

# tikzscale — Absolute resizing of TikZ pictures and PGF plots without scaling text\*

Patrick Häcker<sup>†</sup>

Released 2012/11/02

## 1 Introduction

When dealing with graphics, there are different scaling demands. For *absolute* scaling, a width and/or height is given. Opposed to that, for *relative* scaling, a horizontal and/or vertical scaling factor is needed. This package only is about absolute scaling of tikzpicture environments. The different absolute scaling demands and their solutions are shown in table 1.

The tikzscale package adds and improves certain forms of absolute scaling for TikZ and PGFPlots, respectively. These scaling methods are the ones which are most useful, maybe even the only ones which are needed. During the scaling, the text sizes and line widths are left unscaled, which avoids inconsistency and visual distraction. PGFPlots itself can scale absolutely, but an approximation is used to achieve that. The tikzscale package uses optimization algorithms and warns if the scaling is not exact.

Using tikzscale all relevant scaling methods share the same user interface with the well known `\includegraphics` command, enabling some of its features like automatic file extension detection for TikZ and PGFPlots, too. Furthermore, the `\includegraphics` is improved to look-up relative paths in the correct subdirectory, if a  $\LaTeX$  project is organized in subdirectories.

Relative scaling methods are mostly useless, as the sizes of the used images are often arbitrary, either determined by some resolution for rastered images or some arbitrary unit vector size for vector images, TikZ and PGFPlots. For traditional images and TikZ pictures, only proportional scaling methods giving either a width or a height make sense, as otherwise they get heavily distorted if the original aspect ratio is changed. As PGFPlots can handle different aspect ratios and aspect ratios are normally not predefined for plots, its requirement is the opposite: A width and a height are needed to avoid getting arbitrary sizes. For some special plots, the axis ratio can be given, as well. These requirements lead to the marked blue colors in table 1.

---

\*This file describes version v0.1.2, last revised 2012/11/02.

<sup>†</sup>E-mail: pat\_h@web.de

Table 1: Absolute graphic scaling methods. If multiple methods are available, the most native one is shown. Methods which **approximate** the scaling are shown in orange text color. **Recommended** methods are shown in blue textcolor.

(a) Scaling with scaled text and line widths.

scale	Images	TikZ/PGFPlots
to width	<code>\includegraphics</code>	<code>\resizebox</code>
proportionally	<code>[width=<i>unit</i>]</code>	<code>{<i>width</i>}{!}</code>
to width	<code>\resizebox</code>	<code>\resizebox</code>
keeping height	<code>{<i>width</i>}{\height}</code>	<code>{<i>width</i>}{\height}</code>
to height	<code>\includegraphics</code>	<code>\resizebox</code>
proportionally	<code>[height=<i>unit</i>]</code>	<code>{!}{<i>height</i>}</code>
to height	<code>\resizebox</code>	<code>\resizebox</code>
keeping width	<code>{\width}{<i>height</i>}</code>	<code>{\width}{<i>height</i>}</code>
to width	<code>\includegraphics</code>	<code>\resizebox</code>
and height	<code>[width=<i>unit</i>,height=<i>unit</i>]</code>	<code>{<i>width</i>}{<i>height</i>}</code>

(b) Scaling with unscaled text and line widths without tikzscale.

scale	Images	TikZ	PGFPlots
to width	—	—	<code>[width=<i>unit</i>]</code>
proportionally	—	—	—
to width	—	—	—
keeping height	—	—	—
to height	—	—	<code>[height=<i>unit</i>]</code>
proportionally	—	—	—
to height	—	—	—
keeping width	—	—	—
to width	—	—	<code>[width=<i>unit</i>,height=<i>unit</i>]</code>
and height	—	—	—

(c) Scaling with unscaled text and line widths with tikzscale.

scale	Images	TikZ	PGFPlots
to width	—	<code>\includegraphics</code>	<code>\includegraphics</code>
proportionally	—	<code>[width=<i>unit</i>]</code>	<code>[width=<i>unit</i>]</code>
to width	—	—	—
keeping height	—	—	—
to height	—	<code>\includegraphics</code>	<code>\includegraphics</code>
proportionally	—	<code>[height=<i>unit</i>]</code>	<code>[height=<i>unit</i>]</code>
to height	—	—	—
keeping width	—	—	—
to width	—	—	<code>\includegraphics</code>
and height	—	—	<code>[width=<i>unit</i>,height=<i>unit</i>]</code>

## 2 Usage and Examples

Loading the `tikzscale` package without loading other packages, does not do anything useful.

### 2.1 TikZ

If the `tikzscale` and the `tikz` packages are loaded, the `\includegraphics` command can be used to input and scale a `tikzpicture` environment located in a separate file.

As an example create the following `.tex`-file.

```
\documentclass{minimal}
\usepackage{tikz}
\usepackage{tikzscale}
\begin{document}
  \includegraphics[width=0.5\linewidth]{linewidth.tikz}
\end{document}
```

Furthermore create the following `.tikz`-file and save it as `linewidth.tikz` in the same directory as the above `.tex`-file.

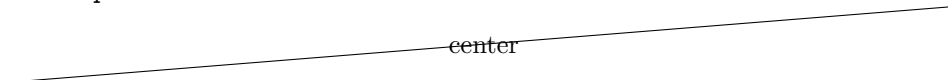
```
\begin{tikzpicture}
  \draw (0,0) - node {center} (\linewidth,1);
\end{tikzpicture}
```

The result of the compiled `.tex`-file should look like this.



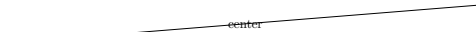
So although the original `tikzpicture` itself has the width of a complete line, it gets proportionally scaled down to half the width while being loaded from the `\includegraphics` command. Neither the line's thickness nor the text `center` are scaled. Compare the output to

```
\input{linewidth.tikz}
```



and

```
\resizebox{0.5\linewidth}{!}{\input{linewidth.tikz}}
```



to see `tikzscale`'s benefit.

