TexVar - Manual

TexVar – LaTeX math calculations

Version: 1.5.14

Sebastian Pech

December 4, 2015

Contents

1 Description

TexVar (short tVar) is a basic LaTeX math calculations tool written in Lua. For integration into LaTeX it has to be used together with LuaLaTeX. Compared to software like Mathcad TexVar is a lot more flexible. You can fill custom designed tables with results, do calculations within text documents and print beautiful LaTeX equations. The current version also supports 2D-plotting with gnuplot.

2 Installation

2.1 Prerequisites

The following software is needed in order to use TexVar:

- Lua 5.1 or higher
- LuaLaTeX (MikTex or Texlive)
- GnuPlot 5.0 (only needed for plotting)

2.2 Installation

Download TexVar from `http://texvar.projectzoo.at/` and copy the folder *tVar* and the file *texvar.sty* (subfolder package) to a location that is visible to LaTeX. (e.g. the folder your *.tex file is in or global LaTeX-folder [1])

2.2.1 Configuration for Gnuplot

In order to use Gnuplot with TexVar you have to allow LuaLaTeX to call external commands during runtime. This works through the command-line switch `--shell-escape`. Your complete call for LuaLaTeX could look like this: `lualatex -synctex=1 -interaction=nonstopmode --shell-escape %.tex`.

3 Getting Started - Hello World

To ensure your installation is working test the following code. When using an luacode based environment like tVar it's important that the commands `\begin{tVar} \end{tVar}` are *not indented.*

```
\documentclass{article}
%
\usepackage{texvar}
\usepackage[fleqn]{amsmath}
%
\begin{document}
\begin{tVar}
#Hello World! I'm using TexVar
tVar.getVersion()
\end{tVar}
\end{document}
```

Output

Hello World! I'm using TexVar Version: 1.5.14

[1]You can find information on global LaTeX-folders at`https://www.math.hmc.edu/computing/support/tex/installing/`

4 General Information on tVar Enviroment and Macro

To execute TexVar commands from LaTeX, the TexVar package offers two options:

- `\begin{tVar} \end{tVar}`, for commands blocks
- `\tv{}`, for inline calls

Both forward the commands to the TexVar interpreter.

```
\begin{tVar}
  a:=10
  b:=3
  c:=a+b
\end{tVar}
The result is: \tv{c:outRES()}
```

Output

$a = 10$

$b = 3$

$c = a + b = 10 + 3 = 13$

The result is: $c = 13$

Generally, there are two key commands the TexVar interpreter searches for [2]:

1. #, for text output
2. :=, for calculations and assignments

Every other command is directly forwarded to the Lua interpreter. This means you can write any Lua code you want inside the tVar environment.

```
\begin{tVar}
function mypow(a,b)
return a^b
end
for i=0,3 do
tex.print("Step " .. mypow(2,i) .. "\\\\")
end
\end{tVar}
```

Output

Step 1
Step 2
Step 4
Step 8

[2] Details on these key commands are discussed in Section 5

5 TexVar - Commands

5.1 Creating Variables

TexVar knows three types of variables:

tVar is the basic type for scalar values.

tMat is used for matrices.

tVec is used for vectors.

The following code generates a variable of each type. Every assignment is made with :=.

```
\begin{tVar}
a:=13
e:={1,2,4}
A:={{1,0},{2,4},{3,9.1}}
\end{tVar}
```

Output

$$a = 13$$

$$\vec{e} = \begin{pmatrix} 1 \\ 2 \\ 4 \end{pmatrix}$$

$$\mathbf{A} = \begin{pmatrix} 1 & 0 \\ 2 & 4 \\ 3 & 9.1 \end{pmatrix}$$

5.1.1 Creating Variables in Equations

New variables can also be created inside equations. The notation for vectors and matrices is explained in Section 5.1. Lua knows - by default - how to do calculations with numbers, so any equation which does not contain tVar objects, can only be printed as it's result. Any mathematical operation with a tVar object and a decimal number results in a tVar value.

```
\begin{tVar}
# Variable a is created from the equation ...
a:=13*2^2
# To preserver the equation some of the ...
a:=tVar:New(13)*2^2
# At first Lua calculates $2^2=4$ ...
# To print the whole ...
a:=13*tVar:New(2)^2
# Matrices and Vectors
c:={{1,2,3},{4,2,6},{1,3,2}}*{1,2,3}*a
\end{tVar}
```

