Manual for Package PGFPLOTS

Version 1.3.1

http://sourceforge.net/projects/pgfplots

Christian Feuersänger*
Institut für Numerische Simulation
Universität Bonn, Germany

March 31, 2010

Abstract

PGFPLOTS draws high–quality function plots in normal or logarithmic scaling with a user-friendly interface directly in TEX. The user supplies axis labels, legend entries and the plot coordinates for one or more plots and PGFPLOTS applies axis scaling, computes any logarithms and axis ticks and draws the plots, supporting line plots, scatter plots, piecewise constant plots, bar plots, area plots, mesh– and surface plots and some more. It is based on Till Tantau's package PGF/TikZ.

Contents

[*] `http://wissrech.ins.uni-bonn.de/people/feuersaenger`

1 Introduction

This package provides tools to generate plots and labeled axes easily. It draws normal plots, logplots and semi-logplots, in two and three dimensions. Axis ticks, labels, legends (in case of multiple plots) can be added with key-value options. It can cycle through a set of predefined line/marker/color specifications. In summary, its purpose is to simplify the generation of high-quality function and/or data plots, and solving the problems of

- consistency of document and font type and font size,
- direct use of TEX math mode in axis descriptions,
- consistency of data and figures (no third party tool necessary),
- inter document consistency using preamble configurations and styles.

Although not necessary, separate `.pdf` or `.eps` graphics can be generated using the `external` library developed as part of Ti*k*Z.

PGFPLOTS is build completely on Ti*k*Z/PGF. Knowledge of Ti*k*Z will simplify the work with PGFPLOTS, although it is not required.

2 About PGFPLOTS: Preliminaries

This section contains information about upgrades, the team, the installation (in case you need to do it manually) and troubleshooting. You may skip it completely except for the upgrade remarks.

However, note that this library requires at least PGF version 2.00. At the time of this writing, many TEX-distributions still contain the older PGF version 1.18, so it may be necessary to install a recent PGF prior to using PGFPLOTS.

2.1 Components

PGFPLOTS comes with two components:

1. the plotting component (which you are currently reading) and
2. the PGFPLOTSTABLE component which simplifies number formatting and postprocessing of numerical tables. It comes as a separate package and has its own manual pgfplotstable.pdf.

2.2 Upgrade remarks

This release provides a lot of improvements which can be found in all detail in `ChangeLog` for interested readers. However, some attention is useful with respect to the following changes.

2.2.1 New Optional Features

1. pgfplots 1.3 comes with user interface improvements. The technical distinction between “behavior options” and “style options” of older versions is no longer necessary (although still fully supported).

2. pgfplots 1.3 has a new feature which allows to *move axis labels tight to tick labels* automatically.

   Since this affects the spacing, it is not enabled be default.

   Use

   ```
   \usepackage{pgfplots}
   \pgfplotsset{compat=1.3}
   ```

   in your preamble to benefit from the improved spacing[1]. Take a look at the next page for the precise definition of `compat`.

3. pgfplots 1.3 now supports reversed axes. It is no longer necessary to use work–arounds with negative units.

   Take a look at the `x dir=reverse` key.

   Existing work–arounds will still function properly. Use `\pgfplotsset{compat=1.3}` together with `x dir=reverse` to switch to the new version.

2.2.2 Old Features Which May Need Attention

1. The `scatter/classes` feature produces proper legends as of version 1.3. This may change the appearance of existing legends of plots with `scatter/classes`.

2. Due to a small math bug in pgf 2.00, you *can’t* use the math expression ‘`-x^2`’. It is necessary to use ‘`0-x^2`’ instead. The same holds for ‘`exp(-x^2)`’; use ‘`exp(0-x^2)`’ instead.

   This will be fixed with pgf $>$ 2.00.

3. Starting with pgfplots 1.1, `\tikzstyle` should *no longer be used* to set pgfplots options.

   Although `\tikzstyle` is still supported for some older pgfplots options, you should replace any occurance of `\tikzstyle` with `\pgfplotsset{`⟨*style name*⟩`/.style={`⟨*key-value-list*⟩`}}` or the associated `/.append style` variant. See section 4.17 for more detail.

I apologize for any inconvenience caused by these changes.

`/pgfplots/compat=1.3|pre 1.3|default|newest` (initially `default`)

The preamble configuration

```
\usepackage{pgfplots}
\pgfplotsset{compat=1.3}
```

allows to choose between backwards compatibility and most recent features.

pgfplots version 1.3 comes with new features which may lead to different spacings compared to earlier versions:

1. Version 1.3 comes with the `xlabel near ticks` feature which places (in this case x) axis labels “near” the tick labels, taking the size of tick labels into account.

   Older versions used `xlabel absolute`, i.e. an absolute distance between the axis and axis labels (now matter how large or small tick labels are).

   The initial setting *keeps backwards compatibility.*

   You are encouraged to use

   ```
   \usepackage{pgfplots}
   \pgfplotsset{compat=1.3}
   ```

   in your preamble.

[1] The `compat=1.3` setting is necessary for new features which move axis labels around like reversed axis. However, pgfplots will still work as it does in earlier versions, your documents will remain the same if you don’t set it explicitly.

2. Version 1.3 fixes a bug with `anchor=south west` which occurred mainly for reversed axes: the anchors where upside–down. This is fixed now.

   If you have written some work–around and want to keep it going, use

   ```
   \pgfplotsset{compat/anchors=pre 1.3}
   ```

   to disable the bug fix.

Use `\pgfplotsset{compat=default}` to restore the factory settings.

The setting `\pgfplotsset{compat=newest}` will always use features of the most recent version. This might result in small changes in the document's appearance.

2.3 The Team

PGFPLOTS has been written mainly by Christian Feuersänger with many improvements of Pascal Wolkotte and Nick Papior Andersen as a spare time project. We hope it is useful and provides valuable plots.

If you are interested in writing something but don't know how, consider reading the auxiliary manual TeX-programming-notes.pdf which comes with PGFPLOTS. It is far from complete, but maybe it is a good starting point (at least for more literature).

2.4 Acknowledgements

I thank God for all hours of enjoyed programming. I thank Pascal Wolkotte and Nick Papior Andersen for their programming efforts and contributions as part of the development team. I thank Stefan Tibus, who contributed the `plot shell` feature. I thank Tom Cashman for the contribution of the `reverse legend` feature. Furthermore, I thank Dr. Schweitzer for many fruitful discussions and Dr. Meine for his ideas and suggestions.

Last but not least, I thank Till Tantau and Mark Wibrow for their excellent graphics (and more) package PGF and Ti*k*Z which is the base of PGFPLOTS.

2.5 Installation and Prerequisites

2.5.1 Licensing

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

A copy of the GNU General Public License can be found in the package file

```
doc/latex/pgfplots/gpl-3.0.txt
```

You may also visit http://www.gnu.org/licenses.

2.5.2 Prerequisites

PGFPLOTS requires PGF with **at least version** 2.0. It is used with

```
\usepackage{pgfplots}
```

in your preamble (see section 3.1 for information about how to use it with ConTEXt and plain TEX).

There are several ways how to teach TEX where to find the files. Choose the option which fits your needs best.

2.5.3 Installation in Windows

Windows users often use MikTEX which downloads the latest stable package versions automatically. You do not need to install anything manually here.

However, MikTEX provides a feature to install packages locally in its own TEX-Directory-Structure (TDS). This is the preferred way if you like to install newer version than those of MikTEX. The basic idea is to unzip PGFPLOTS in a directory of your choice and configure the MikTEX Package Manager to use this specific directory with higher priority than its default paths. If you want to do this, start the MikTEX Settings using “Start ≫ Programs ≫ MikTEX ≫ Settings”. There, use the “Roots” menu section. It contains the MikTEX Package directory as initial configuration. Use “Add” to select the directory in which the unzipped PGFPLOTS tree resides. Then, move the newly added path to the list’s top using the “Up” button. Then press “Ok”. For MikTEX 2.8, you may need to uncheck the “Show MikTEX-maintained root directories” button to see the newly installed path.

MikTEX complains if the provided directory is not TDS conform (see section 2.5.6 for details), so you can’t provide a wrong directory here. This method does also work for other packages, but some packages may need some directory restructuring before MikTEX accepts them.

2.5.4 Installation of Linux Packages

At the time of this writing, I am unaware of PGFPLOTS packages for recent stable Linux distributions. For Ubuntu, there are unofficial Ubuntu Package Repositories which can be added to the Ubuntu Package Tools. The idea is: add a simple URL to the Ubuntu Package Tool, run update and the installation takes place automatically. These URLs are maintained as PPA on Ubuntu Servers.

The PGFPLOTS download area on sourceforge contains recent links about Ubuntu Package Repositories, go to http://sourceforge.net/projects/pgfplots/files and download the readme files with recent links.

2.5.5 Installation in Any Directory - the `TEXINPUTS` Variable

You can simply install PGFPLOTS anywhere on your disc, for example into

```
/foo/bar/pgfplots.
```

Then, you set the `TEXINPUTS` variable to

```
TEXINPUTS=/foo/bar/pgfplots//:
```

The trailing ':' tells TeX to check the default search paths after `/foo/bar/pgfplots`. The double slash '`//`' tells TeX to search all subdirectories.

If the `TEXINPUTS` variable already contains something, you can append the line above to the existing `TEXINPUTS` content.

Furthermore, you should set `TEXDOCS` as well,

```
TEXDOCS=/foo/bar/pgfplots//:
```

so that the TeX-documentation system finds the files `pgfplots.pdf` and `pgfplotstable.pdf` (on some systems, it is then enough to use `texdoc pgfplots`).

Please refer to your operating systems manual for how to set environment variables.

2.5.6 Installation Into a Local TDS Compliant `texmf`-Directory

PGFPLOTS comes in a "TeX Directory Structure" (TDS) conforming directory structure, so you can simply unpack the files into a directory which is searched by TeX automatically. Such directories are `~/texmf` on Linux systems, for example.

Copy PGFPLOTS to a local `texmf` directory like `~/texmf`. You need at least the PGFPLOTS directories `tex/generic/pgfplots` and `tex/latex/pgfplots`. Then, run `texhash` (or some equivalent path–updating command specific to your TeX distribution).

The TDS consists of several sub directories which are searched separately, depending on what has been requested: the sub-directories `doc/latex/`⟨*package*⟩ are used for (LaTeX) documentation, the sub-directories `doc/generic/`⟨*package*⟩ for documentation which apply to LaTeX and other TeX dialects (like plain TeX and ConTeXt which have their own, respective sub-directories) as well.

Similarly, the `tex/latex/`⟨*package*⟩ sub-directories are searched whenever LaTeX packages are requested. The `tex/generic/`⟨*package*⟩ sub-directories are searched for packages which work for LaTeX *and* other TeX dialects.

Do not forget to run `texhash`.

2.5.7 Installation If Everything Else Fails...

If TeX still doesn't find your files, you can copy all `.sty` and all `.code.tex`-files (perhaps all `.def` files as well) into your current project's working directory. In fact, you need all which is in the `tex/latex/pgfplots` and `tex/generic/pgfplots` sub directories.

Please refer to `http://www.ctan.org/installationadvice/` for more information about package installation.

2.6 Troubleshooting – Error Messages

This section discusses some problems which may occur when using PGFPLOTS. Some of the error messages are shown in the index, take a look at the end of this manual (under "Errors").

2.6.1 Problems with available Dimen-registers

To avoid problems with the many required TeX-registers for PGF and PGFPLOTS, you may want to include

```
\usepackage{etex}
```

as first package. This avoids problems with "no room for a new dimen" in most cases. It should work with any modern installation of TeX (it activates the e-TeX extensions).

2.6.2 Dimension Too Large Errors

The core mathematical engine of PGF relies on TeX registers to perform fast arithmetics. To compute $50 + 299$, it actually computes `50pt+299pt` and strips the `pt` suffix of the result. Since TeX registers can only contain numbers up to ± 16384, overflow error messages like "Dimension too large" occur if the result leaves the allowed range. Normally, this should never happen – PGFPLOTS uses a floating point unit with data range $\pm 10^{324}$ and performs all mappings automatically. However, there are some cases where this fails. Some of these cases are:

1. The axis range (for example, for x) becomes *relatively* small. It's no matter if you have absolutely small ranges like $[10^{-17}, 10^{-16}]$. But if you have an axis range like $[1.99999999, 2]$, where a lot of significant digits are necessary, this may be problematic.
2. This may happen as well if you only view a very small portion of the data range.
3. The `axis equal` key will be confused if x and y have a very different scale.
4. You may have found a bug – please contact the developers.

2.6.3 Restrictions for DVI-Viewers and `dvipdfm`

PGF is compatible with

- `latex/dvips`,
- `latex/dvipdfm`,
- `pdflatex`,
- ⋮

However, there are some restrictions: I don't know any DVI viewer which is capable of viewing the output of PGF (and therefor PGFPLOTS as well). After all, DVI has never been designed to draw something different than text and horizontal/vertical lines. You will need to view the postscript file or the pdf-file.

Then, the DVI/pdf combination doesn't support all types of shadings (for example, the `shader=interp` is only available for `dvips` and `pdftex` drivers).

Furthermore, PGF needs to know a *driver* so that the DVI file can be converted to the desired output. Depending on your system, you need the following options:

- `latex/dvips` does not need anything special because `dvips` is the default driver if you invoke `latex`.
- `pdflatex` will also work directly because `pdflatex` will be detected automatically.
- `latex/dvipdfm` requires to use

  ```
  \def\pgfsysdriver{pgfsys-dvipdfm.def}
  %\def\pgfsysdriver{pgfsys-pdftex.def}
  %\def\pgfsysdriver{pgfsys-dvips.def}
  \usepackage{pgfplots}.
  ```

  The uncommented commands could be used to set other drivers explicitly.

Please read the corresponding sections in [5, Section 7.2.1 and 7.2.2] if you have further questions. These sections also contain limitations of particular drivers.

The choice which won't produce any problems at all is `pdflatex`.

2.6.4 Problems with TEX's Memory Capacities

PGFPLOTS can handle small up to medium sized plots. However, TEX has never been designed for data plots – you will eventually face the problem of small memory capacities. See section 6.1 for how to enlarge them.

2.6.5 Problems with Language Settings and Active Characters

Both, PGF and PGFPLOTS use a lot of characters – which may lead to incompatibilities with other packages which define active characters. Compatibility is better than in earlier versions, but may still be an issue. The manual compiles with the `babel` package for english and french, the `german` package does also work. If you experience any trouble, let me know. Sometimes it may work to disable active characters temporarily (`babel` provides such a command).

2.6.6 Other Problems

Please read the mailing list at
`http://sourceforge.net/projects/pgfplots/support`.
Perhaps someone has also encountered your problem before, and maybe he came up with a solution.

Please write a note on the mailing list if you have a different problem. In case it is necessary to contact the authors directly, consider the addresses shown on the title page of this document.

3 User's Guide: Drawing Axes and Plots

3.1 TEX-dialects: LATEX, ConTEXt, plain TEX

PGFPLOTS is compatible with LATEX, ConTEXt and plain TEX. The only difference is how to specify environments. This affects any PGF/TikZ-environments and all PGFPLOTS-environments like axis, semilogxaxis, semilogyaxis and loglogaxis:

LATEX: `\usepackage{pgfplots}` and

```
\begin{tikzpicture}
\begin{axis}
...
\end{axis}
\end{tikzpicture}
```

```
\begin{tikzpicture}
\begin{semilogxaxis}
...
\end{semilogxaxis}
\end{tikzpicture}
```

```
\documentclass[a4paper]{article}

%  for dvipdfm:
% \def\pgfsysdriver{pgfsys-dvipdfm.def}
\usepackage{pgfplots}
\pgfplotsset{compat=1.3}%  <-- moves axis labels near ticklabels (respects tick label widths)

\begin{document}
\begin{figure}
    \centering
    \begin{tikzpicture}
        \begin{loglogaxis}[xlabel=Cost,ylabel=Error]
        \addplot coordinates {
            (5,     8.31160034e-02)
            (17,    2.54685628e-02)
            (49,    7.40715288e-03)
            (129,   2.10192154e-03)
            (321,   5.87352989e-04)
            (769,   1.62269942e-04)
            (1793,  4.44248889e-05)
            (4097,  1.20714122e-05)
            (9217,  3.26101452e-06)
        };
        \addplot coordinates {
            (7,     8.47178381e-02)
            (31,    3.04409349e-02)
            (111,   1.02214539e-02)
            (351,   3.30346265e-03)
            (1023,  1.03886535e-03)
            (2815,  3.19646457e-04)
            (7423,  9.65789766e-05)
            (18943, 2.87339125e-05)
            (47103, 8.43749881e-06)
        };
        \legend{Case 1,Case 2}
        \end{loglogaxis}
    \end{tikzpicture}
    \caption{A larger example}
\end{figure}
\end{document}
```

ConTEXt: `\usemodule[pgfplots]` and

```
\starttikzpicture
\startaxis
...
\stopaxis
\stoptikzpicture
```

```
\starttikzpicture
\startsemilogxaxis
...
\stopsemilogxaxis
\stoptikzpicture
```

A complete ConTEXt–example file can be found in

```
doc/context/pgfplots/pgfplotsexample.tex.
```

plain TeX: `\input pgfplots.tex` and

```
\tikzpicture
\axis
...
\endaxis
\endtikzpicture
```

```
\tikzpicture
\semilogxaxis
...
\endsemilogxaxis
\endtikzpicture
```

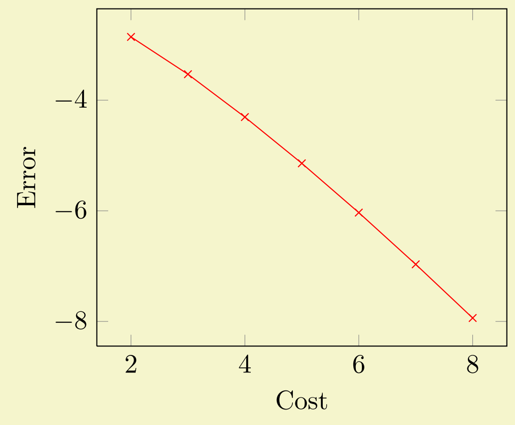
A complete plain–TeX–example file can be found in

```
doc/plain/pgfplots/pgfplotsexample.tex.
```

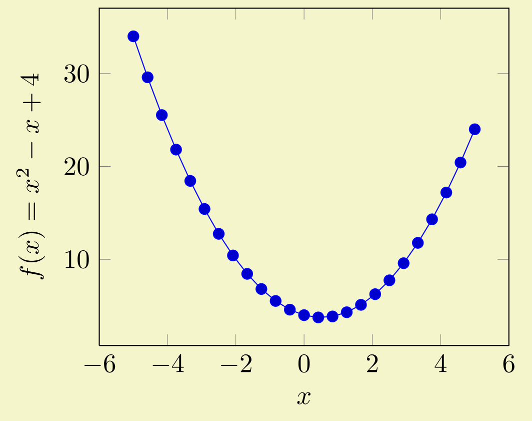
The default system drivers for `dvips` and `pdftex` work without any additional work; for `dvipdfm`, the `pgfsysdriver` macro needs to be redefined manually (see also section 2.6.3).

3.2 A First Plot

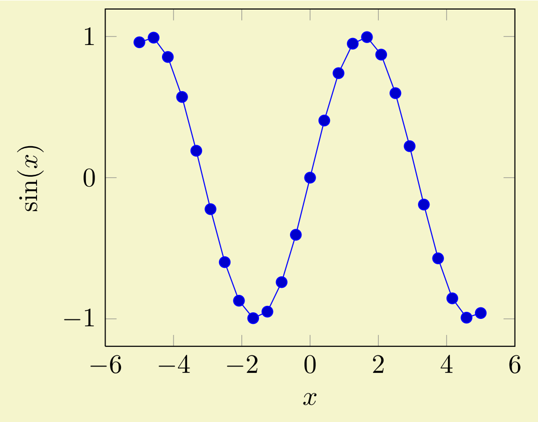
Plotting is done using `\begin{axis} ... \addplot ...; \end{axis}`, where `\addplot` is the main interface to perform plotting operations.



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{axis}[
        xlabel=Cost,
        ylabel=Error]
    \addplot[color=red,mark=x] coordinates {
        (2,-2.8559703)
        (3,-3.5301677)
        (4,-4.3050655)
        (5,-5.1413136)
        (6,-6.0322865)
        (7,-6.9675052)
        (8,-7.9377747)
    };
    \end{axis}
\end{tikzpicture}
```



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{axis}[
        xlabel=$x$,
        ylabel={$f(x) = x^2 - x +4$}
    ]
    %  use TeX as calculator:
    \addplot {x^2 - x +4};
    \end{axis}
\end{tikzpicture}
```

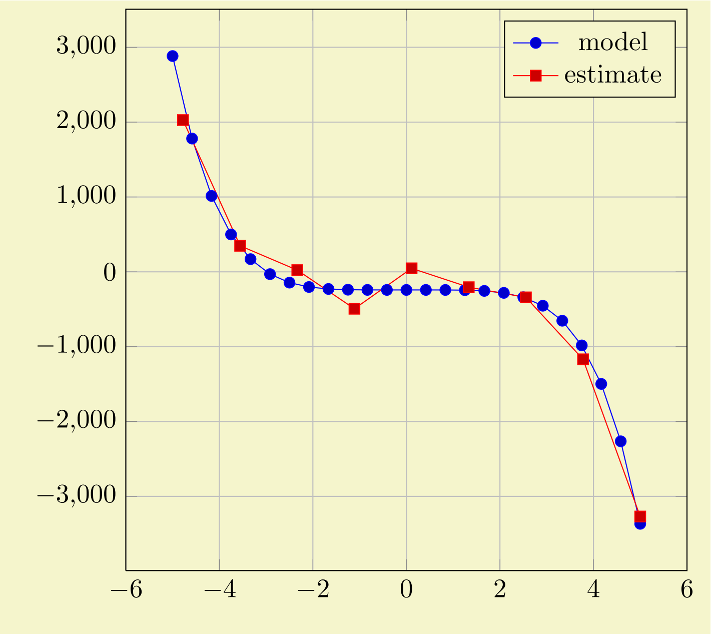


```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{axis}[
        xlabel=$x$,
        ylabel=$\sin(x)$
    ]
    %  invoke external gnuplot as
    %  calculator:
    \addplot gnuplot[id=sin]{sin(x)};
    \end{axis}
\end{tikzpicture}
```

The `plot coordinates`, `plot expression` and `plot gnuplot` commands are three of the several supported ways to create plots, see section 4.2 for more details[2] and the remaining ones (`plot file`, `plot shell`, `plot table` and `plot graphics`). The options '`xlabel`' and '`ylabel`' define axis descriptions.