### 2.2 PGFPlots

#### 2.2.1 Scaling of width and height

If the `pgfplots` package is loaded together with the `tikzscale` package, the user interface is the same. Instead of giving either a width or a height, both have to be given for `pgfplots`. So

```
\input{pgfplots-test.tikz}
```

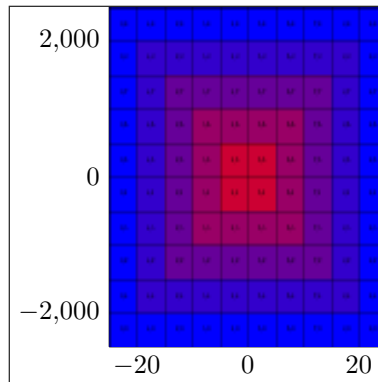


Figure 1: Using options `width=0.4\linewidth` and `height=0.4\linewidth` results in an overall quadratic graphic with overall width and height set to 40% of the linewidth.

```
\begin{tikzpicture}\begin{axis}[width=3cm,height=2cm] ...
```

becomes

```
\includegraphics[width=3cm,height=2cm]{pgfplots-test.tikz}
\begin{tikzpicture}\begin{axis} ...
```

The benefit is a more accurate scaling algorithm, as the scaling with PGFPlots can be quite coarse. Another win is the unified interface, which simplifies the sharing of plots between projects enormously, as one file and thus one plot can be included in different projects with different sizes.

### 2.2.2 Scaling using axis ratio

The scaling described in the previous section scales the whole plot including all axis descriptions and legends to the given width and height. It can thus happen, that the plotted figure has a different size ratio than expected, if the x and y descriptions have different sizes as shown in figure 1. Sometimes, the x-axis and the y-axis should have a specific ratio, e.g. being equal, ignoring the axis description and other things. This is normally achieved by using PGFPlots' option `scale only axis`. Unfortunately, if this option would be used, a plot might be unsharable between two projects, if they have different requirements for the axis ratio. Thus, this option should not be used in this case. Instead, in `\includegraphics` there is a new option `axisratio` which must be used together with either width or height. It scales the whole plot including the axis description to the given width or height as in figure 2 while keeping the graphical part at a given axis ratio, where the ratio is defined by width divided by height. The graphical part is thus not quadratic in general.

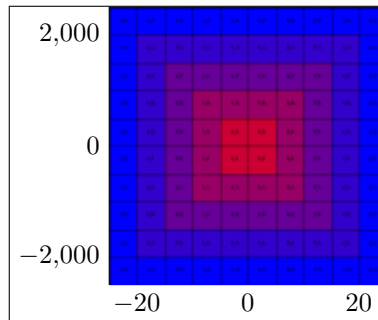


Figure 2: Using options `width=0.4\linewidth` and `axisratio=1` results in an quadratic graphic area with overall width set to 40% of the linewidth. The height follows from these constraints, so that the overall plot is not quadratic in general.

## 2.3 Hints for TikZ and PGFPlots

The whole tikzpicture environment must be in a separate file. This allows sharing of graphics between different  $\text{\TeX}$  projects and a unified user interface via `\includegraphics`. Having tikzpicture environments directly in a .tex-file is not supported, i.e. they do not benefit from the tikzscale package. Multiple tikzpicture environments in one .tikz-file are not supported, either. Put things which always belong together in a shared tikzpicture environment and things which might be used separately in the future in separate files for code sharing across projects. The file ending may be omitted in the `\includegraphics` command, if it is one of .tikz, .TIKZ, .TikZ, .pgf or .PGF. At the moment, use only *either* width *or* height for normal (i.e. non-PGFPlots) tikzpicture environments and use width *and* height or one of both together with *axisratio* for tikzpicture environments containing a PGFPlots' axis environment.

## 2.4 currfile

If the tikzpicture package is loaded together with the `currfile` package, another feature is activated. Suppose you have your project organized in the following directory tree with `directories` shown in blue color:

```
projectDirectory
  main.tex
  firstChapter
    firstChapter.tex
    firstGraphicOfFirstChapter.jpeg
    secondGraphicOfFirstChapter.tikz
  secondChapter
    secondChapter.tex
    firstGraphicOfSecondChapter.tikz
    secondGraphicOfSecondChapter.jpeg
```

Further suppose the chapter.tex files are `\inputted` in main.tex. Calling `\includegraphics{firstGraphicOfFirstChapter.jpeg}` in firstChapter.tex normally does not work. The reason is that the `\input{firstChapter.tex}` command in main.tex copies the content of firstChapter.tex into main.tex, so when the `\includegraphics` command is called, it is called from within project-Directory, thus the relative path lookup of firstGraphicOfFirstChapter.jpeg fails. Instead the command `\includegraphics{firstChapter/firstGraphicOfFirstChapter.jpeg}` can be used (example for a Unix system), but this is tedious and counter-intuitive.

If both `tikzscale` and `currfile` are loaded, the limitation is fixed, so that both `\includegraphics` commands succeed. Note, that this functionality supports the traditional graphic formats, too, and is also available without loading the TikZ or PGFPlots packages, although the package's name might imply otherwise.

### 3 Compatibility

Using both the externalization library and `tikzscale` seems to have a race condition when a makefile is used with multiple jobs (`-jX` with  $X > 1$ ). The probability of getting errors increases with the number of jobs. For  $X = 1$ , obviously, no race condition could be observed. You should either avoid using mode *list and make* or have only one job if you want to be on the safe side.

#### 3.1 Load Order

There is no constraint regarding the load order known, yet. TikZ, PGFPlots and `currfile` might all be loaded or not in all possible combinations and orders before or after `tikzscale`.

#### 3.2 Externalization library

TikZ' externalization library is supported. Its use is highly recommended, as `tikzscale` renders some graphics multiple times to get the correct size. The savings by using the externalization library can thus be huge.

