TexVar - Manual

TexVar – LaTeX math calculations

Version: 1.5.5

Sebastian Pech

November 9, 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*.

```
1   \documentclass{article}
2   %
3   \usepackage{texvar}
4   \usepackage[fleqn]{amsmath}
5   %
6   \begin{document}
7   \begin{tVar}
8   #Hello World! I'm using TexVar
9   tVar.getVersion()
10  \end{tVar}
11  \end{document}
```

Output

Hello World! I'm using TexVar Version: 1.5.5

---

4 General Information on tVar Enviroment

This section gives a general insight on how commands are interpreted and how this affects the code. Generally, there are two key commands the TexVar interpreter searches for [2]:

1. #

2. :=

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

---

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

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

```
1  \begin{tVar}
2  function mypow(a,b)
3  return a^b
4  end
5  for i=0,4 do
6  tex.print("Step " .. mypow(2,i) .. "\\\\")
7  end
8  \end{tVar}
```

Output

Step 1
Step 2
Step 4
Step 8
Step 16

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 :=.

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

Output

$$a = 13.0$$

$$\vec{e} = \begin{pmatrix} 1.0 \\ 2.0 \\ 4.0 \end{pmatrix}$$

$$\mathbf{A} = \begin{pmatrix} 1.0 & 0.0 \\ 2.0 & 4.0 \\ 3.0 & 9.1 \end{pmatrix}$$

5.1.1 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.

```
1  \begin{tVar}
2  a_1_3:=13
3  a__10_2:=3
4  b_1_x__y_2:=12
5  \end{tVar}
```

Output

$$a_{1,3} = 13.0$$

$$a_2^{10} = 3.0$$

$$b_{1,x}^{y,2} = 12.0$$

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 5.1.1.

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

Output

$$f(x,y) = x^{2.0} + y^{2.0} + 4.0$$

$$a = f(2.0, 3.0) + 11.0 = 2.0^{2.0} + 3.0^{2.0} + 4.0 + 11.0 = 28.0$$

5.3 Indexing Matrices

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.

```
1  \begin{tVar}
2  A:={{1,2,6},{2,4,6},{7,6,9}}
3  # Index one element with syntax [row,column]
4  A["1,2"]:outRES()
5  # Index a range
6  A["1:2,1:end"]:outRES()
```

```
 7  # The range 1:end is equal to :
 8  A["1:2,:"]:outRES()
 9  # You can also set matrices this way
10  A["1:2,:"]:={{1,2,3},{4,5,6}}
11  A["1:2,:"]:outRES()
12  \end{tVar}
```

Output

$$\mathbf{A} = \begin{pmatrix} 1.0 & 2.0 & 6.0 \\ 2.0 & 4.0 & 6.0 \\ 7.0 & 6.0 & 9.0 \end{pmatrix}$$

Index one element with syntax [row,column]

$$\mathbf{A}[1,2] = 2.0$$

Index a range

$$\mathbf{A}[1:2,1:\text{end}] = \begin{pmatrix} 1.0 & 2.0 & 6.0 \\ 2.0 & 4.0 & 6.0 \end{pmatrix}$$

The range 1:end is equal to :

$$\mathbf{A}[1:2,:] = \begin{pmatrix} 1.0 & 2.0 & 6.0 \\ 2.0 & 4.0 & 6.0 \end{pmatrix}$$

You can also set matrices this way

$$\mathbf{A}[1:2,:] = \begin{pmatrix} 1.0 & 2.0 & 3.0 \\ 4.0 & 5.0 & 6.0 \end{pmatrix}$$

Adress vector

$$\vec{c} = \mathbf{A}[:,2] = \begin{pmatrix} 2.0 \\ 5.0 \\ 6.0 \end{pmatrix} = \begin{pmatrix} 2.0 \\ 5.0 \\ 6.0 \end{pmatrix}$$

$$\vec{d} = \mathbf{A}[2,:] = \begin{pmatrix} 4.0 & 5.0 & 6.0 \end{pmatrix} = \begin{pmatrix} 4.0 & 5.0 & 6.0 \end{pmatrix}$$

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

$$a = 10.0$$

$$b = 2.0$$

tVar.outputMode = "RES_EQ_N"

$$c = a + b = 10.0 + 2.0 = 12.0$$

tVar.outputMode = "RES_EQ"

$$c = a + b = 12.0$$

tVar.outputMode = "RES"

$$c = 12.0$$

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:
 7  a:=10;
 8  b:=3;
 9  c:=a+b
10  \end{tVar}
```

Output

With output:

$$a = 10.0$$

$$b = 3.0$$

$$c = a + b = 10.0 + 3.0 = 13.0$$

Without output:

$$c = a + b = 10.0 + 3.0 = 13.0$$

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.

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{}`.