Output

Variable a is created from the equation $13 \cdot 2^2$. Since all numbers are just decimal values, Lua calculates the result and creates the tVar object afterwards.

$$a = 52$$

To preserver the equation some of the numbers have to be initialized as tVar objects. In this case it's the number 13.

$$a = 13 \cdot 4 = 13 \cdot 4 = 52$$

At first Lua calculates $2^2 = 4$ and then multiplies the tVar object 13 with 4 which results in a tVar object containing the equation. To print the whole equation, one number from the first calculation has to be transformed into a tVar object.

$$a = 13 \cdot 2^2 = 13 \cdot 2^2 = 52$$

Matrices and Vectors

$$\vec{c} = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 2 & 6 \\ 1 & 3 & 2 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} \cdot a = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 2 & 6 \\ 1 & 3 & 2 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} \cdot 52 = \begin{pmatrix} 728 \\ 1352 \\ 676 \end{pmatrix}$$

5.1.2 Auto-Formatting of variable names

The LaTeX-representation of a variable is automatically generated from the variable name. The first occurrence of an underline starts the subscript the first occurrence of a double underline starts the superscript. Every other underline becomes a comma.

```
\begin{tVar}
a_1_3:=13
a__10_2:=3
b_1_x__y_2:=12
\end{tVar}
```

Output

$$a_{1,3} = 13$$
$$a_2^{10} = 3$$
$$b_{1,x}^{y,2} = 12$$

5.2 Creating Functions

Functions can easily be defined with the following syntax. The auto-formatting of function names and attribute names works according to Section 5.1.2.

```
\begin{tVar}
f(x,y):=x^2+y^2+4
a:=f(2,3)+11
\end{tVar}
```

Output

$$f(x,y) = x^2 + y^2 + 4$$

$$a = f(2,3) + 11 = 2^2 + 3^2 + 4 + 11 = 28$$

5.3 Indexing Matrices and Vectors

The syntax for indexing a matrix is similar to the Matlab syntax. To access a matrix you have to use square brackets and a string as key. The key has to be formatted according to the following examples.

```
\begin{tVar}
A:={{1,2,6},{2,4,6},{7,6,9}}
# Index one element with syntax [row,column]
A["1,2"]:outRES()
# Index a range
A["1:2,1:end"]:outRES()
# The range 1:end is equal to :
A["1:2,:"]:outRES()
# You can also set matrices this way
A["1:2,:"]:={{1,2,3},{4,5,6}}
A["1:2,:"]:outRES()
# Adress vector
c:=A[":,2"]
d:=A["2,:"]
# The above also applies to vectors
{plain
  v_1:={1,4,3}
  #,~
  v_1["2"]:outRES()
  #,~
  v_1["1"]:=9
  v_1:outRES()
}
\end{tVar}
```

Output

$$\mathbf{A} = \begin{pmatrix} 1 & 2 & 6 \\ 2 & 4 & 6 \\ 7 & 6 & 9 \end{pmatrix}$$

Index one element with syntax [row,column]

$$\mathbf{A}[1,2] = 2$$

Index a range

$$\mathbf{A}[1:2,1:\text{end}] = \begin{pmatrix} 1 & 2 & 6 \\ 2 & 4 & 6 \end{pmatrix}$$

