The glmatrix Package

Rene Warnking

Version 1.0.0
2026/04/03

Contents

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

`newMat` `{`⟨*out*⟩`}{`⟨*scalars*⟩`}`

`{`⟨*out*⟩`}`: var name to save the result (matrix) to
`{`⟨*scalars*⟩`}`: 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}}}
```
$\myNewMatrix \mapsto \begin{pmatrix} 0.5 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$
```
\newMat{myNewMatrix}{{{0.5,2,3}}}
```
$\myNewMatrix \mapsto \begin{pmatrix} 0.5 & 2 & 3 \end{pmatrix}$
```
\newMat{myNewMatrix}{{{0.5}{4}{7}}}
```
$\myNewMatrix \mapsto \begin{pmatrix} 0.5 \\ 4 \\ 7 \end{pmatrix}$

`newMatFromRow` `{`⟨*out*⟩`}{`⟨*row*⟩`}`

`{`⟨*out*⟩`}`: var name to save the result (matrix) to
`{`⟨*row*⟩`}`: matrix row that is assigned to out

Description

Create a new matrix from the given row.

`newMatFromRowVecs` `{`⟨*out*⟩`}{`⟨*rows*⟩`}`

`{`⟨*out*⟩`}`: var name to save the result (matrix) to
`{`⟨*rows*⟩`}`: 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` `{`⟨*out*⟩`}{`⟨*columns*⟩`}`

`{`⟨*out*⟩`}`: var name to save the result (matrix) to
`{`⟨*columns*⟩`}`: the columns that are used to construct the matrix

Description

Create a new matrix from the given column vectors.

`newEmptyMat` `{`⟨*out*⟩`}`

`{`⟨*out*⟩`}`: 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` `{`⟨*out*⟩`}{`⟨*scalars*⟩`}`

`{`⟨*out*⟩`}`: var name to save the result (matrix) to
`{`⟨*scalars*⟩`}`: 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` `{`⟨*out*⟩`}{`⟨*matrix*⟩`}{`⟨*vector r*⟩`}`

`{`⟨*out*⟩`}`: var name to save the result (matrix) to
`{`⟨*matrix*⟩`}`: the matrix to append the vector to
`{`⟨*vector r*⟩`}`: 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` `{`⟨*out*⟩`}{`⟨*vector t*⟩`}`

`{`⟨*out*⟩`}`: var name to save the result (matrix) to
`{`⟨*vector t*⟩`}`: 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

$$o = \mathbf{S} \in \mathbb{R}^{4\times4} = \begin{bmatrix} \mathbf{s}_x & 0 & 0 & 0 \\ 0 & \mathbf{s}_y & 0 & 0 \\ 0 & 0 & \mathbf{s}_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

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

$$o = \mathbf{R} \in \mathbb{R}^{3\times3}$$

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}
```

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

`newProjMat` `{`⟨*out*⟩`}`

`{`⟨*out*⟩`}`: var name to save the result (matrix) to

Description

Create a new default projection matrix.

Equation

$o = \mathbf{P} \in \mathbb{R}^{4\times4}$

Example

```
\newProjMat{\myNewMatrix}
```

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

`newProjMatGen` `{`⟨*out*⟩`}`

`{`⟨*out*⟩`}`: var name to save the result (matrix) to

Description

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

Equation

$o = \mathbf{P} \in \mathbb{R}^{4\times4}$

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: frustMatResult
`{`⟨*fovx*⟩`}`: the fovx scalar
`{`⟨*fovy*⟩`}`: the fovy scalar
`{`⟨*near*⟩`}`: the position of the near plane (scalar)
`{`⟨*far*⟩`}`: the psition of the far lane (scalar)

Description

Create a new frustum matrix.

Equation

$o = \mathbf{F} \in \mathbb{R}^{4\times4}$

Example

\newScalar{fovx}{90}
\newScalar{fovy}{90}
\newScalar{near}{1}
\newScalar{far}{6}
\newFrustumMat[myNewMatrix]{\fovx}{\fovy}{\near}{\far}
\myNewMatrix $\mapsto$ $\begin{pmatrix} 0.6174 & 0 & 0 & 0 \\ 0 & 06174 & 0 & 0 \\ 0 & 0 & -1.4 & -2.4 \\ 0 & 0 & -1 & 0 \end{pmatrix}$

`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

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

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

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

Example

\newMat{myNewMatrix}{{{$a_{11}, \dots, a_{1n}$}{$a_{21}, \dots, a_{2n}$}{$a_{m1}, \dots, a_{mn}$}}}
\getMatWidth{myNewMatrix}
\matWidthResult $\mapsto n$

`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

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

Example

\newMat{myNewMatrix}{{{0.5,2,3}{4,5,6}{7,8,9}}}
\transposeMat{\myNewMatrix} $\mapsto \begin{bmatrix} 0.5 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{bmatrix}$

`addMat` `[`⟨*out*⟩`]{`⟨*matrix*⟩`}{`⟨*matrices*⟩`}`

`[`⟨*out*⟩`]`: var name to save the result (matrix) to, default: addMatResult
`{`⟨*matrix*⟩`}`: the matrix to add to
`{`⟨*matrices*⟩`}`: the matrizes 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 matrizes 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 ↦ 
\mulMatScalar[resultName]{\myNewMatrix}{\myNewScalar}
\resultName ↦ 
```

$$\mulMatScalarResult \mapsto \begin{bmatrix} 1 & 4 & 6 \\ 8 & 10 & 12 \\ 14 & 16 & 18 \end{bmatrix}$$

$$\resultName \mapsto \begin{bmatrix} 1 & 4 & 6 \\ 8 & 10 & 12 \\ 14 & 16 & 18 \end{bmatrix}$$

`mulMatVec` `[`⟨*out*⟩`]{`⟨*matrix*⟩`}{`⟨*vector*⟩`}`

`[`⟨*out*⟩`]`: var name to save the result (vector) to, default: mulMatVecResult
`{`⟨*matrix*⟩`}`: the matrix to transform the vetor 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 ↦ 
\mulMatVec[resultName]{\myNewMatrix}{\myNewVec}
\resultName ↦ 
```

$$\mulMatVecResult \mapsto \begin{pmatrix} 5.25 \\ 19 \\ 29.5 \end{pmatrix}$$

$$\resultName \mapsto \begin{pmatrix} 5.25 \\ 19 \\ 29.5 \end{pmatrix}$$

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 matrizes 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

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

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 ↦ 1.5
```

3.2 scalar

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

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

`newScalar` {⟨*out*⟩}{⟨*scalar*⟩}

{⟨*out*⟩}: var name to save the result (scalar) to
{⟨*scalar*⟩}: scalar value that is assigned to out

Description

Create a new scalar and assign the given value.

Example

\newScalar{myNewScalar}{3}
\myNewScalar ↦ 3

`newScalarPi` {⟨*out*⟩}{⟨*scalar*⟩}

{⟨*out*⟩}: var name to save the result (scalar) to
{⟨*scalar*⟩}: 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 ↦ 9.42

`newScalarFraction` `{`⟨*out*⟩`}{`⟨*scalar n*⟩`}{`⟨*scalar d*⟩`}`

`{`⟨*out*⟩`}`: var name to save the result (scalar) to
`{`⟨*scalar n*⟩`}`: nominator
`{`⟨*scalar d*⟩`}`: denominator

Description

Create a scalar from a given fraction

Example

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

`randomScalar` `{`⟨*out*⟩`}{`⟨*lower bound*⟩`}{`⟨*upper bound*⟩`}`

`{`⟨*out*⟩`}`: var name to save the result (scalar) to
`{`⟨*lower bound*⟩`}`: scalar that limits the possible values at the bottom
`{`⟨*upper bound*⟩`}`: 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` `[`⟨*out*⟩`]{`⟨*scalar*⟩`}`

`[`⟨*out*⟩`]`: var name to save the result (scalar) to, default: scalarCopyResult
`{`⟨*scalar*⟩`}`: 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` `{`⟨*out*⟩`}{`⟨*scalar*⟩`}`

`{`⟨*out*⟩`}`: var name to save the result (scalar) to
`{`⟨*scalar*⟩`}`: scalar value that is assigned to out

Description

Set the value of a given scalar.

Example

\newScalar{myNewScalar}{3}
\setScalar{myNewScalar}{4}
\myNewScalar $\mapsto$ 4

`zeroScalar` `{`⟨*out*⟩`}`

`{`⟨*out*⟩`}`: var name to save the result (scalar) to

Description

Set or create a scalar with value 0.

Example

\zeroScalar{myZeroScalar}
\myZeroScalar $\mapsto$ 0

`printScalar` `[`⟨*fraction*⟩`]{`⟨*scalar*⟩`}`

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

Description

Print the value of the given scalar.

Example

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

`printScalarAsPi` `[`⟨*fraction*⟩`]{`⟨*scalar*⟩`}`

`[`⟨*fraction*⟩`]`: whether or not the value should be printed as a fraction (val=1), default: 0
`{`⟨*scalar*⟩`}`: 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 schould be printed

Description

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.

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 schould 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 schould be rounded

Description

Round the value ot the given scalar and save it.

Example

\newScalar{myNewScalar}{0.5}
\roundScalar{\myNewScalar}
\myNewScalar ↦ 1

`ceilScalar` `[`⟨*out*⟩`]{`⟨*scalar*⟩`}`

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

Description

Ceil the value of the given scalar.

Example

\newScalar{myNewScalar}{0.5}
\ceilScalar{\myNewScalar}
\myNewScalar ↦ 1

`floorScalar` `[`⟨*out*⟩`]{`⟨*scalar*⟩`}`

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

Description

Floor the value of the given scalar.

Example

\newScalar{myNewScalar}{0.5}
\floorScalar{\myNewScalar}
\myNewScalar ↦ 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 ↦ 5

`radianToDegree` `[`⟨*out*⟩`]{`⟨*scalar*⟩`}`

`[`⟨*out*⟩`]`: var name to save the result (scalar) to, default: radianToDegreeResult
`{`⟨*scalar*⟩`}`: the scalar which to convert

Description

Convert the value of the given scalar to degree.

Example

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

`toRadian` `[`⟨*out*⟩`]{`⟨*scalar*⟩`}`

`[`⟨*out*⟩`]`: var name to save the result (scalar) to, default: degreeToRadianResult
`{`⟨*scalar*⟩`}`: the scalar which to convert

Description

Convert the value of the given scalar to radian.

Example

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

`addScalar` `[`⟨*out*⟩`]{`⟨*scalar s*⟩`}{`⟨*scalar t*⟩`}`

`[`⟨*out*⟩`]`: var name to save the result (scalar) to, default: scalarAddResult
`{`⟨*scalar s*⟩`}`: the first scalar of the sum
`{`⟨*scalar t*⟩`}`: 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 ↦ 3
```

`subScalar` `[`⟨*out*⟩`]{`⟨*scalar s*⟩`}{`⟨*scalar t*⟩`}`

`[`⟨*out*⟩`]`: 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 ↦ − 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 multply

Description

Multiply the values of two scalars.

