundVecResult \mapsto \begin{pmatrix} 1 \\ 1 \\ 2 \end{pmatrix}
```

`ceilVec` [*out*]{*vector*}

[*out*]: var name to save the result (vector) to, default: ceilVecResult  
 {*vector*}: the vector that should be ceiled

### Description

Ceil the values of the given vector.

### Equation

$$\mathbf{o} = \begin{pmatrix} \lceil \mathbf{v}_1 \rceil \\ \dots \\ \lceil \mathbf{v}_n \rceil \end{pmatrix}$$

### Example

```
\newVec{myNewVec}{0.5, 1, 2}
\ceilVec[ceilVecResult]{\myNewVec}
\ceilVecResult \mapsto \begin{pmatrix} 1 \\ 1 \\ 2 \end{pmatrix}
```

**floorVec** [*out*]{*vector*}

[*out*]: var name to save the result (vector) to, default: floorVecResult  
 {*vector*}: the vector that should be floored

### Description

Floor the values of the given vector.

### Equation

$$o = \begin{pmatrix} \lfloor \mathbf{v}_1 \rfloor \\ \dots \\ \lfloor \mathbf{v}_n \rfloor \end{pmatrix}$$

### Example

```
\newVec{myNewVec}{0.5, 1, 2}
\floorVec[floorVecResult]{\myNewVec}
\floorVecResult \mapsto \begin{pmatrix} 0 \\ 1 \\ 2 \end{pmatrix}
```

**dehomogenVec** [*out*]{*vector*}

[*out*]: var name to save the result (vector) to, default: dehomogenVecResult  
 {*vector*}: the vector that should be dehomogenized

### Description

Dehomogenize the values of the given vector. This divides all vector components by the last component.

### Equation

$$o = \begin{pmatrix} \frac{\mathbf{v}_1}{\mathbf{v}_n} \\ \dots \\ \frac{\mathbf{v}_{n-1}}{\mathbf{v}_n} \end{pmatrix}$$

### Example

```
\newVec{myNewVec}{0.5, 1, 2, 2}
\dehomogenVec[dehomogenVecResult]{\myNewVec}

\dehomogenVecResult \mapsto \begin{pmatrix} 0.25 \\ 0.5 \\ 1 \\ 1 \end{pmatrix}
```

**addVec** [*out*]{*vector*}{*vectors*}

[*out*]: var name to save the result (vector) to, default: addVecResult

{*vector*}: the vector to add to

{*vectors*}: the list of vectors with which to add

### Description

Add the values of 1 to  $m$  vectors to the values of the given vector.

### Equation

$$o = \begin{pmatrix} \mathbf{v}_1 + \mathbf{v}_{s_{1,1}} + \dots + \mathbf{v}_{s_{m,1}} \\ \dots \\ \mathbf{v}_n + \mathbf{v}_{s_{1,n}} + \dots + \mathbf{v}_{s_{m,n}} \end{pmatrix}$$

### Example

```
\newVec{myNewVecA}{0.5, 1, 2}
\newVec{myNewVecB}{0.5, 1, 2}
\addVec{\myNewVecA}{\myNewVecB}

\addVecResult \mapsto \begin{pmatrix} 1.5 \\ 3 \\ 6 \end{pmatrix}

\newVec{myNewVecA}{0.5, 1, 2}
\newVec{myNewVecB}{0.5, 1, 2}
\newVec{myNewVecC}{0.5, 1, 2}
\addVec[resultName]{\myNewVecA}{\myNewVecB, \myNewVecC}

\resultName \mapsto \begin{pmatrix} 1.5 \\ 3 \\ 6 \end{pmatrix}
```

**subVec** [*out*]{*vector*}{*vectors*}

[*out*]: var name to save the result (vector) to, default: subVecResult

{*vector*}: the vector to subtract from

{*vectors*}: the list of vectors with which to subtract

### Description

Subtract the values of 1 to  $m$  vectors from the values of the given vector.

### Equation

$$o = \begin{pmatrix} \mathbf{v}_1 - \mathbf{vS}_{1,1} - \dots - \mathbf{vS}_{m,1} \\ \dots \\ \mathbf{v}_n - \mathbf{vS}_{1,n} - \dots - \mathbf{vS}_{m,n} \end{pmatrix}$$

### Example

```

\newVec{myNewVecA}{0.5, 1, 2}
\newVec{myNewVecB}{0.5, 1, 2}
\subVec{\myNewVecA}{\myNewVecB}
\subVecResult \mapsto \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}
\newVec{myNewVecA}{0.5, 1, 2}
\newVec{myNewVecB}{0.5, 1, 2}
\newVec{myNewVecC}{0.5, 1, 2}
\subVec[resultName]{\myNewVecA}{\myNewVecB, \myNewVecC}
\resultName \mapsto \begin{pmatrix} -0.5 \\ -1 \\ -2 \end{pmatrix}

```

`mulVec` [*out*]{*vector*}{*vectors*}

[*out*]: var name to save the result (vector) to, default: `mulVecResult`

{*vector*}: the vector to multiply

{*vectors*}: the list of vectors with which to multiply

### Description

Multiply the values of 1 to  $m$  vectors with the values of the given vector. See Hadamard product (element-wise product) for more information.

### Equation

$$o = \mathbf{v} \odot \mathbf{vS}_1 \odot \dots \odot \mathbf{vS}_m$$

$$= \begin{pmatrix} \mathbf{v}_1 * \mathbf{vS}_{1,1} * \dots * \mathbf{vS}_{m,1} \\ \dots \\ \mathbf{v}_n * \mathbf{vS}_{1,n} * \dots * \mathbf{vS}_{m,n} \end{pmatrix}$$

### Example

```

\newVec{myNewVecA}{0.5, 1, 2}
\newVec{myNewVecB}{0.5, 1, 2}
\mulVec{\myNewVecA}{\myNewVecB} \mapsto \begin{pmatrix} 0.25 \\ 1 \\ 4 \end{pmatrix}

```

`divVec` [*out*]{*vector*}{*vectors*}

[*out*]: var name to save the result (vector) to, default: `divVecResult`

{*vector*}: the vector to divide

{*vectors*}: the list of vectors to divide with

#### Description

Divide the values of the given vector by the values of 1 to  $m$  vectors (element-wise division).

#### Equation

$$o = \mathbf{v} \oslash \mathbf{vS}_1 \oslash \dots \oslash \mathbf{vS}_m$$

$$= \begin{pmatrix} \mathbf{v}_1 / \mathbf{vS}_{1,1} / \dots / \mathbf{vS}_{m,1} \\ \dots \\ \mathbf{v}_n / \mathbf{vS}_{1,n} / \dots / \mathbf{vS}_{m,n} \end{pmatrix}$$

#### Example

```
\newVec{myNewVecA}{0.5, 1, 2}
\newVec{myNewVecB}{0.5, 1, 2}

\divVec{\myNewVecA}{\myNewVecB} \mapsto \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}
```

`scaleVec` [*out*]{*vector*}{*scalar*}

[*out*]: var name to save the result (vector) to, default: `scaleVecResult`

{*vector*}: the vector to scale

{*scalar*}: the scalar to scale with

#### Description

Multiply the values of the given vector by a scalar.

#### Equation

$$o = \begin{pmatrix} \mathbf{v}_1 * s \\ \dots \\ \mathbf{v}_n * s \end{pmatrix}$$



### Example

```
\newVec{myNewVecA}{0.5, 1, 2}
\newScalar{myNewScalar}{2}
\scaleVec{\myNewVecA}{\myNewScalar}
\scaleVecResult \mapsto \begin{pmatrix} 1 \\ 2 \\ 4 \end{pmatrix}
```

`divVecScalar` [*out*]{*vector*}{*scalar*}

[*out*]: var name to save the result (vector) to, default: `divVecScalarResult`

{*vector*}: the vector to divide

{*scalar*}: the scalar to divide with

### Description

Divide the values of the given vector by a scalar.

### Equation

$$o = \begin{pmatrix} \frac{v_1}{s} \\ \dots \\ \frac{v_n}{s} \end{pmatrix}$$

### Example

```
\newVec{myNewVecA}{0.5, 1, 2}
\newScalar{myNewScalar}{2}
\divVecScalar{\myNewVecA}{\myNewScalar}
\divVecScalarResult \mapsto \begin{pmatrix} 0.25 \\ 0.5 \\ 1 \end{pmatrix}
```

`negateVec` [*out*]{*vector*}

[*out*]: var name to save the result (vector) to, default: `negateVecResult`

{*vector*}: the vector to negate

### Description

Negate the values of the given vector.

### Equation

$$o = \begin{pmatrix} -v_1 \\ \dots \\ -v_n \end{pmatrix}$$

### Example

```
\newVec{myNewVecA}{0.5, 1, 2}
\negateVecResult{\myNewVecA}
\negateVecResult \mapsto \begin{pmatrix} -0.5 \\ -1 \\ -2 \end{pmatrix}
```

`scaleAndAddVec` [*out*]{*vector v*}{*vector w*}{*scalar*}

[*out*]: var name to save the result (vector) to, default: `scaleAndAddVecResult`  
 {*vector v*}: the vector to add to  
 {*vector w*}: the vector that is multiplied and then added  
 {*scalar*}: the scalar that is the multiplier

### Description

Adds two vectors after multiplying the second one by a scalar.

### Equation

$$o = \mathbf{v} + \mathbf{w} * s = \begin{pmatrix} \mathbf{v}_1 + \mathbf{w}_1 * s \\ \dots \\ \mathbf{v}_n + \mathbf{w}_n * s \end{pmatrix}$$

### Example

```
\newVec{myNewVecA}{0.5, 1, 2}
\newVec{myNewVecB}{0.5, 1, 2}
\newScalar{myNewScalar}{2}
\scaleAndAddVec{\myNewVecA}{\myNewVecB}{\myNewScalar}
\scaleAndAddVecResult \mapsto \begin{pmatrix} 1.5 \\ 3 \\ 6 \end{pmatrix}
```

`lerpVec` [*out*]{*vector v*}{*vector w*}{*scalar t*}

[*out*]: var name to save the result (vector) to, default: `lerpVecResult`  
 {*vector v*}: the start vector  
 {*vector w*}: the end vector  
 {*scalar t*}: interpolation amount range [0-1]

### Description

Linearly interpolate between the values of two vectors (element-wise interpolation).

### Equation

$$o = (1 - t) * \mathbf{v} + t * \mathbf{w} = \begin{pmatrix} (1 - t)\mathbf{v}_1 + t * \mathbf{w}_1 \\ \dots \\ (1 - t)\mathbf{v}_n + t * \mathbf{w}_n \end{pmatrix}$$

### Example

```
\newVec{myNewVecA}{0.5, 1, 2}
\newVec{myNewVecB}{1, 2, 4}
\newScalar{myNewScalar}{0.5}
\lerpVec{\myNewVecA}{\myNewVecB}{\myNewScalar}
\lerpVecResult \mapsto \begin{pmatrix} 0.75 \\ 1.5 \\ 3 \end{pmatrix}
```

`slerpVec` [*out*]{*vector v*}{*vector w*}{*scalar t*}

[*out*]: var name to save the result (vector) to, default: `slerpVecResult`

{*vector v*}: the start vector

{*vector w*}: the end vector

{*scalar t*}: interpolation amount range [0-1]

### Description

Performs a spherical linear interpolation between two vectors. This expects vector *v* and vector *w* to be normalized.

### Equation

TODO

`dotVec` [*out*]{*vector v*}{*vector w*}

[*out*]: var name to save the result (scalar) to, default: `dotVecResult`

{*vector v*}: the first vector

{*vector w*}: the second vector

### Description

Calculate the dotproduct of the given two vectors. Other names for this operation are inner product and scalarproduct.

### Equation

$$o = \mathbf{v}_1 * \mathbf{w}_1 + \dots + \mathbf{v}_n * \mathbf{w}_n$$

### Example

```
\newVec{myNewVecA}{0.5, 1, 2}
\newVec{myNewVecB}{1, 2, 4}
\dotVec{myNewVecA}{myNewVecB}
\dotVecResult ↦ 10.5
\dotVec[resultName]{myNewVecA}{myNewVecB}
\resultName ↦ 10.5
```

### Warning

When both vectors do not have the same length

**lenVec** [*out*]{*vector*}

[*out*]: var name to save the result (scalar) to, default: lenVecResult  
 {*vector*}: the vector for which to calculate the length

### Description

Calculate the length of the given vector.

### Equation

$$o = \|\mathbf{v}\| = \sqrt{\mathbf{v}_1^2 + \dots + \mathbf{v}_n^2}$$

### Example

```
\newVec{myNewVecA}{0.5, 1, 2}
\lenVec{myNewVecA}
\lenVecResult ↦ 2.29
```

**sqrLenVec** [*out*]{*vector*}

[*out*]: var name to save the result (scalar) to, default: sqrLenVecResult  
 {*vector*}: the vector for which to calculate the squared length

### Description

Calculate the squared length of the given vector.

### Equation

$$o = \|\mathbf{v}\|^2 = \mathbf{v}_1^2 + \dots + \mathbf{v}_n^2$$

### Example

```
\newVec{myNewVecA}{0.5, 1, 2}
\sqrLenVec{\myNewVecA}
\sqrLenVecResult ↦ 5.25
```

**normalizeVec** [*out*]{*vector*}

[*out*]: var name to save the result (vector) to, default: normalizeVecResult  
 {*vector*}: the vector which to normalize

### Description

Divide all components of the vector by its length.

### Equation

$$o = \begin{pmatrix} \frac{v_1}{\|v\|} \\ \dots \\ \frac{v_n}{\|v\|} \end{pmatrix}$$

### Example

```
\newVec{myNewVecA}{0.5, 1, 2}
\normalizeVec{\myNewVecA}
\normalizeVecResult ↦ \begin{pmatrix} 0.12 \\ 0.24 \\ 0.96 \end{pmatrix}
\normalizeVec[resultName]{\myNewVecA}
\resultName ↦ \begin{pmatrix} 0.12 \\ 0.24 \\ 0.96 \end{pmatrix}
```

**crossVec** [*out*]{*vector v*}{*vector w*}

[*out*]: var name to save the result (vector) to, default: crossVecResult  
 {*vector v*}: the first vector  
 {*vector w*}: the second vector

### Description

Calculate crossproduct for the two given vectors.

### Equation

$$o = \begin{pmatrix} v_2 * w_3 - v_3 * w_2 \\ v_3 * w_1 - v_1 * w_3 \\ v_1 * w_2 - w_2 * w_1 \end{pmatrix}$$

### Example

```
\newVec{myNewVecA}{0.5, 1, 2}
\newVec{myNewVecB}{0.5, 0, 2}
\crossVec{\myNewVecA}{\myNewVecB}
\crossVecResult \mapsto \begin{pmatrix} 2 \\ 0 \\ -0.5 \end{pmatrix}
```

### Warning

When one of both vectors does not have length 3

`crossVecTwoD` [*out*]{*vector v*}{*vector w*}

[*out*]: var name to save the result (vector) to, default: `crossVecResult`  
 {*vector v*}: the first vector  
 {*vector w*}: the second vector

### Description

Calculate crossproduct for the two given vectors.

### Equation

$$o = \mathbf{v}_1 * \mathbf{w}_2 - \mathbf{v}_2 * \mathbf{w}_1$$

### Example

```
\newVec{myNewVecA}{0.5, 1}
\newVec{myNewVecB}{0.5, 0}
\crossVecTwoD{\myNewVecA}{\myNewVecB}
\crossVecResult \mapsto -0.5
```

### Warning

When one of both vectors does not have length 3

`angleVec` [*out*]{*vector v*}{*vector w*}

[*out*]: var name to save the result (scalar) to, default: `angleVecResult`  
 {*vector v*}: the first vector  
 {*vector w*}: the second vector

### Description

Get the angle between the given two vectors in radian.

**Equation**

$$o = \angle(\mathbf{v}, \mathbf{w})$$

**Example**

```
\newVec{myNewVecA}{0.5, 1, 2}
\newVec{myNewVecB}{0.5, 1, 2}
\angleVec{\myNewVecA}{\myNewVecB}
\angleVecResult \mapsto TODOscalar
```

`distVec` [*out*]{*vector v*}{*vector w*}

[*out*]: var name to save the result (scalar) to, default: `distVecResult`  
 {*vector v*}: the first vector  
 {*vector w*}: the second vector

**Description**

Calculates the euclidian distance between the given two vectors.

**Equation**

$$o = \|\mathbf{v} - \mathbf{w}\|$$

**Example**