3.3 Two Plots in the Same Axis

Multiple `\addplot`-commands can be placed into the same axis.



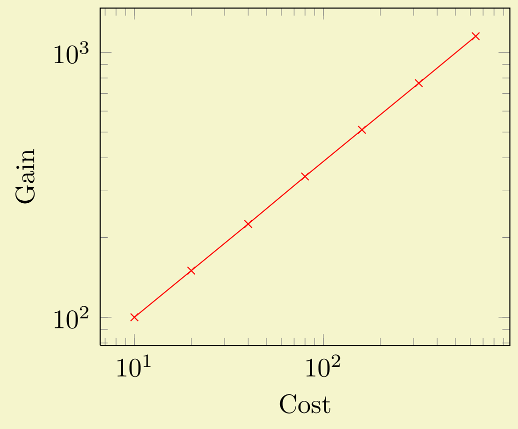
[2] Please note that you need `gnuplot` installed to use `plot gnuplot`.

```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{axis}[
        height=9cm,
        width=9cm,
        grid=major,
    ]

    \addplot gnuplot[id=filesuffix]{(-x**5 - 242)};
    \addlegendentry{model}

    \addplot coordinates {
        (-4.77778,2027.60977)
        (-3.55556,347.84069)
        (-2.33333,22.58953)
        (-1.11111,-493.50066)
        (0.11111,46.66082)
        (1.33333,-205.56286)
        (2.55556,-341.40638)
        (3.77778,-1169.24780)
        (5.00000,-3269.56775)
    };
    \addlegendentry{estimate}
    \end{axis}
\end{tikzpicture}
```

A legend entry is generated if there are `\addlegendentry` commands (or one `\legend` command).

3.4 Logarithmic Plots

Logarithmic plots show $\log x$ versus $\log y$ (or just one logarithmic axis). PGFPLOTS normally uses the natural logarithm, i.e. basis $e \approx 2.718$ (see the key `log basis x`). Now, the axis description also contains minor ticks and the labels are placed at 10^i.

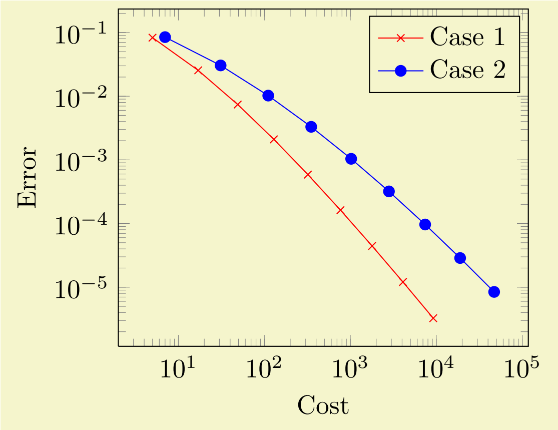


```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{loglogaxis}[xlabel=Cost,ylabel=Gain]
\addplot[color=red,mark=x] coordinates {
    (10,100)
    (20,150)
    (40,225)
    (80,340)
    (160,510)
    (320,765)
    (640,1150)
};
\end{loglogaxis}
\end{tikzpicture}
```

A common application is to visualise scientific data. This is often provided in the format $1.42 \cdot 10^4$, usually written as 1.42e+04. Suppose we have a numeric table named `pgfplots.testtable`, containing

```
Level Cost  Error
1     7     8.471e-02
2     31    3.044e-02
3     111   1.022e-02
4     351   3.303e-03
5     1023  1.038e-03
6     2815  3.196e-04
7     7423  9.657e-05
8     18943 2.873e-05
9     47103 8.437e-06
```

then we can plot `Cost` versus `Error` using



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{loglogaxis}[
    xlabel=Cost,
    ylabel=Error]
\addplot[color=red,mark=x] coordinates {
    (5,    8.31160034e-02)
    (17,   2.54685628e-02)
    (49,   7.40715288e-03)
    (129,  2.10192154e-03)
    (321,  5.87352989e-04)
    (769,  1.62269942e-04)
    (1793, 4.44248889e-05)
    (4097, 1.20714122e-05)
    (9217, 3.26101452e-06)
};

\addplot[color=blue,mark=*]
    table[x=Cost,y=Error] {pgfplots.testtable};

\legend{Case 1,Case 2}
\end{loglogaxis}
\end{tikzpicture}
```

The first plot employs inline coordinates; the second one reads numerical data from file and plots column ‘`Cost`’ versus ‘`Error`’.

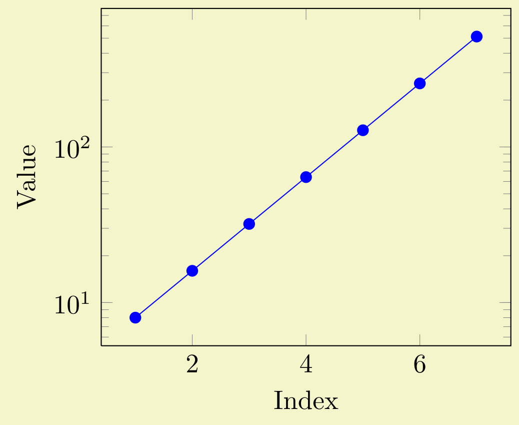
Besided the environment “`loglogaxis`” you can use

- `\begin{axis}...\end{axis}` for normal plots,
- `\begin{semilogxaxis}...\end{semilogxaxis}` for plots which have a normal y axis and a logarithmic x axis,
- `\begin{semilogyaxis}...\end{semilogyaxis}` the same with x and y switched,
- `\begin{loglogaxis}...\end{loglogaxis}` for double–logarithmic plots.

You can also use

```
\begin{axis}[xmode=normal,ymode=log]
...
\end{axis}
```

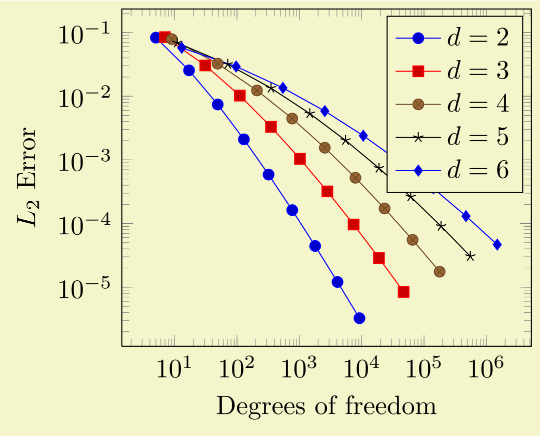
which is the same as `\begin{semilogyaxis}...\end{semilogyaxis}`.



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{semilogyaxis}[
        xlabel=Index,ylabel=Value]

    \addplot[color=blue,mark=*] coordinates {
        (1,8)
        (2,16)
        (3,32)
        (4,64)
        (5,128)
        (6,256)
        (7,512)
    };
    \end{semilogyaxis}%
\end{tikzpicture}%
```

3.5 Cycling Line Styles

You can skip the style arguments for `\addplot[...]` to determine plot specifications from a predefined list:

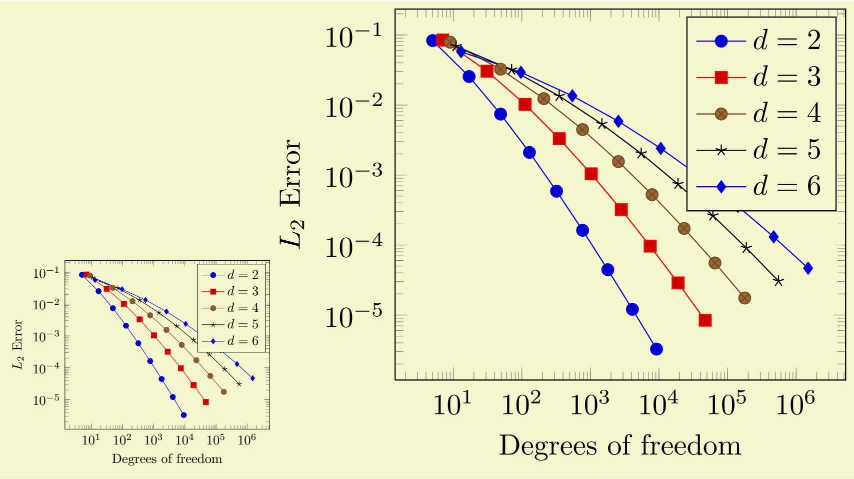


```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{loglogaxis}[
    xlabel={Degrees of freedom},
    ylabel={$L_2$ Error}
]
\addplot coordinates {
    (5,8.312e-02)    (17,2.547e-02)   (49,7.407e-03)
    (129,2.102e-03)  (321,5.874e-04)  (769,1.623e-04)
    (1793,4.442e-05) (4097,1.207e-05) (9217,3.261e-06)
};

\addplot coordinates{
    (7,8.472e-02)    (31,3.044e-02)    (111,1.022e-02)
    (351,3.303e-03)  (1023,1.039e-03)  (2815,3.196e-04)
    (7423,9.658e-05) (18943,2.873e-05) (47103,8.437e-06)
};

\addplot coordinates{
    (9,7.881e-02)     (49,3.243e-02)    (209,1.232e-02)
    (769,4.454e-03)   (2561,1.551e-03)  (7937,5.236e-04)
    (23297,1.723e-04) (65537,5.545e-05) (178177,1.751e-05)
};

\addplot coordinates{
    (11,6.887e-02)    (71,3.177e-02)     (351,1.341e-02)
    (1471,5.334e-03)  (5503,2.027e-03)   (18943,7.415e-04)
    (61183,2.628e-04) (187903,9.063e-05) (553983,3.053e-05)
};

\addplot coordinates{
    (13,5.755e-02)     (97,2.925e-02)     (545,1.351e-02)
    (2561,5.842e-03)   (10625,2.397e-03)  (40193,9.414e-04)
    (141569,3.564e-04) (471041,1.308e-04) (1496065,4.670e-05)
};
\legend{$d=2$,$d=3$,$d=4$,$d=5$,$d=6$}
\end{loglogaxis}
\end{tikzpicture}
```

The `cycle list` can be modified, see the reference below.

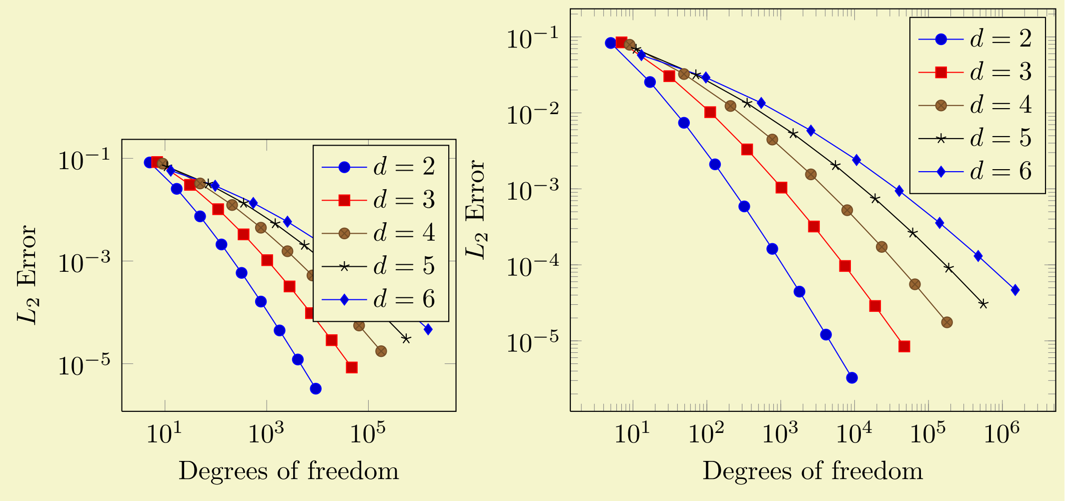
3.6 Scaling Plots

You can use any of the Ti*k*Z options to modify the appearance. For example, the "`scale`" transformation takes the picture as such and scales it.



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}[scale=0.5]
    \begin{loglogaxis}[
        xlabel={Degrees of freedom},
        ylabel={$L_2$ Error}
    ]
    \plotcoords
    \legend{$d=2$,$d=3$,$d=4$,$d=5$,$d=6$}
    \end{loglogaxis}
\end{tikzpicture}

\begin{tikzpicture}[scale=1.1]
    \begin{loglogaxis}[
        xlabel={Degrees of freedom},
        ylabel={$L_2$ Error}
    ]
    \plotcoords
    \legend{$d=2$,$d=3$,$d=4$,$d=5$,$d=6$}
    \end{loglogaxis}
\end{tikzpicture}
```

However, you can also scale plots by assigning a `width=5cm` and/or `height=3cm` argument. This only affects the distance of point coordinates, no font sizes or axis descriptions:



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{loglogaxis}[
        width=6cm,
        xlabel={Degrees of freedom},
        ylabel={$L_2$ Error}
    ]
    \plotcoords
    \legend{$d=2$,$d=3$,$d=4$,$d=5$,$d=6$}
    \end{loglogaxis}
\end{tikzpicture}

\begin{tikzpicture}
    \begin{loglogaxis}[
        width=8cm,
        xlabel={Degrees of freedom},
        ylabel={$L_2$ Error}
    ]
    \plotcoords
    \legend{$d=2$,$d=3$,$d=4$,$d=5$,$d=6$}
    \end{loglogaxis}
\end{tikzpicture}
```

Use the predefined styles `normalsize`, `small`, `footnotesize` to adopt font sizes and ticks automatically.

4 The Reference

4.1 The Axis-Environments

There is an axis environment for linear scaling, two for semi-logarithmic scaling and one for double-logarithmic scaling.

```
\begin{tikzpicture}[⟨options of tikz⟩]
  ⟨environment contents⟩
\end{tikzpicture}
```

This is the graphics environment of Ti*k*Z. It produces a single picture and encloses also every axis.

Instead of using the environment version, there is also a shortcut command

`\tikz{⟨picture content⟩}`

which can be used alternatively.

```
\begin{axis}[⟨options⟩]
  ⟨environment contents⟩
\end{axis}
```

The axis environment for normal plots with linear axis scaling.

The '`every linear axis`' style key can be modified with

```
\pgfplotsset{every linear axis/.append style={...}}
```

to install styles specifically for linear axes. These styles can contain both Ti*k*Z- and PGFPLOTS options.

```
\begin{semilogxaxis}[⟨options⟩]
  ⟨environment contents⟩
\end{semilogxaxis}
```

The axis environment for logarithmic scaling of x and normal scaling of y. Use

```
\pgfplotsset{every semilogx axis/.append style={...}}
```

to install styles specifically for the case with `xmode=log`, `ymode=normal`.

The logarithmic scaling means to apply the natural logarithm (base e) to each x coordinate. Furthermore, ticks will be typeset as $10^{\langle exponent\rangle}$, see section 4.12 for more details.

```
\begin{semilogyaxis}[⟨options⟩]
  ⟨environment contents⟩
\end{semilogyaxis}
```

The axis environment for normal scaling of x and logarithmic scaling of y,

The style '`every semilogy axis`' will be installed for each such plot.

The same remarks as for `semilogxaxis` apply here as well.

```
\begin{loglogaxis}[⟨options⟩]
  ⟨environment contents⟩
\end{loglogaxis}
```

The axis environment for logarithmic scaling of both, x and y axes, As for the other axis possibilities, there is a style '`every loglog axis`' which is installed at the environment's beginning.

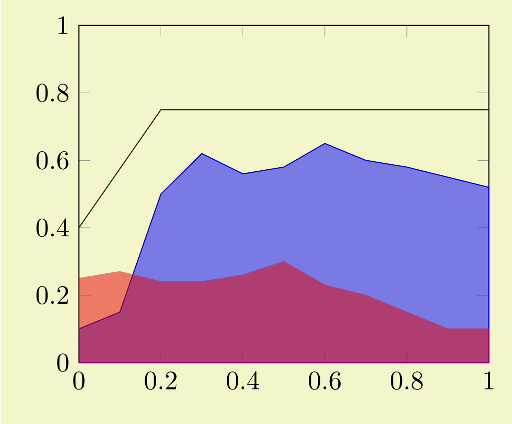
The same remarks as for `semilogxaxis` apply here as well.

They are all equivalent to

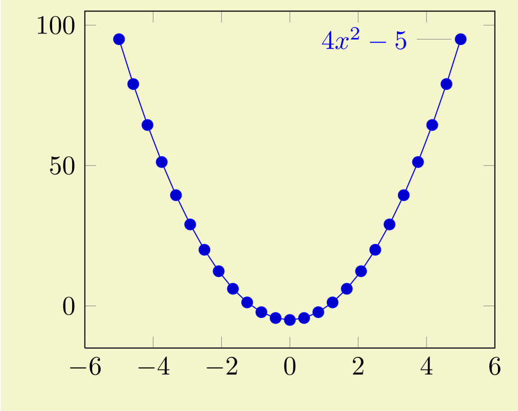
```
\begin{axis}[
    xmode=log|normal,
    ymode=log|normal]
...
\end{axis}
```

with properly set variables '`xmode`' and '`ymode`' (see below).

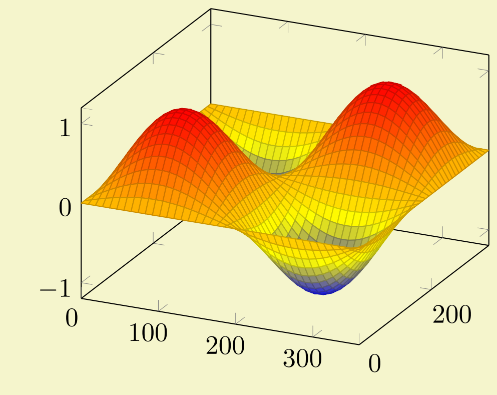
4.2 The \addplot Command: Coordinate Input



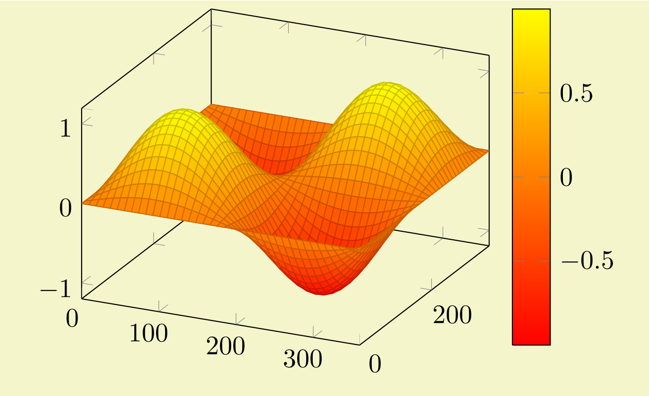
```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}[ymin=0,ymax=1,enlargelimits=false]
\addplot
    [blue!80!black,fill=blue,fill opacity=0.5]
coordinates
{(0,0.1)    (0.1,0.15)  (0.2,0.5)   (0.3,0.62)
 (0.4,0.56) (0.5,0.58)  (0.6,0.65)  (0.7,0.6)
 (0.8,0.58) (0.9,0.55)  (1,0.52)}
|- (axis cs:0,0) -- cycle;

\addplot
    [red,fill=red!90!black,opacity=0.5]
coordinates
{(0,0.25)   (0.1,0.27)  (0.2,0.24)  (0.3,0.24)
 (0.4,0.26) (0.5,0.3)   (0.6,0.23)  (0.7,0.2)
 (0.8,0.15) (0.9,0.1)   (1,0.1)}
|- (axis cs:0,0) -- cycle;

\addplot[green!20!black] coordinates
    {(0,0.4) (0.2,0.75) (1,0.75)};
\end{axis}
\end{tikzpicture}
```



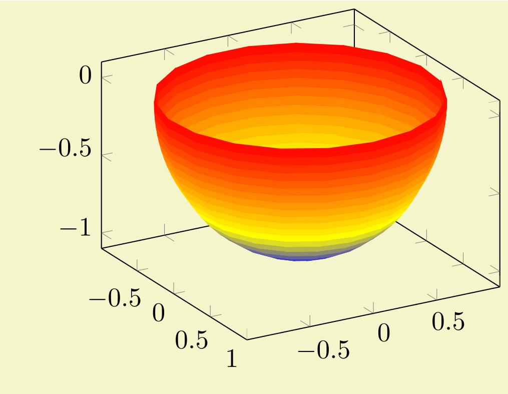
```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}
\addplot+[id=parable,domain=-5:5]
    gnuplot{4*x**2 - 5}
    node[pin=180:{$4x^2-5$}]{};
\end{axis}
\end{tikzpicture}
```



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{axis}
    \addplot3[surf,domain=0:360,samples=40]
        {sin(x)*sin(y)};
    \end{axis}
\end{tikzpicture}
```



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{axis}[colormap/redyellow,colorbar]
    \addplot3[surf,
        domain=0:360,samples=40]
        {sin(x)*sin(y)};
    \end{axis}
\end{tikzpicture}
```