Equation

$o = s * t$

Example

```
\newScalar{myNewScalarA}{2}
\newScalar{myNewScalarB}{3}
\mulScalar{\myNewScalarA}{\myNewScalarB}
\scalarMulResult ↦ 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` `[`⟨*out*⟩`]{`⟨*scalar*⟩`}`

`[`⟨*out*⟩`]`: var name to save the result (scalar) to, default: scalarArccosResult
`{`⟨*scalar*⟩`}`: 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` `[`⟨*out*⟩`]{`⟨*scalar s*⟩`}{`⟨*scalar t*⟩`}{`⟨*scalar u*⟩`}`

`[`⟨*out*⟩`]`: var name to save the result (scalar) to, default: scalarMulAndAddResult
`{`⟨*scalar s*⟩`}`: the scalar to add to
`{`⟨*scalar t*⟩`}`: the scalar that is multiplied and then added
`{`⟨*scalar u*⟩`}`: the scalar that is the muliplier

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` `[`⟨*out*⟩`]{`⟨*scalar i*⟩`}{`⟨*scalar j*⟩`}{`⟨*scalar t*⟩`}`

`[`⟨*out*⟩`]`: var name to save the result (scalar) to, default: lerpScalarResult
`{`⟨*scalar i*⟩`}`: the start scalar

`{`⟨*scalar j*⟩`}`: the end scalar
`{`⟨*scalar t*⟩`}`: interpolation amount range [0-1]

Description

Lineraly 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` `[`⟨*out*⟩`]{`⟨*scalar s*⟩`}{`⟨*scalar t*⟩`}`

`[`⟨*out*⟩`]`: var name to save the result (scalar) to, default: minScalarResult
`{`⟨*scalar s*⟩`}`: the first scalar
`{`⟨*scalar t*⟩`}`: the second scalar

Description

Return the minimum of two scalar values.

Equation

$o = s$ falls $s < t$ sonst t

Example

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

`maxScalar` `[`⟨*out*⟩`]{`⟨*scalar s*⟩`}{`⟨*scalar t*⟩`}`

`[`⟨*out*⟩`]`: var name to save the result (scalar) to, default: maxScalarResult
`{`⟨*scalar s*⟩`}`: first scalar value
`{`⟨*scalar t*⟩`}`: 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 $\mapsto$ 3
\maxScalar[resultName]{\myNewScalarA}{\myNewScalarB}
\resultName $\mapsto$ 3

`betweenScalar` `[`⟨*out*⟩`]{`⟨*scalar*⟩`}{`⟨*lower bound*⟩`}{`⟨*upper bound*⟩`}`

`[`⟨*out*⟩`]`: var name to save the result (scalar) to (0 or 1), default: betweenScalarResult
`{`⟨*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 $\mapsto$ 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
\newVec	`{`⟨*out*⟩`}`, `{`⟨*values*⟩`}`
\newVecPi	`{`⟨*out*⟩`}`, `{`⟨*values*⟩`}`
\randomVec	`{`⟨*out*⟩`}`, `{`⟨*dimensions*⟩`}`, `{`⟨*lower bound*⟩`}`, `{`⟨*upper bound*⟩`}`

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

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

`newVec` {⟨*out*⟩}{⟨*values*⟩}

{⟨*out*⟩}: var name to save the result (vector) to
{⟨*values*⟩}: 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` `{`⟨*out*⟩`}{`⟨*values*⟩`}`

`{`⟨*out*⟩`}`: var name to save the result (vector) to
`{`⟨*values*⟩`}`: 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` `{`⟨*out*⟩`}{`⟨*dimensions*⟩`}{`⟨*lower bound*⟩`}{`⟨*upper bound*⟩`}`

`{`⟨*out*⟩`}`: var name to save the result (vector) to
`{`⟨*dimensions*⟩`}`: scalar that defines the number of dimensions out should have
`{`⟨*lower bound*⟩`}`: scalar that limits the possible values at the bottom
`{`⟨*upper bound*⟩`}`: 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` `[`⟨*out*⟩`]{`⟨*vector*⟩`}`

`[`⟨*out*⟩`]`: var name to save the result (vector) to, default: copyVecResult
`{`⟨*vector*⟩`}`: 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` `[`⟨*dims*⟩`]{`⟨*out*⟩`}`

`[`⟨*dims*⟩`]`: number of dimensions (rows), default: 3
`{`⟨*out*⟩`}`: 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` `{`⟨*out*⟩`}{`⟨*vector*⟩`}{`⟨*scalar*⟩`}`

`{`⟨*out*⟩`}`: var name to save the result (vector) to
`{`⟨*vector*⟩`}`: the currently present values
`{`⟨*scalar*⟩`}`: 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` `[`⟨*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 vector notation.

Equation

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

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

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

Example

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

`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*⟩`}`

`[`⟨*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 as a multiple of π in vector notation.

Equation

$$\begin{pmatrix} \left\{\frac{\mathbf{v}_1}{\pi}\right\} \\ \dots \\ \left\{\frac{\mathbf{v}_n}{\pi}\right\} \end{pmatrix} \pi$$

Example

```
\newVec{myNewVec}{0.5, 1, 2}
\printVecAsPi{\myNewVec}
```
$\mapsto \begin{pmatrix} 0.5\pi \\ \pi \\ 2\pi \end{pmatrix}$

`printVecAsPiT` `[`⟨*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 as a multiple of Pi in transposed notation.

Equation

$$\left(\left\{\frac{\mathbf{v}_1}{\pi}\right\}, \dots, \left\{\frac{\mathbf{v}_n}{\pi}\right\}\right)^{\mathsf{T}} \pi$$

Example

```
\newVec{myNewVec}{0.5, 1, 2}
\printVecAsPi{\myNewVec}
```
$\mapsto (0.5\pi, \pi, 2\pi)^{\mathsf{T}}$

`printVecContent` `{`⟨*vector*⟩`}`

`{`⟨*vector*⟩`}`: the vector which values should be printed

Description

Print the values of the given vector comma separated.

Equation

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

`printVecContentXY` `{`⟨*vector*⟩`}`

`{`⟨*vector*⟩`}`: 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` `{`⟨*vector*⟩`}`

`{`⟨*vector*⟩`}`: 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` `{`⟨*vector*⟩`}`

`{`⟨*vector*⟩`}`: 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` `[`⟨*out*⟩`]{`⟨*vector*⟩`}`

`[`⟨*out*⟩`]`: var name to save the result (scalar) to, default: vecHeightResult
`{`⟨*vector*⟩`}`: 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

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

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 schould be rounded
`{`⟨*digits*⟩`}`: number of digits after comma

Description

Round the values ot the given vector.

Equation

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

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

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

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

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

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

Subract the values of 1 to m vectors from the values of the given vector.

Equation

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

Example

```
\newVec{myNewVecA}{0.5, 1, 2}
\newVec{myNewVecB}{0.5, 1, 2}
\subVec{\myNewVecA}{\myNewVecB}
\subVecResult ↦
\newVec{myNewVecA}{0.5, 1, 2}
\newVec{myNewVecB}{0.5, 1, 2}
\newVec{myNewVecC}{0.5, 1, 2}
\subVec[resultName]{\myNewVecA}{\myNewVecB, \myNewVecC}
\resultName ↦
```

$$\begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

$$\begin{pmatrix} -0.5 \\ -1 \\ -2 \end{pmatrix}$$

`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

$$\begin{aligned} o &= \mathbf{v} \odot \mathbf{vs}_1 \odot \ldots \odot \mathbf{vs}_m \\ &= \begin{pmatrix} \mathbf{v}_1 * \mathbf{vs}_{1,1} * \ldots * \mathbf{vs}_{m,1} \\ \ldots \\ \mathbf{v}_n * \mathbf{vs}_{1,n} * \ldots * \mathbf{vs}_{m,n} \end{pmatrix} \end{aligned}$$

Example

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

$$\begin{pmatrix} 0.25 \\ 1 \\ 4 \end{pmatrix}$$

`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

$$\begin{aligned} o &= \mathbf{v} \oslash \mathbf{vs}_1 \oslash \ldots \oslash \mathbf{vs}_m \\ &= \begin{pmatrix} \mathbf{v}_1 / \mathbf{vs}_{1,1} / \ldots / \mathbf{vs}_{m,1} \\ \ldots \\ \mathbf{v}_n / \mathbf{vs}_{1,n} / \ldots / \mathbf{vs}_{m,n} \end{pmatrix} \end{aligned}$$

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

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

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

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

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

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

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 muliplier

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

Lineraly 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

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

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

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

Example

\newVec{myNewVecA}{0.5, 1, 2}
\sqrLenVec{\myNewVecA}
\sqrLenVecResult $\mapsto$ 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

$$o = \begin{pmatrix} \frac{\mathbf{v}_1}{\|\mathbf{v}\|} \\ \dots \\ \frac{\mathbf{v}_n}{\|\mathbf{v}\|} \end{pmatrix}$$

Example

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

`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

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

Example

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

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

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` `[`⟨*out*⟩`]{`⟨*vector v*⟩`}{`⟨*vector w*⟩`}`

`[`⟨*out*⟩`]`: var name to save the result (scalar) to, default: solveVecEQLResult
`{`⟨*vector v*⟩`}`: the first vector
`{`⟨*vector w*⟩`}`: 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 the result does not match across all components the equation can not be solved.

Equation

$o = \frac{-\mathbf{w}_1}{\mathbf{v}_1} = \ldots = \frac{-\mathbf{w}_n}{\mathbf{v}_n}$

4 Implementation

4.1 matrix

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

\getMatHeight	[⟨*out*⟩], {⟨*matrix*⟩} matHeightResult
\getMatWidth	[⟨*out*⟩], {⟨*matrix*⟩} matWidthResult
\transposeMat	[⟨*out*⟩], {⟨*matrix*⟩} transposeMatResult
\addMat	[⟨*out*⟩], {⟨*matrix*⟩}, {⟨*matrices*⟩} addMatResult
\subMat	[⟨*out*⟩], {⟨*matrix*⟩}, {⟨*matrices*⟩} subMatResult
\mulMatScalar	[⟨*out*⟩], {⟨*matrix*⟩}, {⟨*scalar*⟩} mulMatScalarResult
\mulMatVec	[⟨*out*⟩], {⟨*matrix*⟩}, {⟨*vector*⟩} mulMatVecResult
\mulMat	[⟨*out*⟩], {⟨*matrix M*⟩}, {⟨*matrix N*⟩} mulMatResult
\getMatTranslation	[⟨*out*⟩], {⟨*matrix*⟩} matTranslationResult
\detMat	[⟨*out*⟩], {⟨*matrix*⟩} 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

```
\newcommand\newMat[2]{
    \xintFor* ##1 in #2 \do {%
        \newVec{matRow}{##1}
        \xintifForFirst{%
            \newMatFromRow{#1}{\matRow}
        }{%
            \appendRowToMat{#1}{\csname #1\endcsname}{\matRow}
        }
    }
}
```

`\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

```
\newcommand\newMatFromRow[2]{
    \newScalar{tmpWidth}{\expandafter\xintLength\expandafter{#2}}%
    \FPifeq{\tmpWidth}{1}
        \expandafter\edef\csname #1\endcsname{{\xintCSVtoList{\matRow}}}
    \else
        \expandafter\edef\csname #1\endcsname{\xintCSVtoList{\matRow}}
    \fi
}
```

`\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

```
\newcommand\newMatFromRowVecs[2]{%
    % TODO remove this?
    \FPset\numParam{\expandafter\xintLength{#2}}%
    \FPsub\numParam\numParam{1}
    \FPdiv\numParam\numParam{2}
    \FPadd\numParam\numParam{1}
    % In case only one item/row was given the
    % matrix-datastructure must be adjusted
    \FPifeq{\numParam}{1}
        \expandafter\edef\csname #1\endcsname{\xintCSVtoList{#2}}%
    \else
        \expandafter\edef\csname #1\endcsname{\xintCSVtoList{#2}}%
    \fi
}
```

`\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