```
\newVec{myNewVecA}{0.5, 1, 2}
\newVec{myNewVecB}{0.5, 1, 2}
\distVec{\myNewVecA}{\myNewVecB}
\distVecResult \mapsto TODOscalar
```

**Warning**

When vector *v* and vector *w* do not have the same length

`sqrDistVec` [*out*]{*vector v*}{*vector w*}

[*out*]: var name to save the result (scalar) to, default: `sqrDistVecResult`  
 {*vector v*}: the first vector  
 {*vector w*}: the second vector

**Description**

Calculates the squared euclidian distance between two vectors.

**Equation**

$$o = \|\mathbf{v} - \mathbf{w}\|^2$$

### Example

```
\newVec{myNewVecA}{0.5, 1, 2}
\newVec{myNewVecB}{0.5, 1, 2}
\sqrDistVec{\myNewVecA}{\myNewVecB}
\sqrDistVecResult \mapsto TODOscalar
```

### Warning

When vector v and vector w do not have the same length

**transformVec** [*out*]{*matrix*}{*vector*}

[*out*]: var name to save the result (vector) to, default: transformVecResult

{*matrix*}: the matrix to transform the vector with

{*vector*}: the vector to transform

### Description

Multiply the given vector with a matrix (for transformation).

### Equation

$$o = \mathbf{M} \cdot \mathbf{v} \mid \mathbf{M} \in \mathbb{R}^{m \times n}, \mathbf{v} \in \mathbb{R}^n$$

### Example

```
\newVec{myNewVec}{0.5, 1, 2}
\newMat{myNewMatrix}{\{0.5, 1, 2\}\{4, 5, 6\}\{7, 8, 9\}}
\transformVec{\myNewMatrix}{\myNewVec}
\transformVecResult \mapsto \begin{pmatrix} 5.25 \\ 19 \\ 29.5 \end{pmatrix}
\transformVec[resultName]{\myNewMatrix}{\myNewVec}
\resultName \mapsto \begin{pmatrix} 5.25 \\ 19 \\ 29.5 \end{pmatrix}
```

### Warning

When matrix and vector have a dimension mismatch

**tensorVec** [*out*]{*vector*}

[*out*]: var name to save the result (matrix) to, default: tensorVecResult

{*vector*}: the vector to multiply with itself



### Description

Outer product: Multiply a vector (column) with itself (row) resulting in a matrix. (Dyadic product, Tensor product)

### Equation

$$o = \mathbf{v}\mathbf{v}^T = \mathbf{v} \otimes \mathbf{v}$$

### Warning

When matrix and vector have a dimension mismatch

`minVec` [*out*]{*vector v*}{*vector w*}

[*out*]: var name to save the result (vector) to, default: `minVecResult`

{*vector v*}: the first vector

{*vector w*}: the second vector

### Description

Return the componentwise minimum of the given two vectors.

### Equation

$$o = \begin{pmatrix} \min(\mathbf{v}_1, \mathbf{w}_1) \\ \dots \\ \min(\mathbf{v}_n, \mathbf{w}_n) \end{pmatrix}$$

### Warning

When vector *v* and vector *w* do not have the same length

`maxVec` [*out*]{*vector v*}{*vector w*}

[*out*]: var name to save the result (vector) to, default: `maxVecResult`

{*vector v*}: the first vector

{*vector w*}: the second vector

### Description

Return the componentwise maximum of the given two vectors.

### Equation

$$o = \begin{pmatrix} \max(\mathbf{v}_1, \mathbf{w}_1) \\ \dots \\ \max(\mathbf{v}_n, \mathbf{w}_n) \end{pmatrix}$$

### Warning

When vector  $v$  and vector  $w$  do not have the same length

`solveVecEQL` [ $\langle out \rangle$ ] { $\langle vector v \rangle$ } { $\langle vector w \rangle$ }  
 [ $\langle out \rangle$ ]: var name to save the result (scalar) to, default: solveVecEQLResult  
 { $\langle vector v \rangle$ }: the first vector  
 { $\langle vector w \rangle$ }: the second vector

### Description

Solve the equation  $vectorv * x + vectorw = 0$  for  $x$ . The command expects the input to be two vectors of the same length. The command iterates both vectors componentwise and solves the equation per component. If the result does not match across all components the equation can not be solved.

### Equation

$$0 = \frac{-w_1}{v_1} = \dots = \frac{-w_n}{v_n}$$

## 4 Implementation

### 4.1 matrix

Command	Arguments
<code>\newMat</code>	{ $\langle out \rangle$ }, { $\langle scalars \rangle$ }
<code>\newMatFromRow</code>	{ $\langle out \rangle$ }, { $\langle row \rangle$ }
<code>\newMatFromRowVecs</code>	{ $\langle out \rangle$ }, { $\langle rows \rangle$ }
<code>\newMatFromColVecs</code>	{ $\langle out \rangle$ }, { $\langle columns \rangle$ }
<code>\newEmptyMat</code>	{ $\langle out \rangle$ }
<code>\newDiagMat</code>	{ $\langle out \rangle$ }, { $\langle scalars \rangle$ }
<code>\appendRowToMat</code>	{ $\langle out \rangle$ }, { $\langle matrix \rangle$ }, { $\langle vector r \rangle$ }
<code>\newTransMat</code>	{ $\langle out \rangle$ }, { $\langle vector t \rangle$ }
<code>\newScaleMat</code>	{ $\langle out \rangle$ }, { $\langle vector s \rangle$ }
<code>\newRotMatTwoD</code>	[ $\langle out \rangle$ ], { $\langle scalar \rangle$ } rotMatResult
<code>\newRotMat</code>	[ $\langle out \rangle$ ], { $\langle vector \rangle$ }, { $\langle scalar \rangle$ } rotMatResult
<code>\newLookAtMat</code>	[ $\langle out \rangle$ ], { $\langle right \rangle$ }, { $\langle up \rangle$ }, { $\langle dir \rangle$ }, { $\langle pos \rangle$ } lookAtMatResult
<code>\newProjMat</code>	{ $\langle out \rangle$ }
<code>\newProjMatGen</code>	{ $\langle out \rangle$ }
<code>\newFrustumMat</code>	[ $\langle out \rangle$ ], { $\langle fovx \rangle$ }, { $\langle fovy \rangle$ }, { $\langle near \rangle$ }, { $\langle far \rangle$ } frustMatResult
<code>\printMat</code>	[ $\langle fraction \rangle$ ], { $\langle matrix \rangle$ } 0
<code>\printMatDiag</code>	[ $\langle fraction \rangle$ ], { $\langle matrix \rangle$ } 0

<code>\getMatHeight</code>	<code>[\langle out \rangle], \{\langle matrix \rangle\}</code> matHeightResult
<code>\getMatWidth</code>	<code>[\langle out \rangle], \{\langle matrix \rangle\}</code> matWidthResult
<code>\transposeMat</code>	<code>[\langle out \rangle], \{\langle matrix \rangle\}</code> transposeMatResult
<code>\addMat</code>	<code>[\langle out \rangle], \{\langle matrix \rangle\}, \{\langle matrices \rangle\}</code> addMatResult
<code>\subMat</code>	<code>[\langle out \rangle], \{\langle matrix \rangle\}, \{\langle matrices \rangle\}</code> subMatResult
<code>\mulMatScalar</code>	<code>[\langle out \rangle], \{\langle matrix \rangle\}, \{\langle scalar \rangle\}</code> mulMatScalarResult
<code>\mulMatVec</code>	<code>[\langle out \rangle], \{\langle matrix \rangle\}, \{\langle vector \rangle\}</code> mulMatVecResult
<code>\mulMat</code>	<code>[\langle out \rangle], \{\langle matrix M \rangle\}, \{\langle matrix N \rangle\}</code> mulMatResult
<code>\getMatTranslation</code>	<code>[\langle out \rangle], \{\langle matrix \rangle\}</code> matTranslationResult
<code>\detMat</code>	<code>[\langle out \rangle], \{\langle matrix \rangle\}</code> detMatResult

`\newMat` Create a new matrix from the given values.  
 #1 - out: var name to save the result (matrix) to  
 #2 - scalars: matrix values that are assigned to #1

```

1 \newcommand\newMat[2]{
2   \xintFor* ##1 in #2 \do {%
3     \newVec{matRow}{##1}
4     \xintifForFirst{%
5       \newMatFromRow{#1}{\matRow}
6     }{%
7       \appendRowToMat{#1}{\csname #1\endcsname}{\matRow}
8     }
9   }
10 }
```

`\newMatFromRow` Create a new matrix from the given row.  
 #1 - out: var name to save the result (matrix) to  
 #2 - row: matrix row that is assigned to #1

```

11 \newcommand\newMatFromRow[2]{
12   \newScalar{tmpWidth}{\expandafter\xintLength\expandafter{#2}}%
13   \FPifEq{\tmpWidth}{1}
14     \expandafter\edef\csname #1\endcsname{\xintCSVtoList{\matRow}}
15   \else
16     \expandafter\edef\csname #1\endcsname{\xintCSVtoList{\matRow}}
17   \fi
18 }
```

`\newMatFromRowVecs` Create a new matrix from the given row vectors.  
 #1 - out: var name to save the result (matrix) to  
 #2 - rows: the rows that are used to construct the matrix

```

19 \newcommand\newMatFromRowVecs [2] {%
20     % TODO remove this?
21     \FPset\numParam{\expandafter\xintLength{#2}}%
22     \FPsub\numParam\numParam{1}
23     \FPdiv\numParam\numParam{2}
24     \FPadd\numParam\numParam{1}
25     % In case only one item/row was given the
26     % matrix-datastructure must be adjusted
27     \FPifeq{\numParam}{1}
28         \expandafter\edef\csname #1\endcsname{\xintCSVtoList{#2}}%
29     \else
30         \expandafter\edef\csname #1\endcsname{\xintCSVtoList{#2}}%
31     \fi
32 }
```

`\newMatFromColVecs` Create a new matrix from the given column vectors.  
 #1 - out: var name to save the result (matrix) to  
 #2 - columns: the columns that are used to construct the matrix

```

33 \newcommand\newMatFromColVecs [2] {%
34     \edef\tmpVecList{\xintCSVtoList{#2}}%
35     \copyVec[tmpHeightVec]{\xintNthElt{1}{\tmpVecList}}
36     \getVecHeight[numRows]{\tmpHeightVec}
37     \edef\numColumns{\expandafter\xintLength\expandafter{\tmpVecList}}
38     \edef\colSequence{\xintSeq[+1]{+1}{\numColumns}}
39     \edef\rowSequence{\xintSeq[+1]{+1}{\numRows}}
40     \newEmptyMat{#1}
41     \xintFor* ##1 in \colSequence \do {%
42         \xintFor* ##2 in \rowSequence \do {%
43             \newScalar{tmpValue}{\xintNthElt{##1}{\xintNthElt{##2}{\tmpVecList}}}
44             \xintifForFirst{%
45                 \newVec{matRow}{\tmpValue}
46             }{%
47                 \appendToVec{matRow}{\matRow}{\tmpValue}
48             }
49         }
50     \appendToMat{#1}{\csname #1\endcsname}{\matRow}
51 }
52 }
```

`\newEmptyMat` Create a new empty matrix. Obacht: It is not necessary to call this to create a new matrix before using any of the arithmetic commands.

#1 - out: var name to save the result (matrix) to

```

53 \newcommand\newEmptyMat [1] {
54     \expandafter\edef\csname #1\endcsname{}%
55 }
```

`\newDiagMat` Create a new diagonal matrix from the given vector.

#1 - out: var name to save the result (matrix) to

#2 - scalars: the values that define the diagonal of the matrix

```

56 \newcommand\newDiagMat[2]{
57   \if\noexpand#1\relax
58     \PackageError{glmatrix package}{argument mismatch}{Argument 1 of newDiagMat seems to
59   \else
60     \edef\tmpVector{\xintCSVtoList{#2}}%
61     \newVec{rowA}{\xintNthElt{1}{\tmpVector}, 0, 0}
62     \newVec{rowB}{0, \xintNthElt{2}{\tmpVector}, 0}
63     \newVec{rowC}{0, 0, \xintNthElt{3}{\tmpVector}}
64     \newMatFromRowVecs{#1}{\rowA, \rowB, \rowC}
65   \fi
66 }

```

**\appendRowToMat** Append the given row to the given matrix.

#1 - out: var name to save the result (matrix) to

#2 - matrix: the matrix to append the vector to

#3 - vector r: the row vector that should be appended to the matrix

```

67 \newcommand\appendRowToMat[3]{
68   \newScalar{tmpWidth}{\expandafter\xintLength\expandafter{#3}}%
69   \FPifeq{\tmpWidth}{1}
70     \expandafter\edef\csname #1\endcsname{#2 {\xintCSVtoList{#3}}}%
71   \else
72     \expandafter\edef\csname #1\endcsname{#2 \xintCSVtoList{#3}}%
73   \fi
74 }

```

**\newTransMat** Create translation matrix from the given vector.

#1 - out: var name to save the result (matrix) to

#2 - vector t: the translation vector to construct the matrix from

```

75 \newcommand\newTransMat[2]{%
76   \edef\vecLength{\expandafter\xintLength\expandafter{#2 1}}
77   \edef\innersequence{\xintSeq[+1]{+1}{\expandafter\xintLength\expandafter{#2}+1}}
78   \xintFor* ##1 in \innersequence \do {%
79     \xintifForFirst{%
80       \xintFor* ##2 in \innersequence \do {%
81         \newScalar{tmp}{0}
82         \FPifeq{##2}{\vecLength}
83           \setScalar{tmp}{\xintNthElt{##1}{#2}}
84         \else
85           \fi
86         \FPifeq{##1}{##2}
87           \setScalar{tmp}{1}
88         \else
89           \fi
90       \xintifForFirst{%
91         \newVec{matRow}{\tmp}
92       }{%
93         \appendToVec{matRow}{\matRow}{\tmp}
94       }
95     }
96     \newMatFromRow{#1}{\matRow}

```

```

97     }{%
98         \xintFor* ##2 in \innersequence \do {%
99             \newScalar{tmp}{0}
100            \FPifeq{##2}{\vecLength}
101                \setScalar{tmp}{\xintNthElt{##1}{##2}}
102            \else
103                \fi
104            \FPifeq{##1}{##2}
105                \setScalar{tmp}{1}
106            \else
107                \fi
108            \xintifForFirst{%
109                \newVec{matRow}{\tmp}
110            }{%
111                \appendToVec{matRow}{\matRow}{\tmp}
112            }
113        }
114        \appendRowToMat{#1}{\csname #1\endcsname}{\matRow}
115    }
116 }
117 }

```

`\newScaleMat` Create scaling matrix from the given vector.

#1 - out: var name to save the result (matrix) to

#2 - vector s: the scaling vector to construct the matrix from

```

118 \newcommand\newScaleMat[2]{%
119     \edef\vecLength{\expandafter\xintLength\expandafter{#2 1}}
120     \edef\innersequence{\xintSeq[+1]{+1}{\expandafter\xintLength\expandafter{#2}+1}}
121     \xintFor* ##1 in \innersequence \do {%
122         \xintifForFirst{%
123             \xintFor* ##2 in \innersequence \do {%
124                 \FPiflt{##2}{\vecLength}
125                     \setScalar{tmp}{\xintNthElt{##1}{##2}}
126                 \else
127                     \setScalar{tmp}{1}
128                 \fi
129                 \FPifeq{##1}{##2}
130                 \else
131                     \setScalar{tmp}{0}
132                 \fi
133                 \xintifForFirst{%
134                     \newVec{matRow}{\tmp}
135                 }{%
136                     \appendToVec{matRow}{\matRow}{\tmp}
137                 }
138             }
139             \newMatFromRow{#1}{\matRow}
140         }{%
141             \xintFor* ##2 in \innersequence \do {%
142                 \FPiflt{##2}{\vecLength}
143                     \setScalar{tmp}{\xintNthElt{##1}{##2}}
144                 \else
145                     \setScalar{tmp}{1}

```

```

146         \fi
147         \FPifeq{##1}{##2}
148         \else
149             \setScalar{tmp}{0}
150         \fi
151         \xintifForFirst{%
152             \newVec{matRow}{\tmp}
153         }{%
154             \appendToVec{matRow}{\matRow}{\tmp}
155         }
156     }
157     \appendRowToMat{#1}{\csname #1\endcsname}{\matRow}
158 }
159 }
160 }

```

`\newRotMatTwoD` Create rotation matrix from the given scalar (2D).

#1 - out: var name to save the result (matrix) to

#2 - scalar: the rotation value