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}[view={60}{30}]
    \addplot3[surf,shader=flat,
        samples=20,
        domain=-1:0,y domain=0:2*pi,
        z buffer=sort]
        ({sqrt(1-x^2) * cos(deg(y))},
         {sqrt( 1-x^2 ) * sin(deg(y))},
         x);
\end{axis}
\end{tikzpicture}
```

Inside of an axis environment, the `\addplot` command is the main user interface. It comes in two variants: `\addplot` for two–dimensional visualization and `\addplot3` for three–dimensional visualization.

`\addplot[`⟨*options*⟩`]` ⟨*input data*⟩ ⟨*trailing path commands*⟩;

This is the main plotting command, available within each axis environment. It can be used one or more times within an axis to add plots to the current axis. There is also an `\addplot3` command which is described in section 4.5.

It reads point coordinates from one of the available input sources specified by ⟨*input data*⟩, updates limits, remembers ⟨*options*⟩ for use in a legend (if any) and applies any necessary coordinate transformations (or logarithms).

The ⟨*options*⟩ can be omitted in which case the next entry from the `cycle list` will be inserted as ⟨*options*⟩. These keys characterize the plot's type like linear interpolation, smooth plot, constant interpolation, bar plot, mesh plots, surface plots or whatever and define colors, markers and line specifications[3]. Plot variants like error bars, the number of samples or a sample domain can also be configured in ⟨*options*⟩.

The ⟨*input data*⟩ is one of several coordinate input tools which are described in more detail below. Finally, if `\addplot` successfully processed all coordinates from ⟨*input data*⟩, it generates Ti*k*Z paths to realize the drawing operations. Any ⟨*trailing path commands*⟩ are appended to the final drawing command, allowing to continue the Ti*k*Z path (from the last plot coordinate).

Some more details:

[3] In version 1.2.2 and earlier, there was an explicit distinction between "behaviour" options like error bars, domain, number of samples etc. and "style options" like color, line width, markers etc. This distinction is obsolete now, simply collect everything into ⟨*options*⟩.

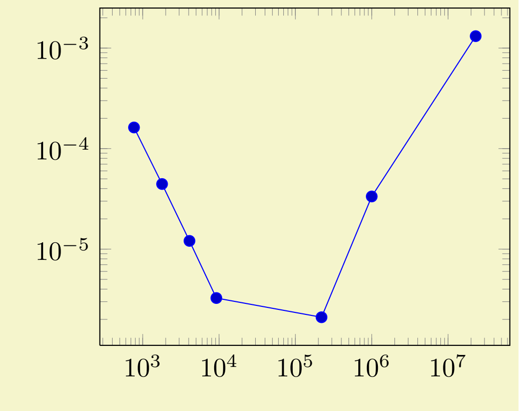
- The style /pgfplots/every axis plot will be installed at the beginning of ⟨*options*⟩. That means you can use

  ```
  \pgfplotsset{every axis plot/.append style={...}}
  ```

  to add options to all your plots - maybe to set line widths to thick. Furthermore, if you have more than one plot inside of an axis, you can also use

  ```
  \pgfplotsset{every axis plot no 3/.append style={...}}
  ```

  to modify options for the plot with number 3 only. The first plot in an axis has number 0.
- The ⟨*options*⟩ are remembered for the legend. They are available as 'current plot style' as long as the path is not yet finished or in associated error bars.
- See subsection 4.6 for a list of available markers and line styles.
- For log plots, PGFPLOTS will compute the natural logarithm $\log(\cdot)$ numerically using a floating point unit developed for this purpose[4]. For example, the following numbers are valid input to \addplot.



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{loglogaxis}
\addplot coordinates {
    (769,   1.6227e-04)
    (1793,  4.4425e-05)
    (4097,  1.2071e-05)
    (9217,  3.2610e-06)
    (2.2e5, 2.1E-6)
    (1e6,   0.00003341)
    (2.3e7, 0.00131415)
};
\end{loglogaxis}
\end{tikzpicture}
```

  You can represent arbitrarily small or very large numbers as long as its logarithm can be represented as a TEX-length (up to about 16384). Of course, any coordinate $x \leq 0$ is not possible since the logarithm of a non-positive number is not defined. Such coordinates will be skipped automatically (using the initial configuration unbounded coords=discard).
- For normal (non–logarithmic) axes, PGFPLOTS applies floating point arithmetics to support large or small numbers like 0.00000001234 or $1.234 \cdot 10^{24}$. Its number range is much larger than TEX's native support for numbers. The relative precision is between 4 and 7 significant decimal digits for the mantissa.

  As soon as the axes limits are completely known, PGFPLOTS applies a transformation which maps these floating point numbers into TEX-precision using transformations

  $$T_x(x) = 10^{s_x} \cdot x - a_x \text{ and } T_y(y) = 10^{s_y} \cdot y - a_y \text{ and (for 3D plots) } T_z(y) = 10^{s_z} \cdot z - a_z$$

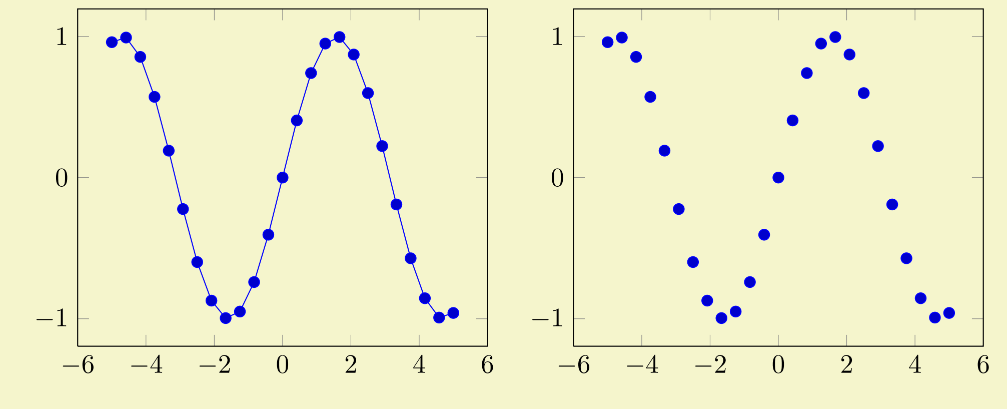
  with properly chosen integers $s_x, s_y, s_z \in \mathbb{Z}$ and shifts $a_x, a_y, a_z \in \mathbb{R}$. Section 4.22 contains a description of disabledatascaling and provides more details about the transformation.
- Some of the coordinate input routines use the powerful \pgfmathparse feature of PGF to read their coordinates, among them plot coordinates, plot expression and plot table. This allows to use mathematical expressions as coordinates which will be evaluated using the floating point routines (this applies to logarithmic and linear scales).
- If you did not specify axis limits manually, \addplot will compute them automatically. The automatic computation of axis limits works as follows:
  1. Every coordinate will be checked. Care has been taken to avoid TEX's limited numerical capabilities.
  2. Since more than one \addplot command may be used inside of \begin{axis}...\end{axis}, all drawing commands will be postponed until \end{axis}.

[4] This floating point unit is available as Ti*k*Z library as part of Ti*k*Z.

\addplot+[⟨*options*⟩] ...;

Does the same like \addplot[⟨*options*⟩] ...; except that ⟨*options*⟩ are *appended* to the arguments which would have been taken for \addplot ... (the element of the default list).

Thus, you can combine cycle list and ⟨*options*⟩.

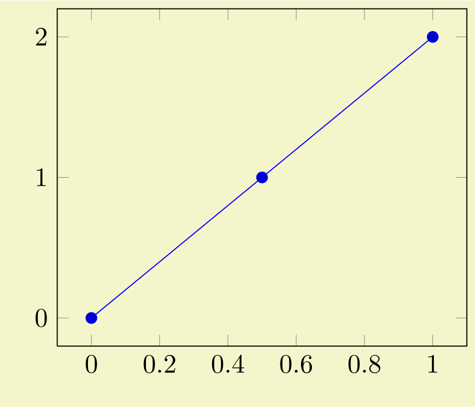


```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}
\addplot {sin(deg(x))};
\end{axis}
\end{tikzpicture}

\begin{tikzpicture}
\begin{axis}
\addplot+[only marks] {sin(deg(x))};
\end{axis}
\end{tikzpicture}
```

4.2.1 Coordinate Lists

\addplot coordinates {⟨*coordinate list*⟩};
\addplot[⟨*options*⟩] coordinates {⟨*coordinate list*⟩} ⟨*trailing path commands*⟩;
\addplot3 ...

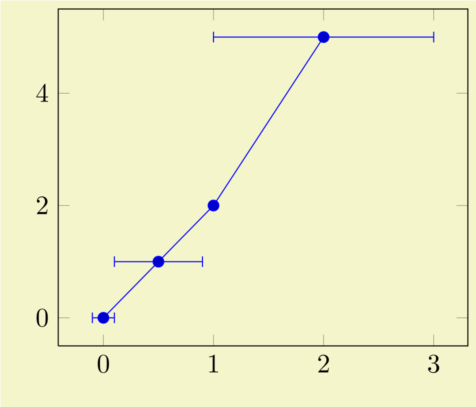
The '`plot coordinates`' command is like that provided by Ti*k*Z and reads its input data from a sequence of point coordinates, encapsulated in round braces.



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}
\addplot coordinates {
    (0,0)
    (0.5,1)
    (1,2)
};
\end{axis}
\end{tikzpicture}
```

The coordinates can be numbers, but they can also contain mathematical expressions like `sin(0.5)` or `\h*8` (assuming you defined `\h` somewhere). However, expressions which involve round braces need to be encapsulated in a further set of curly braces, for example `({sin(0.5)},{cos(0.1)})`.

You can also supply error coordinates (reliability bounds) if you are interested in error bars. Simply append the error coordinates with '`+- (`⟨*ex,ey*⟩`)`' (or `+- (`⟨*ex,ey,ez*⟩`)`) to the associated coordinate:



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}
\addplot+[error bars/.cd,x dir=both,x explicit]
coordinates {
    (0,0)   +- (0.1,0)
    (0.5,1) +- (0.4,0.2)
    (1,2)
    (2,5)   +- (1,0.1)
};
\end{axis}
\end{tikzpicture}
```

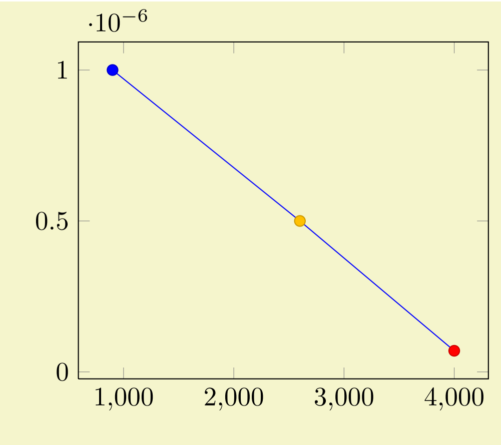
or

```
\addplot coordinates {
     (900,1e-6) +- (0.1,0.2)
    (2600,5e-7) +- (0.2,0.5)
    (4000,7e-8) +- (0.1,0.01)
};
```

These error coordinates are only used in case of error bars, see section 4.11. You will also need to configure whether these values denote absolute or relative errors.

The coordinates as such can be numbers as `+5`, `-1.2345e3`, `35.0e2`, `0.00000123` or `1e2345e-8`. They are not limited to TEX's precision.

Furthermore, `coordinates` allows to define "meta data" for each coordinate. The interpretation of meta data depends on the visualization technique: for scatter plots, meta data can be used to define colors or style associations for every point (see page 57 for an example). Meta data (if any) must be provided after the coordinates and after error bar bounds (if any) in square brackets:



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}
\addplot+[scatter,scatter src=explicit] coordinates {
     (900,1e-6) [1]
    (2600,5e-7) [2]
    (4000,7e-8) [3]
};
\end{axis}
\end{tikzpicture}
```

Please refer to the documentation of `point meta` on page 97 for more information about per point meta data.

`/pgfplots/plot coordinates/math parser=true|false` (initially `true`)

Allows to turn off support for mathematical expressions in every coordinate inside of `plot coordinates`. This might be necessary if coordinates are not in numerical form (or if you'd like to improve speed).

It is necessary to disable `plot coordinates/math parser` if you use some sort of symbolic transformations (i.e. text coordinates).

4.2.2 Reading Coordinates From Files

`\addplot file {`⟨*name*⟩`};`
`\addplot[`⟨*options*⟩`] file {`⟨*name*⟩`}` ⟨*trailing path commands*⟩`;`
`\addplot3 ...`

PGFPLOTS supports two ways to read plot coordinates of external files, and one of them is similar to the Ti*k*Z-command '`plot file`'. It is to be used like

```
\addplot file {datafile.dat};
```

where {⟨*name*⟩} is a text file with at least two columns which will be used as x and y coordinates. Lines starting with '`%`' or '`#`' are ignored. Such files are often generated by GNUPLOT:

```
#Curve 0, 20 points
#x y type
0.00000 0.00000 i
0.52632 0.50235 i
1.05263 0.86873 i
1.57895 0.99997 i
...
9.47368 -0.04889 i
10.00000 -0.54402 i
```

This listing has been copied from [5, section 16.4].

Plot file accepts one optional argument,

```
\addplot file[skip first] {datafile.dat};
```

which allows to skip over a non-comment header line. This allows to read the same input files as `plot table` by skipping over column names. Please note that comment lines do not count as lines here.

The input method `plot file` can also read meta data for every coordinate. As already explained for `plot coordinates` (see above), meta data can be used to change colors or other style parameters for every marker separately. Now, if `point meta` is set to `explicit` or to `explicit symbolic` and the input method is `plot file`, one further element will be read from disk – for every line. Meta data is always the last element which is read. See page 55 for information and examples about per point meta data and page 57 for an application example using `scatter/classes`.

Plot file is very similar to `plot table`: you can achieve the same effect with

```
\addplot table[x index=0,y index=1,header=false] {datafile.dat};
```

Due to its simplicity, `plot file` is slightly faster while `plot table` allows higher flexibility.

Technical note: every opened file will be protocolled into your log file.

`/pgfplots/plot file/skip first=true|false` (initially `false`)
`/pgfplots/plot file/ignore first=true|false` (initially `false`)

The two keys can be provided as arguments to `\addplot file[`⟨*options*⟩`] {`⟨*filename*⟩`};` to skip the first non-comment entry in the file. They are equivalent. If you provide them in this context, the prefix `/pgfplots/plot file` can be omitted.

4.2.3 Reading Coordinates From Tables

`\addplot table [`⟨*column selection*⟩`]{`⟨*file*⟩`};`
`\addplot[`⟨*options*⟩`] table [`⟨*column selection*⟩`]{`⟨*file*⟩`}` ⟨*trailing path commands*⟩`;`
`\addplot3 ...`

The use of '`plot table`' is similar in spirit to '`plot file`', but its flexibility is higher. Given a data file like

```
dof     L2              Lmax            maxlevel
5       8.31160034e-02  1.80007647e-01  2
17      2.54685628e-02  3.75580565e-02  3
49      7.40715288e-03  1.49212716e-02  4
129     2.10192154e-03  4.23330523e-03  5
321     5.87352989e-04  1.30668515e-03  6
769     1.62269942e-04  3.88658098e-04  7
1793    4.44248889e-05  1.12651668e-04  8
4097    1.20714122e-05  3.20339285e-05  9
9217    3.26101452e-06  8.97617707e-06  10
```

one may want to plot '`dof`' versus '`L2`' or '`dof`' versus '`Lmax`'. This can be done by

```
\begin{tikzpicture}
\begin{loglogaxis}[
    xlabel=Dof,
    ylabel=$L_2$ error]
\addplot table[x=dof,y=L2] {datafile.dat};
\end{loglogaxis}
\end{tikzpicture}
```

or

```
\begin{tikzpicture}
\begin{loglogaxis}[
    xlabel=Dof,
    ylabel=$L_infty$ error]
\addplot table[x=dof,y=Lmax] {datafile.dat};
\end{loglogaxis}
\end{tikzpicture}
```

Alternatively, you can load the table *once* and use it *multiple* times:

```
\pgfplotstableread{datafile.dat}\loadedtable
...
\addplot table[x=dof,y=L2] from \loadedtable;
...
\addplot table[x=dof,y=Lmax] from \loadedtable;
...
```

I am not really sure how much time can be saved, but it works anyway. As a rule of thumb, decide as follows:

1. If tables contain few rows and many columns, the `from` ⟨\macro⟩ framework will be more efficient.
2. If tables contain more than 200 data points (rows), you should always use file input (and reload if necessary).

If you do prefer to access columns by column indices instead of column names (or your tables do not have column names), you can also use

```
\addplot table[x index=2,y index=3] {datafile.dat};
\addplot table[x=dof,y index=2] {datafile.dat};
```

Summary and remarks:

- Use `plot table[x={`⟨*column name*⟩`},y={`⟨*column name*⟩`}]` to access column names. Those names are case sensitive and need to exist.
- Use `plot table[x index={`⟨*column index*⟩`},y index={`⟨*column index*⟩`}]` to access column indices. Indexing starts with 0. You may also use an index for x and a column name for y.
- Use `plot table[header=false] {`⟨*file name*⟩`}` if your input file has no column names. Otherwise, the first non-comment line is checked for column names: if all entries are numbers, they are treated as numerical data; if one of them is not a number, all are treated as column names.
- It is possible to read error coordinates from tables as well. Simply add options '`x error`', '`y error`' or '`x error index`'/'`y error index`' to `{`⟨*source columns*⟩`}`. See section 4.11 for details about error bars.
- It is possible to read per point meta data (usable in `scatter src`, see page 55) as has been discussed for `plot coordinates` and `plot file` above. The meta data column can be provided using the `meta` key (or the `meta index` key).
- Use `plot table[`⟨*source columns*⟩`] from {`⟨\macro⟩`}` to use a pre–read table. Tables can be read using