### 4 Further Ideas

- When the externalization library is used, the graphic files get regenerated more often than necessary. This is, because the axis ratio is not saved, thus assumed unknown and the graphic is thus regenerated. For performance reasons if a default axis ratio is used, the plot is not regenerated. If a non-default axis ratio has been used for generating the externalized graphic and the default axis ratio is used in the current run, this is not detected and the graphic has the wrong size at the moment. You can, of course, delete the externalized graphic to generate a new one with the correct size.

- if graphic files are located in a subdirectory, the externalized files should also be in that subdirectory.
- the package can test if a pgfplot is used (needed if normal TikZ graphics should be stretchable) by changing `\tikzscale@width` and or `\tikzscale@height` and measuring. If nothing changes, it must be a normal tikzpicture (the argument does not hold the other way round).
- it may be better to use the **depth as well**
- The final sizing parameters should be saved per figure in the aux file. The first rendering each run should be performed with the aux file's parameters into an sbox. The scaling algorithms should only be called, if the sizing requirements are not met.

## 5 Contributions

- Jake
  - Encouraged the author to create this package.
- Dr. Christian Feuersänger
  - Encouraged the author to create this package and created PGFPlots.
  - Answered many questions and had a lot of good ideas regarding the externalization and beyond.
- David Carlisle
  - Created the `\xcmd` macro for this package, which is used in the documentation.

## 6 Implementation

The basic idea is to first get the correct file name (i.e. find the path and the file extension), then determine the graphic type (i.e. TikZ or something else) and call either the original `includegraphics` command or the `tikzscale` command. Tikzpictures are then plotted into an invisible box and their size is measured. If their measured size differs from the requested size, they are replotted with corrected parameters to get the requested size. The correctly sized plots are then really plotted.

```
\tikzexternal
```

```
1 \AtEndPreamble{%
```

Activate the output of the graphics sizes into the dpth files (one file per graphic). This key is used if the externalization library is activated to check if the scaling is correct, otherwise the code is not needed

```

2 \ifdef{\tikzifexternalizing}{%
3 \pgfkeys{/pgf/images/external info}%
4 }{}%
```

Provide dummy commands, if the externalization library has not been loaded during the preamble.

```

5 \ProvideDocumentCommand{\tikzsetnextfilename}{m}{}%
6 \ProvideDocumentCommand{\tikzsetexternalprefix}{m}{}%
7 \ProvideDocumentCommand{\tikzexternaldisable}{}{}%
8 \ProvideDocumentCommand{\tikzexternalenable}{}{}%
9 \ProvideDocumentCommand{\tikzexternalize@hasbeencalled}{}{0}%
```

Define a test to check if the current graphic should be drawn and scaled.

```

10 \ifboolexpr{test {\ifundef{\tikzexternalize}} or test {\ifdefstring{\tikzexternalize@hasbeencalled}}}
```

Always draw and scale the graphic if externalization library has not been loaded or not been activated.

```

11 \def\tikzscale@ifDrawAndScale#1#2{#1}%
12 }{%
```

If externalization library has been loaded and activated, draw and scale the graphic if it is to be externalized.

```

13 \def\tikzscale@ifDrawAndScale#1#2{%
14 \tikzifexternalizingnext{#1}{#2}%
15 }%
16 }%
17 %\end{macro}
18 %
19 %\begin{macro}{\tikzscale@scale}
20 % \begin{macrocode}
21 \@ifpackageloaded{tikz}{%
```

Set a minimum accuracy tikzscale tries to achieve. TeX's accuracy is limited, thus 0.001 pt cannot always be achieved independent of the number of iterations. The current value is chosen arbitrarily, but it might have to be increased in case a plot is found where the given accuracy cannot be achieved.

```

22 \newlength{\tikzscale@accuracy}%
23 \setlength{\tikzscale@accuracy}{0.01pt}%
```

This is needed in normal TikZ pictures and in PGFPlots, but as the pgfplots package loads the tikz package, it is fine to define it here.

```

24 \def\maxTestIterations{10}%
25 }{}%
26 }
```

This command draws the plot's border at the right text border, so that thick points or label descriptions can reach into the margin. This should be limited to PGFPlots only if activated.

With the option below, the labels can be moved a bit to the left so that they reach to the text margin. `yticklabel style=align=right,inner sep=0pt,xshift=-0.1cm`

`\pgfmathsetglobalmacro` This is a general command, which might be useful for inclusion into the tikz package. It works similar to `\pgfmathsetglobalmacro` but has global scope.

```
27 \def\pgfmathsetglobalmacro#1#2{%
28 \pgfmathparse{#2}%
29 \global\let#1\pgfmathresult%
30 }
```

`\edocsvlist` This is a general command, which might be useful for inclusion into the etoolbox package. It works similar to `\docsvlist` but expands its argument similar to `\def` vs. `\edef`, which is useful if the list is stored in a macro/variable.

```
31 \def\edocsvlist#1{%
32 \edef\tikzscale@edocsvlist{#1}%
33 \expandafter\docsvlist\expandafter{\tikzscale@edocsvlist}%
34 }
```

`\eforcsvlist` These is a general command, which might be useful for inclusion into the etoolbox package. It works similar to `\forcsvlist` but expands its argument similar to `\def` vs. `\edef`, which is useful if the list is stored in a macro/variable.

```
35 \def\eforcsvlist#1#2{%
36 \edef\tikzscale@eforcsvlist{#2}%
37 \expandafter\forcsvlist\expandafter{\expandafter#1\expandafter}\expandafter{\tikzscale@eforcsvlist}%
38 }
```