```

161 \newcommand\newRotMatTwoD[2][rotMatResult]{%
162     \FPcos\tmpCos{#2}
163     \FPSin\tmpSin{#2}
164     \newVec{rowB}{\tmpSin, \tmpCos, 0}
165     \FPmul\tmpSin\tmpSin{-1}
166     \newVec{rowA}{\tmpCos, \tmpSin, 0}
167     \newVec{rowC}{0, 0, 1}
168     \newMatFromRowVecs{#1}{\rowA, \rowB, \rowC}
169 }

```

`\newRotMat` Create a rotation matrix using the rodrigues' matrix

#1 - out: var name to save the result (matrix) to

#2 - vector: the axis to rotate around

#3 - scalar: the angle in degree to rotate with (counterclockwise)

```

170 \newcommand\newRotMat[3][rotMatResult]{%
171     \normalizeVec[normalizedAxis]{#2}
172     \toRadian[angleInRadian]{#3}
173     \FPcos\tmpCos{angleInRadian}
174     \FPSin\tmpSin{angleInRadian}
175     \subScalar[tmpCosMOne]{1}{\tmpCos}
176     \mulScalar[xsquared]{\xintNthElt{1}{\normalizedAxis}}{\xintNthElt{1}{\normalizedAxis}}
177     \mulScalar[xy]{\xintNthElt{1}{\normalizedAxis}}{\xintNthElt{2}{\normalizedAxis}}
178     \mulScalar[xz]{\xintNthElt{1}{\normalizedAxis}}{\xintNthElt{3}{\normalizedAxis}}
179     \mulScalar[ysquared]{\xintNthElt{2}{\normalizedAxis}}{\xintNthElt{2}{\normalizedAxis}}
180     \mulScalar[yz]{\xintNthElt{2}{\normalizedAxis}}{\xintNthElt{3}{\normalizedAxis}}
181     \mulScalar[zsquared]{\xintNthElt{3}{\normalizedAxis}}{\xintNthElt{3}{\normalizedAxis}}
182     %
183     \mulScalar[xsin]{\xintNthElt{1}{\normalizedAxis}}{\tmpSin}
184     \mulScalar[ysin]{\xintNthElt{2}{\normalizedAxis}}{\tmpSin}
185     \mulScalar[zsine]{\xintNthElt{3}{\normalizedAxis}}{\tmpSin}
186     %
187     \mulScalar[oneone]{xsquared}{\tmpCosMOne}
188     \addScalar[oneone]{oneone}{\tmpCos}

```

```

189 \mulScalar[onetwo]{\xy}{\tmpCosMOne}
190 \subScalar[onetwo]{\onetwo}{\zsin}
191 \mulScalar[onethree]{\xz}{\tmpCosMOne}
192 \addScalar[onethree]{\onethree}{\ysin}
193 %
194 \mulScalar[twoone]{\xsquared}{\tmpCosMOne}
195 \addScalar[twoone]{\twoone}{\zsin}
196 \mulScalar[twotwo]{\xy}{\tmpCosMOne}
197 \addScalar[twotwo]{\twotwo}{\tmpCos}
198 \mulScalar[twothree]{\xz}{\tmpCosMOne}
199 \subScalar[twothree]{\twothree}{\xsin}
200 %
201 \mulScalar[threeone]{\xsquared}{\tmpCosMOne}
202 \subScalar[threeone]{\threeone}{\ysin}
203 \mulScalar[threetwo]{\xy}{\tmpCosMOne}
204 \addScalar[threetwo]{\threetwo}{\xsin}
205 \mulScalar[threethree]{\xz}{\tmpCosMOne}
206 \addScalar[threethree]{\threethree}{\tmpCos}
207 %
208 \newVec{rowA}{\oneone, \onetwo, \onethree}
209 \newVec{rowB}{\twoone, \twotwo, \twothree}
210 \newVec{rowC}{\threeone, \threetwo, \threethree}
211 \newMatFromRowVecs{#1}{\rowA, \rowB, \rowC}
212 }

```

`\newLookAtMat` Create a new lookAt matrix.

#1 - out: var name to save the result (matrix) to

#2 - right: the right vector

#3 - up: the up vector

#4 - dir: the direction vector

#5 - pos: the camera position vector

```

213 \newcommand\newLookAtMat[5][lookAtMatResult]{%
214 \dotVec[scalarRC]{#2}{#5}
215 \mulScalar[scalarRC]{\scalarRC}{-1}
216 \copyVec[tmpVecA]{#2}
217 \appendToVec{tmpVecA}{\tmpVecA}{\scalarRC}
218 % \printVec[1]{\tmpVecA}
219 %
220 \dotVec[scalarUC]{#3}{#5}
221 \mulScalar[scalarUC]{\scalarUC}{-1}
222 \copyVec[tmpVecB]{#3}
223 \appendToVec{tmpVecB}{\tmpVecB}{\scalarUC}
224 %
225 \dotVec[scalarDC]{#4}{#5}
226 \copyVec[tmpVecC]{#4}
227 \negateVec[tmpVecC]{\tmpVecC}
228 \appendToVec{tmpVecC}{\tmpVecC}{\scalarDC}
229 %
230 \newVec{tmpVecD}{0, 0, 0, 1}
231 \newMatFromRowVecs{#1}{\tmpVecA, \tmpVecB, \tmpVecC, \tmpVecD}
232 }

```



`\newProjMat` Create a new default projection matrix.

#1 - out: var name to save the result (matrix) to

```
233 \newcommand\newProjMat[1]{%
234     \newVec{tmpVecA}{1, 0, 0, 0}
235     \newVec{tmpVecB}{0, 1, 0, 0}
236     \newVec{tmpVecC}{0, 0, 1, 0}
237     \newVec{tmpVecD}{0, 0, -1, 0}
238     \newMatFromRowVecs{#1}{\tmpVecA, \tmpVecB, \tmpVecC, \tmpVecD}
239 }
```

`\newProjMatGen` Create a new projection matrix from normal and distance from origin.

#1 - out: var name to save the result (matrix) to

```
240 \newcommand\newProjMatGen[3]{%
241     \divScalar[glmNxDelta]{\xintNthElt{1}{#2}}{#3}
242     \divScalar[glmNyDelta]{\xintNthElt{2}{#2}}{#3}
243     \divScalar[glmNzDelta]{\xintNthElt{3}{#2}}{#3}
244     \newVec{tmpVecA}{1, 0, 0, 0}
245     \newVec{tmpVecB}{0, 1, 0, 0}
246     \newVec{tmpVecC}{0, 0, 1, 0}
247     \newVec{tmpVecD}{\glmNxDelta, \glmNyDelta, \glmNzDelta, 0}
248     \newMatFromRowVecs{#1}{\tmpVecA, \tmpVecB, \tmpVecC, \tmpVecD}
249 }
```

`\newFrustumMat` Create a new frustum matrix.

#1 - out: var name to save the result (matrix) to

#2 - fovx: the fovx scalar

#3 - fovy: the fovy scalar

#4 - near: the position of the near plane (scalar)

#5 - far: the position of the far plane (scalar)

```
250 \newcommand\newFrustumMat[5][frustumResult]{%
251     \FPdiv\tmp{#2}{2}
252     \FPtan\frustumRight\tmp
253     \FPmul\frustumRight{#4}\frustumRight
254     \FPmul\frustumLeft\frustumRight{-1}
255     \FPdiv\tmp{#3}{2}
256     \FPtan\frustumTop\tmp
257     \FPmul\frustumTop{#4}\frustumTop
258     \FPmul\frustumBottom\frustumTop{-1}
259     %
260     \FPmul\cellAA{2}{#4} % AA
261     \FPsub\tmp\frustumRight\frustumLeft
262     \FPdiv\cellAA\cellAA\tmp
263     \FPadd\cellAC\frustumRight\frustumLeft % AC
264     \FPdiv\cellAC\cellAC\tmp
265     \FPmul\cellBB{2}{#4} % BB
266     \FPsub\tmp\frustumTop\frustumBottom
267     \FPdiv\cellBB\cellBB\tmp
268     \FPadd\cellBC\frustumTop\frustumBottom % BC
269     \FPdiv\cellBC\cellBC\tmp
270     \FPmul\cellCC{#5}{-1} % CC
271     \FPsub\cellCC\cellCC{#4}
```

```

272 \FPsub\tmp{#5}{#4}
273 \FPdiv\cellCC\cellCC\tmp
274 \FPmul\cellCD{#5}{#4} % CD
275 \FPmul\cellCD{-2}\cellCD
276 \FPdiv\cellCD\cellCD\tmp
277 %
278 \newVec{rowA}{\cellAA, 0, \cellAC, 0}
279 \newVec{rowB}{0, \cellBB, \cellBC, 0}
280 \newVec{rowC}{0, 0, \cellCC, \cellCD}
281 \newVec{rowD}{0, 0, -1, 0}
282 \newMatFromRowVecs{#1}{\rowA, \rowB, \rowC, \rowD}
283 }

```

`\printMat` Print the values of this matrix variable.

#1 - fraction: whether or not the value should be printed as a fraction (val=1)

#2 - matrix: the matrix that should be printed

```

284 \newcommand\printMat[2][0]{%
285   \ensuremath{%
286     \begin{bmatrix}
287       \xintFor* ##1 in #2 \do {%
288         \xintFor* ##2 in {##1} \do {%
289           \xintifForLast{%
290             \xintifForFirst}{&}
291             \FPifeq{#1}{0}
292             \roundScalar{##2}
293             \scalarRoundResult
294           \else
295             \numToFractionA{##2}
296           \fi
297           \\
298         }{%
299           \xintifForFirst}{&}
300           \FPifeq{#1}{0}
301           \roundScalar{##2}
302           \scalarRoundResult
303         \else
304           \numToFractionA{##2}
305         \fi
306       }
307     }
308   }
309 \end{bmatrix}
310 }
311 }

```

`\printMatDiag` Print only the diagonal values of this matrix variable.

#1 - fraction: whether or not the value should be printed as a fraction (val=1)

#2 - matrix: the matrix that should be printed

```

312 \newcommand\printMatDiag[2][0]{%
313   \ensuremath{%
314     \text{diag}\left(
315       \xintNthElt{1}{\xintNthElt{1}{#2}},

```

```

316     \xintNthElt{2}{\xintNthElt{2}{#2}},
317     \xintNthElt{3}{\xintNthElt{3}{#2}}
318     \right)
319   }
320 }

```

**\getMatHeight** Get the height of the given matrix.  
 #1 - out: var name to save the result (scalar) to  
 #2 - matrix: the matrix from which to get the height

```

321 \newcommand\getMatHeight[2][matHeightResult]{%
322   \newScalar{#1}{\expandafter\xintLength\expandafter{#2}}%
323 }

```

**\getMatWidth** Get the width of the given matrix.  
 #1 - out: var name to save the result (scalar) to  
 #2 - matrix: the matrix from which to get the width

```

324 \newcommand\getMatWidth[2][matWidthResult]{%
325   \edef\rowElements{\xintNthElt{1}{#2}}%
326   \newScalar{#1}{\expandafter\xintLength\expandafter{\rowElements}}%
327 }

```

**\transposeMat** Transpose a given Matrix. The parameter can also be a vector, but the result will be a matrix.  
 #1 - out: var name to save the result (matrix) to  
 #2 - matrix: the matrix to transpose

```

328 \newcommand\transposeMat[2][transposeMatResult]{%
329   \getMatWidth{#2}
330   \getMatHeight{#2}
331   \edef\widthSequence{\xintSeq[+1]{+1}{\matWidthResult}}
332   \edef\heightSequence{\xintSeq[+1]{+1}{\matHeightResult}}
333   \newEmptyMat{#1}
334   \xintFor* ##1 in \widthSequence \do {%
335     \xintFor* ##2 in \heightSequence \do {%
336       \FPset\tmpValue{\xintNthElt{##1}{\xintNthElt{##2}{#2}}}
337       \xintifForFirst{%
338         \newVec{rowVec}{\tmpValue}
339       }{%
340         \appendToVec{rowVec}{\rowVec}{\tmpValue}
341       }
342     }
343   \appendToMat{#1}{\csname #1\endcsname}{\rowVec}
344 }
345 }

```

**\addMat** Add two or more matrices.  
 #1 - out: var name to save the result (matrix) to  
 #2 - matrix: the matrix to add to  
 #3 - matrices: the matrices to add

```

346 \newcommand\addMat[3][addMatResult]{%

```

```

347 \if\noexpand#1\relax
348   \PackageError{glmatrix package}{argument mismatch}{Argument 1 of addMat seems to be
349 \else
350 \fi
351 \if\noexpand#2\relax
352 \else
353   \PackageError{glmatrix package}{argument mismatch}{argument 2 of addMat does not see
354 \fi
355 % TODO this does not work if it is a list
356 % \if\noexpand#3\relax
357 % \else
358 %   \PackageError{glmatrix package}{argument mismatch}{argument 3 of addMat does not s
359 % \fi
360 \def\matlist{\xintCSVtoList{#3}}
361 \edef\elemsRow{\xintNthElt{1}{\xintNthElt{1}{\matlist}}}
362 \edef\widthsequence{\xintSeq[+1]{+1}{\expandafter\xintLength\expandafter{\elemsRow}}}%
363 \edef\elemsMat{\xintNthElt{1}{\matlist}}
364 \edef\heightsequence{\xintSeq[+1]{+1}{\expandafter\xintLength\expandafter{\elemsMat}}}%
365 %
366 \newEmptyMat{#1}
367 \xintFor* ##2 in \heightsequence \do {%
368   \xintFor* ##3 in \widthsequence \do {%
369     \FPset\tmpValue{\xintNthElt{##3}{\xintNthElt{##2}{#2}}}
370     \xintifForFirst{%
371       \xintFor* ##1 in \matlist \do {%
372         \FPadd\tmpValue\tmpValue{\xintNthElt{##3}{\xintNthElt{##2}{##1}}}
373       }
374       \newVec{rowVec}{\tmpValue}
375     }{%
376       % \xintifForLast{
377       % }{
378       \xintFor* ##1 in \matlist \do {%
379         \FPadd\tmpValue\tmpValue{\xintNthElt{##3}{\xintNthElt{##2}{##1}}}
380       }
381       \appendToVec{rowVec}{\rowVec}{\tmpValue}
382     } }
383   }
384 }
385 \appendRowToMat{#1}{\csname #1\endcsname}{\rowVec}
386 }
387 }

```

`\subMat` Subtract two or more matrices.

`#1` - out: var name to save the result (matrix) to

`#2` - matrix: the matrix to subtract from

`#3` - matrices: the matrices to subtract

```

388 \newcommand\subMat[3][subMatResult]{%
389 % \getMatWidth{#2}
390 % \edef\widthsequence{\xintSeq[+1]{+1}{\matWidthResult}}%
391 \getMatHeight{#2}
392 \edef\heightsequence{\xintSeq[+1]{+1}{\matHeightResult}}%
393 %
394 \def\matlist{\xintCSVtoList{#3}}

```

```

395 \edef\elemsRow{\xintNthElt{1}{\xintNthElt{1}{\matlist}}}
396 \edef\widthsequence{\xintSeq[+1]{+1}{\expandafter\xintLength\expandafter{\elemsRow}}}%
397 % \edef\elemsMat{\xintNthElt{1}{\matlist}}
398 % \edef\heightsequence{\xintSeq[+1]{+1}{\expandafter\xintLength\expandafter{\elemsMat}}}%
399 %
400 \newEmptyMat{#1}
401 \xintFor* ##2 in \heightsequence \do {%
402   \xintFor* ##3 in \widthsequence \do {%
403     \FPset\tmpValue{\xintNthElt{##3}{\xintNthElt{##2}{#2}}}
404     \xintifForFirst{%
405       \xintFor* ##1 in \matlist \do {%
406         \FPsub\tmpValue\tmpValue{\xintNthElt{##3}{\xintNthElt{##2}{##1}}}
407       }
408       \newVec{rowVec}{\tmpValue}
409     }{%
410       \xintFor* ##1 in \matlist \do {%
411         \FPsub\tmpValue\tmpValue{\xintNthElt{##3}{\xintNthElt{##2}{##1}}}
412       }
413       \appendToVec{rowVec}{\rowVec}{\tmpValue}
414     }
415   }
416   \appendRowToMat{#1}{\csname #1\endcsname}{\rowVec}
417 }
418 }

```

`\mulMatScalar` Multiply the given matrix with a scalar.

`#1` - out: var name to save the result (matrix) to

`#2` - matrix: the matrix to multiply

`#3` - scalar: the scalar to multiply with