  ```
  \pgfplotstableread{datafile.dat}\macroname.
  ```

  The keyword '`from`' can be omitted.
- The accepted input format of those tables is as follows:

  - Columns are usually separated by white spaces (at least one tab or space).
    If you need other column separation characters, you can use the
    col sep=space|tab|comma|colon|semicolon|braces
    option which is documented in all detail in the manual for PgfplotsTable which is part of pgfplots.
  - Any line starting with '#' or '%' is ignored.
  - The first line will be checked if it contains numerical data. If there is a column in the first line which is *no* number, the complete line is considered to be a header which contains column names. Otherwise it belongs to the numerical data and you need to access column indices instead of names.
  - There is future support for a second header line which must start with '`$flags `'. Currently, such a line is ignored. It may be used to provide number formatting hints like precision and number format if those tables shall be typeset using `\pgfplotstabletypeset` (see the manual for PgfplotsTable).
  - The accepted number format is the same as for '`plot coordinates`', see above.
  - If you omit column selectors, the default is to plot the first column against the second. That means `plot table` does exactly the same job as `plot file` for this case.
  - If you need unbalanced columns, simply use `nan` as "empty cell" placeholder. These coordinates will be skipped in plots.
- It is also possible to use **mathematical expressions** together with '`plot table`'. This is documented in all detail in section 4.2.5, but the key idea is to use one of `x expr`, `y expr`, `z expr` or `meta expr` as in '`plot table[x expr=\thisrow{maxlevel}+3,y=L2]`'.
- The enhanced column generation methods provided by PgfplotsTable are also accessible for plots, see the PgfplotsTable manual section "Postprocessing Data in New Columns".

  In this context, you should consider using the key `read completely`, see below.
- Technical note: every opened file will be protocolled into your log file.

Keys To Configure Table Input

The following list of keys allow different methods to select input data or different input formats. Note that the common prefix '`table/`' can be omitted if these keys are set after `\addplot table[`⟨*options*⟩`]`. The `/pgfplots/` prefix can always be omitted when used in a pgfplots method.

/pgfplots/`table/header=true|false` (initially `true`)

Allows to disable header identification for `plot table`. See above.

/pgfplots/`table/x={`⟨*column name*⟩`}`
/pgfplots/`table/y={`⟨*column name*⟩`}`
/pgfplots/`table/z={`⟨*column name*⟩`}`
/pgfplots/`table/x index={`⟨*column index*⟩`}`
/pgfplots/`table/y index={`⟨*column index*⟩`}`
/pgfplots/`table/z index={`⟨*column index*⟩`}`

These keys define the sources for `plot table`. If both, column names and column indices are given, column names are preferred. Column indexing starts with 0. The initial setting is to use `x index=0` and `y index=1`.

Please note that column *aliases* will be considered if unknown column names are used. Please refer to the manual of PgfplotsTable which comes with this package.

/pgfplots/`table/x expr={`⟨*expression*⟩`}`
/pgfplots/`table/y expr={`⟨*expression*⟩`}`
/pgfplots/`table/z expr={`⟨*expression*⟩`}`
/pgfplots/`table/meta expr={`⟨*expression*⟩`}`

These keys allow to combine the mathematical expression parser with file input. They are listed here to complete the list of table keys, but they are described in all detail in section 4.2.5.

The key idea is to provide an ⟨*expression*⟩ which depends on table data (possibly on all columns in one row). Only data within the same row can be used where columns are referenced with `\thisrow{`⟨*column name*⟩`}` or `\thisrowno{`⟨*column index*⟩`}`.

Please refer to section 4.2.5 for details.

/pgfplots/table/x error={⟨*column name*⟩}
/pgfplots/table/y error={⟨*column name*⟩}
/pgfplots/table/z error={⟨*column name*⟩}
/pgfplots/table/x error index={⟨*column index*⟩}
/pgfplots/table/y error index={⟨*column index*⟩}
/pgfplots/table/z error index={⟨*column index*⟩}
/pgfplots/table/x error expr={⟨*math expression*⟩}
/pgfplots/table/y error expr={⟨*math expression*⟩}
/pgfplots/table/z error expr={⟨*math expression*⟩}

These keys define input sources for error bars with explicit error values.

The `x error` method provides an input column name (or alias), the `x error index` method provides input column *indices* and `x error expr` works just as `table/x expr`: it allows arbitrary mathematical expressions which may depend on any number of table columns using `\thisrow`{⟨*col name*⟩}.

Please see section 4.11 for details about the usage of error bars.

/pgfplots/table/meta={⟨*column name*⟩}
/pgfplots/table/meta index={⟨*column index*⟩}

These keys define input sources for per point meta data. Please see page 55 for details about meta data or the documentation for `plot coordinates` and `plot file` for further information.

/pgfplots/table/col sep=space|tab|comma|semicolon|colon|braces (initially `space`)

Allows to choose column separators for `plot table`. Please refer to the manual of PgfplotsTable which comes with this package for details about `col sep`.

/pgfplots/table/read completely={⟨*true,false*⟩} (initially `false`)

Allows to customize `\addplot table`{⟨*file name*⟩} such that it always reads the entire table into memory.

This key has just one purpose, namely to create postprocessing columns on-the-fly and to plot those columns afterwards. This "lazy evaluation" which creates missing columns on-the-fly is documented in the PgfplotsTable manual (in section "Postprocessing Data in New Columns").

Attention: Usually, `\addplot table` only picks required entries, requiring linear runtime complexity. As soon as `read completely` is activated, tables are loaded completely into memory. Due to datastructures issues ("macro append runtime"), the runtime complexity for `read completely` is $O(N^2)$ where N is the number of rows. Thus: use this feature only for "small" tables[5].

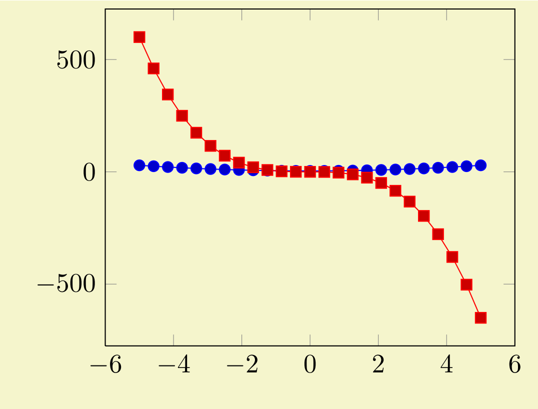
4.2.4 Computing Coordinates with Mathematical Expressions

\addplot {⟨*math expression*⟩} ;
\addplot[⟨*options*⟩] {⟨*math expression*⟩} ⟨*trailing path commands*⟩;
\addplot3 ...

This input method allows to provide mathematical expressions which will be sampled. But unlike `plot gnuplot`, the expressions are evaluated using the math parser of pgf, no external program is required.

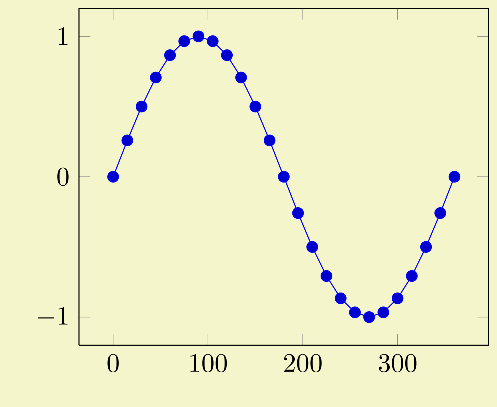
Plot expression samples `x` from the interval $[a, b]$ where a and b are specified with the `domain` key. The number of samples can be configured with `samples`=⟨N⟩ as for plot gnuplot.

[5] This remark might be deprecated; many of the slow routines have been optimized in the meantime to have at least pseudo-linear runtime.



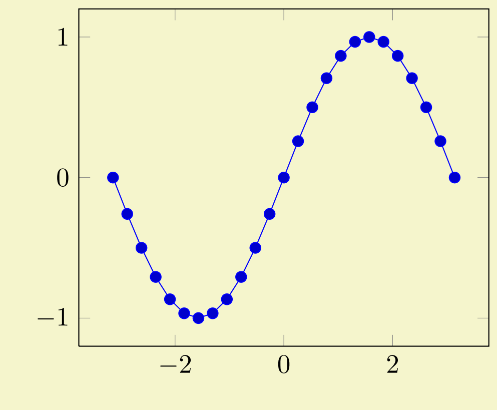
```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}
    \addplot {x^2 + 4};
    \addplot {-5*x^3 - x^2};
\end{axis}
\end{tikzpicture}
```

Please note that PGF's math parser uses degrees for trigonometric functions:



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}
    \addplot+[domain=0:360]
        {sin(x)};
\end{axis}
\end{tikzpicture}
```

If you want to use radians, use



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}
    \addplot+[domain=-pi:pi]
        {sin(deg(x))};
\end{axis}
\end{tikzpicture}
```

to convert the radians to degrees. The plot expression parser also accepts some more options like `samples at={`⟨*coordinate list*⟩`}` or `domain=`⟨*first*⟩`:`⟨*last*⟩ which are described below.

Remarks

1. What really goes on is a loop which assigns the current sample coordinate to the macro `\x`. PGFPLOTS defines a math constant `x` which always has the same value as `\x`.

   In short: it is the same whether you write `\x` or just `x` inside of math expressions.

   The variable name can be customized using `variable=\t` (the backslash is necessary!). Then, `t` will be the same as `\t`.

2. The complete set of math expressions can be found in the PGF manual. The most important mathematical operations are `+`, `-`, `*`, `/`, `abs`, `round`, `floor`, `mod`, `<`, `>`, `max`, `min`, `sin`, `cos`, `tan`, `deg` (conversion from radians to degrees), `rad` (conversion from degrees to radians), `atan`, `asin`, `acos`,

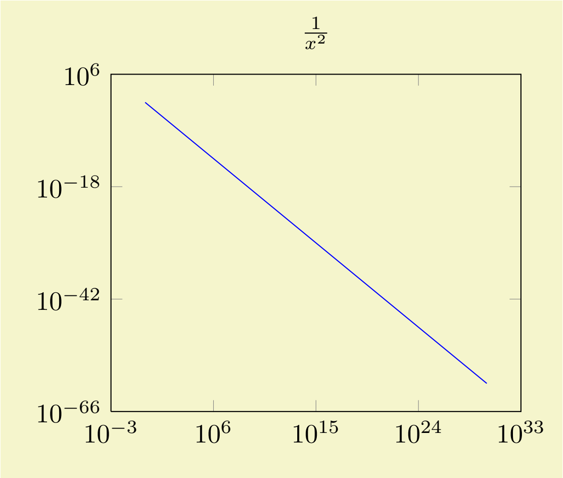
`cot`, `sec`, `cosec`, `exp`, `ln`, `sqrt`, the constants `pi` and `e`, `^` (power operation), `factorial`[6], `rand` (random between −1 and 1), `rnd` (random between 0 and 1), number format conversions `hex`, `Hex`, `oct`, `bin` and some more. The math parser has been written by Mark Wibrow and Till Tantau [5], the FPU routines have been developed as part of PGFPLOTS. The documentation for both parts can be found in [5].

Please note, however, that trigonometric functions are defined in degrees. The character '`^`' is used for exponentiation (not '`**`' as in gnuplot).

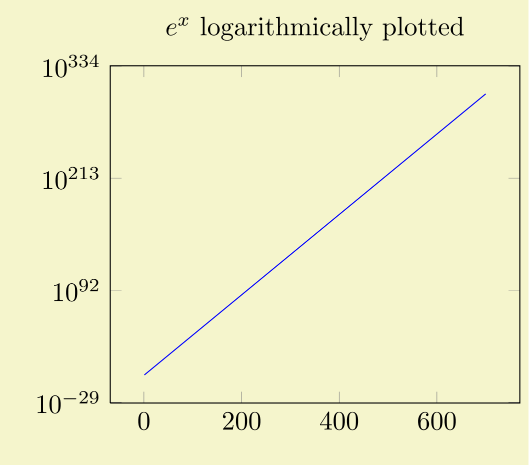
3. If the x axis is logarithmic, samples will be drawn logarithmically.
4. Please note that plot expression does not allow per point meta data (color data).

About the precision and number range: Starting with version 1.2, `plot expression` uses a floating point unit. The FPU provides the full data range of scientific computing with a relative precision between 10^{-4} and 10^{-6}. The `/pgf/fpu` key provides some more details.

In case the `fpu` does not provide the desired mathematical function or is too slow[7], you should consider using the `plot gnuplot` method which invokes the external, freely available program `gnuplot` as desktop calculator.



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{loglogaxis}[
        title={$\frac{1}{x^2}$}]
    \addplot[blue,domain=1:1e30]
        {x^-2};
    \end{loglogaxis}
\end{tikzpicture}
```



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{semilogyaxis}[
        title={$e^x$ logarithmically plotted}]
    \addplot[blue,domain=1:700]
        {exp(x)};
    \end{semilogyaxis}
\end{tikzpicture}
```

```
\addplot expression {⟨math expr⟩};
\addplot[⟨options⟩] expression {⟨math expr⟩} ⟨trailing path commands⟩;
\addplot3 ...
```

The syntax

`\addplot {⟨math expression⟩};`

[6] Starting with PGF versions newer than 2.00, you can use the postfix operator `!` instead of `factorial`.
[7] Or in case you find a bug...

as short-hand equivalent for

`\addplot expression {⟨math expression⟩};`

`\addplot (⟨x expression⟩,⟨y expression⟩) ;`
`\addplot[⟨options⟩] (⟨x expression⟩,⟨y expression⟩) ⟨trailing path commands⟩;`
`\addplot3 ...`

A variant of `\addplot expression` which allows to provide different coordinate expressions for the x and y coordinates. This can be used to generate parameterized plots.

Please note that `\addplot (x,x^2)` is equivalent to `\addplot expression {x^2}`.

Note further that since the complete point expression is surrounded by round braces, round braces for either ⟨*x expression*⟩ or ⟨*y expression*⟩ need special attention. You will need to introduce curly braces additionally to allow round braces:

`\addplot ({⟨x expr⟩}, {⟨y expr⟩}, {⟨z expr⟩});`

`/pgfplots/domain=⟨x1⟩:⟨x2⟩` (initially `[-5:5]`)
`/pgfplots/y domain=⟨y1⟩:⟨y2⟩`
`/pgfplots/domain y=⟨y1⟩:⟨y2⟩`

Sets the function's domain(s) for `plot expression` and `plot gnuplot`. Two dimensional plot expressions are defined as functions $f\colon [x_1,x_2] \to \mathbb{R}$ and $\langle x_1\rangle$ and $\langle x_2\rangle$ are set with `domain`. Three dimensional plot expressions use functions $f\colon [x_1,x_2] \times [y_1,y_2] \to \mathbb{R}$ and $\langle y_1\rangle$ and $\langle y_2\rangle$ are set with `y domain`. If `y domain` is empty, $[y_1,y_2] = [x_1,x_2]$ is assumed for three dimensional plots (see page 68 for details about three dimensional plot expressions).

The keys `y domain` and `domain y` are the same.

The `domain` key won't be used if `samples at` is specified; `samples at` has higher precedence.

Please note that `domain` is not necessarily the same as the axis limits (which are configured with the `xmin`/`xmax` options).

The `domain` keys are *only* relevant for `gnuplot` and `plot expression`. In case you'd like to plot only a subset of other coordinate input routines, consider using the coordinate filter `restrict x to domain`.

Remark for TikZ-users: `/pgfplots/domain` and `/tikz/domain` are independent options. Please prefer the PGFPLOTS variant (i.e. provide `domain` to an axis, `\pgfplotsset` or a plot command). Since older versions also accepted something like `\begin{tikzpicture}[domain=...]`, this syntax is also accepted as long as no PGFPLOTS `domain` key is set.

`/pgfplots/samples={⟨number⟩}` (initially `25`)
`/pgfplots/samples y={⟨number⟩}`

Sets the number of sample points for `plot expression` and `plot gnuplot`. The `samples` key defines the number of samples used for line plots while the `samples y` key is used for mesh plots (three dimensional visualisation, see page 68 for details). If `samples y` is not set explicitly, it uses the value of `samples`.

The `samples` key won't be used if `samples at` is specified; `samples at` has higher precedence.

The same special treatment of `/tikz/samples` and `/pgfplots/samples` as for the `domain` key applies here. See above for details.

`/pgfplots/samples at={⟨coordinate list⟩}`

Sets the x coordinates for `plot expression` explicitly. This overrides `domain` and `samples`.

The `{⟨coordinate list⟩}` is a `\foreach` expression, that means it can contain a simple list of coordinates (comma–separated) but also complex `...` expressions like[8]

```
\pgfplotsset{samples at={5e-5,7e-5,10e-5,12e-5}}
\pgfplotsset{samples at={-5,-4.5,...,5}}
\pgfplotsset{samples at={-5,-3,-1,-0.5,0,...,5}}
```

The same special treatment of `/tikz/samples at` and `/pgfplots/samples at` as for the `domain` key applies here. See above for details.

[8] Unfortunately, the `...` is somewhat restrictive when it comes to extended accuracy. So, if you have particularly small or large numbers (or a small distance), you have to provide a comma–separated list (or use the `domain` key).

Attention: samples at overrides domain, even if domain has been set *after* samples at! Use samples at={} to clear {⟨*coordinate list*⟩} and re-activate domain.

/pgfplots/variable={⟨*variable name*⟩} (initially x)
/pgfplots/variable y={⟨*variable name*⟩} (initially y)

Defines the variables names which will be sampled in domain (with variable) and in domain y (with variable y).

The same variables are used for parametric and for non-parametric plots. Use variable=t to change them if you like (for gnuplot, there is such a distinction; see parametric/var 1d).

Technical remark: Ti*k*Z also uses the variable key. However, it expects a *macro* name, i.e. \x instead of just x. Both possibilities are accepted here.

4.2.5 Mathematical Expressions And File Data

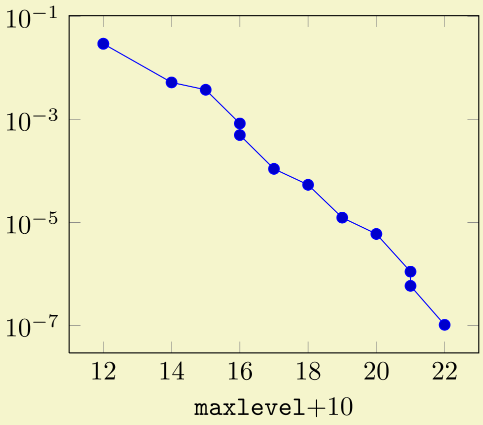
PGFPLOTS allows to combine 'plot table' and 'plot expression' to get both, file input and modifications by means of mathematical expressions.

\addplot table [⟨*column selection and expressions*⟩]{⟨*file*⟩};
\addplot[⟨*options*⟩] table [⟨*column selection and expressions*⟩]{⟨*file*⟩} ⟨*trailing path commands*⟩;
\addplot3 ...

Besides the already discussed possibility to provide a column selection by means of column names (x=⟨*name*⟩ or x index=⟨*index*⟩, see section 4.2.3), it is also possible to provide mathematical expressions as arguments.

Mathematical expressions are specified with x expr=⟨*expression*⟩ inside of ⟨*column selection and expressions*⟩. They can depend on zero, one or more columns of the input file. A column is referenced using the special command '\thisrow{⟨*column name*⟩}' within ⟨*expression*⟩ (or \thisrowno⟨*column index*⟩).

maxlevel	L2
2	$2.97 \cdot 10^{-2}$
2	$2.97 \cdot 10^{-2}$
4	$5.27 \cdot 10^{-3}$
5	$3.8 \cdot 10^{-3}$
6	$8.41 \cdot 10^{-4}$
6	$5.01 \cdot 10^{-4}$
7	$1.11 \cdot 10^{-4}$
8	$5.41 \cdot 10^{-5}$
9	$1.25 \cdot 10^{-5}$
10	$6.01 \cdot 10^{-6}$
11	$1.11 \cdot 10^{-6}$
11	$5.9 \cdot 10^{-7}$
12	$1.03 \cdot 10^{-7}$



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\pgfplotstabletypeset[columns={maxlevel,L2}]{plotdata/newexperiment1.dat}

\begin{tikzpicture}
    \begin{semilogyaxis}[
        xlabel=\texttt{maxlevel}$ + 10$
    ]
    \addplot table
        [x expr=\thisrow{maxlevel}+10, y=L2]
        {plotdata/newexperiment1.dat};
    \end{semilogyaxis}
\end{tikzpicture}
```

Besides x expr, there are keys y expr, z expr and meta expr where the latter allows to provide point meta data (which is used as scatter src or color data for surface plots etc.).

Inside of ⟨*expression*⟩, the following macros can be used to access numerical data cells inside of the input file:

`\thisrow{⟨column name⟩}`

Yields the value of the column designated by `{⟨column name⟩}`. There is no limit on the number of columns which can be part of a mathematical expression, but only values inside of the currently processed *table row* can be used.

It is possible to provide column aliases for `{⟨column name⟩}` as described in the manual of PgfplotsTable.

The argument ⟨*column name*⟩ has to denote either an existing column or one for which a column alias exists (see the manual of PgfplotsTable). If it can't be resolved, the math parser yields an "Unknown function" error message.

`\thisrowno{⟨column index⟩}`

Similar to `\thisrow`, this command yields the value of the column with index `{⟨column index⟩}` (starting with 0).

`\coordindex`

Yields the current index of the table row (starting with 0). This does *not* count header or comment lines.

`\lineno`

Yields the current line number (starting with 0). This does also count header and comment lines.

If `x index`, `x` and `x expr` (or the corresponding keys for `y`, `z` or `meta`) are combined, this is how they interact:

1. Column access via `x` has higher precedence than index access via `x index`.
2. Even if `x expr` is provided, the values of `x index` and `x` are still checked. Any value found using column name access or column index access is made available as `\columnx` (or `\columny`, `\columnz`, `\columnmeta`, resp.). However, the result of `x expr` is used as plot coordinate.

   This allows to access the cell values identified by `x` or `x index` using the "pointer" `\columnx`. I am not sure if this yields any advantage, but it is possible nevertheless. If in doubt, prefer using `\thisrow{⟨column name⟩}`.

Attention: If your table has less rows than two, you may need to set `x index={},y index={}` explicitly. This is a consequence of the fact that column name/index access is still applied even if an expression is provided.

4.2.6 Computing Coordinates with Mathematical Expressions (gnuplot)

```
\addplot gnuplot [⟨further options⟩]{⟨gnuplot code⟩};
\addplot[⟨options⟩] gnuplot [⟨further options⟩]{⟨gnuplot code⟩} ⟨trailing path commands⟩;
\addplot3 ...
```

In contrast to `plot expression`, the `plot gnuplot` command[9] employs the external program `gnuplot` to compute coordinates. The resulting coordinates are written to a text file which will be plotted with `plot file`. PGF checks whether coordinates need to be re-generated and calls `gnuplot` whenever necessary (this is usually the case if you change the number of samples, the argument to `plot gnuplot` or the plotted domain[10]).

The differences between `plot expression` and `plot gnuplot` are:

- `plot expression` does not require any external programs and requires no additional command line options.
- `plot expression` does not produce a lot of temporary files.
- `plot gnuplot` uses radians for trigonometric functions while `plot expression` has degrees.
- `plot gnuplot` is faster.

[9] Note that `plotgnuplot` is actually a re-implementation of the `plotfunction` method known from PGF. It also invokes PGF basic layer commands.

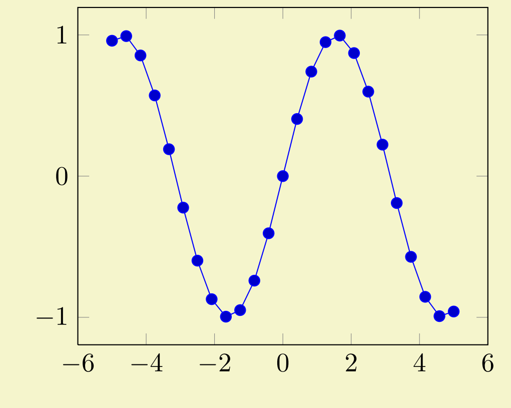
[10] Please note that PGFPLOTS produces slightly different files than Ti*k*Z when used with `plot gnuplot` (it configures high precision output). You should use different `id` for PGFPLOTS and Ti*k*Z to avoid conflicts in such a case.

- plot gnuplot has a larger mathematical library.
- plot gnuplot has a higher accuracy. However, starting with version 1.2, this is no longer a great problem. The new floating point unit for TEX provides reasonable accuracy and the same data range as gnuplot.

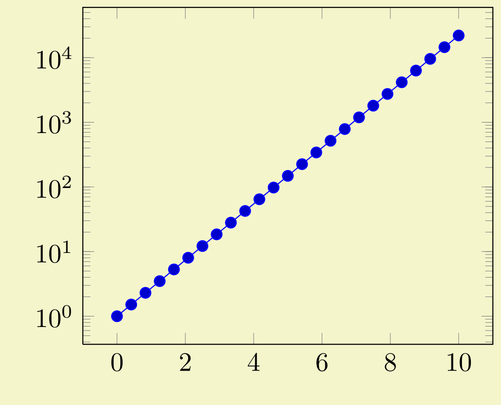
Since system calls are a potential danger, they need to be enabled explicitly using command line options, for example

```
pdflatex -shell-escape filename.tex.
```

Sometimes it is called `shell-escape` or `enable-write18`. Sometimes one needs two slashes – that all depends on your TEX distribution.



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}
\addplot
    gnuplot[id=sin]{sin(x)};
\end{axis}
\end{tikzpicture}
```



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{semilogyaxis}
\addplot gnuplot
    [id=exp,domain=0:10]{exp(x)};
\end{semilogyaxis}
\end{tikzpicture}
```

The ⟨*options*⟩ determine the appearance of the plotted function; these parameters also affect the legend. There is also a set of options which are specific to the gnuplot interface. These options are described in all detail in [5, section 18.6]. A short summary is shown below.