```
\newcommand\newMatFromColVecs[2]{%
    \edef\tmpVecList{\xintCSVtoList{#2}}%
    \copyVec[tmpHeightVec]{{\xintNthElt{1}{\tmpVecList}}}
    \getVecHeight[numRows]{\tmpHeightVec}
    \edef\numColumns{\expandafter\xintLength\expandafter{\tmpVecList}}
    \edef\colSequence{\xintSeq[+1]{+1}{\numColumns}}
    \edef\rowSequence{\xintSeq[+1]{+1}{\numRows}}
    \newEmptyMat{#1}
    \xintFor* ##1 in \colSequence \do {%
        \xintFor* ##2 in \rowSequence \do {%
            \newScalar{tmpValue}{\xintNthElt{##1}{\xintNthElt{##2}{\tmpVecList}}}
            \xintifForFirst{%
                \newVec{matRow}{\tmpValue}
            }{%
                \appendToVec{matRow}{\matRow}{\tmpValue}
            }
        }
        \appendRowToMat{#1}{\csname #1\endcsname}{\matRow}
    }
}
```

`\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

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

`\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

```
\newcommand\newDiagMat[2]{
    \if\noexpand#1\relax
        \PackageError{glmatrix package}{argument mismatch}{Argument 1 of newDiagMat seems to
    \else
        \edef\tmpVector{\xintCSVtoList{#2}}%
        \newVec{rowA}{\xintNthElt{1}{\tmpVector}, 0, 0}
        \newVec{rowB}{0, \xintNthElt{2}{\tmpVector}, 0}
        \newVec{rowC}{0, 0, \xintNthElt{3}{\tmpVector}}
        \newMatFromRowVecs{#1}{\rowA, \rowB, \rowC}
    \fi
}
```

`\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

```
\newcommand\appendRowToMat[3]{
    \newScalar{tmpWidth}{\expandafter\xintLength\expandafter{#3}}%
    \FPifeq{\tmpWidth}{1}
        \expandafter\edef\csname #1\endcsname{#2 {\xintCSVtoList{#3}}}%
    \else
        \expandafter\edef\csname #1\endcsname{#2 \xintCSVtoList{#3}}%
    \fi
}
```

`\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

```
\newcommand\newTransMat[2]{%
    \edef\vecLength{\expandafter\xintLength\expandafter{#2 1}}
    \edef\innersequence{\xintSeq[+1]{+1}{\expandafter\xintLength\expandafter{#2}+1}}
    \xintFor* ##1 in \innersequence \do {%
        \xintifForFirst{%
            \xintFor* ##2 in \innersequence \do {%
                \newScalar{tmp}{0}
                \FPifeq{##2}{\vecLength}
                    \setScalar{tmp}{\xintNthElt{##1}{#2}}
                \else
                \fi
                \FPifeq{##1}{##2}
                    \setScalar{tmp}{1}
                \else
                \fi
                \xintifForFirst{%
                    \newVec{matRow}{\tmp}
                }{%
                    \appendToVec{matRow}{\matRow}{\tmp}
                }
            }
            \newMatFromRow{#1}{\matRow}
```

```
        }{%
            \xintFor* ##2 in \innersequence \do {%
                \newScalar{tmp}{0}
                \FPifeq{##2}{\vecLength}
                    \setScalar{tmp}{\xintNthElt{##1}{#2}}
                \else
                \fi
                \FPifeq{##1}{##2}
                    \setScalar{tmp}{1}
                \else
                \fi
                \xintifForFirst{%
                    \newVec{matRow}{\tmp}
                }{%
                    \appendToVec{matRow}{\matRow}{\tmp}
                }
            }
            \appendRowToMat{#1}{\csname #1\endcsname}{\matRow}
        }
    }
}
```

`\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

```
\newcommand\newScaleMat[2]{%
    \edef\vecLength{\expandafter\xintLength\expandafter{#2 1}}
    \edef\innersequence{\xintSeq[+1]{+1}{\expandafter\xintLength\expandafter{#2}+1}}
    \xintFor* ##1 in \innersequence \do {%
        \xintifForFirst{%
            \xintFor* ##2 in \innersequence \do {%
                \FPiflt{##2}{\vecLength}
                    \setScalar{tmp}{\xintNthElt{##1}{#2}}
                \else
                    \setScalar{tmp}{1}
                \fi
                \FPifeq{##1}{##2}
                \else
                    \setScalar{tmp}{0}
                \fi
                \xintifForFirst{%
                    \newVec{matRow}{\tmp}
                }{%
                    \appendToVec{matRow}{\matRow}{\tmp}
                }
            }
            \newMatFromRow{#1}{\matRow}
        }{%
            \xintFor* ##2 in \innersequence \do {%
                \FPiflt{##2}{\vecLength}
                    \setScalar{tmp}{\xintNthElt{##1}{#2}}
                \else
                    \setScalar{tmp}{1}
```

```
                \fi
                \FPifeq{##1}{##2}
                \else
                    \setScalar{tmp}{0}
                \fi
                \xintifForFirst{%
                    \newVec{matRow}{\tmp}
                }{%
                    \appendToVec{matRow}{\matRow}{\tmp}
                }
            }
            \appendRowToMat{#1}{\csname #1\endcsname}{\matRow}
        }
    }
}
```

`\newRotMatTwoD` Create rotation matrix from the given scalar (2D).
#1 - out: var name to save the result (matrix) to
#2 - scalar: the rotation value

```
\newcommand\newRotMatTwoD[2][rotMatResult]{%
    \FPcos\tmpCos{#2}
    \FPsin\tmpSin{#2}
    \newVec{rowB}{\tmpSin, \tmpCos, 0}
    \FPmul\tmpSin\tmpSin{-1}
    \newVec{rowA}{\tmpCos, \tmpSin, 0}
    \newVec{rowC}{0, 0, 1}
    \newMatFromRowVecs{#1}{\rowA, \rowB, \rowC}
}
```

`\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)

```
\newcommand\newRotMat[3][rotMatResult]{%
    \normalizeVec[normalizedAxis]{#2}
    \toRadian[angleInRadian]{#3}
    \FPcos\tmpCos{\angleInRadian}
    \FPsin\tmpSin{\angleInRadian}
    \subScalar[tmpCosMOne]{1}{\tmpCos}
    \mulScalar[xsquared]{\xintNthElt{1}{\normalizedAxis}}{\xintNthElt{1}{\normalizedAxis}}
    \mulScalar[xy]{\xintNthElt{1}{\normalizedAxis}}{\xintNthElt{2}{\normalizedAxis}}
    \mulScalar[xz]{\xintNthElt{1}{\normalizedAxis}}{\xintNthElt{2}{\normalizedAxis}}
    \mulScalar[ysquared]{\xintNthElt{2}{\normalizedAxis}}{\xintNthElt{2}{\normalizedAxis}}
    \mulScalar[yz]{\xintNthElt{2}{\normalizedAxis}}{\xintNthElt{3}{\normalizedAxis}}
    \mulScalar[zsquared]{\xintNthElt{3}{\normalizedAxis}}{\xintNthElt{3}{\normalizedAxis}}
    %
    \mulScalar[xsin]{\xintNthElt{1}{\normalizedAxis}}{\tmpSin}
    \mulScalar[ysin]{\xintNthElt{2}{\normalizedAxis}}{\tmpSin}
    \mulScalar[zsin]{\xintNthElt{3}{\normalizedAxis}}{\tmpSin}
    %
    \mulScalar[oneone]{\xsquared}{\tmpCosMOne}
    \addScalar[oneone]{\oneone}{\tmpCos}
```

```
    \mulScalar[onetwo]{\xy}{\tmpCosMOne}
    \subScalar[onetwo]{\onetwo}{\zsin}
    \mulScalar[onethree]{\xz}{\tmpCosMOne}
    \addScalar[onethree]{\onethree}{\ysin}
    %
    \mulScalar[twoone]{\xsquared}{\tmpCosMOne}
    \addScalar[twoone]{\twoone}{\zsin}
    \mulScalar[twotwo]{\xy}{\tmpCosMOne}
    \addScalar[twotwo]{\twotwo}{\tmpCos}
    \mulScalar[twothree]{\xz}{\tmpCosMOne}
    \subScalar[twothree]{\twothree}{\xsin}
    %
    \mulScalar[threeone]{\xsquared}{\tmpCosMOne}
    \subScalar[threeone]{\threeone}{\ysin}
    \mulScalar[threetwo]{\xy}{\tmpCosMOne}
    \addScalar[threetwo]{\threetwo}{\xsin}
    \mulScalar[threethree]{\xz}{\tmpCosMOne}
    \addScalar[threethree]{\threethree}{\tmpCos}
    %
    \newVec{rowA}{\oneone, \onetwo, \onethree}
    \newVec{rowB}{\twoone, \twotwo, \twothree}
    \newVec{rowC}{\threeone, \threetwo, \threethree}
    \newMatFromRowVecs{#1}{\rowA, \rowB, \rowC}
}
```

`\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

```
\newcommand\newLookAtMat[5][lookAtMatResult]{%
    \dotVec[scalarRC]{#2}{#5}
    \mulScalar[scalarRC]{\scalarRC}{-1}
    \copyVec[tmpVecA]{#2}
    \appendToVec{tmpVecA}{\tmpVecA}{\scalarRC}
    % \printVec[1]{\tmpVecA}
    %
    \dotVec[scalarUC]{#3}{#5}
    \mulScalar[scalarUC]{\scalarUC}{-1}
    \copyVec[tmpVecB]{#3}
    \appendToVec{tmpVecB}{\tmpVecB}{\scalarUC}
    %
    \dotVec[scalarDC]{#4}{#5}
    \copyVec[tmpVecC]{#4}
    \negateVec[tmpVecC]{\tmpVecC}
    \appendToVec{tmpVecC}{\tmpVecC}{\scalarDC}
    %
    \newVec{tmpVecD}{0, 0, 0, 1}
    \newMatFromRowVecs{#1}{\tmpVecA, \tmpVecB, \tmpVecC, \tmpVecD}
}
```

`\newProjMat` Create a new default projection matrix.
#1 - out: var name to save the result (matrix) to

```
\newcommand\newProjMat[1]{%
    \newVec{tmpVecA}{1, 0, 0, 0}
    \newVec{tmpVecB}{0, 1, 0, 0}
    \newVec{tmpVecC}{0, 0, 1, 0}
    \newVec{tmpVecD}{0, 0, -1, 0}
    \newMatFromRowVecs{#1}{\tmpVecA, \tmpVecB, \tmpVecC, \tmpVecD}
}
```

`\newProjMatGen` Create a new projection matrix from normal and distance from origin.
#1 - out: var name to save the result (matrix) to

```
\newcommand\newProjMatGen[3]{%
    \divScalar[glmNxDelta]{\xintNthElt{1}{#2}}{#3}
    \divScalar[glmNyDelta]{\xintNthElt{2}{#2}}{#3}
    \divScalar[glmNzDelta]{\xintNthElt{3}{#2}}{#3}
    \newVec{tmpVecA}{1, 0, 0, 0}
    \newVec{tmpVecB}{0, 1, 0, 0}
    \newVec{tmpVecC}{0, 0, 1, 0}
    \newVec{tmpVecD}{\glmNxDelta, \glmNyDelta, \glmNzDelta, 0}
    \newMatFromRowVecs{#1}{\tmpVecA, \tmpVecB, \tmpVecC, \tmpVecD}
}
```

`\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 psition of the far lane (scalar)

```
\newcommand\newFrustumMat[5][frustMatResult]{%
    \FPdiv\tmp{#2}{2}
    \FPtan\frustumRight\tmp
    \FPmul\frustumRight{#4}\frustumRight
    \FPmul\frustumLeft\frustumRight{-1}
    \FPdiv\tmp{#3}{2}
    \FPtan\frustumTop\tmp
    \FPmul\frustumTop{#4}\frustumTop
    \FPmul\frustumBottom\frustumTop{-1}
    %
    \FPmul\cellAA{2}{#4} % AA
    \FPsub\tmp\frustumRight\frustumLeft
    \FPdiv\cellAA\cellAA\tmp
    \FPadd\cellAC\frustumRight\frustumLeft % AC
    \FPdiv\cellAC\cellAC\tmp
    \FPmul\cellBB{2}{#4} % BB
    \FPsub\tmp\frustumTop\frustumBottom
    \FPdiv\cellBB\cellBB\tmp
    \FPadd\cellBC\frustumTop\frustumBottom % BC
    \FPdiv\cellBC\cellBC\tmp
    \FPmul\cellCC{#5}{-1} % CC
    \FPsub\cellCC\cellCC{#4}
```