The range 1:end is equal to :

$$\mathbf{A}[1{:}2,:] = \begin{pmatrix} 1 & 2 & 6 \\ 2 & 4 & 6 \end{pmatrix}$$

You can also set matrices this way

$$\mathbf{A}[1{:}2,:] = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$$

Adress vector

$$\vec{c} = \mathbf{A}[:,2] = \begin{pmatrix} 2 \\ 5 \\ 6 \end{pmatrix}$$

$$\vec{d} = \mathbf{A}[2,:] = \begin{pmatrix} 4 & 5 & 6 \end{pmatrix}$$

The above also applies to vectors

$$\vec{v}_1 = \begin{pmatrix} 1 \\ 4 \\ 3 \end{pmatrix},\ \vec{v}_1[2] = 4,\ \vec{v}_1 = \begin{pmatrix} 9 \\ 4 \\ 3 \end{pmatrix}$$

The command `{plain }` groups equations to one math environment. For details see Section 5.4.7

5.4 Output

5.4.1 print()

The `[tVar]:print()` command creates the output according to the global parameter `tVar.outputMode`. The following options are supported:

Output

$$a = 10$$
$$b = 2$$
$$c = a + b = 10 + 2 = 12$$

tVar.outputMode = ”RES_EQ_N”

$$c = c = 12$$

tVar.outputMode = ”RES_EQ”

$$c = c = 12$$

tVar.outputMode = ”RES”

$$c = c = 12$$

If you do your calculations inside `\begin{tVar}` and `\end{tVar}` the `[tVar]:print()` command is automatically added to your calculations. This means `c:=(a+b)` creates an output using the `[tVar]:print()` command. To suppress the automatic output just add a ";" at the end of the line.

```
1  \begin{tVar}
2  # With output:
3  a:=10
4  b:=3
5  c:=a+b
6  # Without output (Lines 7 and 8 produce no output):
7  a:=10;
8  b:=3;
9  c:=a+b
10 \end{tVar}
```

Output

With output:

$$a = 10$$
$$b = 3$$
$$c = a + b = 10 + 3 = 13$$

Without output (Lines 7 and 8 produce no output):

$$c = a + b = 10 + 3 = 13$$

5.4.2 outRES()

The `[tVar]:outRES()` command is equal to the combination of `[tVar]:print()` with `tVar.outputMode = "RES"`. If you do your calculations inside `\begin{tVar}` and `\end{tVar}` the `[tVar]:outRES()` command is automatically added to any assignment of a new variable. That means `a:=10` creates and output using the `[tVar]:outRES()` command.

In some cases you might want to print an equation without assigning it to a variable. This can be achieved by directly calling the function `tVar.outRES([tVar])` (now with a dot) and passing your equation as argument to the function.

```
\begin{tVar}
a:=2
b:=13
tVar.outRES((a+b)/2+b)
\end{tVar}
```

Output

$$a = 2$$
$$b = 13$$

$$\frac{a + b}{2} + b = 20.5$$

This method also works for the output commands listed in Section 5.4.3.

5.4.3 Other Output Commands

For every output mode mentioned in Section 5.4.1 there is an equal output function.

RES_EQ_N `[tVar]:outRES_EQ_N(numbering,environment)`

RES_EQ `[tVar]:outRES_EQ(numbering,environment)`

RES `[tVar]:outRES(numbering,environment)`

The attributes *numbering* and *environment* are boolean values and define if the output is created with numeration and a math environment. Both parameters are optional and can also be defined via the global parameters `tVar.numeration` and `tVar.mathEnviroment`[3]

Additionally there are the functions `[tVar]:out()` for printing the result without the variable name and `[tVar]:outN()` for printing the variable name the equation with numbers and the result.

5.4.4 LaTeX Output

Inside `\begin{tVar}` and `\end{tVar}` the symbol # can be used to print a text in LaTeX. Inside such a text you can use %%[tVar]%% to print a variables name and $$[tVar]$$ to print a variables value.