Some remarks:

- The independent variable for one dimensional plots can be change with the variable option, just as for plot expression. Similarly, the second variable for two dimensional plots can be changed with variable y.

  For parametric plots, the variable names need to be adjusted with parametric/var 1d and parametric/var 2d (since gnuplot uses `t` and `u,v` as initial values for parametric plots).
- Please note that plot gnuplot does not allow per point meta data (color data for each coordinate).
- The generated output file name can be customized with id, see below.

Please refer to [5, section 18.6] for more details about plot function and the gnuplot interaction.

```
\addplot function {⟨gnuplot code⟩};
\addplot[⟨options⟩] function {⟨gnuplot code⟩} ⟨trailing path commands⟩;
```

\addplot3 ...

Use

`\addplot function {⟨gnuplot code⟩};`

as alias for

`\addplot gnuplot {⟨gnuplot code⟩};`

/pgfplots/translate gnuplot=true|false (initially true)

Enables or disables automatic translation of the exponentiation operator '^' to '**'.

This features allows to use ^ in plot gnuplot instead of gnuplot's **.

/pgfplots/parametric=true|false (initially false)

Set this to true if you'd like to use parametric plots with gnuplot. Parametric plots use a comma separated list of expressions to make up $x(t)$, $y(t)$ for a line plot or $x(u, v)$, $y(u, v)$ $z(u, v)$ for a mesh plot (refer to the gnuplot manual for more information about its input methods for parametric plots).

/pgfplots/parametric/var 1d={⟨variable name⟩} (initially t)
/pgfplots/parametric/var 2d={⟨variable name,variable name⟩} (initially u,v)

Allows to change the dummy variables used by parametric gnuplot plots. The initial setting is the one of gnuplot: to use the dummy varialbe 't' for parametric line plots and 'u,v' for parametric mesh plots.

These keys are quite the same as variable and variable y, only for parametric plots. If you like to change variables for non-parametric plots, use variable and/or variable y.

In case you don't want the distinction between parametric and non-parametric plots, use

`\pgfplotsset{parametric/var 1d=,parametric/var 2d=}`.

/tikz/id={⟨unique string identifier⟩}

A unique identifier for the current plot. It is used to generate temporary filenames for gnuplot output.

/tikz/prefix={⟨file name prefix⟩}

A common path prefix for temporary filenames (see [5, section 18.6] for details).

/tikz/raw gnuplot (no value)

Disables the use of samples and domain.

4.2.7 Computing Coordinates with External Programs (shell)

\addplot shell [⟨further options⟩]{⟨shell commands⟩};
\addplot[⟨options⟩] shell [⟨further options⟩]{⟨shell commands⟩} ⟨trailing path commands⟩;
\addplot3 ...

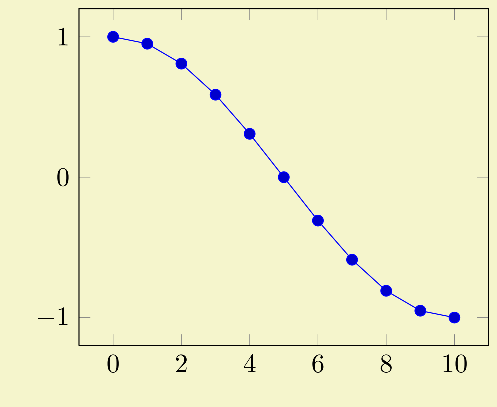
An extension by Stefan Tibus

In contrast to plot gnuplot, the plot shell command allows execution of arbitrary shell commands to compute coordinates. The resulting coordinates are written to a text file which will be plotted with plot file. PGF checks whether coordinates need to be re-generated and executes the ⟨shell commands⟩ whenever necessary.

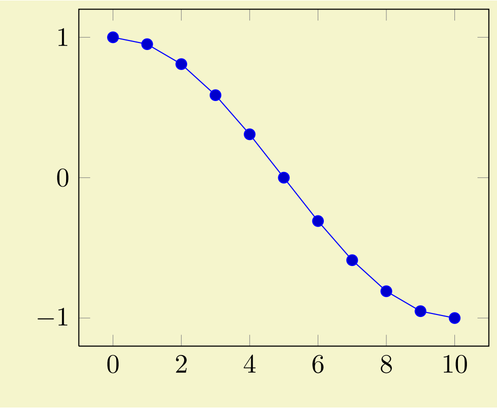
Since system calls are a potential danger, they need to be enabled explicitly using command line options, for example

```
pdflatex -shell-escape filename.tex.
```

Sometimes it is called shell-escape or enable-write18. Sometimes one needs two slashes – that all depends on your TEX distribution.



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}
\addplot
    shell[prefix=pgfshell_,id=cos]{awk 'BEGIN{
        pi=3.14159; N=10;
        for(i=0;i<=N;i++) print i,cos(i/N*pi);}'};
\end{axis}
\end{tikzpicture}
```



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}
\addplot+[prefix=pgfshell_,id=replot]
    shell{cat pgfshell_cos.out};
    %  just reprint the result from above
\end{axis}
\end{tikzpicture}
```

The ⟨*options*⟩ determine the appearance of the plotted function; these parameters also affect the legend. There is also a set of options which are specific to the gnuplot and the shell interface. These options are described in all detail in [5, section 19.6]. A short summary is shown below.

/tikz/id={⟨*unique string identifier*⟩}

A unique identifier for the current plot. It is used to generate temporary filenames for shell output.

/tikz/prefix={⟨*file name prefix*⟩}

A common path prefix for temporary filenames (see [5, section 19.6] for details).

4.2.8 Using External Graphics as Plot Sources

\addplot graphics {⟨*file name*⟩};
\addplot[⟨*options*⟩] graphics {⟨*file name*⟩} ⟨*trailing path commands*⟩;
\addplot3 ...

This plot type allows to extend the plotting capabilities of PGFPLOTS beyond its own limitations. The idea is to generate the graphics as such (for example, a contour plot, a complicated shaded surface[11] or a large point cluster) with an external program like Matlab (tm) or gnuplot. The graphics, however, should *not* contain an axis or descriptions. Then, we use \includegraphics and an PGFPLOTS axis which fits exactly on top of the imported graphics.

Of course, one could do this manually by providing proper scales and such. The operation plot graphics is intended so simplify this process. However the *main difficulty* is to get images with correct bounding box. Typically, you will have to adjust bounding boxes manually.

Let's start with an example: Suppose we use, for example, matlab to generate a surface plot like

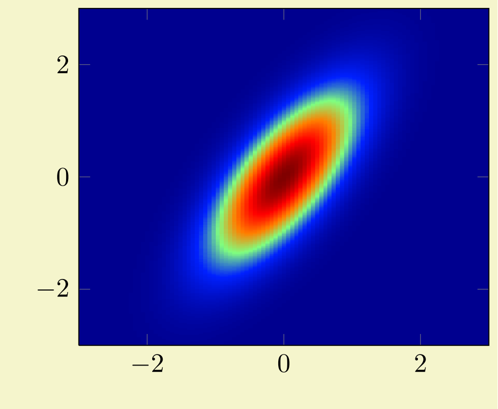
```
[X,Y] = meshgrid( linspace(-3,3,500) );
surf( X,Y, exp(-(X - Y).^2 - X.^2 ) );
shading flat; view(0,90); axis off;
print -dpng external1
```

[11] See also section 4.5.6 for an overview of PGFPLOTS methods to draw shaded surfaces.

which is then found in `external1.png`. The `surf` command of Matlab generates the surface, the following commands disable the axis descriptions, initialise the desired view and export it. Viewing the image in any image tool, we see a lot of white space around the surface – Matlab has a particular weakness in producing tight bounding boxes, as far as I know. Well, no problem: use your favorite image editor and crop the image (most image editors can do this automatically). We could use the free ImageMagick command

```
convert -trim external1.png external1.png
```

to get a tight bounding box. Then, we use



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{axis}[enlargelimits=false,axis on top]
        \addplot graphics
            [xmin=-3,xmax=3,ymin=-3,ymax=3]
            {external1};
    \end{axis}
\end{tikzpicture}
```

to load the graphics[12] just as if we would have drawn it with PGFPLOTS. The `axis on top` simply tells PGFPLOTS to draw the axis on top of any plots (see its description).

Please note that PGFPLOTS offers support for smaller surface plots as well which might be an option – unless the number of samples is too large. See section 4.5.6 for details.

However, external programs have the following advantages here: they are faster, allow more complexity and provide real z buffering which is currently only simulated by PGFPLOTS. Thus, it may help to consider `plot graphics` for complicated surface plots.

Our first test was successful – and not difficult at all because graphics programs can automatically compute the bounding box. There are a couple of free tools available which can compute tight bounding boxes for `.eps` or `.pdf` graphics:

1. The free vector graphics program `inkscape` can help here. Its feature "File ≫ Document Properties: Fit page to selection" computes a tight bounding box around every picture element.

   However, some images may contain a rectangular path which is as large as the bounding box (Matlab (tm) computes such `.eps` images). In this case, use the "Ungroup" method (context menu of `inkscape`) as often as necessary and remove such a path.

   Finally, save as `.eps`.

   However, `inkscape` appears to have problems with postscript fonts – it substitutes them. This doesn't pose problems in this application because fonts shouldn't be part of such images – the descriptions will be drawn by PGFPLOTS.

2. The tool `pdfcrop` removes surrounding whitespace in `.pdf` images and produces quite good bounding boxes.

Adjusting bounding boxes manually In case you don't have tools at hand to provide correct bounding boxes, you can still use TEX to set the bounding box manually. Some viewers like `gv` provide access to low–level image coordinates. The idea is to determine the number of units which need to be removed and communicate these units to `\includegraphics`.

I am aware of the following methods to determine bounding boxes manually:

inkscape I am pretty sure that `inkscape` can do it.

gv The ghost script viewer `gv` always shows the postscript units under the mouse cursor.

[12] Please note that I don't have a Matlab license, so I used `gnuplot` to produce an equivalent replacement graphics.

gimp The graphics program `gimp` usually shows the cursor position in pixels, but it can be configured to display postscript points (`pt`) instead.

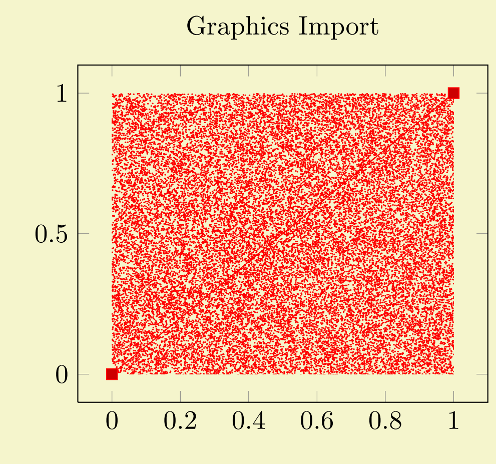
Let's follow this approach in a further example.

We use `gnuplot` to draw a (relatively stupid) example data set. The gnuplot script

```
set samples 30000
set parametric
unset border
unset xtics
unset ytics
set output "external2.eps"
set terminal postscript eps color
plot [t=0:1] rand(0),rand(0) with dots notitle lw 5
```

generates `external2.eps` with a uniform random sample of size 30000. As before, we import this scatter plot into PGFPLOTS using `plot graphics`. Again, the bounding box is too large, so we need to adjust it (`gnuplot` can do this automatically, but we do it anyway to explain the mechanisms):

Using `gv`, I determined that the bounding box needs to be shifted `12` units to the left and `9` down. Furthermore, the right end is `12` units too far off and the top area has about `8` units space wasted. This can be provided to the `trim` option of `\includegraphics`, and we use `clip` to clip the rest away:



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{axis}[axis on top,title=Graphics Import]
        \addplot graphics
            [xmin=0,xmax=1,ymin=0,ymax=1,
            %  trim=left bottom right top
            includegraphics={trim=12 9 12 8,clip}]
            {external2};
        \addplot coordinates {(0,0) (1,1)};
    \end{axis}
\end{tikzpicture}
```

So, `plot graphics` takes a graphics file along with options which can be passed to `\includegraphics`. Furthermore, it provides the information how to embed the graphics into an axis. The axis can contain any other `\addplot` command as well and will be resized properly.

Details about `plot graphics`: The loaded graphics file is drawn with

`\node[/pgfplots/plot graphics/node] {\includegraphics[`⟨*options*⟩`]{`⟨*file name*⟩`}};`

where the `node` style is a configurable style. The node is placed at the coordinate designated by `xmin`, `ymin`.

The ⟨*options*⟩ are any arguments provided to the `includegraphics` key (see below) and `width` and `height` determined such that the graphics fits exactly into the rectangle denoted by the `xmin`, `ymin` and `xmax`, `ymax` coordinates.

The scaling will thus ignore the aspect ratio of the external image and prefer the one used by PGFPLOTS. You will need to provide `width` and `height` to the PGFPLOTS axis to change its scaling. Use the `scale only axis` key in such a case.

Keys To Configure Plot Graphics

The following list of keys configure `\addplot graphics`. Note that the common prefix '`plot graphics/`' can be omitted if these keys are set after `\addplot graphics[`⟨*options*⟩`]`. The `/pgfplots/` prefix can always be omitted when used in a PGFPLOTS method.

`/pgfplots/plot graphics/xmin={`⟨*coordinate*⟩`}`

/pgfplots/plot graphics/ymin={⟨*coordinate*⟩}
/pgfplots/plot graphics/zmin={⟨*coordinate*⟩}
/pgfplots/plot graphics/xmax={⟨*coordinate*⟩}
/pgfplots/plot graphics/ymax={⟨*coordinate*⟩}
/pgfplots/plot graphics/zmax={⟨*coordinate*⟩}

These keys are required for `plot graphics` and provide information about the external data range. The graphics will be squeezed between these coordinates. The arguments are axis coordinates.

/pgfplots/plot graphics/includegraphics={⟨*options*⟩}

A list of options which will be passed as–is to `\includegraphics`. Interesting options include the `trim`=⟨*left*⟩ ⟨*bottom*⟩ ⟨*right*⟩ ⟨*top*⟩ key which reduces the bounding box and `clip` which discards everything outside of the bounding box. The scaling options won't have any effect, they will be overwritten by PGFPLOTS.

/pgfplots/plot graphics/node (style, no value)

A predefined style used for the Ti*k*Z node containing the graphics. The predefined value is

```
\pgfplotsset{
    plot graphics/node/.style={
        transform shape,
        inner sep=0pt,
        outer sep=0pt,
        every node/.style={},
        anchor=south west,
        at={(0pt,0pt)},
        rectangle
    }
}
```

/pgfplots/plot graphics (no value)

This key belongs to the public low–level plotting interface. You won't need it in most cases.

This key is similar to `sharp plot` or `smooth` or `const plot`: it installs a low–level plot–handler which expects exactly two points: the lower left corner and the upper right one. The graphics will be drawn between them. The graphics file name is expected as value of the `/pgfplots/plot graphics/src` key. The other keys described above need to be set correctly (excluding the limits, these are ignored at this level of abstraction). This key can be used independently of an axis.

/pgfplots/plot graphics/lowlevel draw={⟨*width*⟩}{⟨*height*⟩}

A low–level interface for `plot graphics` which actually invokes `\includegraphics`. But there is no magic involved: the command is simply expected to draw a box of dimensions ⟨*width*⟩ × ⟨*height*⟩. The coordinate system has already been shifted correctly.

The initial configuration is

`\includegraphics[`⟨*value of* "`plot graphics/includegraphics`"⟩`,width=#1,height=#2]`
  `{`⟨*value of* "`plot graphics/src`"⟩`}`.

Thus, you can tweak `plot graphics` to place any TEX box of the desired dimensions into an axis between the provided minimum and maximum coordinates. It is not necessary to make use of the graphics file name or the options in the '`includegraphics`' key if you overwrite this lowlevel interface with

`plot graphics/lowlevel draw/.code 2 args={`⟨*code which depends on #1 and #2*⟩`}`.

4.3 About Options: Preliminaries

PGFPLOTS knowns a whole lot of key–value options which can be (re-)defined to activate desired features or modified to apply some fine tuning.
Most keys can be used like

```
\begin{tikzpicture}
\begin{axis}[key=value,key2=value2]
...
\end{axis}
\end{tikzpicture}
```

which changes them for the complete axis. A `key` in this context can be any option defined in this manual, no matter if it has the `/pgfplots/` or the `/tikz/` key prefix. Note that key prefixes can be omitted in almost all cases.

Some keys can be changed individually for each plot:

```
\begin{tikzpicture}
\begin{axis}
\addplot[key=value,key2=value2] ... ;
\addplot+[key=value,key2=value2] ... ; %  keeps the keys which would have been used by default
\end{axis}
\end{tikzpicture}
```

The basic engine to manage key–value pairs is `pgfkeys` which is part of PGF. This engine always has a key name and a key "path", which is somehow similar to file name and directory of files. The common "directory" (key path) of PGFPLOTS is '`/pgfplots/`'. Although the key definitions below provide this full path, it is always (well, almost always) enough to skip this prefix – PGFPLOTS uses it automatically. The same holds for the prefixes '`/tikz/`' which are common for all Ti*k*Z drawing options and '`/pgf/`' which are for the (more or less) low–level commands of PGF. All these prefixes can be omitted.

One important concept is the concept of styles. A style is a key which contains one or more other keys. It can be redefined or modified until it is actually used by the internal routines. Each single component of Ti*k*Z and PGFPLOTS can be configured with styles.

For example,

```
\pgfplotsset{every axis/.append style={line width=1pt}}
```

sets the line width for every axis to `1pt`.

There are several other styles predefined to modify the appearance, see section 4.17.

`\pgfplotsset{`⟨*key-value-list*⟩`}`

Defines or sets all options in `{`⟨*key-value-list*⟩`}`. The ⟨*key-value-list*⟩ can contain any of the options in this manual which have the prefix `/pgfplots/` (you do not need to type the prefix).

It is a shortcut for `\pgfqkeys{/pgfplots}{`⟨*key-value-list*⟩`}`, that means it inserts the prefix `/pgfplots` to any option which has no full path.

This command can be used to define default options for the complete document or a part of the document. For example,

```
\pgfplotsset{
    cycle list={%
        {red, mark=*}, {blue,mark=*},
        {red, mark=x}, {blue,mark=x},
        {red, mark=square*}, {blue,mark=square*},
        {red, mark=triangle*}, {blue,mark=triangle*},
        {red, mark=diamond*}, {blue,mark=diamond*},
        {red, mark=pentagon*}, {blue,mark=pentagon*}
    },
    legend style={
        at={(0.5,-0.2)},
        anchor=north,
        legend columns=2,
        cells={anchor=west},
        font=\footnotesize,
        rounded corners=2pt,
    },
    xlabel=$x$,ylabel=$f(x)$
}
```

can be used to set document–wise styles for line specifications, the legend's style and axis labels.

You can also define new styles (collections of key–value–pairs) with `/.style` and `/.append style`.

```
\pgfplotsset{
    My Style 1/.style={xlabel=$x$, legend entries={1,2,3} },
    My Style 2/.style={xlabel=$X$, legend entries={4,5,6} }
}
```

The `/.style` and `/.append style` key handlers are described in section 4.17 in more detail.

Key handler ⟨*key*⟩/.code={⟨*TEX code*⟩}

Occasionally, the PGFPLOTS user interface offers to replace parts of its routines. This is accomplished using so called "code keys". What it means is to replace the original key and its behavior with new {⟨*TEX code*⟩}. Inside of {⟨*TEX code*⟩}, any command can be used. Furthermore, the #1 pattern will be the argument provided to the key.

I've been invoked with 'this here'

```
\pgfplotsset{
    My Code/.code={I've been invoked with '#1'}}
\pgfplotsset{My Code={this here}}
```

The example defines a (new) key named `My Code`. Essentially, it is nothing else but a `\newcommand`, plugged into the key-value interface. The second statement "invokes" the code key.

Key handler ⟨*key*⟩/.code 2 args={⟨*TEX code*⟩}

As `/.code`, but this handler defines a key which accepts two arguments. When the so defined key is used, the two arguments are available as `#1` and `#2`.

Key handler ⟨*key*⟩/.cd

Each key has a fully qualified name with a (long) prefix, like `/pgfplots/xmin`. However, if the "current directory" is `/pgfplots`, it suffices to write just `xmin`. The `/.cd` key handler changes the "current directory" in this way.

The prefixes `/tikz/` and `/pgfplots/` are checked automatically for any argument provided to `\begin{axis}[`⟨*options*⟩`]` or `\addplot`. So, you won't need to worry about them, just leave them away – and look closer in case the package doesn't identify the option.

4.3.1 Pgfplots Options and Ti*k*Z Options

This section is more or less technical and can be skipped unless one really wants to know more about this topic.

Ti*k*Z options and PGFPLOTS options can be mixed inside of the axis arguments and in any of the associated styles. For example,

```
\pgfplotsset{every axis legend/.append style={
    legend columns=3,font=\Large}}
```

assigns the '`legend columns`' option (a PGFPLOTS option) and uses '`font`' for drawing the legend (a Ti*k*Z option). The point is: `legend columns` needs to be known *before* the legend is typeset whereas `font` needs to be active when the legend is typeset. PGFPLOTS sorts out any key dependencies automatically:

The axis environments will process any known PGFPLOTS options, and all '`every`'–styles will be parsed for PGFPLOTS options. Every unknown option is supposed to be a Ti*k*Z option and will be forward to the associated Ti*k*Z drawing commands. For example, the '`font=\Large`' above will be used as argument to the legend matrix, and the '`font=\Large`' argument in

```
\pgfplotsset{every axis label/.append style={
    ylabel=Error,xlabel=Dof,font=\Large}}
```

will be used in the nodes for axis labels (but not the axis title, for example).

It is an error if you assign incompatible options to axis labels, for example '`xmin`' and '`xmax`' can't be set inside of '`every axis label`'.