```
    \FPsub\tmp{#5}{#4}
    \FPdiv\cellCC\cellCC\tmp
    \FPmul\cellCD{#5}{#4} % CD
    \FPmul\cellCD{-2}\cellCD
    \FPdiv\cellCD\cellCD\tmp
    %
    \newVec{rowA}{\cellAA, 0, \cellAC, 0}
    \newVec{rowB}{0, \cellBB, \cellBC, 0}
    \newVec{rowC}{0, 0, \cellCC, \cellCD}
    \newVec{rowD}{0, 0, -1, 0}
    \newMatFromRowVecs{#1}{\rowA, \rowB, \rowC, \rowD}
}
```

`\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

```
\newcommand\printMat[2][0]{%
    \ensuremath{%
        \begin{bmatrix}
            \xintFor* ##1 in #2 \do {%
                \xintFor* ##2 in {##1} \do {%
                    \xintifForLast{%
                        \xintifForFirst{}{&}
                        \FPifeq{#1}{0}
                            \roundScalar{##2}
                            \scalarRoundResult
                        \else
                            \numToFractionA{##2}
                        \fi
                        \\
                    }{%
                        \xintifForFirst{}{&}
                        \FPifeq{#1}{0}
                            \roundScalar{##2}
                            \scalarRoundResult
                        \else
                            \numToFractionA{##2}
                        \fi
                    }
                }
            }
        \end{bmatrix}
    }
}
```

`\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

```
\newcommand\printMatDiag[2][0]{%
    \ensuremath{%
        \text{diag}\left(
        \xintNthElt{1}{\xintNthElt{1}{#2}},
```

```
        \xintNthElt{2}{\xintNthElt{2}{#2}},
        \xintNthElt{3}{\xintNthElt{3}{#2}}
        \right)
    }
}
```

`\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

```
\newcommand\getMatHeight[2][matHeightResult]{%
    \newScalar{#1}{\expandafter\xintLength\expandafter{#2}}%
}
```

`\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

```
\newcommand\getMatWidth[2][matWidthResult]{%
    \edef\rowElements{\xintNthElt{1}{#2}}%
    \newScalar{#1}{\expandafter\xintLength\expandafter{\rowElements}}%
}
```

`\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

```
\newcommand\transposeMat[2][transposeMatResult]{%
    \getMatWidth{#2}
    \getMatHeight{#2}
    \edef\widthSequence{\xintSeq[+1]{+1}{\matWidthResult}}
    \edef\heightSequence{\xintSeq[+1]{+1}{\matHeightResult}}
    \newEmptyMat{#1}
    \xintFor* ##1 in \widthSequence \do {%
        \xintFor* ##2 in \heightSequence \do {%
            \FPset\tmpValue{\xintNthElt{##1}{\xintNthElt{##2}{#2}}}
            \xintifForFirst{%
                \newVec{rowVec}{\tmpValue}
            }{%
                \appendToVec{rowVec}{\rowVec}{\tmpValue}
            }
        }
        \appendRowToMat{#1}{\csname #1\endcsname}{\rowVec}
    }
}
```

`\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 matrizes to add

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

```
    \if\noexpand#1\relax
        \PackageError{glmatrix package}{argument mismatch}{Argument 1 of addMat seems to be
    \else
    \fi
    \if\noexpand#2\relax
    \else
        \PackageError{glmatrix package}{argument mismatch}{argument 2 of addMat does not see
    \fi
    % TODO this does not work if it is a list
    % \if\noexpand#3\relax
    % \else
    %     \PackageError{glmatrix package}{argument mismatch}{argument 3 of addMat does not s
    % \fi
    \def\matlist{\xintCSVtoList{#3}}
    \edef\elemsRow{\xintNthElt{1}{\xintNthElt{1}{\matlist}}}
    \edef\widthsequence{\xintSeq[+1]{+1}{\expandafter\xintLength\expandafter{\elemsRow}}}%
    \edef\elemsMat{\xintNthElt{1}{\matlist}}
    \edef\heightsequence{\xintSeq[+1]{+1}{\expandafter\xintLength\expandafter{\elemsMat}}}%
    %
    \newEmptyMat{#1}
    \xintFor* ##2 in \heightsequence \do {%
        \xintFor* ##3 in \widthsequence \do {%
            \FPset\tmpValue{\xintNthElt{##3}{\xintNthElt{##2}{#2}}}
            \xintifForFirst{%
                \xintFor* ##1 in \matlist \do {%
                    \FPadd\tmpValue\tmpValue{\xintNthElt{##3}{\xintNthElt{##2}{##1}}}
                }
                \newVec{rowVec}{\tmpValue}
            }{%
                % \xintifForLast{
                % }{
                    \xintFor* ##1 in \matlist \do {%
                        \FPadd\tmpValue\tmpValue{\xintNthElt{##3}{\xintNthElt{##2}{##1}}}
                    }
                    \appendToVec{rowVec}{\rowVec}{\tmpValue}
                % }
            }
        }
        \appendRowToMat{#1}{\csname #1\endcsname}{\rowVec}
    }
}
```

`\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 matrizes to subtract

```
\newcommand\subMat[3][subMatResult]{%
    % \getMatWidth{#2}
    % \edef\widthsequence{\xintSeq[+1]{+1}{\matWidthResult}}%
    \getMatHeight{#2}
    \edef\heightsequence{\xintSeq[+1]{+1}{\matHeightResult}}%
    %
    \def\matlist{\xintCSVtoList{#3}}
```

```
    \edef\elemsRow{\xintNthElt{1}{\xintNthElt{1}{\matlist}}}
    \edef\widthsequence{\xintSeq[+1]{+1}{\expandafter\xintLength\expandafter{\elemsRow}}}%
    % \edef\elemsMat{\xintNthElt{1}{\matlist}}
    % \edef\heightsequence{\xintSeq[+1]{+1}{\expandafter\xintLength\expandafter{\elemsMat}}}
    %
    \newEmptyMat{#1}
    \xintFor* ##2 in \heightsequence \do {%
        \xintFor* ##3 in \widthsequence \do {%
            \FPset\tmpValue{\xintNthElt{##3}{\xintNthElt{##2}{#2}}}
            \xintifForFirst{%
                \xintFor* ##1 in \matlist \do {%
                    \FPsub\tmpValue\tmpValue{\xintNthElt{##3}{\xintNthElt{##2}{##1}}}
                }
                \newVec{rowVec}{\tmpValue}
            }{%
                \xintFor* ##1 in \matlist \do {%
                    \FPsub\tmpValue\tmpValue{\xintNthElt{##3}{\xintNthElt{##2}{##1}}}
                }
                \appendToVec{rowVec}{\rowVec}{\tmpValue}
            }
        }
        \appendRowToMat{#1}{\csname #1\endcsname}{\rowVec}
    }
}
```

`\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

```
\newcommand\mulMatScalar[3][mulMatScalarResult]{%
    \if\noexpand#1\relax
        \PackageError{glmatrix package}{argument mismatch}{Argument 1 of mulMatScalar seems
    \else
    \fi
    \if\noexpand#2\relax
    \else
        \PackageError{glmatrix package}{argument mismatch}{argument 2 of mulMatScalar does n
    \fi
    \edef\elemOne{\xintNthElt{1}{#2}}
    \edef\innersequence{\xintSeq[+1]{+1}{\expandafter\xintLength\expandafter{\elemOne}}}%
    %
    \newEmptyMat{#1}
    \xintFor* ##1 in #2 \do {%
        \xintFor* ##2 in \innersequence \do {%
            \FPset\elem{\xintNthElt{##2}{##1}}
            \FPmul\elem\elem{#3}
            \xintifForFirst{%
                \newVec{rowVec}{\elem}
            }{%
                \appendToVec{rowVec}{\rowVec}{\elem}
            }
        }
        \appendRowToMat{#1}{\csname #1\endcsname}{\rowVec}
```

```
    }
}
```

`\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 vetor with
#3 - vector: the vector to transform

```
\newcommand\mulMatVec[3][mulMatVecResult]{%
    \if\noexpand#1\relax
        \PackageError{glmatrix package}{argument mismatch}{Argument 1 of transformVec seems
    \else
    \fi
    \if\noexpand#2\relax
    \else
        \PackageError{glmatrix package}{argument mismatch}{argument 2 of transformVec does n
    \fi
    \if\noexpand#3\relax
    \else
        \PackageError{glmatrix package}{argument mismatch}{argument 3 of transformVec does n
    \fi
    \getVecHeight{#3}%
    \getMatHeight{#2}%
    \getMatWidth{#2}%
    % Check that matrix and vector can be multiplied
    \FPifeq{\matWidthResult}{\vecHeightResult}
    \else
        \PackageError{glmatrix package}{Dimension mismatch for matrix and vector, could not
    \fi
    % Create iterator sequences from height of vector and matrix
    \edef\vecHeightSequence{\xintSeq[+1]{+1}{\vecHeightResult}}%
    \edef\matHeightSequence{\xintSeq[+1]{+1}{\matHeightResult}}%
    % Iterate all rows of the matrix
    \xintFor* ##2 in \matHeightSequence \do {%
        \FPset\cellVal{0}
        % Differentiate between first and other entries to create the new vector variable
        \xintifForFirst{%
            % Now iterate each entry of the current row
            % Use this indec to access the entry in the matrix and in the vector
            \xintFor* ##3 in \vecHeightSequence \do {%
                \FPmul\tmp{%
                    \xintNthElt{##3}{\xintNthElt{##2}{#2}}%
                }{%
                    \xintNthElt{##3}{#3}%
                }
                % Add to storage variable
                \FPadd\cellVal\cellVal\tmp
            }
            \FPclip\cellVal\cellVal
            \newVec{#1}{\cellVal}
        }{
            \xintFor* ##3 in \vecHeightSequence \do {%
                \FPmul\tmp{%
                    \xintNthElt{##3}{\xintNthElt{##2}{#2}}%
```

```
                }{%
                    \xintNthElt{##3}{#3}%
                }
                \FPadd\cellVal\cellVal\tmp
            }
            \FPclip\cellVal\cellVal
            \appendToVec{#1}{\csname #1\endcsname}{\cellVal}
        }
    }
    % Test if width and height is correct
    \expandafter\getVecHeight\expandafter{\csname #1\endcsname}%
    \FPifeq{\matHeightResult}{\vecHeightResult}
    \else
        \PackageError{glmatrix package}{Vector height does not match expectation}{height of
    \fi
}
```

`\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