`\elseif` This macro provides a conditional which supports an if with an arbitrary amount of elseif (none is also ok) and an optional else. With a simplified syntax (remove the tests and the grouping) this would be worth a separate package.

```
39 \NewDocumentCommand{\elseif}{mm}{%
40 \ifboolexpr{#1}{%
41 #2%
42 \elseif@absorb
43 }{%
44 \elseif@optional
45 }%
46 }
47 \NewDocumentCommand{\elseif@optional}{gg}{%
48 \IfValueTF{#1}{%
49 \IfValueTF{#2}{%
50 \ifboolexpr{#1}{%
51 #2%
52 \elseif@absorb
53 }{%
54 \elseif@optional
55 }%
56 }{%
```

```

57 #1%
58 }%
59 }{}%
60 }
61 \NewDocumentCommand{\elseif@absorb}{g}{%
62 \IfValueTF{#1}{%
63 \elseif@absorb
64 }{}%
65 }

```

T his command is from [Bruno Le Floch](#).

```

66 \ExplSyntaxOn
67 \NewDocumentCommand{\IfNoValueOrSplitEmptyTF}{mmm}{
68 \ifboolexpr{test {\IfNoValueTF{#1}} or test {\tl_if_eq:nnTF{#1}{}}}{
69 #2
70 }{
71 #3
72 }
73 }
74 \ExplSyntaxOff

```

`\activatetikzscale`

```

75 \AtEndPreamble{%

```

Add the TikZ file extensions to the [graphicx file extensions](#).

```

76 \def\tikzscale@tikzFileExtensions{.tikz,.TIKZ,.TikZ,.pgf,.PGF}%
77 % \def\tikzscale@tikzFileExtensions{.tikz,.TIKZ,.TikZ,.pgf,.PGF,.tex,.TEX}%
78 \DeclareGraphicsExtensions{\tikzscale@tikzFileExtensions,\Gin@extensions}%

```

The `\graphicspath` command is used to set additional directories, which are searched for graphics. `\Ginput@path` is used to get the [current content](#).

```

79 \ifdef{\currfiledir}{%
80 % \graphicspath{{\currfiledir}}\Ginput@path}%
81 \def\tikzscale@graphicspath{\currfiledir,}%
82 }{%
83 \def\tikzscale@graphicspath{}}%
84 }%

```

Save the `\includegraphics` [command](#) and replace it by a new more generic command, to have a consistent user interface.

```

85 \LetLtxMacro{\tikzscale@oldincludegraphics}{\includegraphics}%
86 \LetLtxMacro{\includegraphics}{\tikzscale@includegraphics}%

```

Deactivate the new `includegraphics` command inside of `tikzpicture`s, as a `tikzpicture` might load a PNG graphic or something and this should not be scaled by `tikzscale` but by `TikZ` or `PGFPlots`. Besides, the current implementation is not reentrant, so its not a good idea to call the macro recursively. The deactivation must be inside of `tikzpicture`, as a `tikzpicture` can be loaded without using `includegraphics`, thus it cannot be done there.

```

87 \pretocmd{\tikzpicture}{\LetLtxMacro{\includegraphics}{\tikzscale@oldincludegraphics}}{\Pack

```

```

88 \apptocmd{\endtikzpicture}{\LetLtxMacro{\includegraphics}{\tikzscale@includegraphics}}{\Pack
89 }

```

`\includegraphics`

```

90 \NewDocumentCommand{\tikzscale@includegraphics}{0}{m}{%

```

This command uses an empty optional argument for compatibility with the traditional `graphicx` command. Start a group, so that changed variables during processing the current `tikzpicture` due not influence other `tikzpictures`. This is much more convenient, than resetting every single variable. Use `\begingroup` instead of `\bgroup` to simplify finding unmatched braces.

```

91 \begingroup

```

It happened at least once together with externalization, that the deactivation of the new `includegraphics` command did not work, so do it again to be safe (maybe reentrance problem with multiple `tikzpicture` calls?).

```

92 \LetLtxMacro{\includegraphics}{\tikzscale@oldincludegraphics}%

```

There is a leading space character introduced by the externalization library, if the file is input directly. Thus use a trick to avoid that space. Furthermore, TikZ introduces with a specific version an trailing space character. To get rid of all space character issues, just solve the problem here once an for all.

```

93 \edef\tikzscale@restoreEndLineChar{\endlinechar=\the\endlinechar\relax}%
94 \endlinechar=-1%

```

Find the exact file name, as the ending and the path could be omitted.

```

95 \tikzscale@findExactFileName{tikzscale@fileName}{#2}%

```

Check if the found file is a TikZ file.

```

96 \tikzscale@isTikzFile{tikzscale@testTikzFile}{\tikzscale@fileName}%
97 \ifcsdef{tikzscale@testTikzFile}{%
98 \tikzscale@includetikz[#1]{\tikzscale@fileName}%
99 }{%
100 \tikzscale@oldincludegraphics[#1]{\tikzscale@fileName}%
101 }%
102 \tikzscale@restoreEndLineChar
103 \endgroup
104 }%

```

`\tikzscale@findExactFileName` Find the exact file name of a graphic file by testing several paths and file endings if there are degrees of freedom. The file name is saved in the command sequence name given by the first argument.

```

105 \NewDocumentCommand{\tikzscale@findExactFileName}{mm}{%

```

Delete the return variable if it already exists to allow checking if a file has been found.

```

106 \csundef{#1}%

```

Create a helper function used inside the file ending evaluation.

```

107 \def\tikzscale@checkDirectory##1{%
108 \def\tikzscale@checkExtension####1{%

```

```

109 \IfFileExists{##1#2####1}{%
110 \csdef{#1}{##1#2####1}%
111 \listbreak
112 }{}%
113 }%

```

Test all possible file extensions and do not forget that the extension might already be given. \Gin@extensions returns the **current content** set by \DeclareGraphicsExtensions.

```

114 \eforcsvlist{\tikzscale@checkExtension}{{},\Gin@extensions}%
115 }%
116 \eforcsvlist{\tikzscale@checkDirectory}{\tikzscale@graphicspath}%