```

419 \newcommand\mulMatScalar[3][mulMatScalarResult]{%
420   \if\noexpand#1\relax
421     \PackageError{glmatrix package}{argument mismatch}{Argument 1 of mulMatScalar seems
422   }
423   \fi
424   \if\noexpand#2\relax
425     \else
426     \PackageError{glmatrix package}{argument mismatch}{argument 2 of mulMatScalar does n
427   }
428   \fi
429   \edef\elemOne{\xintNthElt{1}{#2}}
430   \edef\innersequence{\xintSeq[+1]{+1}{\expandafter\xintLength\expandafter{\elemOne}}}%
431   %
432   \newEmptyMat{#1}
433   \xintFor* ##1 in #2 \do {%
434     \xintFor* ##2 in \innersequence \do {%
435       \FPset\elem{\xintNthElt{##2}{##1}}
436       \FPmul\elem\elem{#3}
437       \xintifForFirst{%
438         \newVec{rowVec}{\elem}
439       }{%
440         \appendToVec{rowVec}{\rowVec}{\elem}
441       }
442     }
443   }
444   \appendRowToMat{#1}{\csname #1\endcsname}{\rowVec}

```

```
443 }
444 }
```

`\mulMatVec` Multiply the given matrix with a vector (for transformation).

#1 - out: var name to save the result (vector) to

#2 - matrix: the matrix to transform the vector with

#3 - vector: the vector to transform

```
445 \newcommand\mulMatVec[3][mulMatVecResult]{%
446   \if\noexpand#1\relax
447     \PackageError{glmatrix package}{argument mismatch}{Argument 1 of transformVec seems
448   \else
449   \fi
450   \if\noexpand#2\relax
451   \else
452     \PackageError{glmatrix package}{argument mismatch}{argument 2 of transformVec does n
453   \fi
454   \if\noexpand#3\relax
455   \else
456     \PackageError{glmatrix package}{argument mismatch}{argument 3 of transformVec does n
457   \fi
458   \getVecHeight{#3}%
459   \getMatHeight{#2}%
460   \getMatWidth{#2}%
461   % Check that matrix and vector can be multiplied
462   \FPifeq{\matWidthResult}{\vecHeightResult}
463   \else
464     \PackageError{glmatrix package}{Dimension mismatch for matrix and vector, could not
465   \fi
466   % Create iterator sequences from height of vector and matrix
467   \edef\vecHeightSequence{\xintSeq[+1]{+1}{\vecHeightResult}}%
468   \edef\matHeightSequence{\xintSeq[+1]{+1}{\matHeightResult}}%
469   % Iterate all rows of the matrix
470   \xintFor* ##2 in \matHeightSequence \do {%
471     \FPset\cellVal{0}
472     % Differentiate between first and other entries to create the new vector variable
473     \xintifForFirst{%
474       % Now iterate each entry of the current row
475       % Use this indec to access the entry in the matrix and in the vector
476       \xintFor* ##3 in \vecHeightSequence \do {%
477         \FPMul\tmp{%
478           \xintNthElt{##3}{\xintNthElt{##2}{#2}}%
479         }{%
480           \xintNthElt{##3}{#3}%
481         }
482         % Add to storage variable
483         \FPadd\cellVal\cellVal\tmp
484       }
485       \FPclip\cellVal\cellVal
486       \newVec{#1}{\cellVal}
487     }{
488     \xintFor* ##3 in \vecHeightSequence \do {%
489       \FPMul\tmp{%
490         \xintNthElt{##3}{\xintNthElt{##2}{#2}}%
```

```

491         }{%
492         \xintNthElt{##3}{#3}%
493         }
494         \FPadd\cellVal\cellVal\tmp
495     }
496     \FPclip\cellVal\cellVal
497     \appendToVec{#1}{\csname #1\endcsname}{\cellVal}
498 }
499 }
500 % Test if width and height is correct
501 \expandafter\getVecHeight\expandafter{\csname #1\endcsname}%
502 \FPifeq{\matHeightResult}{\vecHeightResult}
503 \else
504     \PackageError{glmatrix package}{Vector height does not match expectation}{height of
505 \fi
506 }

```

`\mulMat` Multiply the given matrix with another matrix.

#1 - out: var name to save the result (matrix) to

#2 - matrix M: the matrix to multiply

#3 - matrix N: the matrix to multiply with

```

507 \newcommand\mulMat[3][mulMatResult]{%
508     \edef\elemsRow{\xintNthElt{1}{#3}}
509     \edef\widthsequence{\xintSeq[+1]{+1}{\expandafter\xintLength\expandafter{\elemsRow}}}%
510     %
511     % TODO remove one of these
512     \edef\elemsMat{#2}
513     \edef\heightsequence{\xintSeq[+1]{+1}{\expandafter\xintLength\expandafter{\elemsMat}}}%
514     %
515     \edef\elemsMat{#3}
516     \edef\heightsequenceB{\xintSeq[+1]{+1}{\expandafter\xintLength\expandafter{\elemsMat}}}%
517     %
518     \edef\elemsRowTmp{\xintNthElt{1}{#2}}
519     \FPset\widthMatA{\expandafter\xintLength\expandafter{\elemsRowTmp}}%
520     \FPset\heightMatB{\expandafter\xintLength\expandafter{\elemsMat}}%
521     % TODO check this
522     \FPifeq{\widthMatA}{\heightMatB}
523     \else
524         \PackageError{glmatrix package}{Dimension mismatch for matrix, could not multiply}{w
525 \fi
526 %
527 \newEmptyMat{#1}
528 \xintFor* ##2 in \heightsequence \do {%
529     \xintFor* ##3 in \widthsequence \do {%
530         \FPset\tmpValue{0}
531         \xintifForFirst{%
532             \xintFor* ##4 in \heightsequenceB \do {%
533                 \FPmul\tmptmp{%
534                     \xintNthElt{##4}{\xintNthElt{##2}{#2}}%
535                 }{%
536                     \xintNthElt{##3}{\xintNthElt{##4}{#3}}%
537                 }
538                 \FPadd\tmpValue\tmpValue\tmptmp

```

```

539     }
540     \newVec{rowVec}{\tmpValue}
541   }{
542     \xintFor* ##4 in \heightsequenceB \do {
543       \FPmul\tmptmp{
544         \xintNthElt{##4}{\xintNthElt{##2}{#2}}
545       }{
546         \xintNthElt{##3}{\xintNthElt{##4}{#3}}
547       }
548       \FPadd\tmpValue\tmpValue\tmptmp
549     }
550     \appendToVec{rowVec}{\rowVec}{\tmpValue}
551   }
552 }
553 \appendRowToMat{#1}{\csname #1\endcsname}{\rowVec}
554 }
555 % Test if width and height is correct
556 \edef\expectedheight{\expandafter\xintLength\expandafter{#2}}
557 \edef\elemsRow{\xintNthElt{1}{#3}}
558 \edef\expectedwidth{\expandafter\xintLength\expandafter{\elemsRow}}
559 %
560 \edef\elemsRow{\xintNthElt{1}{\csname #1\endcsname}}
561 \edef\actualwidth{\expandafter\xintLength\expandafter{\elemsRow}}
562 \edef\elemsMat{\csname #1\endcsname}
563 \edef\actualheight{\expandafter\xintLength\expandafter{\elemsMat}}
564 %
565 \FPifeq{\expectedwidth}{\actualwidth}
566 \else
567   \PackageError{glmatrix package}{Matrix width does not match expectation}{width seems
568 \fi
569 \FPifeq{\expectedheight}{\actualheight}
570 \else
571   \PackageError{glmatrix package}{Matrix height does not match expectation}{height see
572 \fi
573 }

```

`\getMatTranslation` Get the translation component from the given matrix.

`#1` - out: var name to save the result (vector) to

`#2` - matrix: the matrix to extract the translation from

```

574 \newcommand\getMatTranslation[2][matTranslationResult]{
575   \getMatWidth{#2}
576   \getMatHeight{#2}
577   \edef\sequence{\xintSeq[+1]{+1}{\matHeightResult}}
578   \xintFor* ##1 in \sequence \do {
579     \FPset\tmpValue{\xintNthElt{\matWidthResult}{\xintNthElt{##1}{#2}}}
580     \xintifForFirst{
581       \newVec{#1}{\tmpValue}
582     }{
583       \xintifForLast{
584         \appendToVec{#1}{\csname #1\endcsname}{\tmpValue}
585       }
586     }
587   }

```



588 }

`\detMat` Get the determinate of the given matrix.

#1 - out: var name to save the result (scalar) to  
 #2 - matrix: the matrix to calculate the determinate for

```

589 \newcommand\detMat[2][detMatResult]{%
590     \getMatWidth{#2}
591     \getMatHeight{#2}
592     \FPifgt{\matWidthResult}{2}
593         \PackageError{glmatrix package}{Function not yet implemented}{Determinante can only
594     \else
595     \fi
596     \FPifgt{\matHeightResult}{2}
597         \PackageError{glmatrix package}{Function not yet implemented}{Determinante can only
598     \else
599     \fi
600     %
601     \mulScalar[tmpAD]{\xintNthElt{1}{\xintNthElt{1}{#2}}}{\xintNthElt{2}{\xintNthElt{2}{#2}}}
602     \mulScalar[tmpBC]{\xintNthElt{1}{\xintNthElt{2}{#2}}}{\xintNthElt{2}{\xintNthElt{1}{#2}}}
603     \subScalar[#1]{\tmpAD}{\tmpBC}
604 }
```

## 4.2 scalar

Command	Arguments
<code>\newScalar</code>	{ <i>out</i> }, { <i>scalar</i> }
<code>\newScalarPi</code>	{ <i>out</i> }, { <i>scalar</i> }
<code>\newScalarFraction</code>	{ <i>out</i> }, { <i>scalar n</i> }, { <i>scalar d</i> }
<code>\randomScalar</code>	{ <i>out</i> }, { <i>lower bound</i> }, { <i>upper bound</i> }
<code>\copyScalar</code>	[ <i>out</i> ], { <i>scalar</i> }
	scalarCopyResult
<code>\setScalar</code>	{ <i>out</i> }, { <i>scalar</i> }
<code>\zeroScalar</code>	{ <i>out</i> }
<code>\printScalar</code>	[ <i>fraction</i> ], { <i>scalar</i> }
	0
<code>\printScalarAsPi</code>	[ <i>fraction</i> ], { <i>scalar</i> }
	0
<code>\printPiScalarInDegree</code>	[ <i>fraction</i> ], { <i>scalar</i> }
	0
<code>\printScalarRounded</code>	[ <i>digits</i> ], { <i>scalar</i> }
	4
<code>\roundScalar</code>	[ <i>digits</i> ], { <i>scalar</i> }
	4
<code>\ceilScalar</code>	[ <i>out</i> ], { <i>scalar</i> }
	scalarCeilResult
<code>\floorScalar</code>	[ <i>out</i> ], { <i>scalar</i> }
	scalarFloorResult
<code>\clipScalar</code>	[ <i>out</i> ], { <i>scalar</i> }
	clipScalarResult

<code>\radianToDegree</code>	$[\langle out \rangle], \{\langle scalar \rangle\}$ radianToDegreeResult
<code>\toRadian</code>	$[\langle out \rangle], \{\langle scalar \rangle\}$ degreeToRadianResult
<code>\addScalar</code>	$[\langle out \rangle], \{\langle scalar s \rangle\}, \{\langle scalar t \rangle\}$ scalarAddResult
<code>\subScalar</code>	$[\langle out \rangle], \{\langle scalar s \rangle\}, \{\langle scalar t \rangle\}$ scalarSubResult
<code>\mulScalar</code>	$[\langle out \rangle], \{\langle scalar s \rangle\}, \{\langle scalar t \rangle\}$ scalarMulResult
<code>\divScalar</code>	$[\langle out \rangle], \{\langle scalar s \rangle\}, \{\langle scalar t \rangle\}$ scalarDivResult
<code>\powScalar</code>	$[\langle out \rangle], \{\langle scalar s \rangle\}, \{\langle scalar t \rangle\}$ scalarPowResult
<code>\negateScalar</code>	$[\langle out \rangle], \{\langle scalar \rangle\}$ scalarNegateResult
<code>\arccosScalar</code>	$[\langle out \rangle], \{\langle scalar \rangle\}$ scalarArccosResult
<code>\mulAndAddScalar</code>	$[\langle out \rangle], \{\langle scalar s \rangle\}, \{\langle scalar t \rangle\}, \{\langle scalar u \rangle\}$ scalarMulAndAddResult
<code>\lerpScalar</code>	$[\langle out \rangle], \{\langle scalar i \rangle\}, \{\langle scalar j \rangle\}, \{\langle scalar t \rangle\}$ lerpScalarResult
<code>\minScalar</code>	$[\langle out \rangle], \{\langle scalar s \rangle\}, \{\langle scalar t \rangle\}$ minScalarResult
<code>\maxScalar</code>	$[\langle out \rangle], \{\langle scalar s \rangle\}, \{\langle scalar t \rangle\}$ maxScalarResult
<code>\betweenScalar</code>	$[\langle out \rangle], \{\langle scalar \rangle\}, \{\langle lower bound \rangle\}, \{\langle upper bound \rangle\}$ betweenScalarResult
<code>\forEachScalar</code>	$\{\langle input \rangle\}, \{\langle macro \rangle\}, \{\langle args \rangle\}$

`\newScalar` Create a new scalar and assign the given value.  
 #1 - out: var name to save the result (scalar) to  
 #2 - scalar: scalar value that is assigned to out

```

605 \newcommand\newScalar[2]{%
606     \expandafter\edef\csname #1\endcsname{#2}%
607 }
```

`\newScalarPi` Create a new scalar and assign the given value as a multiple of pi.  
 #1 - out: var name to save the result (scalar) to  
 #2 - scalar: scalar value that is multiplied with pi and assigned to out

```

608 \newcommand\newScalarPi[2]{
609     \FPmul\divNumRes{\FPpi}{#2}
610     \expandafter\edef\csname #1\endcsname{\divNumRes}%
611 }
```

`\newScalarFraction` Create a scalar from a given fraction  
 #1 - out: var name to save the result (scalar) to  
 #2 - scalar n: nominator  
 #3 - scalar d: denominator

```

612 \newcommand\newScalarFraction[3]{
613     \FPdiv\divNumRes{#2}{#3}
614     \expandafter\edef\csname #1\endcsname{\divNumRes}%
615 }
```

`\randomScalar` Create a new scalar variable with a random value between lower and upper bound.  
 The value is obtained by creating a random value between 0 and 1 and then interpolating between lower and upper bound.  
 #1 - out: var name to save the result (scalar) to  
 #2 - lower bound: scalar that limits the possible values at the bottom  
 #3 - upper bound: scalar that limits the possible values at the top

```

616 \newcommand\randomScalar[3]{%
617     \FPrandom\tmp
618     \lerpScalar[#1]{#2}{#3}{\tmp}
619 }
```

`\copyScalar` Copy the value of one scalar variable to another.  
 #1 - out: var name to save the result (scalar) to  
 #2 - scalar: scalar which value is assigned to out

```

620 \newcommand\copyScalar[2][scalarCopyResult]{%
621     \setScalar{#1}{#2}
622 }
```

`\setScalar` Set the value of a given scalar.  
 #1 - out: var name to save the result (scalar) to  
 #2 - scalar: scalar value that is assigned to out

```

623 \newcommand\setScalar[2]{%
624     \newScalar{#1}{#2}
625 }
```

`\zeroScalar` Set or create a scalar with value 0.  
 #1 - out: var name to save the result (scalar) to

```

626 \newcommand\zeroScalar[1]{%
627     \expandafter\edef\csname #1\endcsname{0}%
628 }
```

`\printScalar` Print the value of the given scalar.  
 #1 - fraction: whether or not the value should be printed as a fraction (val=1)  
 #2 - scalar: the scalar that should be printed

```

629 \newcommand\printScalar[2][0]{%
630     \FPifeq{#1}{0}%
631         \roundScalar{#2}%
632         \scalarRoundResult%
633     \else%
```

```

634     \numToFractionA{#2}%
635     \fi%
636 }

```

**\printScalarAsPi** Print the value of the given scalar as a multiple of Pi.  
 #1 - fraction: whether or not the value should be printed as a fraction (val=1)  
 #2 - scalar: the scalar that should be printed