```
\newcommand\mulMat[3][mulMatResult]{%
    \edef\elemsRow{\xintNthElt{1}{#3}}
    \edef\widthsequence{\xintSeq[+1]{+1}{\expandafter\xintLength\expandafter{\elemsRow}}}%
    %
    % TODO remove one of these
    \edef\elemsMat{#2}
    \edef\heightsequence{\xintSeq[+1]{+1}{\expandafter\xintLength\expandafter{\elemsMat}}}%
    %
    \edef\elemsMat{#3}
    \edef\heightsequenceB{\xintSeq[+1]{+1}{\expandafter\xintLength\expandafter{\elemsMat}}}%
    %
    \edef\elemsRowTmp{\xintNthElt{1}{#2}}
    \FPset\widthMatA{\expandafter\xintLength\expandafter{\elemsRowTmp}}%
    \FPset\heightMatB{\expandafter\xintLength\expandafter{\elemsMat}}%
    % TODO check this
    \FPifeq{\widthMatA}{\heightMatB}
    \else
        \PackageError{glmatrix package}{Dimension mismatch for matrix, could not multiply}{w
    \fi
    %
    \newEmptyMat{#1}
    \xintFor* ##2 in \heightsequence \do {%
        \xintFor* ##3 in \widthsequence \do {%
            \FPset\tmpValue{0}
            \xintifForFirst{%
                \xintFor* ##4 in \heightsequenceB \do {%
                    \FPmul\tmptmp{%
                        \xintNthElt{##4}{\xintNthElt{##2}{#2}}%
                    }{%
                        \xintNthElt{##3}{\xintNthElt{##4}{#3}}%
                    }
                    \FPadd\tmpValue\tmpValue\tmptmp
```

```
                }
                \newVec{rowVec}{\tmpValue}
            }{%
                \xintFor* ##4 in \heightsequenceB \do {%
                    \FPmul\tmptmp{%
                        \xintNthElt{##4}{\xintNthElt{##2}{#2}}%
                    }{%
                        \xintNthElt{##3}{\xintNthElt{##4}{#3}}%
                    }
                    \FPadd\tmpValue\tmpValue\tmptmp
                }
                \appendToVec{rowVec}{\rowVec}{\tmpValue}
            }
        }
        \appendRowToMat{#1}{\csname #1\endcsname}{\rowVec}
    }
    % Test if width and height is correct
    \edef\expectedheight{\expandafter\xintLength\expandafter{#2}}%
    \edef\elemsRow{\xintNthElt{1}{#3}}
    \edef\expectedwidth{\expandafter\xintLength\expandafter{\elemsRow}}%
    %
    \edef\elemsRow{\xintNthElt{1}{\csname #1\endcsname}}
    \edef\actualwidth{\expandafter\xintLength\expandafter{\elemsRow}}%
    \edef\elemsMat{\csname #1\endcsname}
    \edef\actualheight{\expandafter\xintLength\expandafter{\elemsMat}}%
    %
    \FPifeq{\expectedwidth}{\actualwidth}
    \else
        \PackageError{glmatrix package}{Matrix width does not match expectation}{width seems
    \fi
    \FPifeq{\expectedheight}{\actualheight}
    \else
        \PackageError{glmatrix package}{Matrix height does not match expectation}{height see
    \fi
}
```

`\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

```
\newcommand\getMatTranslation[2][matTranslationResult]{%
    \getMatWidth{#2}
    \getMatHeight{#2}
    \edef\sequence{\xintSeq[+1]{+1}{\matHeightResult}}
    \xintFor* ##1 in \sequence \do {%
        \FPset\tmpValue{\xintNthElt{\matWidthResult}{\xintNthElt{##1}{#2}}}
        \xintifForFirst{%
            \newVec{#1}{\tmpValue}
        }{%
            \xintifForLast{}{
                \appendToVec{#1}{\csname #1\endcsname}{\tmpValue}
            }
        }
    }
```

```
}
```

`\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

```
\newcommand\detMat[2][detMatResult]{%
    \getMatWidth{#2}
    \getMatHeight{#2}
    \FPifgt{\matWidthResult}{2}
        \PackageError{glmatrix package}{Function not yet implemented}{Determinante can only
    \else
    \fi
    \FPifgt{\matHeightResult}{2}
        \PackageError{glmatrix package}{Function not yet implemented}{Determinante can only
    \else
    \fi
    %
    \mulScalar[tmpAD]{\xintNthElt{1}{\xintNthElt{1}{#2}}}{\xintNthElt{2}{\xintNthElt{2}{#2}}
    \mulScalar[tmpBC]{\xintNthElt{1}{\xintNthElt{2}{#2}}}{\xintNthElt{2}{\xintNthElt{1}{#2}}
    \subScalar[#1]{\tmpAD}{\tmpBC}
}
```

4.2 scalar

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

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

`\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

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

`\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

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

`\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

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

`\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

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

`\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

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

`\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

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

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

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

`\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

```
\newcommand\printScalar[2][0]{%
    \FPifeq{#1}{0}%
        \roundScalar{#2}%
        \scalarRoundResult%
    \else%
```

```
        \numToFractionA{#2}%
    \fi%
}
```

`\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

```
\newcommand\printScalarAsPi[2][0]{%
    \FPdiv\divNumRes{#2}{\FPpi}
    \FPifeq{#1}{0}
        \roundScalar{\divNumRes}
        \scalarRoundResult
    \else
        \numToFractionA{\divNumRes}
    \fi
    \pi
}
```

`\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 schould be printed

```
\newcommand\printPiScalarInDegree[2][0]{%
    \FPmul\tmp{#2}{180}
    \FPdiv\divNumRes{\tmp}{\FPpi}
    \FPifeq{#1}{0}
        \roundScalar{\divNumRes}
        \scalarRoundResult
    \else
        \numToFractionA{\divNumRes}
    \fi
    ^\circ
}
```

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

```
\newcommand\printScalarRounded[2][4]{%
    \FPround\tmp{#2}{#1}
    \FPclip\tmp\tmp
    \tmp
}
```

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

```
\newcommand\roundScalar[2][4]{%
    \FPround\tmp{#2}{#1}%
    \FPclip\tmp\tmp%
    \newScalar{scalarRoundResult}{\tmp}%
}
```

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

```
\newcommand\ceilScalar[2][scalarCeilResult]{
    \FPtrunc\tmp{#2}{0}
    \FPset\tmpSave{\tmp}
    %
    \FPifpos{#2}
        \FPadd\tmpSave{\tmp}{1}
    \else
    \fi
    \FPifint{#2}
        \FPset\tmpSave{\tmp}
    \else
    \fi
    \newScalar{#1}{\tmpSave}
}
```

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

```
\newcommand\floorScalar[2][scalarFloorResult]{
    \FPtrunc\tmp{#2}{0}
    \FPset\tmpSave{\tmp}
    %
    \FPifneg{#2}
        \FPsub\tmpSave{\tmp}{1}
    \else
    \fi
    \FPifint{#2}
        \FPset\tmpSave{\tmp}
    \else
    \fi
    \newScalar{#1}{\tmpSave}
}
```

`\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

```
\newcommand\clipScalar[2][clipScalarResult]{
    \FPclip\tmp{#2}
    \newScalar{#1}{\tmp}
}
```

`\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

```
\newcommand\radianToDegree[2][radianToDegreeResult]{
    \FPmul\tmp{#2}{180}
    \FPdiv\tmp{\tmp}{\FPpi}
    \newScalar{#1}{\tmp}
}
```

`\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

```
\newcommand\toRadian[2][degreeToRadianResult]{
    \FPdiv\tmp{#2}{180}
    \FPmul\tmp{\tmp}{\FPpi}
    \newScalar{#1}{\tmp}
}
```

`\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

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

`\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

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

`\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 multply

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

`\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

```
\newcommand\divScalar[3][scalarDivResult]{%
    \FPifeq{#3}{0}
        \PackageError{glmatrix package}{In divScalar function: division by zero}{parameter 3
    \else
    \fi
    \FPdiv\tmp{#2}{#3}
    \newScalar{#1}{\tmp}
}
```

`\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

```
\newcommand\powScalar[3][scalarPowResult]{%
    \FPpow\tmp{#2}{#3}
    \newScalar{#1}{\tmp}
}
```

`\negateScalar` Negate the value of the given scalar.
#1 - out: var name to save the result (scalar) to
#2 - scalar: the scalar to negate

```
\newcommand\negateScalar[2][scalarNegateResult]{%
    \FPneg\tmp{#2}
    \newScalar{#1}{\tmp}
}
```

`\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

```
\newcommand\arccosScalar[2][scalarArccosResult]{%
    \FPround\tmp{#2}{10}
    \FPclip\tmp\tmp
    \FParccos\tmp\tmp
    \newScalar{#1}{\tmp}
}
```

`\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 muliplier

```
\newcommand\mulAndAddScalar[4][scalarMulAndAddResult]{%
    \FPmul\tmp{#3}{#4}
    \FPadd\tmp{#2}{\tmp}
    \newScalar{#1}{\tmp}
}
```

`\lerpScalar` Lineraly 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]

```
\newcommand\lerpScalar[4][lerpScalarResult]{%
    \FPsub\tmpA{1}{#4}
    \FPmul\tmpA{#2}\tmpA
    \FPmul\tmpB{#3}{#4}
    \FPadd\tmpA\tmpA\tmpB
    \newScalar{#1}{\tmpA}
}
```

`\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

```
\newcommand\minScalar[3][minScalarResult]{%
    % TODO use \FPmin#1#2#3
    \FPiflt{#2}{#3}
        \expandafter\edef\csname #1\endcsname{#2}%
    \else
        \expandafter\edef\csname #1\endcsname{#3}%
    \fi
}
```

`\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

```
\newcommand\maxScalar[3][maxScalarResult]{%
    % TODO use \FPmax#1#2#3
    \FPiflt{#2}{#3}
        \expandafter\edef\csname #1\endcsname{#3}%
    \else
        \expandafter\edef\csname #1\endcsname{#2}%
    \fi
}
```

`\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

```
\newcommand\betweenScalar[4][betweenScalarResult]{%
    \FPset\betweenScalarResult{0}
    \FPiflt{#2}{#3}
    \else
        \FPadd\betweenScalarResult\betweenScalarResult{1}
    \fi
    \FPifgt{#2}{#4}
    \else
        \FPadd\betweenScalarResult\betweenScalarResult{1}
    \fi
    % TODO
    \FPifeq{\betweenScalarResult}{2}
        % \FPset\betweenScalarResult{1}
        \expandafter\edef\csname #1\endcsname{1}%
    \else
        % \FPset\betweenScalarResult{0}
        \expandafter\edef\csname #1\endcsname{0}%
    \fi
}
```

`\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

```
\newcommand\forEachScalar[3]{
    \FPiflt{#2}{10}
    \else
        \PackageError{glmatrix package}{In forEach scalar}{selection function that is not im
    \fi
    %
    \def\varList{\xintCSVtoList{#1}}
    \def\paramList{\xintCSVtoList{#3}}
    \xintFor* ##1 in \varList \do {%
        \FPifeq{#2}{1}
            \addScalar[##1]{\csname ##1\endcsname}{\xintNthElt{1}{\paramList}}
        \else
        \fi
        \FPifeq{#2}{2}
            \subScalar[##1]{\csname ##1\endcsname}{\xintNthElt{1}{\paramList}}
        \else
        \fi
        \FPifeq{#2}{3}
            \mulScalar[##1]{\csname ##1\endcsname}{\xintNthElt{1}{\paramList}}
        \else
        \fi
        \FPifeq{#2}{4}
            \divScalar[##1]{\csname ##1\endcsname}{\xintNthElt{1}{\paramList}}
        \else
        \fi
        \FPifeq{#2}{5}
            \powScalar[##1]{\csname ##1\endcsname}{\xintNthElt{1}{\paramList}}
        \else
        \fi
        \FPifeq{#2}{6}
            \negateScalar[##1]{\csname ##1\endcsname}
        \else
        \fi
        \FPifeq{#2}{7}
            \lerpScalar[##1]{\csname ##1\endcsname}{\xintNthElt{1}{\paramList}}{\xintNthElt{
        \else
        \fi
        \FPifeq{#2}{8}
            \mulAndAddScalar[##1]{\csname ##1\endcsname}{\xintNthElt{1}{\paramList}}{\xintNt
        \else
        \fi
        \FPifeq{#2}{9}
            \arccosScalar[##1]{\csname ##1\endcsname}
        \else
        \fi
    }
}
```

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