LaTeX commands used within # have to be escaped. For example `\section{}` has to be written as `\\section{}`.

```
\begin{tVar}
#\\section*{First Section}
a_1_2:=22.4:setUnit("m")
# Linebreaks are a bit confusing \\\\
# The variable %%a_1_2%% has the value $$a_1_2$$
\end{tVar}
```

Output

First Subsection

$a_{1,2} = 22.4\ m$

Linebreaks are a bit confusing
The variable $a_{1,2}$ has the value $22.4\ m$

5.4.5 Precise Manipulation of Output

The following functions manipulate equations directly and can be used for detailed formatting.

```
\begin{tVar}
a:=10:setUnit("m")
b:=3:setUnit("m")
c:=((((a+b):bracR()^2):bracB()*5):bracC()*22):setUnit("m^2")
#In case you do a really long calculation you can insert a linebreak
    \\\\
c:=((a+a+a+a+a+a+a+a+b+b):CRLF("+")+b+b+b:CRLF_EQ("+")+b+b+b+a+a+a+a)
    :CRLFb("=")
tVar.debugMode = "off"
```

[3]In case `tVar.mathEnviroment` is set to ”” no math environment is used.

Output

$a = 10\ m$

$b = 3\ m$

$c = \left\{\left[(a+b)^2\right]\cdot 5\right\}\cdot 22 = \left\{\left[(10+3)^2\right]\cdot 5\right\}\cdot 22 = 18590\ m^2$

In case you do a really long calculation you can insert a linebreak

$$\begin{aligned} c &= a+a+a+a+a+a+a+a+a+b+b+b+b+b+ \\ &+b+b+b+a+a+a+a= \\ &= 10+10+10+10+10+10+10+10+3+3+ \\ &+3+3+3+3+3+3+10+10+10+10 = 144 \end{aligned}$$

`[tVar]:bracR()`	Surrounds the [tVar] object with round brackets.
`[tVar]:bracB()`	Surrounds the [tVar] object with boxed brackets.
`[tVar]:bracC()`	Surrounds the [tVar] object with curly brackets.
`[tVar]:CRLF ([string])`	Inserts a line break in N after the [tVar] object and adds the string before and after the line break.
`[tVar]:CRLFb([string])`	Inserts a line break in N before the [tVar] object and adds the string before and after the line break.
`[tVar]:CRLF_EQ ([string])`	Inserts a line break in EQ after the [tVar] object and adds the string before and after the line break.
`[tVar]:CRLFb_EQ([string])`	Inserts a line break in EQ before the [tVar] object and adds the string before and after the line break.
`[tVar]:clean()`	Removes the calculation history from an object.
`[tVar]:setUnit([string])`	Sets the unit for a tVar object.
`[tVar]:setFormat([string])`	Sets the numberformat for a tVar Object. For details see Section 5.4.6

5.4.6 Number Format

The number format can be controlled globally and locally. The function `[tVar]:setFormat([string])` defines the local number format of a tVar object. In case no local format was defined, tVar falls back to the global format set with `tVar.numberFormat = [string]`.

Common values are:

Output

Variable a is set to 192.6345
Integer `%d`

$a = 192$

Exponential `%.2E`

$$a = 1.93 \cdot 10^{2}$$

Float `%.3f`

$$a = 192.635$$

There is one case where TexVar automatically changes the number format: When the output precision return 0 but the calculation precision is unequal to 0 the number format is changed to `%.3E`.

Output

Variable a is set to 0.00001. Default number format is `%.3f`. Calculation precision is 10

$$a = 1 \cdot 10^{-5}$$

$$b = 123432$$

$$c = a \cdot b = 1 \cdot 10^{-5} \cdot 123432 = 1.234$$

5.4.7 Grouping Math Environments

By default TexVar creates a new mathematical environment for every output (except plain text-output with #). To group equations into one environment, you can enclose them with curly brackets. The group operator automatically adds an alignment symbol at the beginning of the line and a line-break at the end. If you want to suppress this behavior you can open the group with the command `{plain`.

```
\begin{tVar}
{
  a:=1
  b:=10
  c:=3
}
d:=(a+b):bracR()/c
#It's also possible to create environments without line-breaks and
    alignment:
{plain
f:=3
#,~
g:=f+d
}
\end{tVar}
```

Output

$$a = 1$$
$$b = 10$$
$$c = 3$$

$$d = \frac{(a+b)}{c} = \frac{(1+10)}{3} = 3.667$$

It's also possible to create environments without line-breaks and alignment:

$$f = 3,\ g = f + d = 3 + 3.667 = 6.667$$

5.5 Global Parameters

Global parameters can be set during runtime and affect all commands.