```

If no file has been found, return the given file name, as includegraphics should try its best.

```

117 \ifcsundef{#1}{%
118 \csdef{#1}{#2}%
119 }{}%
120 }

```

**\tikzscale@findExactFileName** The first argument is the macro name (without backslash), which gets defined if the file is a tikzfile. The second argument is the file name.

```

121 \NewDocumentCommand{\tikzscale@isTikzFile}{mm}{%

```

Create a helper function used inside the evaluation.

```

122 \def\do##1{%
123 \IfEndWith{#2}{##1}{%
124 \csdef{#1}{}%
125 \listbreak
126 }{}%
127 }%

```

Delete macro so that defining it is really indicating something.

```

128 \csundef{#1}%
129 \edocsvlist{\tikzscale@tikzFileExtensions}%
130 }

```

**\pgfkeys** This is **similarly** done.

```

131 \pgfkeys{
132 /tikzscale/.is family, /tikzscale,
133 width/.code = {\pgfmathsetmacro{\requestedWidth}{#1}},
134 width/.value required,
135 height/.code = {\pgfmathsetmacro{\requestedHeight}{#1}},
136 height/.value required,
137 axisratio/.code = {\pgfmathsetmacro{\requestedAxisRatio}{#1}},
138 axisratio/.value required
139 }

```

```

\tikzscale@includetikz \tikzscale@includetikz{\filename}
\tikzscale@includetikz[\width=1cm]{\filename}
\tikzscale@includetikz[\height=1cm]{\filename}
\tikzscale@includetikz[\height=1cm,width=1cm]{\filename}

```

`\tikzscale@includetikz[(width=1cm,height=1cm)]{(filename)}`

This command allows the inclusion of a tikz file like a graphics file. Thus instead of writing `\includegraphics[width=\linewidth]fileWithoutEnding` write `\tikzscale@includetikz[width=\linewidth]fileWithoutEnding`. If only one of width or height are given, scale proportionally to fulfill the requirement. If both are given, scale non-proportionally to required width and height. Therefore, for normal tikzpictures only give either width or height, as the aspect ratio is already determined by the coordinate limits in the tikzpicture, but give width and height for PGFPlots, as the aspect ratio is unknown for these plots. `\NewEnviron` could be used to handle something like `verbose` in a tikzpicture, but at the moment, this is unsupported.

```
140 \NewDocumentCommand{\tikzscale@includetikz}{0{m}}{%
```

Check the keys here already, as they are needed both to see if already externalized files fulfill their requirements and to handle unexternalized files.

```
141 \pgfkeys{/tikzscale, #1}%
```

Draw and scale the graphic if it can or should not be loaded from the externalized file.

```
142 \tikzscale@ifDrawAndScale{%
```

```
143 \tikzscale@includetikzUnexternalized[#1]{#2}%
```

```
144 }{%
```

Try to load a dpth file to get the sizes `pgfexternalwidth` and `pgfexternalheight` of the externalized graphic

```
145 \ifboolexpr{test {\ifdef{\tikzexternalgetnextfilename}}}{%
```

```
146 \tikzexternalgetnextfilename{\tikzscale@externalizationName}%
```

```
147 \pgfexternalreaddpth{\tikzscale@externalizationName}%
```

Check if the sizes are still correct, i.e. agree with the sizes of the externalized PDF graphic. The axis ratio does not have to be checked, as it is always correct. This is because if it gets changed, the width and the height get changed, too, and one of these changes gets detected.

```
148 \ifdef{\requestedWidth}{%
```

```
149 \ifdef{\pgfexternalwidth}{%
```

```
150 \tikzscale@ifSizeDifference{\requestedWidth - \pgfexternalwidth}{%
```

```
151 \tikzset{external/remake next}%
```

```
152 % \PackageWarning{tikzscale}{Regenerate \tikzscale@externalizationName \MessageBreak because
```

```
153 }{}%
```

```
154 }{%
```

```
155 \tikzset{external/remake next}%
```

```
156 % \PackageWarning{tikzscale}{Regenerate \tikzscale@externalizationName \MessageBreak because
```

```
157 }%
```

```
158 }{}%
```

```
159 \ifdef{\requestedHeight}{%
```

```
160 \ifdef{\pgfexternalheight}{%
```

```
161 \tikzscale@ifSizeDifference{\requestedHeight - \pgfexternalheight}{%
```

```
162 \tikzset{external/remake next}%
```

```
163 % \PackageWarning{tikzscale}{Regenerate \tikzscale@externalizationName \MessageBreak because
```

```
164 }{}%
```

```

165 }{%
166 \tikzset{external/remake next}%
167 % \PackageWarning{tikzscale}{Regenerate \tikzscale@externalizationName \MessageBreak because
168 }%
169 }{%
170 \ifdef{\requestedAxisRatio}{%
171 \ifdef{\tikzscale@oldAxisRatio}{%
172 \tikzscale@ifSizeDifference{\requestedAxisRatio - \tikzscale@oldAxisRatio}{%
173 \tikzset{external/remake next}%
174 % \PackageWarning{tikzscale}{Regenerate \tikzscale@externalizationName \MessageBreak because
175 }{%
176 \global\undef{\tikzscale@oldAxisRatio}%
177 }{%
178 \tikzset{external/remake next}%
179 % \PackageWarning{tikzscale}{Regenerate \tikzscale@externalizationName \MessageBreak because
180 }%
181 }{%
182 }{%
183 \tikzset{external/remake next}%
184 \PackageWarning{tikzscale}{Generate \tikzscale@externalizationName \MessageBreak because no fi
185 }%
186 \input{#2}%
187 }%
188 }

```

scale@includetikzUnexternalized

```

189 \NewDocumentCommand{\tikzscale@includetikzUnexternalized}{0{m}}{%
190 \elseif{test {\ifundef{\requestedWidth}} and test {\ifundef{\requestedHeight}} and test {\ifun