```
\newcommand\fractionToNum[3]{%
    \FProot\tmpA{#3}{2}
    \FPmul\tmpA\tmpA{#2}
    \FPdiv\tmpB{#1}\tmpA
    \FPset\fractionNumRes{\tmpB}
    \ensuremath{
        \tmpB
    }
}
```

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

```
\newcommand\solutionC{%
    \FPset\finalmul{1}
    \FPset\tmptmptmptmp\tmptmptmp
    %
    \edef\innersequence{\xintSeq[+1]{+1}{+10}}%
    \xintFor* ##1 in \innersequence \do {%
        \FPtrunc\floored\tmptmptmptmp{0}
        \FPsub\remainder\tmptmptmptmp\floored
        \FPifgt{\remainder}{0.00001}
            \FPdiv\mulmul{1}{\remainder}
            \FPmul\finalmul\finalmul\mulmul
        \else
            \xintBreakFor
        \fi
        \FPset\tmptmptmptmp\mulmul
    }
    \FPround\finalmul\finalmul{8}
    \FPclip\finalmul\finalmul
    \FPset\finalmultmp\finalmul
    % The result will be a multiplicator that is able to round
    % the number to a whole number, but the result might not be a
    % nice number to root e.g. 8/sqrt(6), 0.09375 will result in 32
    % 0.09375 * 32 = 3, but sqrt(32) = 5,65.. but given this we know
    % that the searched for number must be a multiple of 32, in this case 64
    % Note: rounding to 6 & 36 does not work since 0.09375 * 36 = 3.375
    \xintFor* ##1 in \innersequence \do {%
        \FProot\finalmulsqrt\finalmultmp{2}
        \FPround\finalmulsqrt\finalmulsqrt{4}
        \FPclip\finalmulsqrt\finalmulsqrt
        \FPifint{\finalmulsqrt}
            \xintBreakFor
        \else
            \FPadd\finalmultmp\finalmultmp\finalmul
        \fi
    }
    \FPset\finalmul\finalmultmp
    \FProot\finalmulsqrt\finalmul{2}
    \FPround\finalmulsqrt\finalmulsqrt{4}
    \FPclip\finalmulsqrt\finalmulsqrt
    \FPmul\divisor\divisor\finalmul
    \FPmul\dividend\dividend\finalmulsqrt
}
```

\numToFraction

```
\newcommand\numToFraction[1]{%
    \FPset\dividend{1}
    \FPset\divisor{#1}
    \FPset\minusSign{1}
    \FPiflt{\divisor}{0}
        \FPset\minusSign{-1}
    \else
    \fi
    \ensuremath{
        % reverse fraction
        \FPdiv\divisor{1}{#1}
        % Reverse root
        % Use FPmul instead of FPpow to allow for neg values
        \FPmul\divisor\divisor\divisor
        \FPset\tmptmptmp\divisor
        \FPround\divisor\divisor{10}
        \FPclip\divisor\divisor
        \FPifint{\divisor}
        \else
            \solutionC
        \fi
        %
        \FPround\divisor\divisor{4}
        \FPclip\divisor\divisor
        \FPmul\dividend\dividend\minusSign
        \FPround\dividend\dividend{4}
        \FPclip\dividend\dividend
        % TODO this should not be necessary
        \FPset\outputVar{#1}
        % Create fraction from dividend and divisor
        \FPset\outputVar{\frac{\dividend}{\sqrt{\divisor}}}
        % If the sqrt can be resolved
        \FPset\tmpDivisor{\divisor}
        \FProot\tmpDivisor\tmpDivisor{2}
        \FPround\tmpDivisor\tmpDivisor{8}
        \FPclip\tmpDivisor\tmpDivisor
        \FPifint{\tmpDivisor}
            \FPset\outputVar{\frac{\dividend}{\tmpDivisor}}
        \else
        \fi
        %
        \FPset\inputVar{#1}
        \FPmul\inputVar\inputVar\inputVar
        \FPmul\inputVar\inputVar\minusSign
        \FPround\inputVar\inputVar{8}
        \FPclip\inputVar\inputVar
        \FPifint{\inputVar}
            \FPset\outputVar{\,\sqrt{\inputVar}}
        \else
        \fi
        % If #1 was an int all along
        \FPset\inputVarA{#1}
        \FPround\inputVarA\inputVarA{10}
```

```
        \FPclip\inputVarA\inputVarA
        \FPifint{\inputVarA}
            \FPset\outputVar{\inputVarA}
        \else
        \fi
        \outputVar
    }
}
```

\numToFractionA

```
\newcommand\numToFractionA[1]{%
    \FPset\inputVarA{#1}
    \FPround\inputVarA\inputVarA{10}
    \FPclip\inputVarA\inputVarA
    % Prevent division by 0
    \FPifint{\inputVarA}
        \FPset\outputVar{\inputVarA}
        \outputVar
    \else
        \numToFraction{#1}
    \fi
}
```

4.3 vector

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

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

\sqrDistVec	[⟨*out*⟩], {⟨*vector v*⟩}, {⟨*vector w*⟩} sqrDistVecResult
\transformVec	[⟨*out*⟩], {⟨*matrix*⟩}, {⟨*vector*⟩} transformVecResult
\tensorVec	[⟨*out*⟩], {⟨*vector*⟩} tensorVecResult
\minVec	[⟨*out*⟩], {⟨*vector v*⟩}, {⟨*vector w*⟩} minVecResult
\maxVec	[⟨*out*⟩], {⟨*vector v*⟩}, {⟨*vector w*⟩} maxVecResult
\solveVecEQL	[⟨*out*⟩], {⟨*vector v*⟩}, {⟨*vector w*⟩} 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

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

`\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

```
\newcommand\newVecPi[2]{%
    \def\tmpVec{\xintCSVtoList{#2}}
    \xintFor* ##1 in \tmpVec \do {%
        \newScalarPi{tmp}{##1}
        \xintifForFirst{%
            \newVec{#1}{\tmp}
        }{%
            \appendToVec{#1}{\csname #1\endcsname}{\tmp}
        }
    }
}
```

`\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

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

```
    \edef\innersequence{\xintSeq[+1]{+1}{#2}}%
    \xintFor* ##1 in \innersequence \do {%
        \FPrandom\tmp
        \lerpScalar{#3}{#4}{\tmp}
        \xintifForFirst{%
            \newVec{#1}{\lerpScalarResult}
        }{%
            \appendToVec{#1}{\csname #1\endcsname}{\lerpScalarResult}
        }
    }
}
```

`\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

```
\newcommand\copyVec[2][copyVecResult]{
    \xintFor* ##1 in #2 \do {%
        \xintifForFirst{%
            \newVec{#1}{##1}
        }{%
            \appendToVec{#1}{\csname #1\endcsname}{##1}
        }
    }
}
```

`\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

```
\newcommand\zeroVec[2][3]{%
    \edef\sequence{\xintSeq[+1]{+1}{#1}}%
    \xintFor* ##1 in \sequence \do {%
        \xintifForFirst{%
            \newVec{#2}{0}
        }{%
            \appendToVec{#2}{\csname #2\endcsname}{0}
        }
    }
}
```

`\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

```
\newcommand\appendToVec[3]{%
    \expandafter\edef\csname #1\endcsname{#2 \xintCSVtoList{#3}}%
}
```

`\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

```
\newcommand\printVec[2][0]{%
    \if\noexpand#2\relax
    \else
        \PackageError{glmatrix package}{argument mismatch}{Argument 1 of printVec seems to b
    \fi
    \ensuremath{%
        \begin{pmatrix}
            \xintFor* ##1 in {#2} \do {%
                \xintifForLast{
                    \FPifeq{#1}{0}
                        \roundScalar{##1}
                        \scalarRoundResult
                    \else
                        \numToFractionA{##1}
                    \fi
                }{%
                    \FPifeq{#1}{0}
                        \roundScalar{##1}
                        \scalarRoundResult
                    \else
                        \numToFractionA{##1}
                    \fi
                    \\
                }
            }
        \end{pmatrix}
    }
}
```

`\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

```
\newcommand\printVecT[2][0]{%
    \if\noexpand#2\relax
    \else
        \PackageError{glmatrix package}{argument mismatch}{Argument 1 of printVecT seems to
    \fi
    \ensuremath{\trans{\left(
        \xintFor* ##1 in {#2} \do {%
            \xintifForLast{
                \FPifeq{#1}{0}
                    \roundScalar{##1}
                    \scalarRoundResult
                \else
                    \numToFractionA{##1}
                \fi
            }{%
                \FPifeq{#1}{0}
                    \roundScalar{##1}
                    \scalarRoundResult
```

```
                \else
                    \numToFractionA{##1}
                \fi
                ,
            }
        }
    \right)}}%
}
```

`\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

```
\newcommand\printVecAsPoint[2][0]{%
    \ensuremath{%
        \begin{bmatrix}
            \xintFor* ##1 in {#2} \do {%
                \FPifeq{#1}{0}
                    \roundScalar{##1}
                    \scalarRoundResult
                \else
                    \numToFractionA{##1}
                \fi
                \xintifForLast{}{%
                    \\
                }
            }
        \end{bmatrix}
    }
}
```

`\printVecAsPi` Print the values of the given vector as a multiple of π 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

```
\newcommand\printVecAsPi[2][0]{%
    \ensuremath{%
        \begin{pmatrix}
            \xintFor* ##1 in #2 \do {%
                \printScalarAsPi[#1]{##1}
                % \FPifeq{#1}{0}
                %     \roundScalar{##1}
                %     \scalarRoundResult
                % \else
                %     \numToFractionA{##1}
                % \fi
                \xintifForLast{}{%
                    \\
                }
            }
        \end{pmatrix}
    }
}
```

\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

```
\newcommand\printVecAsPiT[2][0]{%
    \ensuremath{\trans{\left(
        \xintFor* ##1 in #2 \do {%
            \printScalarAsPi[#1]{##1}
            \xintifForLast{}{%
                ,
            }
        }
    \right)}}
}
```

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

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

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

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

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

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

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

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

\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

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

\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

```
\newcommand\printScalarPComp[3][0] {%
    \FPifeq{\expandafter\xintLength\expandafter{#2}}{\expandafter\xintLength\expandafter{#3}
    \else
        \PackageError{glmatrix package}{In scalar product}{vector A and B do not have same l
    \fi
    \edef\innersequence{\xintSeq[+1]{+1}{\expandafter\xintLength\expandafter{#2}}}%
    \ensuremath{
        \xintFor* ##1 in \innersequence \do {%
            \FPifeq{#1}{0}
                \printScalarRounded{\xintNthElt{##1}{#2}}*\printScalarRounded{\xintNthElt{##
            \else
                \numToFractionA{\xintNthElt{##1}{#2}}*\numToFractionA{\xintNthElt{##1}{#3}}
            \fi
            \xintifForLast{}{%
                +
            }
        }
    }
}
```

\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

```
\newcommand\printCrossPComp[3][0] {%
    \FPifeq{\expandafter\xintLength\expandafter{#2}}{3}
    \else
        \PackageError{glmatrix package}{In cross product}{vector A is not of length 3}
    \fi
    \FPifeq{\expandafter\xintLength\expandafter{#3}}{3}
    \else
        \PackageError{glmatrix package}{In cross product}{vector A is not of length 3}
    \fi
    \FPset\aa{\xintNthElt{1}{#2}}
    \FPset\ab{\xintNthElt{2}{#2}}
    \FPset\ac{\xintNthElt{3}{#2}}
    \FPset\ba{\xintNthElt{1}{#3}}
    \FPset\bb{\xintNthElt{2}{#3}}
    \FPset\bc{\xintNthElt{3}{#3}}
    %
    \ensuremath{
        \FPifeq{#1}{0}
            \begin{pmatrix}
                \printScalarRounded{\ab}*\printScalarRounded{\bc} - \printScalarRounded{\ac}
                \printScalarRounded{\ac}*\printScalarRounded{\ba} - \printScalarRounded{\aa}
                \printScalarRounded{\aa}*\printScalarRounded{\bb} - \printScalarRounded{\ab}
            \end{pmatrix}
        \else
```