Tab. 1: Global parameters with default values and description

`tVar.numFormat = "%.3f"`	Defines the number format for printing.
`tVar.mathEnviroment = "align"`	Defines the environment used around equations.
`tVar.outputMode = "RES_EQ_N"`	Defines the outputmode (Section 5.4.1).
`tVar.numeration = true`	Disables and enables numeration of equations.
`tVar.decimalSeparator = "."`	Defines the decimal separator.
`tMat.texStyle = "mathbf"`	Defines the style for matrices.
`tMat.eqTexAsMatrix = false`	Enables and disables output of a matrix as variable name or matrix with variable names
`tVec.texStyle = "vec"`	Defines the style for vectors.
`tVar.calcPrecision = 10`	Defines the how many decimal places are used for comparison.
`tVar.disableOutput = false`	Disables the complete output.
`tVar.autocutZero = true`	Removes trailing zeros from a decimal number.
`tVar.autocutDecimalSep = true`	In case `tVar.autocutDecimalSep` is true remove the decimal separator if all trailing zeros have been removed. Else show number with one decimal zero. (Only works if `tVar.autocutZero = true`)
`tVar.debugMode = "off"`	In case debugMode is set to "on" the equations are printed as LaTeXcode.
`tVar.logInterp = false`	Enables logging of interpreted commands. Creates a file tVarLog.log
`tVar.coloredOutput = false`	Prints all variables with value=nil red.

5.5.1 Details on automatic removal of trailing zeros

The following example shows the difference between the global parameters `tVar.autocutDecimalSep` and `tVar.autocutZero`. If `tVar.autocutZero` is disabled the number format from `tVar.numFormat` get's applied.

```
\begin{tVar}
#Default settings
{
a:=2.0
b:=3.3
c:=tVar.sqrt(a^2+b^2)
}
#Disable tVar.autocutDecimalSep
tVar.autocutDecimalSep = false
```

```
{
a:=2.0
b:=3.3
c:=tVar.sqrt(a^2+b^2)
}
#Disable tVar.autocutZero
tVar.autocutZero = false
{
a:=2.0
b:=3.3
c:=tVar.sqrt(a^2+b^2)
}
\end{tVar}
```

Output

Default settings

$$a = 2$$
$$b = 3.3$$
$$c = \sqrt{a^2 + b^2} = \sqrt{2^2 + 3.3^2} = 3.859$$

Disable tVar.autocutDecimalSep

$$a = 2.0$$
$$b = 3.3$$
$$c = \sqrt{a^{2.0} + b^{2.0}} = \sqrt{2.0^{2.0} + 3.3^{2.0}} = 3.859$$

Disable tVar.autocutZero

$$a = 2.000$$
$$b = 3.300$$
$$c = \sqrt{a^{2.000} + b^{2.000}} = \sqrt{2.000^{2.000} + 3.300^{2.000}} = 3.859$$

5.6 Plotting with Gnuplot

Plotting in TexVar is support via Gnuplot. The code describing the figure is generated in TexVar and is sent to Gnuplot which creates the graphics.

5.6.1 Configuration

For enabling plotting the path to the Gnuplot executable has to be set via a global parameter. By default its set to `tPlot.gnuplot_library ="gnuplot"`. In case you work on a windows system and want to specify the absolute path to your Gnuplot install the command has to be `tPlot.gnuplot_library = [==[""WINDOWSPATH"]==]`.