```
1  \begin{tVar}
2  #\\section*{First Section}
3  a_1_2:=22.4
4  # Linebreaks are a bit confusing \\\\
5  # The variable %%a_1_2%% has the value $$a_1_2$$
6  \end{tVar}
```

Output

First Subsection

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

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

5.4.5 Precise Manipulation of Output

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

`[tVar]:bracR()`	Surrounds the [tVar] object with round 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.

---

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

```
1  \begin{tVar}
2  a:=10:setUnit("m")
3  b:=3:setUnit("m")
4  c:=((a+b):bracR()^2):setUnit("m^2")
5  #In case you do a really long calculation you can insert a linebreak
6  c:=((a+a+a+a+b+b):CRLF("+")+b+b+b:CRLF_EQ("+")+b+b+a+a+a):CRLFb("=")
7  \end{tVar}
```

Output

$$a = 10.0\ m$$

$$b = 3.0\ m$$

$$c = (a + b)^{2.0} = (10.0 + 3.0)^{2.0} = 169.0\ m^2$$

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

$$c = a + a + a + a + a + a + a + a + b + b + b + b + b +$$
$$+ b + b + b + a + a + a + a =$$
$$= 10.0 + 10.0 + 10.0 + 10.0 + 10.0 + 10.0 + 10.0 + 10.0 + 3.0 + 3.0 +$$
$$+ 3.0 + 3.0 + 3.0 + 3.0 + 3.0 + 3.0 + 10.0 + 10.0 + 10.0 + 10.0 = 144.0$$

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.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.

5.6 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.

```
1  \begin{tVar}
2  f(x):=1/x
3
4  plot1=tPlot:New()
5  plot1.xlabel = "{/ Symbol e}_c"
6  plot1.ylabel = "{/ Symbol s}_c"
7  plot1.steps = 0.001
8
9  plot1.xtics = "0.1"
10 plot1.xrange = "[0:1]"
11 plot1.yrange = "[0:16]"
12 plot1.conf.size = "6cm ,6cm"
13 plot1:add(f,"f(x)", "with line lt 1 lc 2")
14 plot1:add({{0.2,6},{0.5,12}}, "pt", "with points lc 1")
15 #\\begin{center}
16 plot1:plot()
17 #\\end{center}
18 \end{tVar}
```

Output

$$f(x) = \frac{1.0}{x}$$
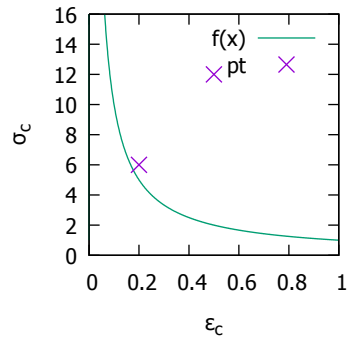
Currently TexVar only supports 2D plots and functions with one attribute. If you want to print a function with more than one attribute you can create a helper function.
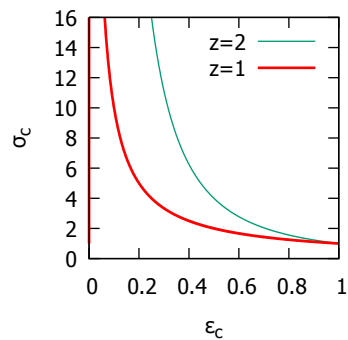
```
1   \begin{tVar}
2   -- use plot1 as template
3   plot2=tPlot:New(plot1)
4
5   f(x,z):=(1/x)^z
6   -- helper functions with fixed z values
7   f_h_1(x):=f(x,2);
8   f_h_2(x):=f(x,1);
9
10  plot2:add(f_h_1,"z=2","with line lt 1 lc 2")
11  plot2:add(f_h_2,"z=1","with line lt 1 lc 3")
12
13  #\\begin{center}
14  plot2:plot()
15  #\\end{center}
16  \end{tVar}
```

Output

$$f(x,z) = \frac{1.0^{\,z}}{x}$$

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.

```
1   \begin{tVar}
2   function mycalcFactorial(n)
3     -- no tVar objects here just numbers
4     if n<=1 then return 1 end
5     return n*mycalcFactorial(n-1)
6   end
7
8   -- link it
9   myfact = tVar.link(mycalcFactorial,"","!")
10
11  a:=10
12  b:=myfact(a)
13  \end{tVar}
```

Output

$$a = 10.0$$

$$b = a! = 10.0! = 3628800.0$$

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`:

```
1   --- calculates inverse tangens with with appr. quadrant
2   --
3   -- @param opposite (tVar,number) values
4   -- @param adjacent (tVar,number) values
5   -- @return (tVar)
6   tVar.atan2 = tVar.link(math.atan2,"\\text{atan2}\\left(","\\right)")
```

Output

$$c = \text{atan2}\left(3.0; 4.0\right) = \text{atan2}\left(3.0; 4.0\right) = 0.644$$

---

6 Examples

6.1 U-Value

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