```

637 \newcommand\printScalarAsPi[2][0]{%
638     \FPdiv\divNumRes{#2}{\FPpi}
639     \FPifeq{#1}{0}
640         \roundScalar{\divNumRes}
641         \scalarRoundResult
642     \else
643         \numToFractionA{\divNumRes}
644     \fi
645     \pi
646 }

```

**\printPiScalarInDegree** Print the value of the given scalar in degree. This macro does do a conversion and therefore expects the input to be in radian. A degree sign is automatically added.  
 #1 - fraction: whether or not the value should be printed as a fraction (val=1)  
 #2 - scalar: the value that should be printed

```

647 \newcommand\printPiScalarInDegree[2][0]{%
648     \FPmul\tmp{#2}{180}
649     \FPdiv\divNumRes{\tmp}{\FPpi}
650     \FPifeq{#1}{0}
651         \roundScalar{\divNumRes}
652         \scalarRoundResult
653     \else
654         \numToFractionA{\divNumRes}
655     \fi
656     ^\circ
657 }

```

**\printScalarRounded** Print the rounded value of the given scalar.  
 #1 - digits: number of digits after comma  
 #2 - scalar: the value that should be printed

```

658 \newcommand\printScalarRounded[2][4]{%
659     \FPround\tmp{#2}{#1}
660     \FPclip\tmp\tmp
661     \tmp
662 }

```

**\roundScalar** Round the value of the given scalar and save it.  
 #1 - digits: number of digits after comma  
 #2 - scalar: the value that should be rounded

```

663 \newcommand\roundScalar[2][4]{%
664     \FPround\tmp{#2}{#1}%
665     \FPclip\tmp\tmp%
666     \newScalar{scalarRoundResult}{\tmp}%
667 }

```

`\ceilScalar` Ceil the value of the given scalar.  
 #1 - out: var name to save the result (scalar) to  
 #2 - scalar: the scalar that should be ceiled

```

668 \newcommand\ceilScalar[2][scalarCeilResult]{
669   \FPtrunc\tmp{#2}{0}
670   \FPset\tmpSave{\tmp}
671   %
672   \FPifpos{#2}
673     \FPadd\tmpSave{\tmp}{1}
674   \else
675     \fi
676   \FPifint{#2}
677     \FPset\tmpSave{\tmp}
678   \else
679     \fi
680   \newScalar{#1}{\tmpSave}
681 }
```

`\floorScalar` Floor the value of the given scalar.  
 #1 - out: var name to save the result (scalar) to  
 #2 - scalar: the scalar that should be floored

```

682 \newcommand\floorScalar[2][scalarFloorResult]{
683   \FPtrunc\tmp{#2}{0}
684   \FPset\tmpSave{\tmp}
685   %
686   \FPifneg{#2}
687     \FPsub\tmpSave{\tmp}{1}
688   \else
689     \fi
690   \FPifint{#2}
691     \FPset\tmpSave{\tmp}
692   \else
693     \fi
694   \newScalar{#1}{\tmpSave}
695 }
```

`\clipScalar` Remove all trailing zeros from the value of the given scalar.  
 #1 - out: var name to save the result (scalar) to  
 #2 - scalar: the scalar from which the zeros should be removed

```

696 \newcommand\clipScalar[2][clipScalarResult]{
697   \FPclip\tmp{#2}
698   \newScalar{#1}{\tmp}
699 }
```

`\radianToDegree` Convert the value of the given scalar to degree.  
 #1 - out: var name to save the result (scalar) to  
 #2 - scalar: the scalar which to convert

```

700 \newcommand\radianToDegree[2][radianToDegreeResult]{
701   \FPmul\tmp{#2}{180}
702   \FPdiv\tmp{\tmp}{\FPpi}
703   \newScalar{#1}{\tmp}
704 }
```

`\toRadian` Convert the value of the given scalar to radian.  
 #1 - out: var name to save the result (scalar) to  
 #2 - scalar: the scalar which to convert

```

705 \newcommand\toRadian[2][degreeToRadianResult]{
706     \FPdiv\tmp{#2}{180}
707     \FPmul\tmp{\tmp}{\FPpi}
708     \newScalar{#1}{\tmp}
709 }
```

`\addScalar` Add the values of two scalars.  
 #1 - out: var name to save the result (scalar) to  
 #2 - scalar s: the first scalar of the sum  
 #3 - scalar t: the second scalar of the sum

```

710 \newcommand\addScalar[3][scalarAddResult]{%
711     \FPadd\tmp{#2}{#3}
712     \newScalar{#1}{\tmp}
713 }
```

`\subScalar` Subtract the values of two scalars.  
 #1 - out: var name to save the result (scalar) to  
 #2 - scalar s: the scalar to subtract from  
 #3 - scalar t: the scalar to subtract

```

714 \newcommand\subScalar[3][scalarSubResult]{%
715     \FPsub\tmp{#2}{#3}
716     \newScalar{#1}{\tmp}
717 }
```

`\mulScalar` Multiply the values of two scalars.  
 #1 - out: var name to save the result (scalar) to  
 #2 - scalar s: the scalar to multiply  
 #3 - scalar t: the scalar with which to multiply

```

718 \newcommand\mulScalar[3][scalarMulResult]{
719     \FPmul\tmp{#2}{#3}
720     \newScalar{#1}{\tmp}
721 }
```

`\divScalar` Divide the values of two scalars.  
 #1 - out: var name to save the result (scalar) to  
 #2 - scalar s: the scalar to divide  
 #3 - scalar t: the scalar to divide with

```

722 \newcommand\divScalar[3][scalarDivResult]{%
723     \FPifeq{#3}{0}
724         \PackageError{glmatrix package}{In divScalar function: division by zero}{parameter 3}
725     \else
726         \fi
727     \FPdiv\tmp{#2}{#3}
728     \newScalar{#1}{\tmp}
729 }
```

```

\powScalar Raise the values of one scalar by another.
#1 - out: var name to save the result (scalar) to
#2 - scalar s: the scalar to raise
#3 - scalar t: the scalar with which to raise
730 \newcommand\powScalar[3][scalarPowResult]{%
731   \FPpow\tmp{#2}{#3}
732   \newScalar{#1}{\tmp}
733 }

\negateScalar Negate the value of the given scalar.
#1 - out: var name to save the result (scalar) to
#2 - scalar: the scalar to negate
734 \newcommand\negateScalar[2][scalarNegateResult]{%
735   \FPneg\tmp{#2}
736   \newScalar{#1}{\tmp}
737 }

\arccosScalar Get the arccos of the value of the given scalar.
#1 - out: var name to save the result (scalar) to
#2 - scalar: the scalar to calculate the arccos for
738 \newcommand\arccosScalar[2][scalarArccosResult]{%
739   \FPround\tmp{#2}{10}
740   \FPclip\tmp\tmp
741   \FParccos\tmp\tmp
742   \newScalar{#1}{\tmp}
743 }

\mulAndAddScalar Adds two scalars after multiplying the second one by a scalar.
#1 - out: var name to save the result (scalar) to
#2 - scalar s: the scalar to add to
#3 - scalar t: the scalar that is multiplied and then added
#4 - scalar u: the scalar that is the multiplier
744 \newcommand\mulAndAddScalar[4][scalarMulAndAddResult]{%
745   \FPmul\tmp{#3}{#4}
746   \FPadd\tmp{#2}{\tmp}
747   \newScalar{#1}{\tmp}
748 }

\lerpScalar Linearly interpolate between values of two scalars.
#1 - out: var name to save the result (scalar) to
#2 - scalar i: the start scalar
#3 - scalar j: the end scalar
#4 - scalar t: interpolation amount range [0-1]
749 \newcommand\lerpScalar[4][lerpScalarResult]{%
750   \FPsub\tmpA{1}{#4}
751   \FPmul\tmpA{#2}{\tmpA}
752   \FPmul\tmpB{#3}{#4}
753   \FPadd\tmpA\tmpA\tmpB
754   \newScalar{#1}{\tmpA}
755 }

```

```

\minScalar Return the minimum of two scalar values.
#1 - out: var name to save the result (scalar) to
#2 - scalar s: the first scalar
#3 - scalar t: the second scalar
756 \newcommand\minScalar[3][minScalarResult]{%
757   % TODO use \FPmin#1#2#3
758   \FPiflt{#2}{#3}
759     \expandafter\edef\csname #1\endcsname{#2}%
760   \else
761     \expandafter\edef\csname #1\endcsname{#3}%
762   \fi
763 }

```

```

\maxScalar Return the maximum of two scalar values.
#1 - out: var name to save the result (scalar) to
#2 - scalar s: first scalar value
#3 - scalar t: second scalar value
764 \newcommand\maxScalar[3][maxScalarResult]{%
765   % TODO use \FPmax#1#2#3
766   \FPiflt{#2}{#3}
767     \expandafter\edef\csname #1\endcsname{#3}%
768   \else
769     \expandafter\edef\csname #1\endcsname{#2}%
770   \fi
771 }

```

```

\betweenScalar Return whether a scalar lies between two other values.
#1 - out: var name to save the result (scalar) to (0 or 1)
#2 - scalar: the scalar to test
#3 - lower bound: #2 needs to be greater then this for this macro to return 1
#4 - upper bound: #2 needs to be smaller than this for this macro to return 1
772 \newcommand\betweenScalar[4][betweenScalarResult]{%
773   \FPset\betweenScalarResult{0}
774   \FPiflt{#2}{#3}
775   \else
776     \FPadd\betweenScalarResult\betweenScalarResult{1}
777   \fi
778   \FPifgt{#2}{#4}
779   \else
780     \FPadd\betweenScalarResult\betweenScalarResult{1}
781   \fi
782   % TODO
783   \FPifeq{\betweenScalarResult}{2}
784     % \FPset\betweenScalarResult{1}
785     \expandafter\edef\csname #1\endcsname{1}%
786   \else
787     % \FPset\betweenScalarResult{0}
788     \expandafter\edef\csname #1\endcsname{0}%
789   \fi
790 }

```



`\forEachScalar` Apply the given function to a list of variables.  
`#1` - input: list of commands (var names) that should be altered  
`#2` - macro: the chosen function  
`#3` - args: params for function

```

791 \newcommand\forEachScalar [3] {
792   \FPiflt{#2}{10}
793   \else
794     \PackageError{glmatrix package}{In forEach scalar}{selection function that is not im
795   \fi
796   %
797   \def\varList{\xintCSVtoList{#1}}
798   \def\paramList{\xintCSVtoList{#3}}
799   \xintFor* ##1 in \varList \do {%
800     \FPifeq{#2}{1}
801       \addScalar[##1]{\csname ##1\endcsname}{\xintNthElt{1}{\paramList}}
802     \else
803     \fi
804     \FPifeq{#2}{2}
805       \subScalar[##1]{\csname ##1\endcsname}{\xintNthElt{1}{\paramList}}
806     \else
807     \fi
808     \FPifeq{#2}{3}
809       \mulScalar[##1]{\csname ##1\endcsname}{\xintNthElt{1}{\paramList}}
810     \else
811     \fi
812     \FPifeq{#2}{4}
813       \divScalar[##1]{\csname ##1\endcsname}{\xintNthElt{1}{\paramList}}
814     \else
815     \fi
816     \FPifeq{#2}{5}
817       \powScalar[##1]{\csname ##1\endcsname}{\xintNthElt{1}{\paramList}}
818     \else
819     \fi
820     \FPifeq{#2}{6}
821       \negateScalar[##1]{\csname ##1\endcsname}
822     \else
823     \fi
824     \FPifeq{#2}{7}
825       \lerpScalar[##1]{\csname ##1\endcsname}{\xintNthElt{1}{\paramList}}{\xintNthElt{
826     \else
827     \fi
828     \FPifeq{#2}{8}
829       \mulAndAddScalar[##1]{\csname ##1\endcsname}{\xintNthElt{1}{\paramList}}{\xintNt
830     \else
831     \fi
832     \FPifeq{#2}{9}
833       \arccosScalar[##1]{\csname ##1\endcsname}
834     \else
835     \fi
836   }
837 }
```

`\fractionToNum` Convert a fraction with root to a decimal number

```

838 \newcommand\fractionToNum[3]{%
839   \FProot\tmpA{#3}{2}
840   \FPMul\tmpA\tmpA{#2}
841   \FPdiv\tmpB{#1}\tmpA
842   \FPset\fractionNumRes{\tmpB}
843   \ensuremath{
844     \tmpB
845   }
846 }

```

`\solutionC` Convert a decimal number to a fraction with root

```

847 \newcommand\solutionC{%
848   \FPset\finalmul{1}
849   \FPset\tmptmptmptmp\tmptmptmp
850   %
851   \edef\innersequence{\xintSeq[+1]{+1}{+10}}%
852   \xintFor* ##1 in \innersequence \do {%
853     \FPtrunc\floored\tmptmptmptmp{0}
854     \FPsub\remainder\tmptmptmptmp\floored
855     \FPifgt{\remainder}{0.00001}
856       \FPdiv\mulmul{1}{\remainder}
857       \FPMul\finalmul\finalmul\mulmul
858     \else
859       \xintBreakFor
860     \fi
861     \FPset\tmptmptmptmp\mulmul
862   }
863   \FPround\finalmul\finalmul{8}
864   \FPclip\finalmul\finalmul
865   \FPset\finalmultmp\finalmul
866   % The result will be a multiplicator that is able to round
867   % the number to a whole number, but the result might not be a
868   % nice number to root e.g. 8/sqrt(6), 0.09375 will result in 32
869   % 0.09375 * 32 = 3, but sqrt(32) = 5,65.. but given this we know
870   % that the searched for number must be a multiple of 32, in this case 64
871   % Note: rounding to 6 & 36 does not work since 0.09375 * 36 = 3.375
872   \xintFor* ##1 in \innersequence \do {%
873     \FProot\finalmulsqrt\finalmultmp{2}
874     \FPround\finalmulsqrt\finalmulsqrt{4}
875     \FPclip\finalmulsqrt\finalmulsqrt
876     \FPifint{\finalmulsqrt}
877     \xintBreakFor
878     \else
879       \FPadd\finalmultmp\finalmultmp\finalmul
880     \fi
881   }
882   \FPset\finalmul\finalmultmp
883   \FProot\finalmulsqrt\finalmul{2}
884   \FPround\finalmulsqrt\finalmulsqrt{4}
885   \FPclip\finalmulsqrt\finalmulsqrt
886   \FPMul\divisor\divisor\finalmul
887   \FPMul\dividend\dividend\finalmulsqrt
888 }

```

`\numToFraction`

```

889 \newcommand\numToFraction[1]{%
890     \FPset\dividend{1}
891     \FPset\divisor{#1}
892     \FPset\minusSign{1}
893     \FPiflt{\divisor}{0}
894         \FPset\minusSign{-1}
895     \else
896     \fi
897     \ensuremath{
898         % reverse fraction
899         \FPdiv\divisor{1}{#1}
900         % Reverse root
901         % Use FPMul instead of FPow to allow for neg values
902         \FPMul\divisor\divisor\divisor
903         \FPset\temptptmp\divisor
904         \FPround\divisor\divisor{10}
905         \FPclip\divisor\divisor
906         \FPifint{\divisor}
907     \else
908         \solutionC
909     \fi
910     %
911     \FPround\divisor\divisor{4}
912     \FPclip\divisor\divisor
913     \FPMul\dividend\dividend\minusSign
914     \FPround\dividend\dividend{4}
915     \FPclip\dividend\dividend
916     % TODO this should not be necessary
917     \FPset\outputVar{#1}
918     % Create fraction from dividend and divisor
919     \FPset\outputVar{\frac{\dividend}{\sqrt{\divisor}}}
920     % If the sqrt can be resolved
921     \FPset\tmpDivisor{\divisor}
922     \FProot\tmpDivisor\tmpDivisor{2}
923     \FPround\tmpDivisor\tmpDivisor{8}
924     \FPclip\tmpDivisor\tmpDivisor
925     \FPifint{\tmpDivisor}
926         \FPset\outputVar{\frac{\dividend}{\tmpDivisor}}
927     \else
928     \fi
929     %
930     \FPset\inputVar{#1}
931     \FPMul\inputVar\inputVar\inputVar
932     \FPMul\inputVar\inputVar\minusSign
933     \FPround\inputVar\inputVar{8}
934     \FPclip\inputVar\inputVar
935     \FPifint{\inputVar}
936         \FPset\outputVar{\,\sqrt{\inputVar}}
937     \else
938     \fi
939     % If #1 was an int all along
940     \FPset\inputVarA{#1}
941     \FPround\inputVarA\inputVarA{10}

```