The Gnuplot terminal is by default `tPlot.terminal = "pdf enhanced color font 'Helvetica ,12'"` and the file extension is `tPlot.FileExtension = "pdf"`.

5.6.2 Creating a Plot

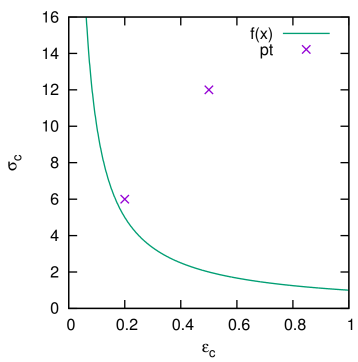
Every parameter set through `[tPlot].*` is directly translated to a gnuplot command. Actually tPlot is only aware of the following commands

`tPlot:New(present[tPlot])`	Creates a new plot. Present is an optional parameter. If a value gets passed the configuration of the plot is used as template for the new plot.
`[tPlot].steps = 0.1`	Resolution for functions
`[tPlot].conf.size = "14cm,8cm"`	Size of the plot
`[tPlot].xrange = "[0,10]"`	Range of the x axis and min and max value for function creation.
`[tPlot]:add(f or {{1,2},{3,2}},"f(x)", "with line lt 1 lc 2")`	Adds a functions or points to the plot.
`[tPlot]:plot()`	Generates the plot and returns the includegraphics command.

```
\begin{tVar}
f(x):=1/x

plot1=tPlot:New()
plot1.xlabel = "{/ Symbol e}_c"
plot1.ylabel = "{/ Symbol s}_c"
plot1.steps = 0.001

plot1.xtics = "0.1"
plot1.xrange = "[0:1]"
plot1.yrange = "[0:16]"
plot1.conf.size = "6cm ,6cm"
plot1:add(f,"f(x)", "with line lt 1 lc 2")
plot1:add({{0.2,6},{0.5,12}}, "pt", "with points lc 1")
#\\begin{center}
plot1:plot()
#\\end{center}
\end{tVar}
```

Output

$$f(x) = \frac{1.000}{x}$$

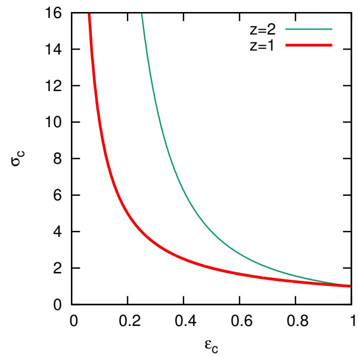


Currently TexVar only supports 2D plots and functions with one attribute. If you want to print a function with more than one attribute you can create a helper function.

```
\begin{tVar}
-- use plot1 as template
plot2=tPlot:New(plot1)

f(x,z):=(1/x)^z
-- helper functions with fixed z values
f_h_1(x):=f(x,2);
f_h_2(x):=f(x,1);

plot2:add(f_h_1,"z=2","with line lt 1 lc 2")
plot2:add(f_h_2,"z=1","with line lt 1 lc 3")

#\\begin{center}
plot2:plot()
#\\end{center}
\end{tVar}
```

Output

$$f(x, z) = \frac{1.000}{x}^{z}$$



5.7 Mathematical Commands

The following subsections are just a listing of currently implemented mathematical functions.

5.7.1 General

These functions can be used with every tVar object.

- tVar.sqrt([tVar],[tVar])
- tVar.PI
- tVar.abs([tVar])
- tVar.acos([tVar])
- tVar.cos([tVar])
- tVar.cosh([tVar])
- tVar.asin([tVar])
- tVar.sin([tVar])

- tVar.sinh([tVar])
- tVar.atan([tVar])
- tVar.tan([tVar])
- tVar.tanh([tVar])
- tVar.ceil([tVar])
- tVar.floor([tVar])
- tVar.exp([tVar])
- tVar.ln([tVar])
- tVar.log10([tVar])
- tVar.rad([tVar])
- tVar.deg([tVar])
- tVar.atan2(X[tVar],Y[tVar])
- tVar.fact([tVar])

5.7.2 Matrices and Vectors

These functions can only be used with tMat or tVec objects.