```
            \begin{pmatrix}
                \numToFractionA{\ab}*\numToFractionA{\bc} - \numToFractionA{\ac}*\numToFract
                \numToFractionA{\ac}*\numToFractionA{\ba} - \numToFractionA{\aa}*\numToFract
                \numToFractionA{\aa}*\numToFractionA{\bb} - \numToFractionA{\ab}*\numToFract
            \end{pmatrix}
        \fi
    }
}
```

`\roundVec` Round the values ot the given vector.
#1 - out: var name to save the result (vector) to
#2 - vector: the vector that schould be rounded
#3 - digits: number of digits after comma

```
\newcommand\roundVec[3][roundVecResult]{%
    \xintFor* ##3 in #2 \do {%
        \FPround\tmp{##3}{#3}
        \xintifForFirst{%
            \newVec{#1}{\tmp}
        }{%
            \appendToVec{#1}{\csname #1\endcsname}{\tmp}
        }
    }
}
```

`\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

```
\newcommand\ceilVec[2][ceilVecResult]{
    \xintFor* ##1 in #2 \do {%
        \ceilScalar[tmp]{##1}
        \xintifForFirst{%
            \newVec{#1}{\tmp}
        }{%
            \appendToVec{#1}{\csname #1\endcsname}{\tmp}
        }
    }
}
```

`\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

```
\newcommand\floorVec[2][floorVecResult]{
    \xintFor* ##1 in #2 \do {%
        \floorScalar[tmp]{##1}
        \xintifForFirst{%
            \newVec{#1}{\tmp}
        }{%
            \appendToVec{#1}{\csname #1\endcsname}{\tmp}
        }
    }
}
```

`\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

```
\newcommand\dehomogenVec[2][dehomogenVecResult]{
    \getVecHeight{#2}
    \FPset\tmpDiv{\xintNthElt{\vecHeightResult}{#2}}
    \edef\sequence{\xintSeq[+1]{+1}{\vecHeightResult}}%
    \xintFor* ##1 in \sequence \do {%
        \xintifForFirst{%
            \FPdiv\divResult{\xintNthElt{##1}{#2}}{\tmpDiv}
            \newVec{tmpVec}{\divResult}
        }{%
            % \xintifForLast{
            % }{
                \FPdiv\divResult{\xintNthElt{##1}{#2}}{\tmpDiv}
                \appendToVec{tmpVec}{\tmpVec}{\divResult}
            % }
        }
    }
    \copyVec[#1]{\tmpVec}
}
```

`\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

```
\newcommand\addVec[3][addVecResult]{%
    \def\vecList{\xintCSVtoList{#3}}
    \getVecHeight{#2}
    \edef\sequence{\xintSeq[+1]{+1}{\vecHeightResult}}%
    % Iterate each row / dimension
    \xintFor* ##1 in \sequence \do {%
        % Assign value of start vector (param #2)
        \FPset\sumValue{\xintNthElt{##1}{#2}}%
        % If this is the first iteration create a new vector
        \xintifForFirst{%
            % Iterate all vectors that are part of param #3
            \xintFor* ##2 in \vecList \do {%
                % Multiply with row value of current vector
                \FPadd\sumValue\sumValue{\xintNthElt{##1}{##2}}%
            }
            \newVec{#1}{\sumValue}
        }{%
            \xintFor* ##2 in \vecList \do {%
                \FPadd\sumValue\sumValue{\xintNthElt{##1}{##2}}%
            }
            \appendToVec{#1}{\csname #1\endcsname}{\sumValue}
        }
    }
}
```

`\subVec` Subract 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

```
\newcommand\subVec[3][subVecResult]{%
    \if\noexpand#1\relax
        \PackageError{glmatrix package}{argument mismatch}{Argument 1 of subVec seems to be
    \else
    \fi
    \if\noexpand#2\relax
    \else
        \PackageError{glmatrix package}{argument mismatch}{argument 2 of subVec does not see
    \fi
    % TODO does not work on list of commands
    % \if\noexpand#3\relax
    % \else
    %     \PackageError{glmatrix package}{argument mismatch}{argument 3 of subVec does not s
    % \fi
    %
    \def\vecList{\xintCSVtoList{#3}}
    \getVecHeight{#2}
    \edef\sequence{\xintSeq[+1]{+1}{\vecHeightResult}}%
    % Iterate each row / dimension
    \xintFor* ##1 in \sequence \do {%
        % Assign value of start vector (param #2)
        \FPset\subValue{\xintNthElt{##1}{#2}}%
        % If this is the first iteration create a new vector
        \xintifForFirst{%
            % Iterate all vectors that are part of param #3
            \xintFor* ##2 in \vecList \do {%
                % Multiply with row value of current vector
                \FPsub\subValue\subValue{\xintNthElt{##1}{##2}}%
            }
            \newVec{#1}{\subValue}
        }{%
            \xintFor* ##2 in \vecList \do {%
                \FPsub\subValue\subValue{\xintNthElt{##1}{##2}}%
            }
            \appendToVec{#1}{\csname #1\endcsname}{\subValue}
        }
    }
}
```

`\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

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

```
    % Iterate each row / dimension
    \xintFor* ##1 in \sequence \do {%
        % Assign value of start vector (param #2)
        \FPset\mulValue{\xintNthElt{##1}{#2}}%
        % If this is the first iteration create a new vector
        \xintifForFirst{%
            % Iterate all vectors that are part of param #3
            \xintFor* ##2 in \vecList \do {%
                % Multiply with row value of current vector
                \FPmul\mulValue\mulValue{\xintNthElt{##1}{##2}}%
            }
            \newVec{#1}{\mulValue}
        }{%
            \xintFor* ##2 in \vecList \do {%
                \FPmul\mulValue\mulValue{\xintNthElt{##1}{##2}}%
            }
            \appendToVec{#1}{\csname #1\endcsname}{\mulValue}
        }
    }
}
```

`\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

```
\newcommand\divVec[3][divVecResult]{
    \def\vecList{\xintCSVtoList{#3}}
    \getVecHeight{#2}
    \edef\sequence{\xintSeq[+1]{+1}{\vecHeightResult}}%
    % Iterate each row / dimension
    \xintFor* ##1 in \sequence \do {%
        % Assign value of start vector (param #2)
        \FPset\divValue{\xintNthElt{##1}{#2}}%
        % If this is the first iteration create a new vector
        \xintifForFirst{%
            % Iterate all vectors that are part of param #3
            \xintFor* ##2 in \vecList \do {%
                \FPifeq{\xintNthElt{##1}{##2}}{0}
                    \PackageError{glmatrix package}{In divVec function: division by zero}{yo
                \else
                \fi
                % Divide with row value of current vector
                \FPdiv\divValue\divValue{\xintNthElt{##1}{##2}}%
            }
            \newVec{#1}{\divValue}
        }{%
            \xintFor* ##2 in \vecList \do {%
                \FPifeq{\xintNthElt{##1}{##2}}{0}
                    \PackageError{glmatrix package}{In divVec function: division by zero}{yo
                \else
                \fi
                \FPdiv\divValue\divValue{\xintNthElt{##1}{##2}}%
```

```
            }
            \appendToVec{#1}{\csname #1\endcsname}{\divValue}
        }
    }
}
```

`\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

```
\newcommand\scaleVec[3][scaleVecResult] {%
    \xintFor* ##1 in #2 \do {%
        \FPmul\tmpVal{##1}{#3}
        \xintifForFirst{%
            \newVec{#1}{\tmpVal}
        }{%
            \appendToVec{#1}{\csname #1\endcsname}{\tmpVal}
        }
    }
}
```

`\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

```
\newcommand\divVecScalar[3][divVecScalarResult] {%
    \if\noexpand#1\relax
        \PackageError{glmatrix package}{argument mismatch}{Argument 1 of divVecScalar seems
    \else
    \fi
    \if\noexpand#2\relax
    \else
        \PackageError{glmatrix package}{argument mismatch}{argument 2 of divVecScalar does n
    \fi
    \FPifeq{#3}{0}
        \PackageError{glmatrix package}{In div vector by scalar}{divisor is 0}
    \else
    \fi
    \xintFor* ##1 in #2 \do {%
        \FPdiv\divNumRes{##1}{#3}
        \xintifForFirst{%
            \newVec{#1}{\divNumRes}
        }{%
            \appendToVec{#1}{\csname #1\endcsname}{\divNumRes}
        }
    }
}
```

`\negateVec` Negate the values of the given vector.
#1 - out: var name to save the result (vector) to
#2 - vector: the vector to negate

```
\newcommand\negateVec[2][negateVecResult]{
    \xintFor* ##1 in #2 \do {%
        \FPneg\tmp{##1}
        \xintifForFirst{%
            \newVec{#1}{\tmp}
        }{%
            \appendToVec{#1}{\csname #1\endcsname}{\tmp}
        }
    }
}
```

`\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 muliplier

```
\newcommand\scaleAndAddVec[4][scaleAndAddVecResult]{
    \scaleVec{#3}{#4}
    \addVec[#1]{#2}{\scaleVecResult}
}
```

`\lerpVec` Lineraly 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]

```
\newcommand\lerpVec[4][lerpVecResult]{
    % \def\vecList{\xintCSVtoList{#3}}
    \getVecHeight{#2}
    \edef\sequence{\xintSeq[+1]{+1}{\vecHeightResult}}%
    % Iterate each row / dimension
    \xintFor* ##1 in \sequence \do {%
        % Assign value of start vector (param #2)
        % \FPset\divValue{\xintNthElt{##1}{#2}}%
        % If this is the first iteration create a new vector
        \xintifForFirst{%
            \lerpScalar{\xintNthElt{##1}{#2}}{\xintNthElt{##1}{#3}}{#4}
            \newVec{#1}{\lerpScalarResult}
        }{%
            \lerpScalar{\xintNthElt{##1}{#2}}{\xintNthElt{##1}{#3}}{#4}
            \appendToVec{#1}{\csname #1\endcsname}{\lerpScalarResult}
        }
    }
}
```

`\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]

```
\newcommand\slerpVec[4][slerpVecResult] {%
    \angleVec{#2}{#3}
    \FPifeq{\angleVecResult}{0}
        \lerpVec[#1]{#2}{#3}{#4}
    \else
        \FPsin\angleTotal{\angleVecResult}
        \FPsub\ratioA{1}{#4}
        \FPmul\ratioA\ratioA{\angleVecResult}
        \FPsin\ratioA\ratioA
        \FPdiv\ratioA\ratioA\angleTotal
        \FPmul\ratioB{#4}{\angleVecResult}
        \FPsin\ratioB\ratioB
        \FPdiv\ratioB\ratioB\angleTotal
        \scaleVec[tmpA]{#2}{\ratioA}
        \scaleVec[tmpB]{#3}{\ratioB}
        \addVec[#1]{\tmpA}{\tmpB}
    \fi
}
```

`\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