```

942     \FPclip\inputVarA\inputVarA
943     \FPifint{\inputVarA}
944         \FPset\outputVar{\inputVarA}
945     \else
946     \fi
947     \outputVar
948 }
949 }

```

#### `\numToFractionA`

```

950 \newcommand\numToFractionA[1]{%
951     \FPset\inputVarA{#1}
952     \FPround\inputVarA\inputVarA{10}
953     \FPclip\inputVarA\inputVarA
954     % Prevent division by 0
955     \FPifint{\inputVarA}
956         \FPset\outputVar{\inputVarA}
957         \outputVar
958     \else
959         \numToFraction{#1}
960     \fi
961 }

```

### 4.3 vector

Command	Arguments
<code>\newVec</code>	{ <i>out</i> }, { <i>values</i> }
<code>\newVecPi</code>	{ <i>out</i> }, { <i>values</i> }
<code>\randomVec</code>	{ <i>out</i> }, { <i>dimensions</i> }, { <i>lower bound</i> }, { <i>upper bound</i> }
<code>\copyVec</code>	[ <i>out</i> ], { <i>vector</i> } copyVecResult
<code>\zeroVec</code>	[ <i>dims</i> ], { <i>out</i> } 3
<code>\appendToVec</code>	{ <i>out</i> }, { <i>vector</i> }, { <i>scalar</i> }
<code>\printVec</code>	[ <i>fraction</i> ], { <i>vector</i> } 0
<code>\printVecT</code>	[ <i>fraction</i> ], { <i>vector</i> } 0
<code>\printVecAsPoint</code>	[ <i>fraction</i> ], { <i>vector</i> } 0
<code>\printVecAsPi</code>	[ <i>fraction</i> ], { <i>vector</i> } 0
<code>\printVecAsPiT</code>	[ <i>fraction</i> ], { <i>vector</i> } 0
<code>\printVecContent</code>	{ <i>vector</i> }
<code>\printVecContentXY</code>	{ <i>vector</i> }
<code>\printVecContentXYZ</code>	{ <i>vector</i> }
<code>\printVecContentXYZW</code>	{ <i>vector</i> }

<code>\getVecHeight</code>	$[\langle out \rangle], \{\langle vector \rangle\}$ vecHeightResult
<code>\printScalarPComp</code>	$[\langle fraction \rangle], \{\langle vector v \rangle\}, \{\langle vector w \rangle\}$ 0
<code>\printCrossPComp</code>	$[\langle fraction \rangle], \{\langle vector v \rangle\}, \{\langle vector w \rangle\}$ 0
<code>\roundVec</code>	$[\langle out \rangle], \{\langle vector \rangle\}, \{\langle digits \rangle\}$ roundVecResult
<code>\ceilVec</code>	$[\langle out \rangle], \{\langle vector \rangle\}$ ceilVecResult
<code>\floorVec</code>	$[\langle out \rangle], \{\langle vector \rangle\}$ floorVecResult
<code>\dehomogenVec</code>	$[\langle out \rangle], \{\langle vector \rangle\}$ dehomogenVecResult
<code>\addVec</code>	$[\langle out \rangle], \{\langle vector \rangle\}, \{\langle vectors \rangle\}$ addVecResult
<code>\subVec</code>	$[\langle out \rangle], \{\langle vector \rangle\}, \{\langle vectors \rangle\}$ subVecResult
<code>\mulVec</code>	$[\langle out \rangle], \{\langle vector \rangle\}, \{\langle vectors \rangle\}$ mulVecResult
<code>\divVec</code>	$[\langle out \rangle], \{\langle vector \rangle\}, \{\langle vectors \rangle\}$ divVecResult
<code>\scaleVec</code>	$[\langle out \rangle], \{\langle vector \rangle\}, \{\langle scalar \rangle\}$ scaleVecResult
<code>\divVecScalar</code>	$[\langle out \rangle], \{\langle vector \rangle\}, \{\langle scalar \rangle\}$ divVecScalarResult
<code>\negateVec</code>	$[\langle out \rangle], \{\langle vector \rangle\}$ negateVecResult
<code>\scaleAndAddVec</code>	$[\langle out \rangle], \{\langle vector v \rangle\}, \{\langle vector w \rangle\}, \{\langle scalar \rangle\}$ scaleAndAddVecResult
<code>\lerpVec</code>	$[\langle out \rangle], \{\langle vector v \rangle\}, \{\langle vector w \rangle\}, \{\langle scalar t \rangle\}$ lerpVecResult
<code>\slerpVec</code>	$[\langle out \rangle], \{\langle vector v \rangle\}, \{\langle vector w \rangle\}, \{\langle scalar t \rangle\}$ slerpVecResult
<code>\dotVec</code>	$[\langle out \rangle], \{\langle vector v \rangle\}, \{\langle vector w \rangle\}$ dotVecResult
<code>\lenVec</code>	$[\langle out \rangle], \{\langle vector \rangle\}$ lenVecResult
<code>\sqrLenVec</code>	$[\langle out \rangle], \{\langle vector \rangle\}$ sqrLenVecResult
<code>\normalizeVec</code>	$[\langle out \rangle], \{\langle vector \rangle\}$ normalizeVecResult
<code>\crossVec</code>	$[\langle out \rangle], \{\langle vector v \rangle\}, \{\langle vector w \rangle\}$ crossVecResult
<code>\crossVecTwoD</code>	$[\langle out \rangle], \{\langle vector v \rangle\}, \{\langle vector w \rangle\}$ crossVecResult
<code>\angleVec</code>	$[\langle out \rangle], \{\langle vector v \rangle\}, \{\langle vector w \rangle\}$ angleVecResult
<code>\distVec</code>	$[\langle out \rangle], \{\langle vector v \rangle\}, \{\langle vector w \rangle\}$ distVecResult

<code>\sqrDistVec</code>	$[\langle out \rangle], \{\langle vector v \rangle\}, \{\langle vector w \rangle\}$ sqrDistVecResult
<code>\transformVec</code>	$[\langle out \rangle], \{\langle matrix \rangle\}, \{\langle vector \rangle\}$ transformVecResult
<code>\tensorVec</code>	$[\langle out \rangle], \{\langle vector \rangle\}$ tensorVecResult
<code>\minVec</code>	$[\langle out \rangle], \{\langle vector v \rangle\}, \{\langle vector w \rangle\}$ minVecResult
<code>\maxVec</code>	$[\langle out \rangle], \{\langle vector v \rangle\}, \{\langle vector w \rangle\}$ maxVecResult
<code>\solveVecEQL</code>	$[\langle out \rangle], \{\langle vector v \rangle\}, \{\langle vector w \rangle\}$ solveVecEQLResult

`\newVec` Create a new vector and assign the given values.  
 #1 - out: var name to save the result (vector) to  
 #2 - values: values that are assigned to #1

```

962 \newcommand\newVec[2]{%
963     \if\noexpand#1\relax
964         \PackageError{glmatrix package}{argument mismatch}{Argument 1 of newVec seems to be
965     \else
966         \expandafter\edef\csname #1\endcsname{\xintCSVtoList{#2}}%
967     \fi
968 }
```

`\newVecPi` Create a new vector and assign the given values as a multiple of pi.  
 #1 - out: var name to save the result (vector) to  
 #2 - values: values that are multiplied with pi and assigned to #1

```

969 \newcommand\newVecPi[2]{%
970     \def\tmpVec{\xintCSVtoList{#2}}
971     \xintfor* ##1 in \tmpVec \do {%
972         \newScalarPi{tmp}{##1}
973         \xintifForFirst{%
974             \newVec{#1}{\tmp}
975         }{%
976             \appendToVec{#1}{\csname #1\endcsname}{\tmp}
977         }
978     }
979 }
```

`\randomVec` Create a new vector and assign random values between lower and upper bound to its components. The value is obtained by creating a random value between 0 and 1 and then interpolating between lower and upper bound.  
 #1 - out: var name to save the result (vector) to  
 #2 - dimensions: scalar that defines the number of dimensions #1 should have  
 #3 - lower bound: scalar that limits the possible values at the bottom  
 #4 - upper bound: scalar that limits the possible values at the top

```

980 \newcommand\randomVec[4]{
```

```

981 \edef\innersequence{\xintSeq[+1]{+1}{#2}}%
982 \xintFor* ##1 in \innersequence \do {%
983     \FPrandom\tmp
984     \lerpScalar{#3}{#4}{\tmp}
985     \xintifForFirst{%
986         \newVec{#1}{\lerpScalarResult}
987     }{%
988         \appendToVec{#1}{\csname #1\endcsname}{\lerpScalarResult}
989     }
990 }
991 }

```

**\copyVec** Copy values from one vector variable to another.  
 #1 - out: var name to save the result (vector) to  
 #2 - vector: vector which values are assigned to #1

```

992 \newcommand\copyVec[2][copyVecResult]{
993     \xintFor* ##1 in #2 \do {%
994         \xintifForFirst{%
995             \newVec{#1}{##1}
996         }{%
997             \appendToVec{#1}{\csname #1\endcsname}{##1}
998         }
999     }
1000 }

```

**\zeroVec** Create or override a vector with all its components being 0. Obacht: It is not necessary to call this to create a new vector. This is just for convenience when requiring a vector with initialized with zero.

```

#1 - dims: number of dimensions (rows)
#2 - out: var name to save the result (vector) to
1001 \newcommand\zeroVec[2][3]{%
1002     \edef\sequence{\xintSeq[+1]{+1}{#1}}%
1003     \xintFor* ##1 in \sequence \do {%
1004         \xintifForFirst{%
1005             \newVec{#2}{0}
1006         }{%
1007             \appendToVec{#2}{\csname #2\endcsname}{0}
1008         }
1009     }
1010 }

```

**\appendToVec** Append a dimension with the given value to an already defined vector.

```

#1 - out: var name to save the result (vector) to
#2 - vector: the currently present values
#3 - scalar: the scalar to add to #2
1011 \newcommand\appendToVec[3]{%
1012     \expandafter\edef\csname #1\endcsname{#2 \xintCSVtoList{#3}}%
1013 }

```

`\printVec` Print the values of the given vector in vector notation.

#1 - fraction: whether or not the value should be printed as a fraction (val=1)

#2 - vector: the vector that should be printed

```

1014 \newcommand\printVec[2][0]{%
1015   \if\noexpand#2\relax
1016   \else
1017     \PackageError{glmatrix package}{argument mismatch}{Argument 1 of printVec seems to b
1018   \fi
1019   \ensuremath{%
1020     \begin{pmatrix}
1021       \xintFor* ##1 in {#2} \do {%
1022         \xintifForLast{
1023           \FPifeq{#1}{0}
1024             \roundScalar{##1}
1025             \scalarRoundResult
1026           \else
1027             \numToFractionA{##1}
1028           \fi
1029         }{%
1030           \FPifeq{#1}{0}
1031             \roundScalar{##1}
1032             \scalarRoundResult
1033           \else
1034             \numToFractionA{##1}
1035           \fi
1036         \\
1037       }
1038     }
1039   \end{pmatrix}
1040 }
1041 }
```

`\printVecT` Print the values of the given vector in transposed notation.

#1 - fraction: whether or not the value should be printed as a fraction (val=1)

#2 - vector: the vector that should be printed

```

1042 \newcommand\printVecT[2][0]{%
1043   \if\noexpand#2\relax
1044   \else
1045     \PackageError{glmatrix package}{argument mismatch}{Argument 1 of printVecT seems to
1046   \fi
1047   \ensuremath{\trans{\left(
1048     \xintFor* ##1 in {#2} \do {%
1049       \xintifForLast{
1050         \FPifeq{#1}{0}
1051           \roundScalar{##1}
1052           \scalarRoundResult
1053         \else
1054           \numToFractionA{##1}
1055         \fi
1056       }{%
1057         \FPifeq{#1}{0}
1058           \roundScalar{##1}
1059           \scalarRoundResult
```



```

1060         \else
1061             \numToFractionA{##1}
1062         \fi
1063         ,
1064     }
1065 }
1066 \right)}}}%
1067 }

```

`\printVecAsPoint` Print the values of the given vector in point notation.

#1 - fraction: whether or not the value should be printed as a fraction (val=1)  
 #2 - vector: the vector that should be printed

```

1068 \newcommand\printVecAsPoint[2][0]{%
1069     \ensuremath{%
1070         \begin{bmatrix}
1071             \xintFor* ##1 in {#2} \do {%
1072                 \FPifeq{#1}{0}
1073                     \roundScalar{##1}
1074                     \scalarRoundResult
1075                 \else
1076                     \numToFractionA{##1}
1077                 \fi
1078             \xintifForLast}{-}{%
1079                 \\
1080             }
1081         }
1082     \end{bmatrix}
1083 }
1084 }

```

`\printVecAsPi` Print the values of the given vector as a multiple of  $\pi$  in vector notation.

#1 - fraction: whether or not the value should be printed as a fraction (val=1)  
 #2 - vector: the vector that should be printed

```

1085 \newcommand\printVecAsPi[2][0]{%
1086     \ensuremath{%
1087         \begin{pmatrix}
1088             \xintFor* ##1 in #2 \do {%
1089                 \printScalarAsPi[#1]{##1}
1090                 % \FPifeq{#1}{0}
1091                 %     \roundScalar{##1}
1092                 %     \scalarRoundResult
1093                 % \else
1094                 %     \numToFractionA{##1}
1095                 % \fi
1096             \xintifForLast}{-}{%
1097                 \\
1098             }
1099         }
1100     \end{pmatrix}
1101 }
1102 }

```

`\printVecAsPiT` Print the values of the given vector as a multiple of Pi in transposed notation.  
 #1 - fraction: whether or not the value should be printed as a fraction (val=1)  
 #2 - vector: the vector that should be printed

```

1103 \newcommand\printVecAsPiT[2][0]{%
1104     \ensuremath{\trans{\left(
1105         \xintFor* ##1 in #2 \do {%
1106             \printScalarAsPi[##1]{##1}
1107             \xintifForLast}{-%
1108                 ,
1109             }
1110         }
1111     \right)}}
1112 }
```

`\printVecContent` Print the values of the given vector comma separated.  
 #1 - vector: the vector which values should be printed

```

1113 \newcommand\printVecContent[1]{%
1114     \xintNthElt{1}{#1}, \xintNthElt{3}{#1}, \xintNthElt{2}{#1}%
1115 }
```

`\printVecContentXY` Returns the x and y component of the given vector.  
 #1 - vector: the vector which values should be printed

```

1116 \newcommand\printVecContentXY[1]{%
1117     \xintNthElt{1}{#1}, \xintNthElt{2}{#1}
1118 }
```

`\printVecContentXYZ` Returns the x, y and z component of the given vector.  
 #1 - vector: the vector which values should be printed

```

1119 \newcommand\printVecContentXYZ[1]{%
1120     \xintNthElt{1}{#1}, \xintNthElt{2}{#1}, \xintNthElt{3}{#1}%
1121 }
```

`\printVecContentXYZW` Returns the x, y, z and w component of the given vector  
 #1 - vector: the vector which values should be printed

```

1122 \newcommand\printVecContentXYZW[1]{%
1123     \xintNthElt{1}{#1}, \xintNthElt{2}{#1}, \xintNthElt{3}{#1}, \xintNthElt{4}{#1}
1124 }
```

`\getVecHeight` Returns the number of components that given vector has (number of dimensions).  
 #1 - out: var name to save the result (scalar) to  
 #2 - vector: the vector of which to return the height

```

1125 \newcommand\getVecHeight[2][vecHeightResult]{%
1126     \newScalar{##1}{\expandafter\xintLength\expandafter{##2}}
1127 }
```

```

\printScalarPComp Print the intermediate components of the scalarproduct calculation.
#1 - fraction: whether or not the value should be printed as a fraction (val=1)
#2 - vector v: the first vector
#3 - vector w: the second vector
1128 \newcommand\printScalarPComp[3][0] {%
1129     \FPifeq{\expandafter\xintLength\expandafter{#2}}{\expandafter\xintLength\expandafter{#3}}
1130     \else
1131         \PackageError{glmatrix package}{In scalar product}{vector A and B do not have same length}
1132     \fi
1133     \edef\innersequence{\xintSeq[+1]{+1}{\expandafter\xintLength\expandafter{#2}}}%
1134     \ensuremath{
1135         \xintFor* ##1 in \innersequence \do {%
1136             \FPifeq{#1}{0}
1137                 \printScalarRounded{\xintNthElt{##1}{#2}}*\printScalarRounded{\xintNthElt{##1}{#3}}
1138             \else
1139                 \numToFractionA{\xintNthElt{##1}{#2}}*\numToFractionA{\xintNthElt{##1}{#3}}
1140             \fi
1141             \xintifForLast{}{%
1142                 +
1143             }
1144         }
1145     }
1146 }

```

```

\printCrossPComp Print the intermediate components of the crossproduct calculation.
#1 - fraction: whether or not the value should be printed as a fraction (val=1)
#2 - vector v: the first vector
#3 - vector w: the second vector
1147 \newcommand\printCrossPComp[3][0] {%
1148     \FPifeq{\expandafter\xintLength\expandafter{#2}}{3}
1149     \else
1150         \PackageError{glmatrix package}{In cross product}{vector A is not of length 3}
1151     \fi
1152     \FPifeq{\expandafter\xintLength\expandafter{#3}}{3}
1153     \else
1154         \PackageError{glmatrix package}{In cross product}{vector A is not of length 3}
1155     \fi
1156     \FPset\aa{\xintNthElt{1}{#2}}
1157     \FPset\ab{\xintNthElt{2}{#2}}
1158     \FPset\ac{\xintNthElt{3}{#2}}
1159     \FPset\ba{\xintNthElt{1}{#3}}
1160     \FPset\bb{\xintNthElt{2}{#3}}
1161     \FPset\bc{\xintNthElt{3}{#3}}
1162     %
1163     \ensuremath{
1164         \FPifeq{#1}{0}
1165             \begin{pmatrix}
1166                 \printScalarRounded{\ab}*\printScalarRounded{\bc} - \printScalarRounded{\ac}
1167                 \printScalarRounded{\ac}*\printScalarRounded{\ba} - \printScalarRounded{\aa}
1168                 \printScalarRounded{\aa}*\printScalarRounded{\bb} - \printScalarRounded{\ab}
1169             \end{pmatrix}
1170         \else

```

```

1171         \begin{pmatrix}
1172             \numToFractionA{\ab}*\numToFractionA{\bc} - \numToFractionA{\ac}*\numToFract
1173             \numToFractionA{\ac}*\numToFractionA{\ba} - \numToFractionA{\aa}*\numToFract
1174             \numToFractionA{\aa}*\numToFractionA{\bb} - \numToFractionA{\ab}*\numToFract
1175         \end{pmatrix}
1176     \fi
1177 }
1178 }

```

`\roundVec` Round the values of the given vector.