4.4 Two Dimensional Plot Types

PGFPLOTS supports several two-dimensional line-plots like piecewise linear line plots, piecewise constant plots, smoothed plots, bar plots and comb plots. Most of them use the PGF plot handler library directly, see [5, section 18.8].

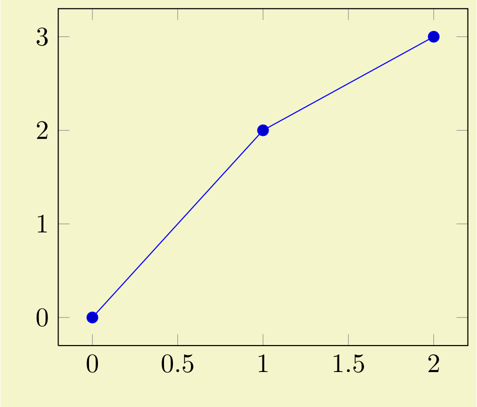
Plot types are part of the plot style, so they are set with options. Most of the basic 2d plot types are part of Ti*k*Z, see [5, section 18.8], and are probably known to users of Ti*k*Z. They are documented here as well.

4.4.1 Linear Plots

/tikz/sharp plot (no value)

`\addplot+[sharp plot]`

Linear ('sharp') plots are the default. Point coordinates are simply connected by straight lines.



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}
    \addplot+[sharp plot] coordinates
        {(0,0) (1,2) (2,3)};
\end{axis}
\end{tikzpicture}
```

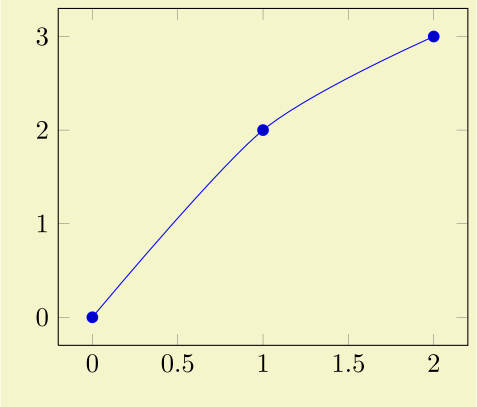
The '+' here means to use the normal plot cycle list and append '`sharp plot`' to its option list.

4.4.2 Smooth Plots

`/tikz/smooth` (no value)

`\addplot+[smooth]`

Smooth plots interpolate smoothly between successive points.



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}
    \addplot+[smooth] coordinates
        {(0,0) (1,2) (2,3)};
\end{axis}
\end{tikzpicture}
```

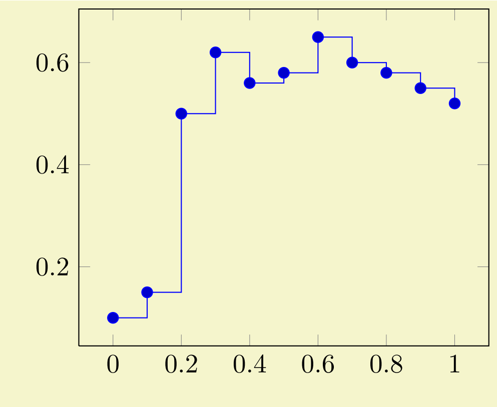
4.4.3 Constant Plots

Constant plots draw lines parallel to the x-axis to connect coordinates. The discontinuous edges may be drawn or not, and marks may be placed on left or right ends.

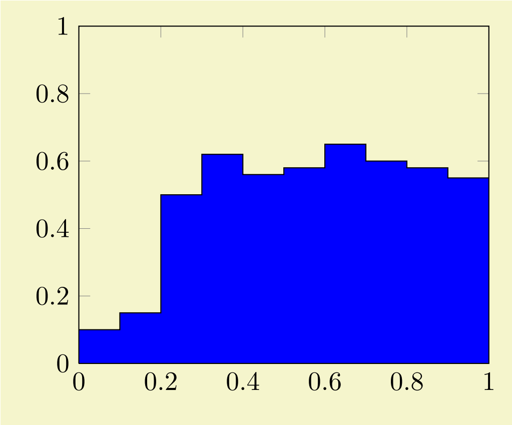
`/tikz/const plot` (no value)

`\addplot+[const plot]`

Connects all points with horizontal and vertical lines. Marks are placed left-handed on horizontal line segments, causing the plot to be right-sided continuous at all data points.



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}
\addplot+[const plot]
coordinates
{(0,0.1)    (0.1,0.15)  (0.2,0.5)   (0.3,0.62)
 (0.4,0.56) (0.5,0.58)  (0.6,0.65)  (0.7,0.6)
 (0.8,0.58) (0.9,0.55)  (1,0.52)};
\end{axis}
\end{tikzpicture}
```



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}[ymin=0,ymax=1,enlargelimits=false]
\addplot
    [const plot,fill=blue,draw=black]
coordinates
{(0,0.1)    (0.1,0.15)  (0.2,0.5)   (0.3,0.62)
 (0.4,0.56) (0.5,0.58)  (0.6,0.65)  (0.7,0.6)
 (0.8,0.58) (0.9,0.55)  (1,0.52)}
    \closedcycle;
\end{axis}
\end{tikzpicture}
```

`/tikz/const plot mark left` (no value)

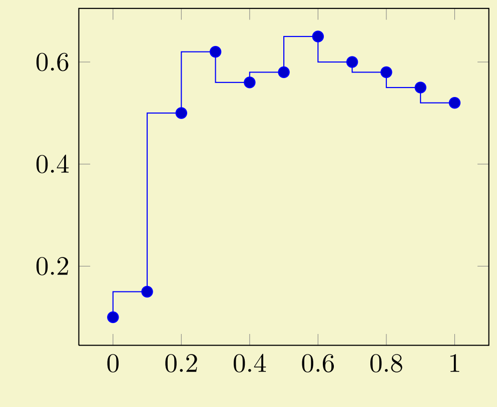
`\addplot+[const plot mark left]`

An alias for '`const plot`'.

`/tikz/const plot mark right` (no value)

`\addplot+[const plot mark right]`

A variant which places marks on the right of each line segment, causing plots to be left-sided continuous at coordinates.

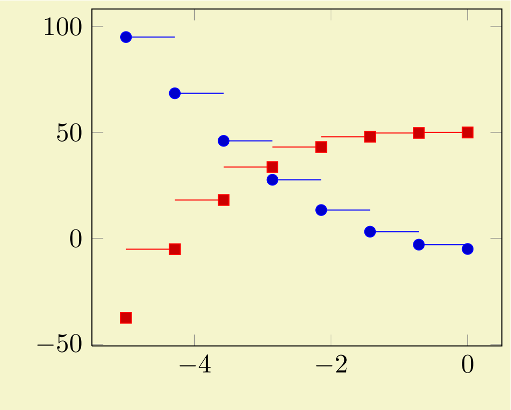


```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}
\addplot+[const plot mark right]
coordinates
{(0,0.1)    (0.1,0.15)  (0.2,0.5)   (0.3,0.62)
 (0.4,0.56) (0.5,0.58)  (0.6,0.65)  (0.7,0.6)
 (0.8,0.58) (0.9,0.55)  (1,0.52)};
\end{axis}
\end{tikzpicture}
```

`/tikz/jump mark left` (no value)

`\addplot+[jump mark left]`

A variant of '`const plot mark left`' which does not draw vertical lines.



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}[samples=8]
\addplot+[jump mark left,domain=-5:0]
    {4*x^2 - 5};

\addplot+[jump mark right,domain=-5:0]
    {0.7*x^3 + 50};
\end{axis}
\end{tikzpicture}
```

/tikz/jump mark right (no value)

\addplot+[jump mark right]

A variant of '`const plot mark right`' which does not draw vertical lines.

4.4.4 Bar Plots

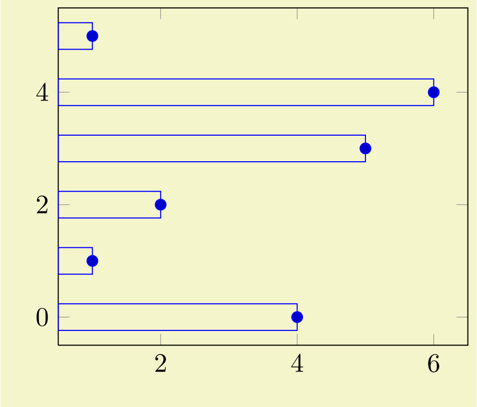
Bar plots place horizontal or vertical bars at coordinates. Multiple bar plots in one axis can be stacked on top of each other or aligned next to each other.

/tikz/xbar (no value)

\addplot+[xbar]

Places horizontal bars between the $(y = 0)$ line and each coordinate.

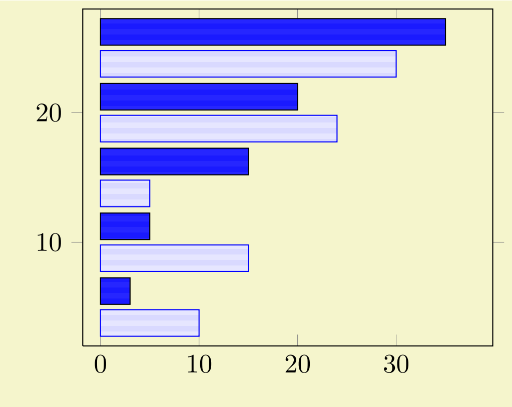
This option is used on a per-plot basis and configures only the visualization of coordinates. The figure-wide style `/pgfplots/xbar` also sets reasonable options for ticks, legends and multiple plots.



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}
\addplot+[xbar] coordinates
    {(4,0) (1,1) (2,2)
     (5,3) (6,4) (1,5)};
\end{axis}
\end{tikzpicture}
```

Bars are centered at plot coordinates with width `bar width`. Using bar plots usually means more than just a different way of how to connect coordinates, for example to draw ticks outside of the axis, change the legend's appearance or introduce shifts if multiple `\addplot` commands appear.

There is a preconfigured style for `xbar` which is installed automatically if you provide `xbar` as argument to the axis environment which provides this functionality.



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}[xbar,enlargelimits=0.15]
\addplot
[draw=blue,pattern=horizontal lines light blue]
coordinates
    {(10,5) (15,10) (5,15) (24,20) (30,25)};

\addplot
[draw=black,pattern=horizontal lines dark blue]
coordinates
    {(3,5) (5,10) (15,15) (20,20) (35,25)};
\end{axis}
\end{tikzpicture}
```

Here `xbar` yields `/pgfplots/xbar` because it is an argument to the axis, not to a single plot.

Besides line-, fill- and colorstyles, bars can be configured with `bar width` and `bar shift`, see below.

`/pgfplots/xbar={`⟨*shift for multiple plots*⟩`}` (style, default `2pt`)

This style sets `/tikz/xbar` *and* some commonly used options concerning horizontal bars for the complete axis. This is automatically done if you provide `xbar` as argument to an axis argument, see above.

The `xbar` style defines shifts if multiple plots are placed into one axis. It draws bars adjacent to each other, separated by `{`⟨*shift for multiple plots*⟩`}`. Furthermore, it sets the style `bar cycle list` and sets tick and legend appearance options.

The style is defined as follows.

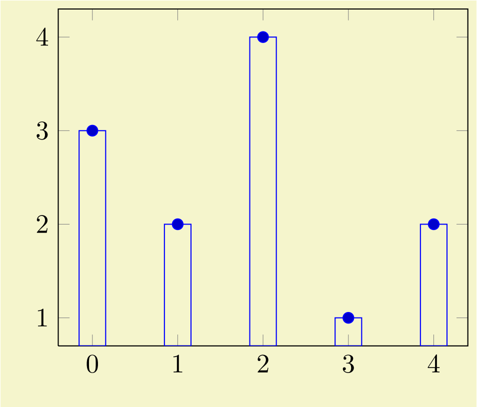
```
\pgfplotsset{
    /pgfplots/xbar/.style={
        /tikz/xbar,
        bar cycle list,
        tick align=outside,
        xbar legend,
        /pgf/bar shift={%
            %  total width = n*w + (n-1)*skip
            %  i.e. subtract half for centering
            -0.5*(\numplots*\pgfplotbarwidth + (\numplots-1)*#1)  +
            %  the '0.5*w' is for centering
            (.5+\plotnum)*\pgfplotbarwidth + \plotnum*#1%
        },
    },
}
```

The formular for `bar shift` assigns shifts dependent on the total number of plots and the current plot's number. It is designed to fill a total width of $n \cdot$`bar width`$+(n-1)\cdot$`{`⟨*shift for multiple plots*⟩`}`. The 0.5 compensates for centering.

`/tikz/ybar` (no value)

`\addplot+[ybar]`

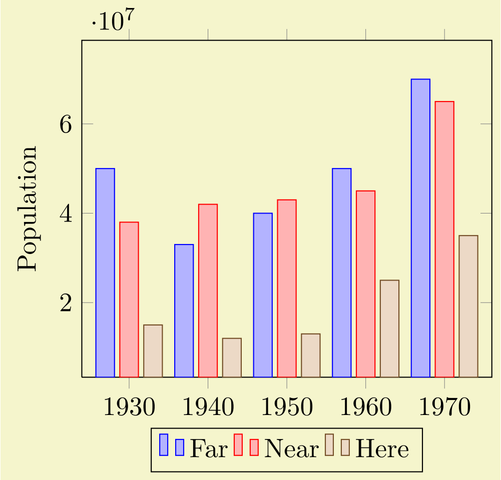
Like `xbar`, this option generates bar plots. It draws vertical bars between the ($x = 0$) line and each input coordinate.



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}
\addplot+[ybar] plot coordinates
    {(0,3) (1,2) (2,4) (3,1) (4,2)};
\end{axis}
\end{tikzpicture}
```

The example above simply changes how input coordinates shall be visualized. As mentioned for `xbar`, one usually needs modified legends and shifts for multiple bars in the same axis.

There is a predefined style which installs these customizations when provided to the axis-environment:

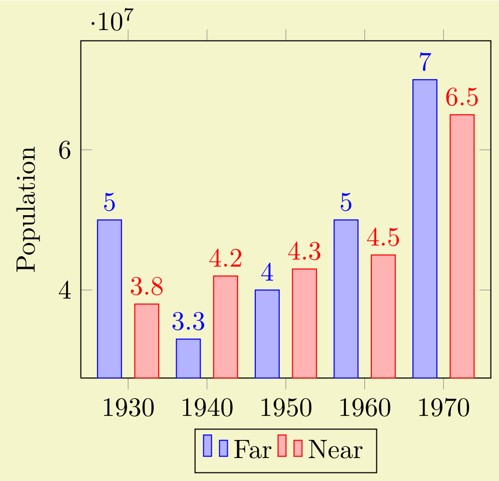


```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}[
    x tick label style={
        /pgf/number format/1000 sep=},
    ylabel=Population,
    enlargelimits=0.15,
    legend style={at={(0.5,-0.15)},
        anchor=north,legend columns=-1},
    ybar,
    bar width=7pt,
]
\addplot
    coordinates {(1930,50e6) (1940,33e6)
         (1950,40e6) (1960,50e6) (1970,70e6)};

\addplot
    coordinates {(1930,38e6) (1940,42e6)
        (1950,43e6) (1960,45e6) (1970,65e6)};

\addplot
    coordinates {(1930,15e6) (1940,12e6)
        (1950,13e6) (1960,25e6) (1970,35e6)};
\legend{Far,Near,Here}
\end{axis}
\end{tikzpicture}
```

Here `ybar` yields `/pgfplots/ybar` because it is an argument to the axis, not to a single plot.

As for `xbar`, the bar width and shift can be configured with `bar width` and `bar shift`. However, the bar shift is better provided as argument to `/pgfplots/ybar` since this style will overwrite the bar shift. Thus, prefer `/pgfplots/ybar=4pt` to set the bar shift.

Sometimes it is useful to write the y values directly near the bars. This can be realized using the `nodes near coords` method:



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}[
    x tick label style={
        /pgf/number format/1000 sep=},
    ylabel=Population,
    enlargelimits=0.15,
    legend style={at={(0.5,-0.15)},
        anchor=north,legend columns=-1},
    ybar=5pt,% configures ‘bar shift’
    bar width=9pt,
    nodes near coords,
    point meta=y *10^-7 % the displayed number
]
\addplot
    coordinates {(1930,50e6) (1940,33e6)
         (1950,40e6) (1960,50e6) (1970,70e6)};

\addplot
    coordinates {(1930,38e6) (1940,42e6)
        (1950,43e6) (1960,45e6) (1970,65e6)};

\legend{Far,Near}
\end{axis}
\end{tikzpicture}
```

/pgfplots/ybar={⟨*shift for multiple plots*⟩} (style, default 2pt)

As /pgfplots/xbar, this style sets the /tikz/ybar option to draw vertical bars, but it also provides commonly used options for vertical bars.

If you supply ybar to an axis environment, /pgfplots/ybar will be chosen instead of /tikz/ybar.

It changes the legend, draws ticks outside of the axis lines and draws multiple \addplot arguments adjacent to each other; block–centered at the x coordinate and separated by {⟨*shift for multiple plots*⟩}. It will also install the bar shift for every node near coord. Furthermore, it installs the style bar cycle list. It is defined similarly to /pgfplots/xbar.

/pgfplots/bar cycle list (no value)

A style which installs cycle lists for multiple bar plots.

```
\pgfplotsset{
    /pgfplots/bar cycle list/.style={/pgfplots/cycle list={%
        {blue,fill=blue!30!white,mark=none},%
        {red,fill=red!30!white,mark=none},%
        {brown!60!black,fill=brown!30!white,mark=none},%
        {black,fill=gray,mark=none},%
        }
    },
}
```

/pgf/bar width={⟨*dimension*⟩} (initially 10pt)

Configures the width used by xbar and ybar. It is accepted to provide mathematical expressions.

/pgf/bar shift={⟨*dimension*⟩} (initially 0pt)

Configures a shift for xbar and ybar. Use bar shift together with bar width to draw multiple bar plots into the same axis. It is accepted to provide mathematical expressions.

/tikz/ybar interval (no value)

\addplot+[ybar interval]

This plot type produces vertical bars with width (and shift) relatively to intervals of coordinates.

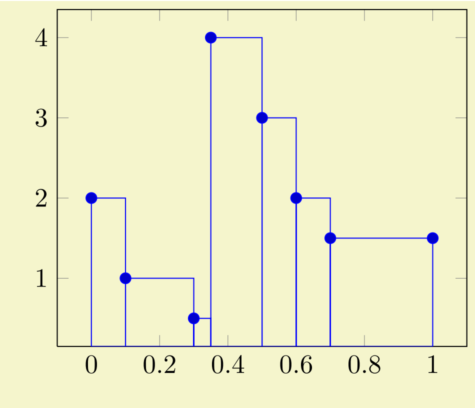
There is one conceptional difference when working with intervals: an interval is defined by *two* coordinates. Since ybar has one value for each interval, the ith bar is defined by

1. the y value of the ith coordinates,
2. the x value of the ith coordinate as left interval boundary,

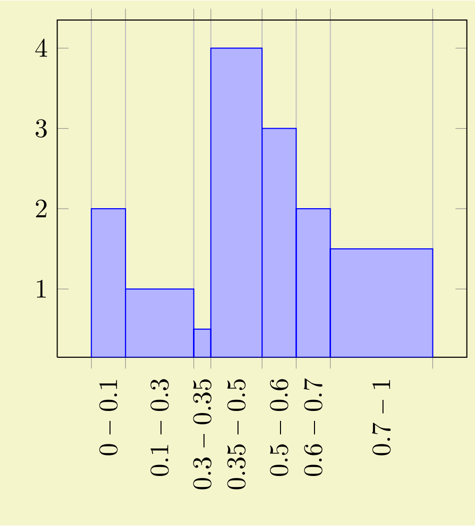
3. the x value of the $(i + 1)$th coordinate as right interval boundary.

Consequently, there is *one coordinate too much*: the last coordinate will *only* be used to determine the interval width; its y value doesn't influence the bar appearance.

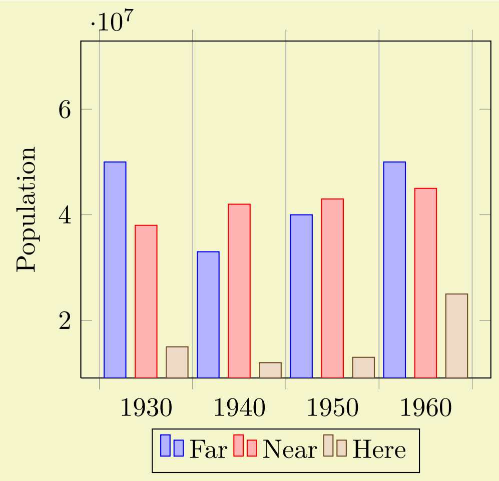
It is installed on a per-plot basis and configures *only* the visualization of coordinates. See the style `/pgfplots/ybar interval` which configures the appearance of the complete figure.



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}
\addplot+[ybar interval] plot coordinates
    {(0,2) (0.1,1) (0.3,0.5) (0.35,4) (0.5,3)
     (0.6,2) (0.7,1.5) (1,1.5)};
\end{axis}
\end{tikzpicture}
```



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}[ybar interval,
    xtick=data,
    xticklabel interval boundaries,
    x tick label style=
        {rotate=90,anchor=east}
    ]
\addplot coordinates
    {(0,2) (0.1,1) (0.3,0.5) (0.35,4) (0.5,3)
     (0.6,2) (0.7,1.5) (1,1.5)};
\end{axis}
\end{tikzpicture}
```



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}[
    x tick label style={
        /pgf/number format/1000 sep=},
    ylabel=Population,
    enlargelimits=0.05,
    legend style={at={(0.5,-0.15)},
        anchor=north,legend columns=-1},
    ybar interval=0.7,
]
\addplot
    coordinates {(1930,50e6) (1940,33e6)
         (1950,40e6) (1960,50e6) (1970,70e6)};

\addplot
    coordinates {(1930,38e6) (1940,42e6)
        (1950,43e6) (1960,45e6) (1970,65e6)};

\addplot
    coordinates {(1930,15e6) (1940,12e6)
        (1950,13e6) (1960,25e6) (1970,35e6)};
\legend{Far,Near,Here}
\end{axis}
\end{tikzpicture}
```

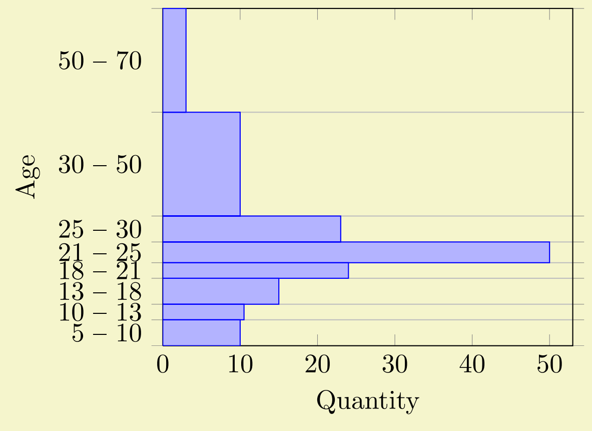
/pgfplots/**ybar interval**={⟨*relative width*⟩} (style, default 1)

A style which is intended to install options for `ybar interval` for a complete figure. This includes tick and legend appearance, management of multiple bar plots in one figure and a more adequate `cycle list` using the style `bar cycle list`.