```

If no option is given, directly load the content, as nothing should get scaled.

```

191 \input{#2}%
192 }{test {\ifdef{\requestedWidth}} and test {\ifdef{\requestedHeight}}}{%

```

If width and height are given, the content must be a pgfplot, so scale it. The plot currently only had approximately the given size without calling the `resizeTo` macro, due to a (known) bug in PGFPlots.

```

193 \tikzscale@resizePlotTo{#2}%
194 }{test {\ifdef{\requestedAxisRatio}}}{%
195 \tikzscale@includeAxisRatio{#2}%
196 }{test {\ifundef{\requestedAxisRatio}}}{%

```

If only either width or height is given it can be a normal `tikzpicture` or a plot with `axisratio=1`. Let's guess that it is a plot with default `axisratio`. If the guess is wrong, the called function detects that scaling the plot does not work and automatically calls `\tikzscale@includeNormalTikzpicture`.

```

197 \def\requestedAxisRatio{1}%
198 \tikzscale@includeAxisRatio{#2}%
199 }{%
200 % Everything else results in an error.
201 \tikzscale@invalidKeyError{#2}%

```

```

202 }%
203 }

\tikzscale@preparePlot
204 \NewDocumentCommand{\tikzscale@preparePlot}{*}{%
Set a scaling factor or a width and height for the plot, which will be loaded. The
\tikzset and \pgfplotsset commands have local scope. The internal redefini-
tion of the style is correct, because if one tikzpicture includes another one, the
scaling factor is reset so that it does not get squared in the inner one. Note that
if a user-defined style thus is ignored in this special case. The styles are defined
here, so that files which are inputted without the \ncludegraphics command are
not affected.
205 \pgfplotsset{every axis/.append style={width=\tikzscale@width,height=\tikzscale@height,every a
206 }
207 \NewDocumentCommand{\tikzscale@prepareTikzpicture}{*}{%
208 \tikzset{every picture/.style={scale=\tikzscale@scale,every picture/.style={}}}%
209 }

\tikzscale@includeNormalTikzpicture \tikzscale@includeNormalTikzpicture{\file name}
210 \NewDocumentCommand{\tikzscale@includeNormalTikzpicture}{m}{%
211 \tikzscale@prepareTikzpicture
212 \elseif{test {\ifdef{\requestedWidth}} and test {\ifundef{\requestedHeight}}}{%
213 \def\requestedSize{\requestedWidth}%
214 \tikzscale@scaleTikzpictureTo{\wd}{\input{#1}}{#1}%
215 }{test {\ifundef{\requestedWidth}} and test {\ifdef{\requestedHeight}}}{%
216 \def\requestedSize{\requestedHeight}%
217 \tikzscale@scaleTikzpictureTo{\ht}{\input{#1}}{#1}%
218 }{%
219 \tikzscale@invalidKeyError{#1}%
220 }%
221 }

\tikzscale@invalidKeyError
222 \NewDocumentCommand{\tikzscale@invalidKeyError}{m}{%
223 \PackageError{tikzscale}{Invalid key for TikZ graphic}{Change key #1 into a valid key.}%
224 }

\tikzscale@includeAxisRatio \tikzscale@includeAxisRatio{\file name}
225 \NewDocumentCommand{\tikzscale@includeAxisRatio}{m}{%
Try to set initial sizes close to the requested sizes, to improve the optimization's
speed.
226 \pgfplotsset{every axis/.append style={scale only axis,every axis/.style={}}}%
227 \elseif{test {\ifdef{\requestedWidth}} and test {\ifundef{\requestedHeight}}}{%
228 \let\requestedSize\requestedWidth
229 \def\tikzscale@width{\requestedWidth}%
230 \pgfmathsetmacro{\tikzscale@height}{\requestedWidth / \requestedAxisRatio}%
231 \tikzscale@resizePlotWithAxesRatioTo{\wd}{\tikzscale@width}{\input{#1}}{#1}%

```

```

232 }{test {\ifundef{\requestedWidth}} and test {\ifdef{\requestedHeight}}}{%
233 \let\requestedSize\requestedHeight
234 \def\tikzscale@height{\requestedHeight}%
235 \pgfmathsetmacro{\tikzscale@width}{\requestedHeight * \requestedAxisRatio}%
236 \tikzscale@resizePlotWithAxesRatioTo{\ht}{\tikzscale@height}{\input{#1}}{#1}%
237 }{%
238 \tikzscale@invalidKeyError{#1}%
239 }%
240 }

```

`\tikzscale@scaleTikzpictureTo` `\scalteTo{<\wd or \ht>}{<to-be-scaled content>}{<file name>}` The first argument determines if a specific width or a specific height should be achieved by scaling.

```

241 \NewDocumentCommand{\tikzscale@scaleTikzpictureTo}{mmm}{%

```

Deactivate the externalization, as the measurements to determine the correct size should not be externalized.

```

242 \tikzscale@conditionalDisableExternalization

```

When scaling a tikzpicture, only the drawings are scaled, but nodes are not scaled. So in general, there are horizontal or vertical areas, where the picture contains only unscaled nodes, and areas where the picture contains scalable drawings. Mathematically all scaled and all unscaled areas can be combined, so that there is one area of fixed size and one variable sized area. Thus scaling only by multiplication of a factor is incorrect in general. To do the correct scaling, the fixed area size must be known. As there are two unknown parameters, i.e. fixed area size and variable area size, the fixed area size can be calculated by measuring the tikzpicture with two different scalings. A special scaling factor is used, to get the size close to the final size minimizing numerical and logical errors.

```

243 \def\tikzscale@scale{1}%
244 \tikzscale@measureSize{\measuredFirst}{#1}{#2}%
245 \pgfmathsetmacro{\tikzscale@scale}{\requestedSize/\measuredFirst}%
246 \tikzscale@measureSize{\measuredSecond}{#1}{#2}%