```

#1 - out: var name to save the result (vector) to
#2 - vector: the vector that should be rounded
#3 - digits: number of digits after comma
1179 \newcommand\roundVec[3][roundVecResult]{%
1180     \xintFor* ##3 in #2 \do {%
1181         \FPround\tmp{##3}{#3}
1182         \xintifForFirst{%
1183             \newVec{#1}{\tmp}
1184         }{%
1185             \appendToVec{#1}{\csname #1\endcsname}{\tmp}
1186         }
1187     }
1188 }

```

`\ceilVec` Ceil the values of the given vector.

```

#1 - out: var name to save the result (vector) to
#2 - vector: the vector that should be ceiled
1189 \newcommand\ceilVec[2][ceilVecResult]{
1190     \xintFor* ##1 in #2 \do {%
1191         \ceilScalar[tmp]{##1}
1192         \xintifForFirst{%
1193             \newVec{#1}{\tmp}
1194         }{%
1195             \appendToVec{#1}{\csname #1\endcsname}{\tmp}
1196         }
1197     }
1198 }

```

`\floorVec` Floor the values of the given vector.

```

#1 - out: var name to save the result (vector) to
#2 - vector: the vector that should be floored
1199 \newcommand\floorVec[2][floorVecResult]{
1200     \xintFor* ##1 in #2 \do {%
1201         \floorScalar[tmp]{##1}
1202         \xintifForFirst{%
1203             \newVec{#1}{\tmp}
1204         }{%
1205             \appendToVec{#1}{\csname #1\endcsname}{\tmp}
1206         }
1207     }
1208 }

```

`\dehomogenVec` Dehomogenize the values of the given vector. This divides all vector components by the last component.

```

#1 - out: var name to save the result (vector) to
#2 - vector: the vector that should be dehomogenized
1209 \newcommand\dehomogenVec[2][dehomogenVecResult]{
1210   \getVecHeight{#2}
1211   \FPset\tmpDiv{\xintNthElt{\vecHeightResult}{#2}}
1212   \edef\sequence{\xintSeq[+1]{+1}{\vecHeightResult}}%
1213   \xintFor* ##1 in \sequence \do {%
1214     \xintifForFirst{%
1215       \FPdiv\divResult{\xintNthElt{##1}{#2}}{\tmpDiv}
1216       \newVec{tmpVec}{\divResult}
1217     }{%
1218       % \xintifForLast{
1219       % }{
1220       \FPdiv\divResult{\xintNthElt{##1}{#2}}{\tmpDiv}
1221       \appendToVec{tmpVec}{\tmpVec}{\divResult}
1222     % }
1223   }
1224 }
1225 \copyVec{#1}{\tmpVec}
1226 }
```

`\addVec` Add the values of 1 to  $m$  vectors to the values of the given vector.

```

#1 - out: var name to save the result (vector) to
#2 - vector: the vector to add to
#3 - vectors: the list of vectors with which to add
1227 \newcommand\addVec[3][addVecResult]{%
1228   \def\vecList{\xintCSVtoList{#3}}
1229   \getVecHeight{#2}
1230   \edef\sequence{\xintSeq[+1]{+1}{\vecHeightResult}}%
1231   % Iterate each row / dimension
1232   \xintFor* ##1 in \sequence \do {%
1233     % Assign value of start vector (param #2)
1234     \FPset\sumValue{\xintNthElt{##1}{#2}}%
1235     % If this is the first iteration create a new vector
1236     \xintifForFirst{%
1237       % Iterate all vectors that are part of param #3
1238       \xintFor* ##2 in \vecList \do {%
1239         % Multiply with row value of current vector
1240         \FPadd\sumValue\sumValue{\xintNthElt{##1}{##2}}%
1241       }
1242       \newVec{#1}{\sumValue}
1243     }{%
1244       \xintFor* ##2 in \vecList \do {%
1245         \FPadd\sumValue\sumValue{\xintNthElt{##1}{##2}}%
1246       }
1247       \appendToVec{#1}{\csname #1\endcsname}{\sumValue}
1248     }
1249   }
1250 }
```

`\subVec` Subtract the values of 1 to  $m$  vectors from the values of the given vector.

#1 - out: var name to save the result (vector) to

#2 - vector: the vector to subtract from

#3 - vectors: the list of vectors with which to subtract

```

1251 \newcommand\subVec[3][subVecResult]{%
1252   \if\noexpand#1\relax
1253     \PackageError{glmatrix package}{argument mismatch}{Argument 1 of subVec seems to be
1254   \else
1255     \fi
1256   \if\noexpand#2\relax
1257     \else
1258       \PackageError{glmatrix package}{argument mismatch}{argument 2 of subVec does not see
1259     \fi
1260   % TODO does not work on list of commands
1261   % \if\noexpand#3\relax
1262   % \else
1263   %   \PackageError{glmatrix package}{argument mismatch}{argument 3 of subVec does not s
1264   % \fi
1265   %
1266   \def\vecList{\xintCSVtoList{#3}}
1267   \getVecHeight{#2}
1268   \edef\sequence{\xintSeq[+1]{+1}{\vecHeightResult}}%
1269   % Iterate each row / dimension
1270   \xintFor* ##1 in \sequence \do {%
1271     % Assign value of start vector (param #2)
1272     \FPset\subValue{\xintNthElt{##1}{#2}}%
1273     % If this is the first iteration create a new vector
1274     \xintifForFirst{%
1275       % Iterate all vectors that are part of param #3
1276       \xintFor* ##2 in \vecList \do {%
1277         % Multiply with row value of current vector
1278         \FPsub\subValue\subValue{\xintNthElt{##1}{##2}}%
1279       }
1280       \newVec{#1}{\subValue}
1281     }{%
1282       \xintFor* ##2 in \vecList \do {%
1283         \FPsub\subValue\subValue{\xintNthElt{##1}{##2}}%
1284       }
1285       \appendToVec{#1}{\csname #1\endcsname}{\subValue}
1286     }
1287   }
1288 }
```

`\mulVec` Multiply the values of 1 to  $m$  vectors with the values of the given vector. See Hadamard product (element-wise product) for more information.

#1 - out: var name to save the result (vector) to

#2 - vector: the vector to multiply

#3 - vectors: the list of vectors with which to multiply

```

1289 \newcommand\mulVec[3][mulVecResult]{
1290   \def\vecList{\xintCSVtoList{#3}}
1291   \getVecHeight{#2}
1292   \edef\sequence{\xintSeq[+1]{+1}{\vecHeightResult}}%
```

```

1293 % Iterate each row / dimension
1294 \xintFor* ##1 in \sequence \do {%
1295     % Assign value of start vector (param #2)
1296     \FPset\mulValue{\xintNthElt{##1}{#2}}%
1297     % If this is the first iteration create a new vector
1298     \xintifForFirst{%
1299         % Iterate all vectors that are part of param #3
1300         \xintFor* ##2 in \vecList \do {%
1301             % Multiply with row value of current vector
1302             \FPmul\mulValue\mulValue{\xintNthElt{##1}{##2}}%
1303         }
1304         \newVec{#1}{\mulValue}
1305     }{%
1306     \xintFor* ##2 in \vecList \do {%
1307         \FPmul\mulValue\mulValue{\xintNthElt{##1}{##2}}%
1308     }
1309     \appendToVec{#1}{\csname #1\endcsname}{\mulValue}
1310 }
1311 }
1312 }

```

`\divVec` Divide the values of the given vector by the values of 1 to  $m$  vectors (element-wise division).

#1 - out: var name to save the result (vector) to

#2 - vector: the vector to divide

#3 - vectors: the list of vectors to divide with

```

1313 \newcommand\divVec[3][divVecResult]{
1314     \def\vecList{\xintCSVtoList{#3}}
1315     \getVecHeight{#2}
1316     \edef\sequence{\xintSeq[+1]{+1}{\vecHeightResult}}%
1317     % Iterate each row / dimension
1318     \xintFor* ##1 in \sequence \do {%
1319         % Assign value of start vector (param #2)
1320         \FPset\divValue{\xintNthElt{##1}{#2}}%
1321         % If this is the first iteration create a new vector
1322         \xintifForFirst{%
1323             % Iterate all vectors that are part of param #3
1324             \xintFor* ##2 in \vecList \do {%
1325                 \FPifeq{\xintNthElt{##1}{##2}}{0}
1326                 \PackageError{glmatrix package}{In divVec function: division by zero}{yo
1327             }
1328             \fi
1329             % Divide with row value of current vector
1330             \FPdiv\divValue\divValue{\xintNthElt{##1}{##2}}%
1331         }
1332         \newVec{#1}{\divValue}
1333     }{%
1334     \xintFor* ##2 in \vecList \do {%
1335         \FPifeq{\xintNthElt{##1}{##2}}{0}
1336         \PackageError{glmatrix package}{In divVec function: division by zero}{yo
1337     }
1338     \fi
1339     \FPdiv\divValue\divValue{\xintNthElt{##1}{##2}}%

```

```

1340     }
1341     \appendToVec{#1}{\csname #1\endcsname}{\divValue}
1342   }
1343 }
1344 }

```

`\scaleVec` Multiply the values of the given vector by a scalar.

```

#1 - out: var name to save the result (vector) to
#2 - vector: the vector to scale
#3 - scalar: the scalar to scale with
1345 \newcommand\scaleVec[3][scaleVecResult] {%
1346   \xintFor* ##1 in #2 \do {%
1347     \FPmul\tmpVal{##1}{#3}
1348     \xintifForFirst{%
1349       \newVec{#1}{\tmpVal}
1350     }{%
1351       \appendToVec{#1}{\csname #1\endcsname}{\tmpVal}
1352     }
1353   }
1354 }

```

`\divVecScalar` Divide the values of the given vector by a scalar.

```

#1 - out: var name to save the result (vector) to
#2 - vector: the vector to divide
#3 - scalar: the scalar to divide with
1355 \newcommand\divVecScalar[3][divVecScalarResult] {%
1356   \if\noexpand#1\relax
1357     \PackageError{glmatrix package}{argument mismatch}{Argument 1 of divVecScalar seems
1358   \else
1359   \fi
1360   \if\noexpand#2\relax
1361   \else
1362     \PackageError{glmatrix package}{argument mismatch}{argument 2 of divVecScalar does m
1363   \fi
1364   \FPifeq{#3}{0}
1365     \PackageError{glmatrix package}{In div vector by scalar}{divisor is 0}
1366   \else
1367   \fi
1368   \xintFor* ##1 in #2 \do {%
1369     \FPdiv\divNumRes{##1}{#3}
1370     \xintifForFirst{%
1371       \newVec{#1}{\divNumRes}
1372     }{%
1373       \appendToVec{#1}{\csname #1\endcsname}{\divNumRes}
1374     }
1375   }
1376 }

```

`\negateVec` Negate the values of the given vector.

```

#1 - out: var name to save the result (vector) to
#2 - vector: the vector to negate

```



```

1377 \newcommand\negateVec[2][negateVecResult]{
1378   \xintFor* ##1 in #2 \do {%
1379     \FPneg\tmp{##1}
1380     \xintifForFirst{%
1381       \newVec{#1}{\tmp}
1382     }{%
1383       \appendToVec{#1}{\csname #1\endcsname}{\tmp}
1384     }
1385   }
1386 }

```

`\scaleAndAddVec` Adds two vectors after multiplying the second one by a scalar.

#1 - out: var name to save the result (vector) to  
 #2 - vector v: the vector to add to  
 #3 - vector w: the vector that is multiplied and then added  
 #4 - scalar: the scalar that is the multiplier

```

1387 \newcommand\scaleAndAddVec[4][scaleAndAddVecResult]{
1388   \scaleVec{#3}{#4}
1389   \addVec{#1}{#2}{\scaleVecResult}
1390 }

```

`\lerpVec` Linearly interpolate between the values of two vectors (element-wise interpolation).

#1 - out: var name to save the result (vector) to  
 #2 - vector v: the start vector  
 #3 - vector w: the end vector  
 #4 - scalar t: interpolation amount range [0-1]

```

1391 \newcommand\lerpVec[4][lerpVecResult]{
1392   % \def\vecList{\xintCSVtoList{#3}}
1393   \getVecHeight{#2}
1394   \edef\sequence{\xintSeq[+1]{+1}{\vecHeightResult}}%
1395   % Iterate each row / dimension
1396   \xintFor* ##1 in \sequence \do {%
1397     % Assign value of start vector (param #2)
1398     % \FPset\divValue{\xintNthElt{##1}{#2}}%
1399     % If this is the first iteration create a new vector
1400     \xintifForFirst{%
1401       \lerpScalar{\xintNthElt{##1}{#2}}{\xintNthElt{##1}{#3}}{#4}
1402       \newVec{#1}{\lerpScalarResult}
1403     }{%
1404       \lerpScalar{\xintNthElt{##1}{#2}}{\xintNthElt{##1}{#3}}{#4}
1405       \appendToVec{#1}{\csname #1\endcsname}{\lerpScalarResult}
1406     }
1407   }
1408 }

```

`\slerpVec` Performs a spherical linear interpolation between two vectors. This expects vector v and vector w to be normalized.

#1 - out: var name to save the result (vector) to  
 #2 - vector v: the start vector

```

#3 - vector w: the end vector
#4 - scalar t: interpolation amount range [0-1]
1409 \newcommand\slerpVec[4][slerpVecResult] {%
1410   \angleVec{#2}{#3}
1411   \FPifeq{\angleVecResult}{0}
1412     \lerpVec{#1}{#2}{#3}{#4}
1413   \else
1414     \FPSin\angleTotal{\angleVecResult}
1415     \FPsub\ratioA{1}{#4}
1416     \FPMul\ratioA\ratioA{\angleVecResult}
1417     \FPSin\ratioA\ratioA
1418     \FPdiv\ratioA\ratioA\angleTotal
1419     \FPMul\ratioB{#4}{\angleVecResult}
1420     \FPSin\ratioB\ratioB
1421     \FPdiv\ratioB\ratioB\angleTotal
1422     \scaleVec[tmpA]{#2}{\ratioA}
1423     \scaleVec[tmpB]{#3}{\ratioB}
1424     \addVec{#1}{\tmpA}{\tmpB}
1425   \fi
1426 }

```

`\dotVec` Calculate the dotproduct of the given two vectors. Other names for this operation are inner product and scalarproduct.

`#1` - out: var name to save the result (scalar) to

`#2` - vector v: the first vector

`#3` - vector w: the second vector