/tikz/**xbar interval** (no value)

\addplot+[**xbar interval**]

As `ybar interval`, just for horizontal bars.



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}[
    xmin=0,xmax=53,
    ylabel=Age,
    xlabel=Quantity,
    enlargelimits=false,
    ytick=data,
    yticklabel interval boundaries,
    xbar interval,
]
\addplot
    coordinates {(10,5) (10.5,10) (15,13)
        (24,18) (50,21) (23,25) (10,30)
        (3,50) (3,70)};
\end{axis}
\end{tikzpicture}
```

/pgfplots/**xbar interval**={⟨*relative width*⟩} (style, default 1)

A style which is intended to install options for `xbar interval` for a complete figure, see the style `/pgfplots/ybar interval` for details.

/pgfplots/**xticklabel interval boundaries** (no value)
/pgfplots/**yticklabel interval boundaries** (no value)
/pgfplots/**zticklabel interval boundaries** (no value)

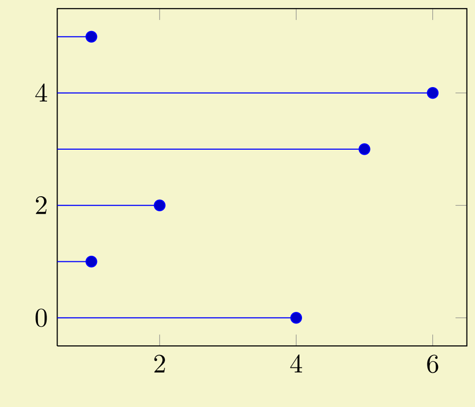
These are style keys which set `x tick label as interval` (see page 166 for details) and configure the tick appearance to be {⟨*start*⟩} – {⟨*end*⟩} for each tick interval.

4.4.5 Comb Plots

Comb plots are very similar to bar plots except that they employ single horizontal/vertical lines instead of rectangles.

/tikz/**xcomb** (no value)

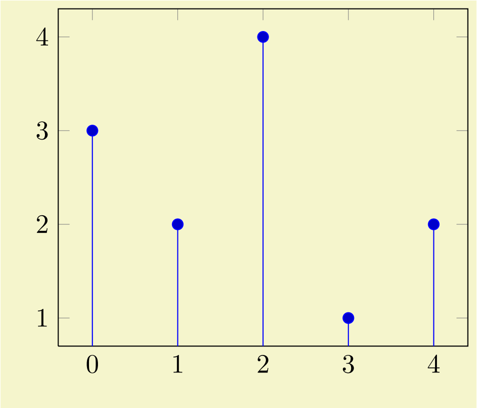
\addplot+[**xcomb**]



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}
\addplot+[xcomb] coordinates
    {(4,0) (1,1) (2,2)
     (5,3) (6,4) (1,5)};
\end{axis}
\end{tikzpicture}
```

/tikz/**ycomb** (no value)

\addplot+[ycomb]

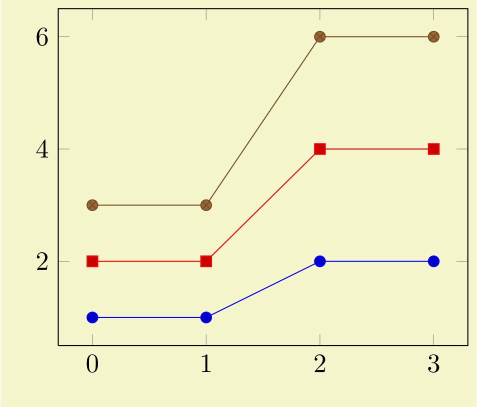


```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}
\addplot+[ycomb] plot coordinates
    {(0,3) (1,2) (2,4) (3,1) (4,2)};
\end{axis}
\end{tikzpicture}
```

4.4.6 Stacked Plots

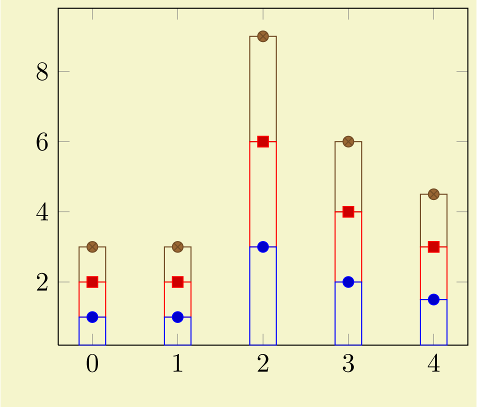
/pgfplots/stack plots=x|y|false (initially false)

Allows stacking of plots in either x or y direction. Stacking means to add either x- or y coordinates of successive \addplot commands on top of each other.

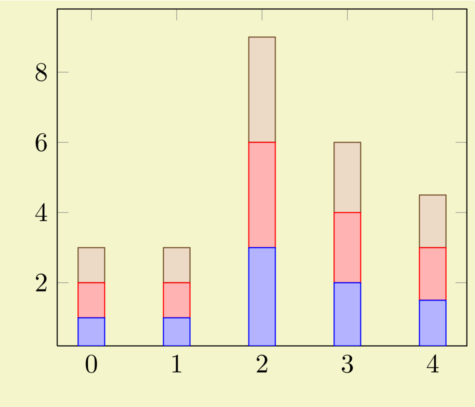


```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{axis}[stack plots=y]
    \addplot coordinates
        {(0,1) (1,1) (2,2) (3,2)};
    \addplot coordinates
        {(0,1) (1,1) (2,2) (3,2)};
    \addplot coordinates
        {(0,1) (1,1) (2,2) (3,2)};
    \end{axis}
\end{tikzpicture}
```

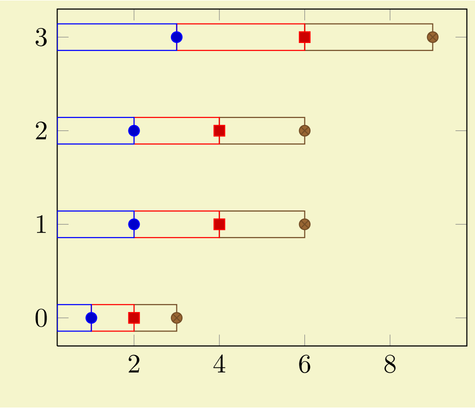
stack plots is particularly useful for bar plots. The following examples demonstrate its functionality. Normally, it is advisable to use the styles ybar stacked and xbar stacked which also set some other options.



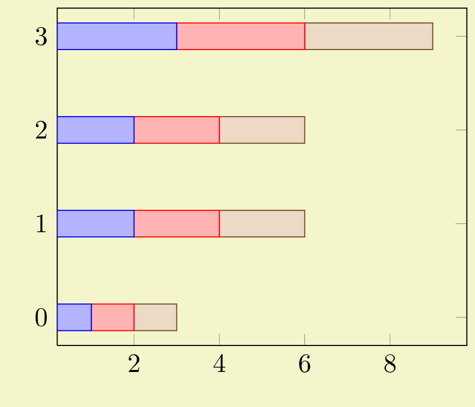
```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{axis}[stack plots=y,/tikz/ybar]
    \addplot coordinates
        {(0,1) (1,1) (2,3) (3,2) (4,1.5)};
    \addplot coordinates
        {(0,1) (1,1) (2,3) (3,2) (4,1.5)};
    \addplot coordinates
        {(0,1) (1,1) (2,3) (3,2) (4,1.5)};
    \end{axis}
\end{tikzpicture}
```



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{axis}[ybar stacked]
    \addplot coordinates
        {(0,1) (1,1) (2,3) (3,2) (4,1.5)};
    \addplot coordinates
        {(0,1) (1,1) (2,3) (3,2) (4,1.5)};
    \addplot coordinates
        {(0,1) (1,1) (2,3) (3,2) (4,1.5)};
    \end{axis}
\end{tikzpicture}
```



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{axis}[stack plots=x,/tikz/xbar]
    \addplot coordinates
        {(1,0) (2,1) (2,2) (3,3)};
    \addplot coordinates
        {(1,0) (2,1) (2,2) (3,3)};
    \addplot coordinates
        {(1,0) (2,1) (2,2) (3,3)};
    \end{axis}
\end{tikzpicture}
```



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}

\begin{tikzpicture}
    \begin{axis}[xbar stacked]
    \addplot coordinates
        {(1,0) (2,1) (2,2) (3,3)};
    \addplot coordinates
        {(1,0) (2,1) (2,2) (3,3)};
    \addplot coordinates
        {(1,0) (2,1) (2,2) (3,3)};
    \end{axis}
\end{tikzpicture}
```

The current implementation for `stack plots` does *not* interpolate missing coordinates. That means stacking will fail if the plots have different grids.

`/pgfplots/stack dir=plus|minus` (initially `plus`)

Configures the direction of `stack plots`. The value `plus` will add coordinates of successive plots while `minus` subtracts them.

`/pgfplots/reverse stacked plots=true|false` (initially `true, default true`)

Configures the sequence in which stacked plots are drawn. This is more or less a technical detail which should not be changed in any normal case.

The motivation is as follows: suppose multiple `\addplot` commands are stacked on top of each other and they are processed in the order of appearance. Than, the second plot could easily draw its lines (or fill area) on top of the first one - hiding its marker or line completely. Therefor, PGFPLOTS reverses the sequence of drawing commands.

This has the side-effect that any normal Ti*k*Z-paths inside of an axis will also be processed in reverse sequence.

/pgfplots/xbar stacked=plus|minus (style, default plus)

A figure-wide style which enables stacked horizontal bars (i.e. xbar and stack plots=x). It also adjusts the legend and tick appearance and assigns a useful cycle list.

/pgfplots/ybar stacked=plus|minus (style, default plus)

A figure-wide style which enables stacked vertical bars (i.e. ybar and stack plots=y). It also adjusts the legend and tick appearance and assigns a useful cycle list.

/pgfplots/xbar interval stacked=plus|minus (style, default plus)

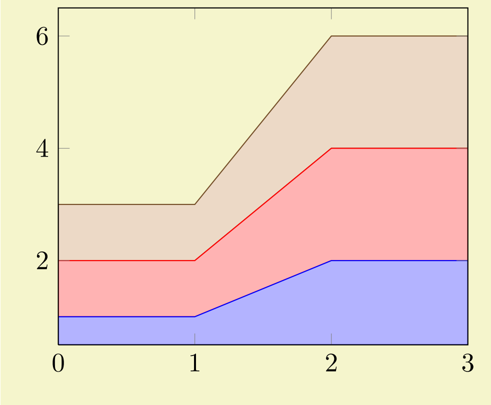
A style similar to /pgfplots/xbar stacked for the interval based bar plot variant.

/pgfplots/ybar interval stacked=plus|minus (style, default plus)

A style similar to /pgfplots/ybar stacked for the interval based bar plot variant.

4.4.7 Area Plots

Area plots are a combination of \closedcycle and stack plots. They can be combined with any other plot type.



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{axis}[
        stack plots=y,
        area style,
        enlarge x limits=false]
    \addplot coordinates
        {(0,1) (1,1) (2,2) (3,2)}
        \closedcycle;
    \addplot coordinates
        {(0,1) (1,1) (2,2) (3,2)}
        \closedcycle;
    \addplot coordinates
        {(0,1) (1,1) (2,2) (3,2)}
        \closedcycle;
    \end{axis}
\end{tikzpicture}
```

Area plots may need modified legends, for example using the area legend key. Furthermore, one may want to consider the axis on top key such that filled areas do not overlap ticks and grid lines.

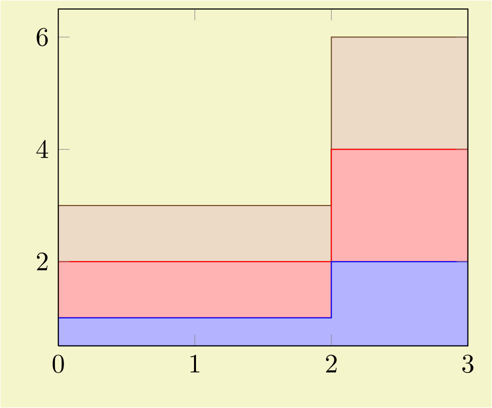
/pgfplots/area style (style, no value)

A style which sets

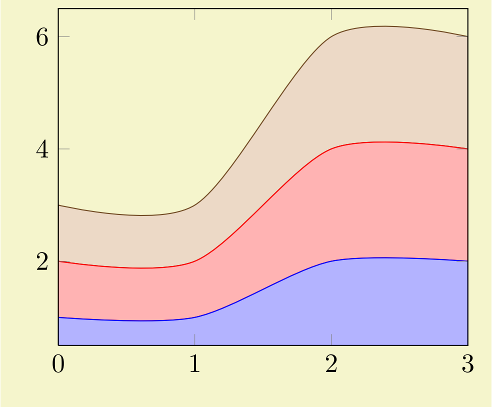
```
\pgfplotsset{
    /pgfplots/area style/.style={%
        area cycle list,
        area legend,
        axis on top,
    }}
```

/pgfplots/area cycle list (style, no value)

A style which installs a cycle list suitable for area plots. The initial configuration of this style simply invokes the bar cycle list which does also provide filled plot styles.



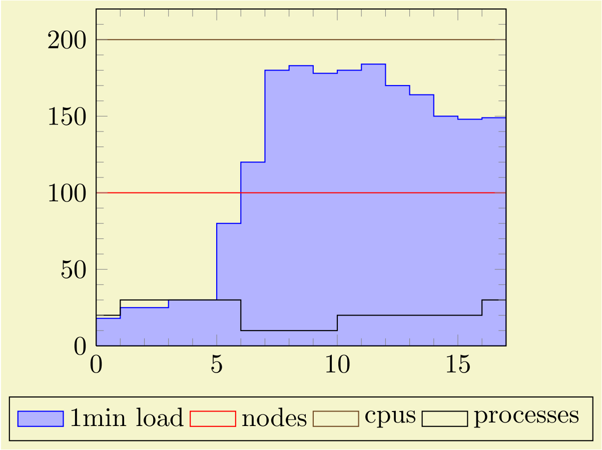
```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{axis}[
        const plot,
        stack plots=y,
        area style,
        enlarge x limits=false]
    \addplot coordinates
        {(0,1) (1,1) (2,2) (3,2)}
        \closedcycle;
    \addplot coordinates
        {(0,1) (1,1) (2,2) (3,2)}
        \closedcycle;
    \addplot coordinates
        {(0,1) (1,1) (2,2) (3,2)}
        \closedcycle;
    \end{axis}
\end{tikzpicture}
```



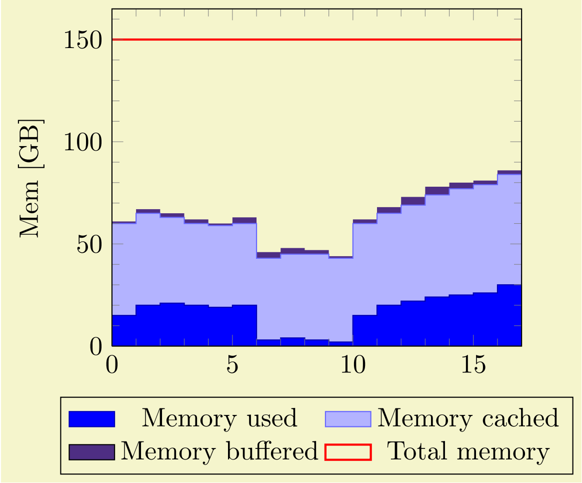
```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{axis}[
        smooth,
        stack plots=y,
        area style,
        enlarge x limits=false]
    \addplot coordinates
        {(0,1) (1,1) (2,2) (3,2)}
        \closedcycle;
    \addplot coordinates
        {(0,1) (1,1) (2,2) (3,2)}
        \closedcycle;
    \addplot coordinates
        {(0,1) (1,1) (2,2) (3,2)}
        \closedcycle;
    \end{axis}
\end{tikzpicture}
```

time	1minload	nodes	cpus	processes	memused	memcached	membuf	memtotal
0	18	100	200	20	15	45	1	150
1	25	100	200	30	20	45	2	150
2	25	100	200	30	21	42	2	150
3	30	100	200	30	20	40	2	150
4	30	100	200	30	19	40	1	150
5	80	100	200	30	20	40	3	150
6	120	100	200	10	3	40	3	150
7	180	100	200	10	4	41	3	150
8	183	100	200	10	3	42	2	150
9	178	100	200	10	2	41	1	150
10	180	100	200	20	15	45	2	150
11	184	100	200	20	20	45	3	150
12	170	100	200	20	22	47	4	150
13	164	100	200	20	24	50	4	150
14	150	100	200	20	25	52	3	150
15	148	100	200	20	26	53	2	150
16	149	100	200	30	30	54	2	150
17	154	100	200	30	35	55	1	150

```
\pgfplotstableread{pgfplots.timeseries.dat}\loadedtable
\pgfplotstabletypeset\loadedtable
```



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\pgfplotstableread
    {pgfplots.timeseries.dat}
    {\loadedtable}

\begin{tikzpicture}
    \begin{axis}[
        ymin=0,
        minor tick num=4,
        enlarge x limits=false,
        axis on top,
        every axis plot post/.append style=
            {mark=none},
        const plot,
        legend style={
            area legend,
            at={(0.5,-0.15)},
            anchor=north,
            legend columns=-1}]

    \addplot[draw=blue,fill=blue!30!white]
     table[x=time,y=1minload] from \loadedtable
        \closedcycle;
    \addplot table[x=time,y=nodes] from \loadedtable;
    \addplot table[x=time,y=cpus] from \loadedtable;
    \addplot table[x=time,y=processes]
        from \loadedtable;
    \legend{1min load,nodes,cpus,processes}
    \end{axis}
\end{tikzpicture}
```



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\pgfplotstableread{pgfplots.timeseries.dat}\loadedtable

\begin{tikzpicture}
    \begin{axis}[
        ymin=0,
        minor tick num=4,
        enlarge x limits=false,
        const plot,
        axis on top,
        stack plots=y,
        cycle list={%
            {blue!70!black,fill=blue},%
            {blue!60!white,fill=blue!30!white},%
            {draw=none,fill={rgb:red,138;green,82;blue,232}},%
            {red,thick}%
        },
        ylabel={Mem [GB]},
        legend style={
            area legend,
            at={(0.5,-0.15)},
            anchor=north,
            legend columns=2}]

    \addplot table[x=time,y=memused]      from \loadedtable \closedcycle;
    \addplot table[x=time,y=memcached]    from \loadedtable \closedcycle;
    \addplot table[x=time,y=membuf]       from \loadedtable \closedcycle;
    \addplot+[stack plots=false]
             table[x=time,y=memtotal]     from \loadedtable;
    \legend{Memory used,Memory cached,Memory buffered,Total memory}
    \end{axis}
\end{tikzpicture}
```

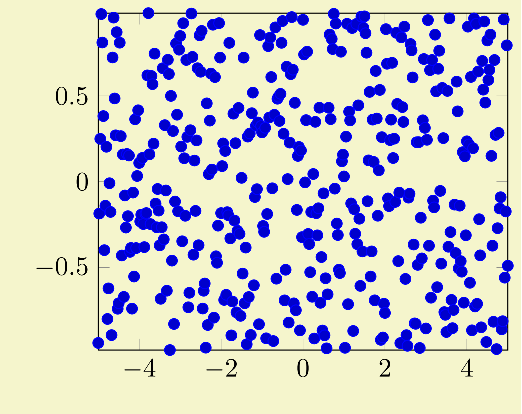
4.4.8 Scatter Plots

The most simple scatter plots produce the same as the line plots above – but they contain only markers. They are enabled using the `only marks` key of Ti*k*Z.

`/tikz/only marks` (no value)

`\addplot+[only marks]`

Draws a simple scatter plot: all markers have the same appearance.