```

It can happen, that there are no variable areas. Furthermore, the original size could already fit. Avoid numerical problems in both cases by directly drawing the picture. Do not compare the float values directly, as TeX's precision is quite limited.

```

247 \tikzscale@ifSizeDifference{\measuredSecond - \requestedSize}{%

```

If a plot is not scalable (e.g. consisting of a node only), but is not correctly scaled, exit with an error.

```

248 \tikzscale@ifSizeDifference{\measuredFirst - \measuredSecond}{%
249 }{%

```

```

250 \PackageError{tikzscale}{Requested to scale unscalable graphic}{Do not set width or height for
251 }%

```

We know, that the variable sized area scales with the scaling factor, thus it holds  $\text{\scale} * \text{\variableFirst} = \text{\variableSecond}$ , with  $\text{\variableFirst}$

$= \text{\texttt{\textbackslash measuredFirst}} - \text{\texttt{\textbackslash fixedSize}}$  and  $\text{\texttt{\textbackslash variableSecond}} = \text{\texttt{\textbackslash measuredSecond}} - \text{\texttt{\textbackslash fixedSize}}$ , which can be solved by substitution and results in

```
252 \pgfmathsetmacro{\fixedSize}{(\tikzscale@scale*\measuredFirst - \measuredSecond) / (\tikzscale@scale - 1)}
```

Now, to get the correct scaling factor, only take the variable areas into account, as it holds  $\text{\texttt{\textbackslash scaleFinal}} = \text{\texttt{\textbackslash variableSizeFinal}} / \text{\texttt{\textbackslash variableSizeOriginal}}$  with  $\text{\texttt{\textbackslash variableSizeFinal}} = \text{\texttt{\textbackslash requestedSize}} - \text{\texttt{\textbackslash fixedSize}}$  and  $\text{\texttt{\textbackslash variableSizeOriginal}} = \text{\texttt{\textbackslash measuredFirst}} - \text{\texttt{\textbackslash fixedSize}}$ , which results in

```
253 \pgfmathsetmacro{\tikzscale@scale}{(\requestedSize - \fixedSize) / (\measuredFirst - \fixedSize)}
```

Additionally or alternatively the brute force approach to iteratively improve the solution can be used.

```
254 \foreach \l in {1,...,\maxTestIterations}{%
255 \tikzscale@measureSize{\measuredIntermediate}{\l}{\l}%
```

Optimize until the absolute difference is small enough, although the (relative) size ratios are used to calculate a new scaling factor.

```
256 \tikzscale@ifSizeDifference{\measuredIntermediate-\requestedSize}{%
```

First divide before multiply to avoid overflowing (at 16384).

```
257 \pgfmathparsemacro{\errorRatio}{\measuredIntermediate/\requestedSize}%
258 \pgfmathsetglobalmacro{\tikzscale@scale}{\tikzscale@scale/\errorRatio}%
259 }{%
260 \breakforeach%
261 }%
262 }%
```

Externalize the graphic with the final size.

```
263 \tikzscale@conditionalEnableExternalization{\l}{\l}%
```

Finally, include the picture. Do it via a new measurement to be able to warn if it does not fit good enough.

```
264 \tikzscale@measureSize{\measuredFinal}{\l}{\l}%
265 \usebox{\tikzscale@measuredSize}%
266 \tikzscale@warnIfSizeDifference{\measuredFinal}{\requestedSize}{\l}%
267 }{%
```

Externalize the graphic with the final size.

```
268 \tikzscale@conditionalEnableExternalization{\l}{\l}%
269 #2%
270 }%
271 }
```

```
\tikzscale@resizePlotTo \tikzscale@resizePlotTo{\file name}
```

```
272 \NewDocumentCommand{\tikzscale@resizePlotTo}{m}{%
273 \def\fileName{\l}%
274 \def\content{\input{\l}}%
275 \tikzscale@preparePlot
276 \def\tikzscale@width{\requestedWidth}%
277 \def\tikzscale@height{\requestedHeight}%
```

Deactivate the externalization, as the measurements to determine the correct size should not be externalized.

```
278 \tikzscale@conditionalDisableExternalization
```

Improve the solution iteratively until it is good enough.

```
279 \foreach \l in {1,...,\maxTestIterations}{%
```

Using the box allows measuring the width and height with one rendering run.

```
280 \sbox{\tikzscale@measuredSize}{\content}%
```

Determine the remaining error and check if it is larger than a threshold.

```
281 \pgfmathsetmacro{\widthDifference}{\wd\tikzscale@measuredSize - \requestedWidth}%
```

```
282 \pgfmathsetmacro{\heightDifference}{\ht\tikzscale@measuredSize - \requestedHeight}%
```

Output error in current iteration for debugging.

```
283 % widthDifference: \widthDifference, heightDifference: \heightDifference\\% Debugging
```

Check if the remaining error is larger than a threshold.

```
284 \ifboolexpr{test {\tikzscale@ifSizeDifference{\widthDifference}} or test {\tikzscale@ifSizeDif
```

Correct the dimension by the error. Use a global assignment, as each iteration in the loop is put into a separate group.

```
285 \pgfmathsetglobalmacro{\tikzscale@width}{\tikzscale@width - \widthDifference}%
```

```
286 \pgfmathsetglobalmacro{\tikzscale@height}{\tikzscale@height - \heightDifference}%
```

```
287 }{%
```

```
288 \breakforeach
```

```
289 }%
```

```
290 }%
```

Externalize the graphic with the final size.

```
291 \tikzscale@conditionalEnableExternalization{\fileName}%
```