```

1427 \newcommand\dotVec[3][dotVecResult] {%
1428   \if\noexpand#1\relax
1429     \PackageError{glmatrix package}{argument mismatch}{Argument 1 of dotVec seems to be
1430   \else
1431     \fi
1432   \if\noexpand#2\relax
1433     \else
1434       \PackageError{glmatrix package}{argument mismatch}{argument 2 of dotVec does not see
1435     \fi
1436   \if\noexpand#3\relax
1437     \else
1438       \PackageError{glmatrix package}{argument mismatch}{argument 3 of dotVec does not see
1439     \fi
1440   \FPifeq{\xintLength\expandafter{#2}}{\xintLength\expandafter{#3}}
1441   \else
1442     \PackageError{glmatrix package}{In dot product}{vector A and B do not have same leng
1443   \fi
1444   \FPset\res{0}
1445   \FPset\tmp{0}
1446   \edef\innersequence{\xintSeq[+1]{+1}{\expandafter\xintLength\expandafter{#2}}}%
1447   \xintFor* ##1 in \innersequence \do {%
1448     \FPset\valueA{\xintNthElt{##1}{#2}}
1449     \FPset\valueB{\xintNthElt{##1}{#3}}
1450     \FPMul\tmp\valueA\valueB
1451     \FPadd\res\res\tmp
1452   }
1453   \newScalar{#1}{\res}

```

1454 }

`\lenVec` Calculate the length of the given vector.

#1 - out: var name to save the result (scalar) to  
 #2 - vector: the vector for which to calculate the length

```

1455 \newcommand\lenVec[2][lenVecResult] {%
1456     \if\noexpand#1\relax
1457         \PackageError{glmatrix package}{argument mismatch}{Argument 1 of lenVec seems to be
1458     \else
1459     \fi
1460     \if\noexpand#2\relax
1461     \else
1462         \PackageError{glmatrix package}{argument mismatch}{argument 2 of lenVec does not see
1463     \fi
1464     \FPset\normTotal{0}
1465     \FPset\iterVal{0}
1466     % Iterate each value, multiply it with itself and sum up
1467     \xintFor* ##1 in #2 \do {%
1468         % No idea why this needs to be abs,
1469         % but negative values are a problem here
1470         \FPabs\iterVal{##1}
1471         \FPpow\tmpNormB\iterVal{2}
1472         \FPadd\normTotal\normTotal\tmpNormB
1473     }
1474     % Evaluate root for numbered result
1475     \FProot\normRooted\normTotal{2}
1476     % Set output var
1477     \newScalar{#1}{\normRooted}
1478 }
```

`\sqrLenVec` Calculate the squared length of the given vector.

#1 - out: var name to save the result (scalar) to  
 #2 - vector: the vector for which to calculate the squared length

```

1479 \newcommand\sqrLenVec[2][sqrLenVecResult] {%
1480     \FPset\normTotal{0}
1481     \FPset\iterVal{0}
1482     % Iterate each value, multiply it with itself and sum up
1483     \xintFor* ##1 in #2 \do {%
1484         % No idea why this needs to be abs,
1485         % but negative values are a problem here
1486         \FPabs\iterVal{##1}
1487         \FPpow\tmpNormB\iterVal{2}
1488         \FPadd\normTotal\normTotal\tmpNormB
1489     }
1490     % Set output var
1491     \newScalar{#1}{\normTotal}
1492 }
```

`\normalizeVec` Divide all components of the vector by its length.

#1 - out: var name to save the result (vector) to  
 #2 - vector: the vector which to normalize

```

1493 \newcommand\normalizeVec[2][normalizeVecResult] {%
1494     \if\noexpand#1\relax
1495         \PackageError{glmatrix package}{argument mismatch}{Argument 1 of normalizeVec seems
1496     \else
1497     \fi
1498     \if\noexpand#2\relax
1499     \else
1500         \PackageError{glmatrix package}{argument mismatch}{argument 2 of normalizeVec does n
1501     \fi
1502     \lenVec{#2}
1503     \divVecScalar[#1]{#2}{\lenVecResult}
1504 }

```

`\crossVec` Calculate crossproduct for the two given vectors.

#1 - out: var name to save the result (vector) to

#2 - vector v: the first vector

#3 - vector w: the second vector

```

1505 \newcommand\crossVec[3][crossVecResult] {%
1506     \FPifeq{\expandafter\xintLength\expandafter{#2}}{3}
1507     \else
1508         \PackageError{glmatrix package}{In cross product}{vector A is not of length 3}
1509     \fi
1510     \FPifeq{\expandafter\xintLength\expandafter{#3}}{3}
1511     \else
1512         \PackageError{glmatrix package}{In cross product}{vector A is not of length 3}
1513     \fi
1514     \FPset\aa{\xintNthElt{1}{#2}}
1515     \FPset\ab{\xintNthElt{2}{#2}}
1516     \FPset\ac{\xintNthElt{3}{#2}}
1517     \FPset\ba{\xintNthElt{1}{#3}}
1518     \FPset\bb{\xintNthElt{2}{#3}}
1519     \FPset\bc{\xintNthElt{3}{#3}}
1520     \FPset\tmp{0}
1521     % x component
1522     \FPset\scalarX{0}
1523     \FPmul\scalarX\ab\bc
1524     \FPmul\tmp\ac\bb
1525     \FPsub\scalarX\scalarX\tmp
1526     % y component
1527     \FPset\scalarY{0}
1528     \FPmul\scalarY\ac\ba
1529     \FPmul\tmp\aa\bc
1530     \FPsub\scalarY\scalarY\tmp
1531     % z component
1532     \FPset\scalarZ{0}
1533     \FPmul\scalarZ\aa\bb
1534     \FPmul\tmp\ab\ba
1535     \FPsub\scalarZ\scalarZ\tmp
1536     \newVec{#1}{\scalarX,\scalarY,\scalarZ}
1537 }

```

`\crossVecTwoD` Calculate crossproduct for the two given vectors.

```

#1 - out: var name to save the result (vector) to
#2 - vector v: the first vector
#3 - vector w: the second vector
1538 \newcommand\crossVecTwoD[3][crossVecResult] {%
1539   \FPifeq{\expandafter\xintLength\expandafter{#2}}{2}
1540   \else
1541     \PackageError{glmatrix package}{In cross product}{vector A is not of length 2}
1542   \fi
1543   \FPifeq{\expandafter\xintLength\expandafter{#3}}{2}
1544   \else
1545     \PackageError{glmatrix package}{In cross product}{vector A is not of length 2}
1546   \fi
1547   \FPset\aa{\xintNthElt{1}{#2}}
1548   \FPset\ab{\xintNthElt{2}{#2}}
1549   \FPset\ba{\xintNthElt{1}{#3}}
1550   \FPset\bb{\xintNthElt{2}{#3}}
1551   \FPmul\scalarUxVy{\xintNthElt{1}{#2}}{\xintNthElt{2}{#3}}
1552   \FPmul\scalarUyVx{\xintNthElt{2}{#2}}{\xintNthElt{1}{#3}}
1553   \FPsub\scalarResult{\scalarUxVy}{\scalarUyVx}
1554   \newScalar{#1}{\scalarResult}
1555 }

```

`\angleVec` Get the angle between the given two vectors in radian.

```

#1 - out: var name to save the result (scalar) to
#2 - vector v: the first vector
#3 - vector w: the second vector
1556 \newcommand\angleVec[3][angleVecResult] {%
1557   \dotVec[dotVW]{#2}{#3}
1558   \lenVec[normV]{#2}
1559   \lenVec[normW]{#3}
1560   \mulScalar[mulNormVW]{\normV}{\normW}
1561   \divScalar{dotVW}{mulNormVW}
1562   \arccosScalar[#1]{\scalarDivResult}
1563 }

```

`\distVec` Calculates the euclidian distance between the given two vectors.

```

#1 - out: var name to save the result (scalar) to
#2 - vector v: the first vector
#3 - vector w: the second vector
1564 \newcommand\distVec[3][distVecResult] {%
1565   \sqrDistVec{#2}{#3}
1566   \FProot\res\sqrDistVecResult{2}
1567   \newScalar{#1}{\res}
1568 }

```

`\sqrDistVec` Calculates the squared euclidian distance between two vectors.

```

#1 - out: var name to save the result (scalar) to
#2 - vector v: the first vector
#3 - vector w: the second vector
1569 \newcommand\sqrDistVec[3][sqrDistVecResult] {%

```

```

1570 \getVecHeight[getVecHeightA]{#2}
1571 \getVecHeight[getVecHeightB]{#3}
1572 \FPifeq{\getVecHeightA}{\getVecHeightB}
1573 \else
1574     \PackageError{glmatrix package}{In sqrt distance}{vector #2 and #3 do not have same
1575 \fi
1576 \FPset\res{0}
1577 \FPset\tmp{0}
1578 \edef\sequence{\xintSeq[+1]{+1}{\getVecHeightA}}%
1579 \xintFor* ##1 in \sequence \do {%
1580     \FPset\valueA{\xintNthElt{##1}{#2}}
1581     \FPset\valueB{\xintNthElt{##1}{#3}}
1582     \FPsub\tmp\valueA\valueB
1583     \FPMul\tmp\tmp\tmp
1584     \FPadd\res\res\tmp
1585 }
1586 \newScalar{#1}{\res}
1587 }

```

`\transformVec` Multiply the given vector with a matrix (for transformation).

`#1` - out: var name to save the result (vector) to

`#2` - matrix: the matrix to transform the vector with

`#3` - vector: the vector to transform

```

1588 \newcommand\transformVec[3][transformVecResult] {%
1589     \if\noexpand#1\relax
1590         \PackageError{glmatrix package}{argument mismatch}{Argument 1 of transformVec seems
1591 \else
1592     \fi
1593     \if\noexpand#2\relax
1594     \else
1595         \PackageError{glmatrix package}{argument mismatch}{argument 2 of transformVec does n
1596     \fi
1597     \if\noexpand#3\relax
1598     \else
1599         \PackageError{glmatrix package}{argument mismatch}{argument 3 of transformVec does n
1600     \fi
1601     \getVecHeight{#3}%
1602     \getMatHeight{#2}%
1603     \getMatWidth{#2}%
1604     % Check that matrix and vector can be multiplied
1605     \FPifeq{\matWidthResult}{\vecHeightResult}
1606     \else
1607         \PackageError{glmatrix package}{Dimension mismatch for matrix and vector, could not
1608     \fi
1609     % Create iterator sequences from height of vector and matrix
1610     \edef\vecHeightSequence{\xintSeq[+1]{+1}{\vecHeightResult}}%
1611     \edef\matHeightSequence{\xintSeq[+1]{+1}{\matHeightResult}}%
1612     % Iterate all rows of the matrix
1613     \xintFor* ##2 in \matHeightSequence \do {%
1614         \FPset\cellVal{0}
1615         % Differentiate between first and other entries to create the new vector variable
1616         \xintifForFirst{%
1617             % Now iterate each entry of the current row

```

```

1618         % Use this indec to access the entry in the matrix and in the vector
1619         \xintFor* ##3 in \vecHeightSequence \do {%
1620             \FPmul\tmp{%
1621                 \xintNthElt{##3}{\xintNthElt{##2}{#2}}%
1622             }{%
1623                 \xintNthElt{##3}{#3}%
1624             }
1625             % Add to storage variable
1626             \FPadd\cellVal\cellVal\tmp
1627         }
1628         \FPclip\cellVal\cellVal
1629         \newVec{#1}{\cellVal}
1630     }{
1631         \xintFor* ##3 in \vecHeightSequence \do {%
1632             \FPmul\tmp{%
1633                 \xintNthElt{##3}{\xintNthElt{##2}{#2}}%
1634             }{%
1635                 \xintNthElt{##3}{#3}%
1636             }
1637             \FPadd\cellVal\cellVal\tmp
1638         }
1639         \FPclip\cellVal\cellVal
1640         \appendToVec{#1}{\csname #1\endcsname}{\cellVal}
1641     }
1642 }
1643 % Test if width and height is correct
1644 \expandafter\getVecHeight\expandafter{\csname #1\endcsname}%
1645 \FPifeq{\matHeightResult}{\vecHeightResult}
1646 \else
1647     \PackageError{glmatrix package}{Vector height does not match expectation}{height of
1648 \fi
1649 }

```

`\tensorVec` Outer product: Multiply a vector (column) with itself (row) resulting in a matrix.  
(Dyadic product, Tensor product)

#1 - out: var name to save the result (matrix) to  
#2 - vector: the vector to multiply with itself

```

1650 \newcommand\tensorVec[2][tensorVecResult]{%
1651     \newMatFromRowVecs{rowMatrix}{#2}
1652     \newMatFromColVecs{colMatrix}{#2}
1653     \mulMat{#1}{\colMatrix}{\rowMatrix}
1654 }

```

`\minVec` Return the componentwise minimum of the given two vectors.

#1 - out: var name to save the result (vector) to  
#2 - vector v: the first vector  
#3 - vector w: the second vector

```

1655 \newcommand\minVec[3][minVecResult]{
1656     \if\noexpand#1\relax
1657         \PackageError{glmatrix package}{argument mismatch}{Argument 1 of minVec seems to be
1658     \else
1659     \fi

```

```

1660 \FPifseq{\expandafter\xintLength\expandafter{#2}}{\expandafter\xintLength\expandafter{#3}}
1661 \else
1662   \PackageError{glmatrix package}{incompatible size}{vector A and B of minVec do not h
1663 \fi
1664 \edef\innersequence{\xintSeq[+1]{+1}{\expandafter\xintLength\expandafter{#2}}}%
1665 \xintFor* ##1 in \innersequence \do {%
1666   \minScalar{\xintNthElt{##1}{#2}}{\xintNthElt{##1}{#3}}
1667   \xintifForFirst{%
1668     \newVec{#1}{\minScalarResult}
1669   }{%
1670     \appendToVec{#1}{\csname #1\endcsname}{\minScalarResult}
1671   }
1672 }
1673 }

```

`\maxVec` Return the componentwise maximum of the given two vectors.

#1 - out: var name to save the result (vector) to

#2 - vector v: the first vector

#3 - vector w: the second vector

```

1674 \newcommand\maxVec[3][maxVecResult]{
1675   \if\noexpand#1\relax
1676     \PackageError{glmatrix package}{argument mismatch}{Argument 1 of maxVec seems to be
1677   \else
1678     \fi
1679   \FPifseq{\expandafter\xintLength\expandafter{#2}}{\expandafter\xintLength\expandafter{#3}}
1680   \else
1681     \PackageError{glmatrix package}{incompatible size}{vector A and B of maxVec do not h
1682   \fi
1683   \edef\innersequence{\xintSeq[+1]{+1}{\expandafter\xintLength\expandafter{#2}}}%
1684   \xintFor* ##1 in \innersequence \do {%
1685     \maxScalar{\xintNthElt{##1}{#2}}{\xintNthElt{##1}{#3}}
1686     \xintifForFirst{%
1687       \newVec{#1}{\maxScalarResult}
1688     }{%
1689       \appendToVec{#1}{\csname #1\endcsname}{\maxScalarResult}
1690     }
1691   }
1692 }

```

`\solveVecEQL` Solve the equation  $vector v * x + vector w = 0$  for  $x$ . The command expects the input to be two vectors of the same length. The command iterates both vectors componentwise and solves the equation per component. If the the result does not match across all components the equation can not be solved.

#1 - out: var name to save the result (scalar) to

#2 - vector v: the first vector

#3 - vector w: the second vector

```

1693 \newcommand\solveVecEQL[3][solveVecEQLResult]{
1694   \getVecHeight{#3}%
1695   \edef\vecHeightSequence{\xintSeq[+1]{+1}{\vecHeightResult}}%
1696   \xintFor* ##1 in \vecHeightSequence \do {%
1697     % \FPneg\tmpVar{\xintNthElt{##1}{#3}}
1698     \FPsolve\glCurSol{\xintNthElt{##1}{#2}}{\xintNthElt{##1}{#3}}

```



```
1699     \xintifForFirst{%
1700         \setScalar{#1}{\glCurSol}
1701     }{%
1702         \FPifeq{\csname #1\endcsname}{\glCurSol}
1703         \else
1704             \PackageError{glmatrix package}{Could not solve equation}{varying results fo
1705         \fi
1706     }
1707 }
1708 }
```

## 5 License

Released under BSD 3-Clause License

BSD 3-Clause License

Copyright (c) 2026, Rene Warnking

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.