```
1   \newcommand{\msKpW}{\tfrac{m^2K}{W}}
2   \newcommand{\WpmsK}{\tfrac{W}{m^2K}}
3   \newcommand{\WpmK}{\tfrac{W}{mK}}
4   \newcommand{\m}{m}
5
6   Calculating the U-Value for an element with two layers and resistance
        of surface internal and external.\\
7   \begin{tVar}
8   -- define variables
9   #\\noindent Resistance of surface
10  R_se:= 0.3:setUnit("\\msKpW")
11  R_si:= 0.13:setUnit("\\msKpW")
12
13  #Parameters for elements
14  d_1 := 0.2:setUnit("\\m")
15  \\lambda_1 := 0.035:setUnit("\\WpmK")
16
17  d_2 := 0.1:setUnit("\\m")
18  \\lambda_2 := 0.5:setUnit("\\WpmK")
19
20  #Calculate thermal resistance
21  R := (R_se + d_1/lambda_1 + d_2/lambda_2 + R_si):setName("R"):setUnit
        ("\\msKpW")
22
23  #Calculate U-Value
24  U:=(1/R):setUnit("\\WpmsK")
25  \end{tVar}
```

Output

Calculating the U-Value for an element with two layers and resistance of surface internal and external.
Resistance of surface

$$R_{se} = 0.3 \ \tfrac{m^2K}{W}$$

$$R_{si} = 0.13 \ \tfrac{m^2K}{W}$$

Parameters for elements

$$d_1 = 0.2 \ m$$

$$\lambda_1 = 0.035 \ \tfrac{W}{mK}$$

$$d_2 = 0.1 \ m$$

$$\lambda_2 = 0.5 \ \tfrac{W}{mK}$$

Calculate thermal resistance

$$R = R_{se} + \frac{d_1}{\lambda_1} + \frac{d_2}{\lambda_2} + R_{si} = 0.3 + \frac{0.2}{0.035} + \frac{0.1}{0.5} + 0.13 = 6.344 \ \frac{m^2 K}{W}$$

Calculate U-Value

$$U = \frac{1.0}{R} = \frac{1.0}{6.344} = 0.158 \ \frac{W}{m^2 K}$$

6.2 Rotating a Vector

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

```
1  \begin{tVar}
2  tMat.eqTexAsMatrix = true
3
4  #Rotation angle in degree
5  \\theta:=tVar.rad(45)
6
7  #Rotation matrix in R2
8  A := {{tVar.cos(theta),-tVar.sin(theta)},{tVar.sin(theta),tVar.cos(
       theta)}}:outRES_EQ()
9  e_x:={1,0}
10
11
12 tMat.eqTexAsMatrix = false
13 e:=(A*e_x)
14 \end{tVar}
```

Output

Rotation angle in degree

$$\theta = \mathrm{rad}\,(45.0) = \mathrm{rad}\,(45.0) = 0.785$$

Rotation matrix in R2

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

$$\vec{e}_x = \begin{pmatrix} 1.0 \\ 0.0 \end{pmatrix}$$

$$\vec{e} = \mathbf{A} \cdot \vec{e}_x = \begin{pmatrix} 0.707 & (-0.707) \\ 0.707 & 0.707 \end{pmatrix} \cdot \begin{pmatrix} 1.0 \\ 0.0 \end{pmatrix} = \begin{pmatrix} 0.707 \\ 0.707 \end{pmatrix}$$

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.

```
1   \begin{tVar}
2   tVar.outputMode = "RES_EQ"
3
4   function angleBetweenVectors(a,b,disablePrinting)
5     if disablePrinting then
6       tVar.disableOutput = true
7     end
8     #Calculate the length of the vectors
9     len_a:=tVar.sqrt(a*a)
10    len_b:=tVar.sqrt(b*b)
11
12    #Normalize the vectors
13    a_n:=(a/len_a)
14    b_n:=(b/len_b)
15
16    \\alpha:=tVar.deg(tVar.acos(a_n*b_n)):setUnit("^{\\circ}")
17    if disablePrinting then
18      tVar.disableOutput = false
19    end
20    return alpha
21  end
22
23  #With output
24  v_1:={1,0.4,0.5}
25  v_2:={0.3,1,0}
26
27  \\alpha_1:=angleBetweenVectors(v_1,v_2)
28  #Without output
29  v_3:={4,0.2,5}
30  v_4:={9.3,8,1}
31
32  \\alpha_2:=angleBetweenVectors(v_3,v_4,true)
33  \end{tVar}
```

Output

With output

$$\vec{v}_1 = \begin{pmatrix} 1.0 \\ 0.4 \\ 0.5 \end{pmatrix}$$

$$\vec{v}_2 = \begin{pmatrix} 0.3 \\ 1.0 \\ 0.0 \end{pmatrix}$$

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

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

$$\vec{b}_n = \frac{\vec{v}_2}{len_b} = \begin{pmatrix} 0.287 \\ 0.958 \\ 0.0 \end{pmatrix}$$

$$\alpha = \deg\left(\mathrm{acos}\left(\vec{a}_n \cdot \vec{b}_n\right)\right) = 55.622\,°$$

$$\alpha_1 = \alpha = 55.622\,°$$

Without output

$$\vec{v}_3 = \begin{pmatrix} 4.0 \\ 0.2 \\ 5.0 \end{pmatrix}$$

$$\vec{v}_4 = \begin{pmatrix} 9.3 \\ 8.0 \\ 1.0 \end{pmatrix}$$

$$\alpha_2 = \alpha = 56.255\,°$$