- [tMat]:T()
- [tMat]:Det()
- [tMat]:Inv()
- tVec.crossP([tVec],[tVec])

5.8 The Link Function

If you want to use your own functions within TexVar, you can use the Link function to link them to tVar functions. This only applies to functions that don't use TexVar objects for calculation. If you want to write a TexVar function see Section 5.2. The following code shows the implementation of the factorial function with the link function.

```
\begin{tVar}
function mycalcFactorial(n)
  -- no tVar objects here just numbers
  if n<=1 then return 1 end
  return n*mycalcFactorial(n-1)
end

-- link it
myfact = tVar.link(mycalcFactorial,"","!")

a:=10
b:=myfact(a)
\end{tVar}
```

Output

$a = 10.000$

$b = a! = 10.000! = 3628800.000$

The link function takes as first attribute the function you want to link as second a string to be added before the attributes and as third a string to be added after the attributes.

All Lua math functions are implemented this way.

For example the Lua function **`math.atan2`**:

```
--- calculates inverse tangens with with appr. quadrant
--
-- @param opposite (tVar,number) values
-- @param adjacent (tVar,number) values
-- @return (tVar)
tVar.atan2 = tVar.link(math.atan2,"\\text{atan2}\\left(","\\right)")
```

Output

$c = \text{atan2}\left(3.000; 4.000\right) = \text{atan2}\left(3.000; 4.000\right) = 0.644$

---

6 Examples

6.1 U-Value

This is a very simple example using only the basic functionality of TexVar.

```
\newcommand{\msKpW}{\tfrac{m^2K}{W}}
\newcommand{\WpmsK}{\tfrac{W}{m^2K}}
\newcommand{\WpmK}{\tfrac{W}{mK}}
\newcommand{\m}{m}
Calculating the U-Value for an element with two layers.\\
\begin{tVar}
  #\\noindent Resistance of surface
{plain
  R_se:= 0.3:setUnit("\\msKpW")
  #,~
  R_si:= 0.13:setUnit("\\msKpW")
}
  #Parameters for elements
{
  d_1 := 0.2:setUnit("\\m")
  \\lambda_1 := 0.035:setUnit("\\WpmK")
  d_2 := 0.1:setUnit("\\m")
  \\lambda_2 := 0.5:setUnit("\\WpmK")
}
  #Calculate thermal resistance
  R := (R_se + d_1/lambda_1 + d_2/lambda_2 + R_si):setUnit("\\msKpW")
  #Calculate U-Value
  U:=(1/R):setUnit("\\WpmsK")
\end{tVar}
```

Output

Calculating the U-Value for an element with two layers.
Resistance of surface

$$R_{se} = 0.300\ \tfrac{m^2K}{W},\ R_{si} = 0.130\ \tfrac{m^2K}{W}$$

Parameters for elements

$$d_1 = 0.200\ m$$
$$\lambda_1 = 0.035\ \tfrac{W}{mK}$$
$$d_2 = 0.100\ m$$
$$\lambda_2 = 0.500\ \tfrac{W}{mK}$$

Calculate thermal resistance

$$R = R_{se} + \frac{d_1}{\lambda_1} + \frac{d_2}{\lambda_2} + R_{si} = 0.300 + \frac{0.200}{0.035} + \frac{0.100}{0.500} + 0.130 = 6.344\ \tfrac{m^2K}{W}$$

Calculate U-Value

$$U = \frac{1.000}{R} = \frac{1.000}{6.344} = 0.158\ \tfrac{W}{m^2K}$$

6.2 Rotating a Vector

This example shows the usage of the parameter `tMat.eqTexAsMatrix` as mentioned in Section 5.5.

```
\begin{tVar}
tMat.eqTexAsMatrix = true

#Rotation angle in degree
\\theta:=tVar.rad(45)

#Rotation matrix in R2
{
A := {{tVar.cos(theta),-tVar.sin(theta)},{tVar.sin(theta),tVar.cos(
    theta)}}:outRES_EQ()
e_x:={1,0}
}


tMat.eqTexAsMatrix = false
e:=(A*e_x)
\end{tVar}
```