Finally, include the picture. Do it via a new measurement to be able to warn if it does not fit good enough.

```
292 \sbox{\tikzscale@measuredSize}{\content}%
```

```
293 \usebox{\tikzscale@measuredSize}%
```

```
294 \tikzscale@warnIfSizeDifference{\requestedWidth}{\wd\tikzscale@measuredSize}{\fileName's width
```

```
295 \tikzscale@warnIfSizeDifference{\requestedHeight}{\ht\tikzscale@measuredSize}{\fileName's heigh
```

```
296 }
```

```
scale@resizePlotWithAxesRatioTo \tikzscale@resizePlotWithAxesRatioTo{(\wd or \ht)}{(\tikzscale@width or  
 \tikzscale@height)}{(\to-be-scaled content)}{(\file name)} The first argument de-  
termines if a specific width or a specific height should be achieved by resizing.
```

```
297 \NewDocumentCommand{\tikzscale@resizePlotWithAxesRatioTo}{mmm}{%
```

```
298 \def\dimension{#1}%
```

```
299 \def\variable{#2}%
```

```
300 \def\content{#3}%
```

```
301 \def\fileName{#4}%
```

```
302 \gdef\tikzscale@oldSizeDifference{0pt}%
```

```
303 \tikzscale@preparePlot
```

Deactivate the externalization, as the measurements to determine the correct size should not be externalized.

```
304 \tikzscale@conditionalDisableExternalization
```

Improve the solution iteratively until it is good enough.

```
305 \foreach \l in {1,...,\maxTestIterations}{%
```

```
306 \tikzscale@measureSize{\measuredSize}{\dimension}{\content}%
```

Determine the remaining error and check if it is larger than a threshold.

```
307 \pgfmathsetmacro{\sizeDifference}{\measuredSize - \requestedSize}%
```

Output error in current iteration for debugging.

```
308 % sizeDifference: \sizeDifference\\% Debugging
```

Optimize if the absolute difference is too large.

```
309 \tikzscale@ifSizeDifference{\sizeDifference}{%
```

```
310 \ifdefstring{\dimension}{\wd}{%
```

```
311 \pgfmathsetglobalmacro{\tikzscale@width}{\tikzscale@width - \sizeDifference}%
```

```
312 \pgfmathsetglobalmacro{\tikzscale@height}{\tikzscale@width / \requestedAxisRatio}%
```

```
313 }{%
```

```
314 \pgfmathsetglobalmacro{\tikzscale@height}{\tikzscale@height - \sizeDifference}%
```

```
315 \pgfmathsetglobalmacro{\tikzscale@width}{\tikzscale@height * \requestedAxisRatio}%
```

```
316 }%
```

```
317 \tikzscale@ifSizeDifference{\sizeDifference-\tikzscale@oldSizeDifference}{%
```

```
318 }{%
```

```
319 \tikzscale@includeNormalTikzpicture{#4}%
```

```
320 \gdef\tikzscale@alreadyIncluded{true}%
```

```
321 \breakforeach
```

```
322 }%
```

```
323 \pgfmathsetglobalmacro{\tikzscale@oldSizeDifference}{\sizeDifference}%
```

```
324 }{%
```

```
325 \breakforeach
```

```
326 }%
```

```
327 }%
```

```
328 \ifdef{\tikzscale@alreadyIncluded}{%
```

```
329 \global\undef\tikzscale@alreadyIncluded%
```

```
330 }{%
```

Externalize the graphic with the final size.

```
331 \tikzscale@conditionalEnableExternalization{\fileName}%
```

Finally, include the picture. Do it via a new measurement to be able to warn if it does not fit good enough.

```
332 \tikzscale@measureSize{\measuredFinal}{\dimension}{\content}%
```

```
333 \usebox{\tikzscale@measuredSize}%
```

```
334 \tikzscale@warnIfSizeDifference{\measuredFinal}{\requestedSize}{\fileName}%
```

```
335 }%
```

```
336 }
```

```
\tikzscale@measuredSize
```

```
337 \newsavebox{\tikzscale@measuredSize}
```

```

\measureSize{<result variable name>}{<\wd or \ht>}{<to-be-measured content>}
338 \def\tikzscale@measureSize#1#2#3{%
339 \sbox{\tikzscale@measuredSize}{#3}%
340 \pgfmithsetmacro{#1}{#2\tikzscale@measuredSize}%
341 }

\tikzscale@ifSizeDifference \tikzscale@ifSizeDifference{<size>}{<executed if true>}{<executed if false>}
342 \def\tikzscale@ifSizeDifference#1#2#3{%
343 \pgfmithparse{abs(#1)}%
344 \ifdimgreater{\pgfmithresult pt}{\tikzscale@accuracy}{%
345 #2%
346 }{%
347 #3%
348 }%
349 }%

\tikzscale@measuredSize \tikzscale@warnIfSizeDifference{<firstSize>}{<secondSize>}{<file name>}
350 \def\tikzscale@warnIfSizeDifference#1#2#3{%
351 \tikzscale@ifSizeDifference{#1-#2}{%
352 \PackageWarning{tikzscale}{Scaling of #3 was only\MessageBreak accurate to \pgfmithresult pt}%
353 }{%
354 }

conditionalDisableExternalization
355 \NewDocumentCommand{\tikzscale@conditionalDisableExternalization}{}{%
356 \tikzexternaldisable
357 }

conditionalEnableExternalization Activate externalization of TikZ graphics iff it had been active before definitely
disabling it for measurement purposes. The argument contains the file name.
358 \NewDocumentCommand{\tikzscale@conditionalEnableExternalization}{m}{%
For the externalization, set correct file name and only externalize the graphic with
the final size. This produces a known bug
359 % \tikzsetnextfilename{#1}%
Get the current directory as a string and use it as an prefix, so that the graphic's
PDF is generated in a subdirectory if the tikz file is located in a subdirectory, too.
This is necessary, as the PDF file is searched for in the subdirectory in this case.
This might be unnecessary due to the newly created path lookup logic.
360 % \expandafter\tikzsetexternalprefix\expandafter{\tikzscale@pwd}%
361 % \expandnext{\tikzsetexternalprefix}{\tikzscale@pwd}%
362 \tikzexternalenable
363 }

```