```
\newcommand\dotVec[3][dotVecResult] {%
    \if\noexpand#1\relax
        \PackageError{glmatrix package}{argument mismatch}{Argument 1 of dotVec seems to be
    \else
    \fi
    \if\noexpand#2\relax
    \else
        \PackageError{glmatrix package}{argument mismatch}{argument 2 of dotVec does not see
    \fi
    \if\noexpand#3\relax
    \else
        \PackageError{glmatrix package}{argument mismatch}{argument 3 of dotVec does not see
    \fi
    \FPifeq{\expandafter\xintLength\expandafter{#2}}{\expandafter\xintLength\expandafter{#3}
    \else
        \PackageError{glmatrix package}{In dot product}{vector A and B do not have same leng
    \fi
    \FPset\res{0}
    \FPset\tmp{0}
    \edef\innersequence{\xintSeq[+1]{+1}{\expandafter\xintLength\expandafter{#2}}}%
    \xintFor* ##1 in \innersequence \do {%
        \FPset\valueA{\xintNthElt{##1}{#2}}
        \FPset\valueB{\xintNthElt{##1}{#3}}
        \FPmul\tmp\valueA\valueB
        \FPadd\res\res\tmp
    }
    \newScalar{#1}{\res}
```

```
}
```

`\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

```
\newcommand\lenVec[2][lenVecResult] {%
    \if\noexpand#1\relax
        \PackageError{glmatrix package}{argument mismatch}{Argument 1 of lenVec seems to be
    \else
    \fi
    \if\noexpand#2\relax
    \else
        \PackageError{glmatrix package}{argument mismatch}{argument 2 of lenVec does not see
    \fi
    \FPset\normTotal{0}
    \FPset\iterVal{0}
    % Iterate each value, multiply it with itself and sum up
    \xintFor* ##1 in #2 \do {%
        % No idea why this needs to be abs,
        % but negative values are a problem here
        \FPabs\iterVal{##1}
        \FPpow\tmpNormB\iterVal{2}
        \FPadd\normTotal\normTotal\tmpNormB
    }
    % Evaluate root for numbered result
    \FProot\normRooted\normTotal{2}
    % Set output var
    \newScalar{#1}{\normRooted}
}
```

`\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

```
\newcommand\sqrLenVec[2][sqrLenVecResult] {%
    \FPset\normTotal{0}
    \FPset\iterVal{0}
    % Iterate each value, multiply it with itself and sum up
    \xintFor* ##1 in #2 \do {%
        % No idea why this needs to be abs,
        % but negative values are a problem here
        \FPabs\iterVal{##1}
        \FPpow\tmpNormB\iterVal{2}
        \FPadd\normTotal\normTotal\tmpNormB
    }
    % Set output var
    \newScalar{#1}{\normTotal}
}
```

`\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

```
\newcommand\normalizeVec[2][normalizeVecResult] {%
    \if\noexpand#1\relax
        \PackageError{glmatrix package}{argument mismatch}{Argument 1 of normalizeVec seems
    \else
    \fi
    \if\noexpand#2\relax
    \else
        \PackageError{glmatrix package}{argument mismatch}{argument 2 of normalizeVec does n
    \fi
    \lenVec{#2}
    \divVecScalar[#1]{#2}{\lenVecResult}
}
```

`\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

```
\newcommand\crossVec[3][crossVecResult] {%
    \FPifeq{\expandafter\xintLength\expandafter{#2}}{3}
    \else
        \PackageError{glmatrix package}{In cross product}{vector A is not of length 3}
    \fi
    \FPifeq{\expandafter\xintLength\expandafter{#3}}{3}
    \else
        \PackageError{glmatrix package}{In cross product}{vector A is not of length 3}
    \fi
    \FPset\aa{\xintNthElt{1}{#2}}
    \FPset\ab{\xintNthElt{2}{#2}}
    \FPset\ac{\xintNthElt{3}{#2}}
    \FPset\ba{\xintNthElt{1}{#3}}
    \FPset\bb{\xintNthElt{2}{#3}}
    \FPset\bc{\xintNthElt{3}{#3}}
    \FPset\tmp{0}
    % x component
    \FPset\scalarX{0}
    \FPmul\scalarX\ab\bc
    \FPmul\tmp\ac\bb
    \FPsub\scalarX\scalarX\tmp
    % y component
    \FPset\scalarY{0}
    \FPmul\scalarY\ac\ba
    \FPmul\tmp\aa\bc
    \FPsub\scalarY\scalarY\tmp
    % z component
    \FPset\scalarZ{0}
    \FPmul\scalarZ\aa\bb
    \FPmul\tmp\ab\ba
    \FPsub\scalarZ\scalarZ\tmp
    \newVec{#1}{\scalarX,\scalarY,\scalarZ}
}
```

`\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

```
\newcommand\crossVecTwoD[3][crossVecResult] {%
    \FPifeq{\expandafter\xintLength\expandafter{#2}}{2}
    \else
        \PackageError{glmatrix package}{In cross product}{vector A is not of length 2}
    \fi
    \FPifeq{\expandafter\xintLength\expandafter{#3}}{2}
    \else
        \PackageError{glmatrix package}{In cross product}{vector A is not of length 2}
    \fi
    \FPset\aa{\xintNthElt{1}{#2}}
    \FPset\ab{\xintNthElt{2}{#2}}
    \FPset\ba{\xintNthElt{1}{#3}}
    \FPset\bb{\xintNthElt{2}{#3}}
    \FPmul\scalarUxVy{\xintNthElt{1}{#2}}{\xintNthElt{2}{#3}}
    \FPmul\scalarUyVx{\xintNthElt{2}{#2}}{\xintNthElt{1}{#3}}
    \FPsub\scalarResult{\scalarUxVy}{\scalarUyVx}
    \newScalar{#1}{\scalarResult}
}
```

`\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

```
\newcommand\angleVec[3][angleVecResult] {%
    \dotVec[dotVW]{#2}{#3}
    \lenVec[normV]{#2}
    \lenVec[normW]{#3}
    \mulScalar[mulNormVW]{\normV}{\normW}
    \divScalar{\dotVW}{\mulNormVW}
    \arccosScalar[#1]{\scalarDivResult}
}
```

`\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

```
\newcommand\distVec[3][distVecResult] {%
    \sqrDistVec{#2}{#3}
    \FProot\res\sqrDistVecResult{2}
    \newScalar{#1}{\res}
}
```

`\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

```
\newcommand\sqrDistVec[3][sqrDistVecResult] {%
```

```
    \getVecHeight[getVecHeightA]{#2}
    \getVecHeight[getVecHeightB]{#3}
    \FPifeq{\getVecHeightA}{\getVecHeightB}
    \else
        \PackageError{glmatrix package}{In sqrt distance}{vector #2 and #3 do not have same
    \fi
    \FPset\res{0}
    \FPset\tmp{0}
    \edef\sequence{\xintSeq[+1]{+1}{\getVecHeightA}}%
    \xintFor* ##1 in \sequence \do {%
        \FPset\valueA{\xintNthElt{##1}{#2}}
        \FPset\valueB{\xintNthElt{##1}{#3}}
        \FPsub\tmp\valueA\valueB
        \FPmul\tmp\tmp\tmp
        \FPadd\res\res\tmp
    }
    \newScalar{#1}{\res}
}
```

`\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 vetor with
#3 - vector: the vector to transform

```
\newcommand\transformVec[3][transformVecResult] {%
    \if\noexpand#1\relax
        \PackageError{glmatrix package}{argument mismatch}{Argument 1 of transformVec seems
    \else
    \fi
    \if\noexpand#2\relax
    \else
        \PackageError{glmatrix package}{argument mismatch}{argument 2 of transformVec does n
    \fi
    \if\noexpand#3\relax
    \else
        \PackageError{glmatrix package}{argument mismatch}{argument 3 of transformVec does n
    \fi
    \getVecHeight{#3}%
    \getMatHeight{#2}%
    \getMatWidth{#2}%
    % Check that matrix and vector can be multiplied
    \FPifeq{\matWidthResult}{\vecHeightResult}
    \else
        \PackageError{glmatrix package}{Dimension mismatch for matrix and vector, could not
    \fi
    % Create iterator sequences from height of vector and matrix
    \edef\vecHeightSequence{\xintSeq[+1]{+1}{\vecHeightResult}}%
    \edef\matHeightSequence{\xintSeq[+1]{+1}{\matHeightResult}}%
    % Iterate all rows of the matrix
    \xintFor* ##2 in \matHeightSequence \do {%
        \FPset\cellVal{0}
        % Differentiate between first and other entries to create the new vector variable
        \xintifForFirst{%
            % Now iterate each entry of the current row
```

```
            % Use this indec to access the entry in the matrix and in the vector
            \xintFor* ##3 in \vecHeightSequence \do {%
                \FPmul\tmp{%
                    \xintNthElt{##3}{\xintNthElt{##2}{#2}}%
                }{%
                    \xintNthElt{##3}{#3}%
                }
                % Add to storage variable
                \FPadd\cellVal\cellVal\tmp
            }
            \FPclip\cellVal\cellVal
            \newVec{#1}{\cellVal}
        }{
            \xintFor* ##3 in \vecHeightSequence \do {%
                \FPmul\tmp{%
                    \xintNthElt{##3}{\xintNthElt{##2}{#2}}%
                }{%
                    \xintNthElt{##3}{#3}%
                }
                \FPadd\cellVal\cellVal\tmp
            }
            \FPclip\cellVal\cellVal
            \appendToVec{#1}{\csname #1\endcsname}{\cellVal}
        }
    }
    % Test if width and height is correct
    \expandafter\getVecHeight\expandafter{\csname #1\endcsname}%
    \FPifeq{\matHeightResult}{\vecHeightResult}
    \else
        \PackageError{glmatrix package}{Vector height does not match expectation}{height of
    \fi
}
```

`\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

```
\newcommand\tensorVec[2][tensorVecResult]{%
    \newMatFromRowVecs{rowMatrix}{#2}
    \newMatFromColVecs{colMatrix}{#2}
    \mulMat[#1]{\colMatrix}{\rowMatrix}
}
```

`\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

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

```
    \FPifeq{\expandafter\xintLength\expandafter{#2}}{\expandafter\xintLength\expandafter{#3}
    \else
        \PackageError{glmatrix package}{incompatible size}{vector A and B of minVec do not h
    \fi
    \edef\innersequence{\xintSeq[+1]{+1}{\expandafter\xintLength\expandafter{#2}}}%
    \xintFor* ##1 in \innersequence \do {%
        \minScalar{\xintNthElt{##1}{#2}}{\xintNthElt{##1}{#3}}
        \xintifForFirst{%
            \newVec{#1}{\minScalarResult}
        }{%
            \appendToVec{#1}{\csname #1\endcsname}{\minScalarResult}
        }
    }
}
```

`\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

```
\newcommand\maxVec[3][maxVecResult]{
    \if\noexpand#1\relax
        \PackageError{glmatrix package}{argument mismatch}{Argument 1 of maxVec seems to be
    \else
    \fi
    \FPifeq{\expandafter\xintLength\expandafter{#2}}{\expandafter\xintLength\expandafter{#3}
    \else
        \PackageError{glmatrix package}{incompatible size}{vector A and B of maxVec do not h
    \fi
    \edef\innersequence{\xintSeq[+1]{+1}{\expandafter\xintLength\expandafter{#2}}}%
    \xintFor* ##1 in \innersequence \do {%
        \maxScalar{\xintNthElt{##1}{#2}}{\xintNthElt{##1}{#3}}
        \xintifForFirst{%
            \newVec{#1}{\maxScalarResult}
        }{%
            \appendToVec{#1}{\csname #1\endcsname}{\maxScalarResult}
        }
    }
}
```

`\solveVecEQL` 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 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

```
\newcommand\solveVecEQL[3][solveVecEQLResult]{
    \getVecHeight{#3}%
    \edef\vecHeightSequence{\xintSeq[+1]{+1}{\vecHeightResult}}%
    \xintFor* ##1 in \vecHeightSequence \do {%
        % \FPneg\tmpVar{\xintNthElt{##1}{#3}}
        \FPlsolve\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