Output

Rotation angle in degree

$$\theta = \mathrm{rad}\,(45.000) = \mathrm{rad}\,(45.000) = 0.785$$

Rotation matrix in R2

$$\mathbf{A} = \begin{pmatrix} \cos(\theta) & (-\sin(\theta)) \\ \sin(\theta) & \cos(\theta) \end{pmatrix} = \begin{pmatrix} 0.707 & (-0.707) \\ 0.707 & 0.707 \end{pmatrix}$$

$$\vec{e}_x = \begin{pmatrix} 1.000 \\ 0.000 \end{pmatrix}$$

$$\vec{e} = \mathbf{A} \cdot \vec{e}_x = \begin{pmatrix} 0.707 & (-0.707) \\ 0.707 & 0.707 \end{pmatrix} \cdot \begin{pmatrix} 1.000 \\ 0.000 \end{pmatrix} = \begin{pmatrix} 0.707 \\ 0.707 \end{pmatrix}$$

6.3 Vector Calculations - Custom Function

This example shows how to create a custom TexVar function for calculating the angle between two vectors. The function has an extra parameter for disabling printing.

```
\begin{tVar}

tVar.outputMode = "RES_EQ"
function angleBetweenVectors(a,b,disablePrinting)
  if disablePrinting then
    tVar.disableOutput = true
  end

  #Calculate the length of the vectors

  len_a:=tVar.sqrt(a*a)
  len_b:=tVar.sqrt(b*b)

  #Normalize the vectors

  a_n:=(a/len_a)
  b_n:=(b/len_b)

  \\alpha:=tVar.deg(tVar.acos(a_n*b_n)):setUnit("^{\\circ}")

  if disablePrinting then
    tVar.disableOutput = false
  end

  return alpha
end

#With output
{
  v_1:={1,0.4,0.5}
  v_2:={0.3,1,0}
}

\\alpha_1:=angleBetweenVectors(v_1,v_2)

#Without output
{
  v_3:={4,0.2,5}
  v_4:={9.3,8,1}
}
\\alpha_2:=angleBetweenVectors(v_3,v_4,true)
\end{tVar}
```

Output

With output

$$\vec{v}_1 = \begin{pmatrix} 1.000 \\ 0.400 \\ 0.500 \end{pmatrix}$$

$$\vec{v}_2 = \begin{pmatrix} 0.300 \\ 1.000 \\ 0.000 \end{pmatrix}$$

Calculate the length of the vectors

$$len_a = \sqrt{\vec{v}_1 \cdot \vec{v}_1} = 1.187$$

$$len_b = \sqrt{\vec{v}_2 \cdot \vec{v}_2} = 1.044$$

Normalize the vectors

$$\vec{a}_n = \frac{\vec{v}_1}{len_a} = \begin{pmatrix} 0.842 \\ 0.337 \\ 0.421 \end{pmatrix}$$

$$\vec{b}_n = \frac{\vec{v}_2}{len_b} = \begin{pmatrix} 0.287 \\ 0.958 \\ 0.000 \end{pmatrix}$$

$$\alpha = \deg\left(\operatorname{acos}\left(\vec{a}_n \cdot \vec{b}_n\right)\right) = 55.622\ ^\circ$$

$$\alpha_1 = \alpha = 55.622\ ^\circ$$

Without output

$$\vec{v}_3 = \begin{pmatrix} 4.000 \\ 0.200 \\ 5.000 \end{pmatrix}$$

$$\vec{v}_4 = \begin{pmatrix} 9.300 \\ 8.000 \\ 1.000 \end{pmatrix}$$

$$\alpha_2 = \alpha = 56.255\ ^\circ$$