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{axis}[enlargelimits=false]
    \addplot+[only marks,samples=400]
        {rand};
    \end{axis}
\end{tikzpicture}
```

The `only marks` visualization style simply draws marks at every coordinate. Marks can be set with `mark=`⟨*mark name*⟩ and marker options like size and color can be specified using the `mark options={`⟨*style options*⟩`}` key (or by modifying the `every mark` style). The available markers along with the accepted style options can be found in section 4.6 on page 80.

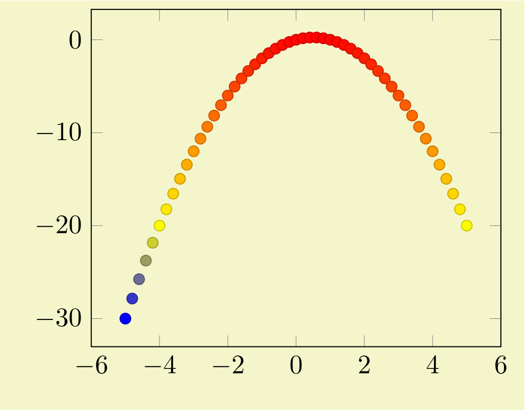
More sophisticated scatter plots change the marker appearance for each data point. An example is that marker colors depend on the magnitude of function values $f(x)$ or other provided coordinates. The term “scatter plot” will be used for this type of plots in the following sections.

Scatter plots require “source” coordinates. These source coordinates can be the y coordinate, or explicitly provided additional values.

/pgfplots/scatter (no value)

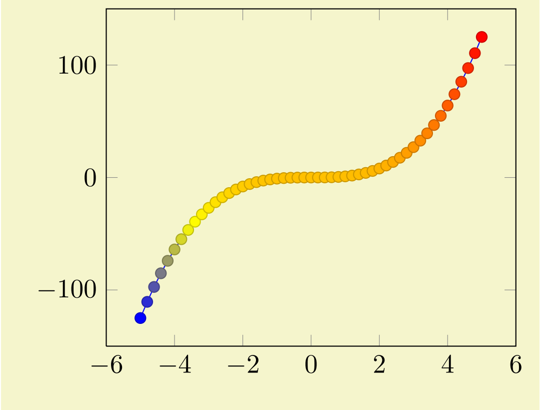
\addplot+[scatter]

Enables marker appearance modifications. The default implementation acquires “source coordinates” for every data point (see scatter src below) and maps them linearly into the current color map. The resulting color is used as draw and fill color of the marker.



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{axis}
    \addplot+[scatter,only marks,
        samples=50,scatter src=y]
       {x-x^2};
    \end{axis}
\end{tikzpicture}
```

The key scatter is simply a boolean variable which enables marker modifications. It applies only to markers and it can be combined with any other plot type.



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{axis}
    \addplot+[scatter,
        samples=50,scatter src=y]
       {x^3};
    \end{axis}
\end{tikzpicture}
```

Scatter plots can be configured using a set of options. One of them is mandatory, the rest allows fine grained control over marker appearance options.

/pgfplots/scatter src=none|⟨*expression*⟩|x|y|z|f(x)|explicit|explicit symbolic (initially none)

This key is necessary for any scatter plot and it is set to f(x) as soon as scatter is activated and no different choice has been made. It needs to be provided as {⟨*option*⟩} for \addplot to configure the value used to determine marker appearances. Actually, scatter src is nothing but an alias for point meta, so the main documentation for this key is on page 97. However, we summarize the choices here together with scatter plot examples.

Usually, scatter src provides input data (numeric or string type) which is used to determine colors and other style options for markers. The default configuration expects numerical data which is mapped linearly into the current color map.

The value of scatter src determines how to get this data: the choices x, y and z will use either the x, y or z coordinates to determine marker options. Any coordinate filters, logarithms or stacked-plot computations have already been applied to these values (use rawx, rawy and rawz for unprocessed values). The special choice f(x) is the same as y for two dimensional plots and the same as z for three dimensional plots. The choice explicit expects the scatter source data as additional coordinate from the coordinate input streams (see section 4.2.1 for how to provide input meta data or below for some small examples). They will be treated as numerical data. The choice explicit symbolic also expects scatter source data as additional meta information for each input coordinate, but it treats them as strings, not as numerical data. Consequently, no arithmetics is performed. It is task of the scatter plot style to do something with it. See, for example, the scatter/classes style below. Finally, it is possible to provide an arbitrary mathematical expression which involves zero, one or more of the values x (the current x coordinate), y (the current y coordinate) or z (the current z coordinate, if any).

If data is read from tables, mathematical expressions might also involve \thisrow{⟨*column name*⟩} or \thisrowno{⟨*column index*⟩} to access any of the table cells in the current row.

Here are examples for how to provide data for the choices explicit and explicit symbolic.

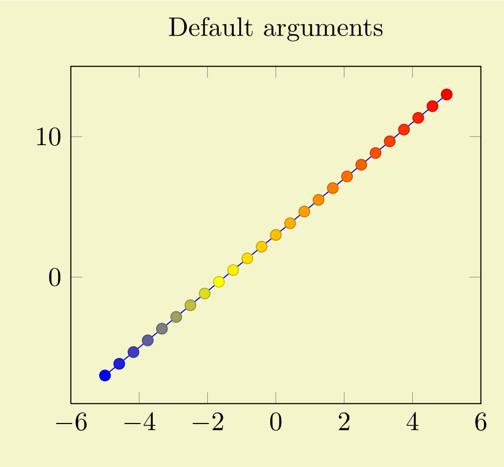
```
\begin{tikzpicture}
    \begin{axis}
        %  provide color data explicitly using [<data>]
        %  behind coordinates:
        \addplot+[scatter,scatter src=explicit]
            coordinates {
                (0,0) [1.0e10]
                (1,2) [1.1e10]
                (2,3) [1.2e10]
                (3,4) [1.3e10]
                %  ...
            };

        %  Assumes a datafile.dat like
        %  xcolname  ycolname    colordata
        %  0         0           0.001
        %  1         2           0.3
        %  2         2.1         0.4
        %  3         3           0.5
        %  ...
        %  the file may have more columns.
        \addplot+[scatter,scatter src=explicit]
            table[x=xcolname,y=ycolname,meta=colordata]
                {datafile.dat};
        %  Same data as last example:
        \addplot+[scatter,scatter src=\thisrow{colordata}+\thisrow{ycolname}]
            table[x=xcolname,y=ycolname]
                {datafile.dat};

        %  Assumes a datafile.dat like
        %  0         0           0.001
        %  1         2           0.3
        %  2         2.1         0.4
        %  3         3           0.5
        %  ...
        %  the first three columns will be used here:
        \addplot+[scatter,scatter src=explicit]
            file {datafile.dat};
    \end{axis}
\end{tikzpicture}
```

Please note that scatter src≠none results in computational work even if scatter=false.

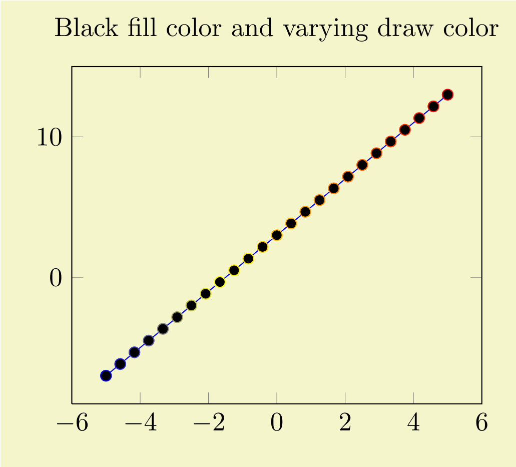
/pgfplots/scatter/use mapped color={⟨*options for each marker*⟩} (style, initially draw=mapped color!80!black,fill=mapped color)

This style is installed by default. When active, it recomputes the color mapped color for every processed point coordinate by transforming the scatter src coordinates into the current color map linearly. Then, it evaluates the options provided as {⟨*options for each marker*⟩} which are expected to depend on mapped color.

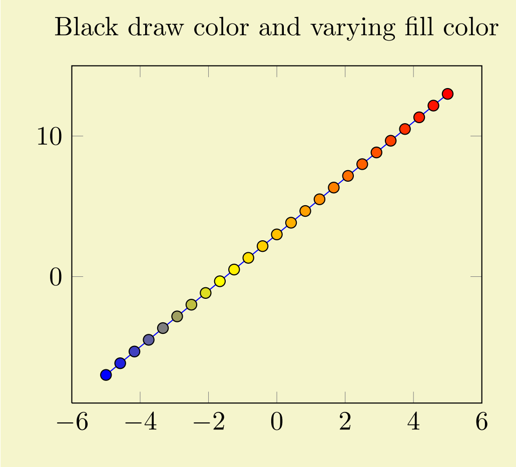
The user interface for color maps is described in section 4.6.5.



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}[title=Default arguments]
\addplot+[scatter,scatter src=y]
    {2*x+3};
\end{axis}
\end{tikzpicture}
```



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}[
    title=Black fill color and varying draw color,
    scatter/use mapped color=
        {draw=mapped color,fill=black}]
\addplot+[scatter,scatter src=y]
    {2*x+3};
\end{axis}
\end{tikzpicture}
```



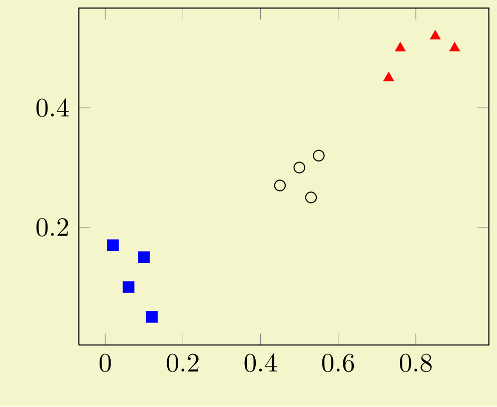
```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}[
    title=Black draw color and varying fill color,
    scatter/use mapped color=
        {draw=black,fill=mapped color}]
\addplot+[scatter,scatter src=y]
    {2*x+3};
\end{axis}
\end{tikzpicture}
```

This key is actually a style which redefines `@pre marker code` and `@post marker code` (see below).

Remark: The style `use mapped color` *re*defines `@pre marker code` and `@post marker code`. There is a starred variant `use mapped color*` which *appends* the functionality while keeping the old marker code.

/pgfplots/`scatter/classes`={⟨*styles for each class name*⟩}

A scatter plot style which visualizes points using several classes. The style assumes that every point coordinate has a class label attached, that means the choice `scatter src=explicit symbolic` is assumed[13]. A class label can be a number, but it can also be a symbolic constant. Given class labels for every point, {⟨*styles for each class name*⟩} contains a comma-separated list which associates appearance options to each class label.



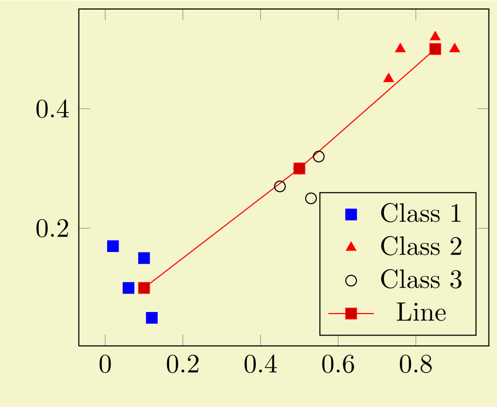
```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}[scatter/classes={
    a={mark=square*,blue},%
    b={mark=triangle*,red},%
    c={mark=o,draw=black}}]

    %  \addplot[] is better than \addplot+[] here:
    %  it avoids scalings of the cycle list
    \addplot[scatter,only marks,
        scatter src=explicit symbolic]
        coordinates {
            (0.1,0.15)  [a]
            (0.45,0.27) [c]
            (0.02,0.17) [a]
            (0.06,0.1)  [a]
            (0.9,0.5)   [b]
            (0.5,0.3)   [c]
            (0.85,0.52) [b]
            (0.12,0.05) [a]
            (0.73,0.45) [b]
            (0.53,0.25) [c]
            (0.76,0.5)  [b]
            (0.55,0.32) [c]
        };
\end{axis}
\end{tikzpicture}
```

In this example, the coordinate `(0.1,0.15)` has the associated label '`a`' while `(0.45,0.27)` has the label '`c`' (see section 4.2 for details about specifying point meta data). Now, The argument to `scatter/classes` contains styles for every label – for label '`a`', square markers will be drawn in color blue.

The generation of a legend works as for a normal plot – but `scatter/classes` requires one legend entry for every provided class. It communicates the class labels to the legend automatically. It works as if there had been different `\addplot` commands, one for every class label.

It is also possible to provide `scatter/classes` as argument to a single plot, allowing different scatter plots in one axis.

[13] If `scatter src` is not `explicit symbolic`, we expect a numeric argument which is rounded to the nearest integer. The resulting integer is used a class label. If that fails, the numeric argument is truncated to the nearest integer. If that fails as well, the point has no label.



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}[legend pos=south east]
    % The data file contains:
    % x     y      label
    % 0.1   0.15   a
    % 0.45  0.27   c
    % 0.02  0.17   a
    % 0.06  0.1    a
    % 0.9   0.5    b
    % 0.5   0.3    c
    % 0.85  0.52   b
    % 0.12  0.05   a
    % 0.73  0.45   b
    % 0.53  0.25   c
    % 0.76  0.5    b
    % 0.55  0.32   c
    \addplot[
        scatter/classes={
            a={mark=square*,blue},%
            b={mark=triangle*,red},%
            c={mark=o,draw=black,fill=black}%
        },
        scatter,only marks,
        scatter src=explicit symbolic]
    table[x=x,y=y,meta=label]
            {plotdata/scattercl.dat};

    \addplot coordinates
        {(0.1,0.1) (0.5,0.3) (0.85,0.5)};
    \legend{Class 1,Class 2,Class 3,Line}
\end{axis}
\end{tikzpicture}
```

In general, the format of {⟨*styles for each class name*⟩} is a comma separated list of ⟨*label*⟩={⟨*style options*⟩}.

Attention: The keys `every mark` and `mark options` have *no effect* when used inside of {⟨*styles for each class name*⟩}! So, instead of assigning `mark options`, you can simply provide the options directly. They apply only to markers anyway.

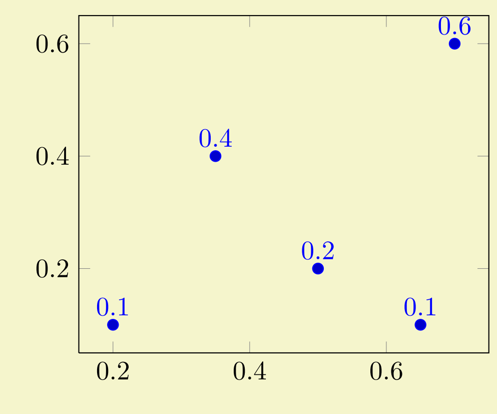
Remark: To use `\label` and `\ref` in conjunction with `scatter/classes`, you can provide the class labels as optional arguments to `\label` in square brackets:

```
\addplot[
    scatter/classes={
        a={mark=square*,blue},%
        b={mark=triangle*,red},%
        c={mark=o,draw=black,fill=black}%
    },
    scatter,only marks,
    scatter src=explicit symbolic]
    %  [and coordinate input here... ]
    ;

\label[a]{label:for:first:class}
\label[b]{label:for:second:class}
\label[c]{label:for:third:class}


...
First class is \ref{label:for:first:class}, second is \ref{label:for:second:class}.
```

Remark: The style `scatter/classes` *re*defines `@pre marker code` and `@post marker code`. There is a starred variant `scatter/classes*` which *appends* the functionality while keeping the old marker code.

/pgfplots/`nodes near coords={⟨`*content*`⟩}` (default `\pgfmathprintnumber\pgfplotspointmeta`)

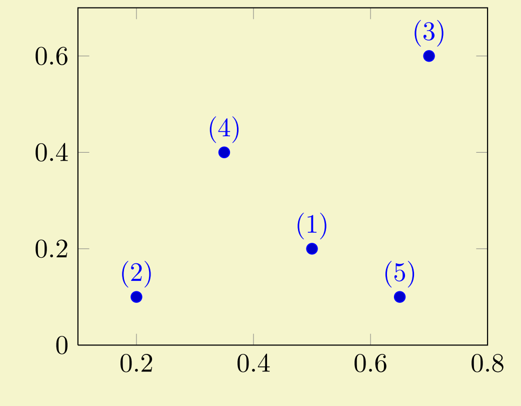
A scatter plot style which places text nodes near every coordinate.



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}[nodes near coords]
    \addplot+[only marks] coordinates {
        (0.5,0.2) (0.2,0.1) (0.7,0.6)
        (0.35,0.4) (0.65,0.1)};
\end{axis}
\end{tikzpicture}
```

The {⟨*content*⟩} is, if nothing else has been specified, the content of the "point meta", displayed using the default ⟨*content*⟩=\pgfmathprintnumber{\pgfplotspointmeta}. The macro \pgfplotspointmeta contains whatever has been selected by the point meta key, it defaults to the y coordinate for two dimensional plots and the z coordinate for three dimensional plots.

Since point meta=explicit symbolic allows to treat string data, you can provide textual descriptions which will be shown inside of the generated nodes[14]:



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}[nodes near coords,enlargelimits=0.2]
    \addplot+[only marks,
        point meta=explicit symbolic]
    coordinates {
        (0.5,0.2) [(1)]
        (0.2,0.1) [(2)]
        (0.7,0.6) [(3)]
        (0.35,0.4) [(4)]
        (0.65,0.1) [(5)]
    };
\end{axis}
\end{tikzpicture}
```

The square brackets are the way to provide explicit point meta for plot coordinates. Please refer to the documentation of plot file and plot table for how to get point meta from files.

The style nodes near coords might be useful for bar plots, see ybar for an example of nodes near coords.

Remarks and Details:

- nodes near coords uses the same options for line styles and colors as the current plot. This may be changed using the style every node near coord, see below.
- nodes near coords is actually one of the scatter plot styles. It redefines scatter/@pre marker code to generate several Ti*k*Z \node commands.

  In order to use nodes near coords together with other scatter plot styles (like scatter/use mapped color or scatter/classes), you may append a star to each of these keys. The variant nodes near coords* will *append* code to scatter/@pre marker code without overwriting the previous value.
- Consider using enlargelimits together with nodes near coords if text is clipped away.
- Currently nodes near coords does not work satisfactory for ybar interval or xbar interval, sorry.

[14] In this case, the \pgfmathprintnumber will be skipped automatically.

/pgfplots/`every node near coord` (style, no value)

A style used for every node generated by `nodes near coords`. It is initially empty.

/pgfplots/`nodes near coords align={⟨alignment method⟩}` (initially `auto`)

Specifies how to align nodes generated by `nodes near coords`.

Possible choices for `{⟨alignment⟩}` are

auto Uses `horizontal` if the x coordinates are shown or `vertical` in all other cases. This checks the current value of `point meta`.

horizontal uses `left` if `\pgfplotspointmeta` < 0 and `right` otherwise.

vertical uses `below` if `\pgfplotspointmeta` < 0 and `above` otherwise.

Its also possible to provide any Ti*k*Z alignment option such as `anchor=north east`, `below` or something like that. It is also allowed if multiple options are provided.

/pgfplots/`scatter/@pre marker code/.code={⟨...⟩}`
/pgfplots/`scatter/@post marker code/.code={⟨...⟩}`

These two keys constitute the public interface which determines the marker appearance depending on scatter source coordinates.

Redefining them allows fine grained control even over marker types, line styles and colors.

The scatter plot algorithm works as follows:

1. The scatter source coordinates form a data stream whose data limits are computed additionally to the axis limits. This step is skipped for `symbolic` meta data.
2. Before any markers are drawn, a linear coordinate transformation from these data limits to the interval $[0.0, 1000.0]$ is initialised.
3. Every scatter source coordinate[15] will be transformed linearly and the result is available as macro `\pgfplotspointmetatransformed` $\in [0.0, 1000.0]$.

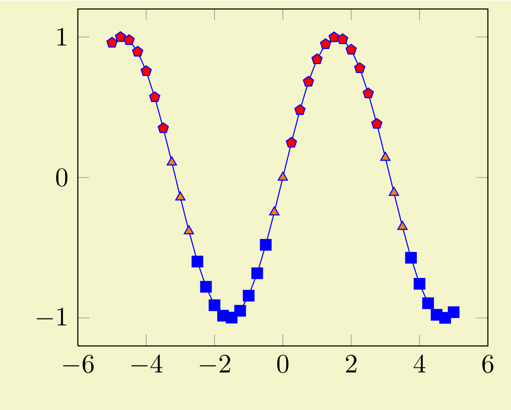
   The decision is thus based on per thousands of the data range. The transformation is skipped for `symbolic` meta data (and the meta data is simply contained in the mentioned macro).
4. The PGF coordinate system is translated such that `(0pt,0pt)` is the plot coordinate.
5. The code of `scatter/@pre marker code` is evaluated (without arguments).
6. The standard code which draws markers is evaluated.
7. The code of `scatter/@post marker code` is evaluated (without arguments).

The idea is to generate a set of appearance keys which depends on `\pgfplotspointmetatransformed`. Then, a call to `\scope[⟨generated keys⟩]` as `@pre` code and the associated `\endscope` as `@post` code will draw markers individually using `[⟨generated keys⟩]`.

A technical example is shown below. It demonstrates how to write user defined routines, in this case a three–class system[16].

[15] During the evaluation, the public macros `\pgfplotspointmeta` and `\pgfplotspointmetarange` indicate the source coordinate and the source coordinate range in the format $a : b$ (for log–axis, they are given in fixed point representation and for linear axes in floating point).

[16] Please note that you don't need to copy this particular example: the multiple–class example is also available as predefined style `scatter/classes`.



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
%  Low-Level scatter plot interface Example:
%  use three different marker classes
%  0% - 30%   : first class
%  30% - 60%  : second class
%  60% - 100% : third class
\begin{axis}[
scatter/@pre marker code/.code={%
   \ifdim\pgfplotspointmetatransformed pt<300pt
      \def\markopts{mark=square*,fill=blue}%
   \else
      \ifdim\pgfplotspointmetatransformed pt<600pt
         \def\markopts{mark=triangle*,fill=orange}%
      \else
         \def\markopts{mark=pentagon*,fill=red}%
      \fi
   \fi
   \expandafter\scope\expandafter[\markopts]
},%
scatter/@post marker code/.code={%
   \endscope
}]

\addplot+[scatter,scatter src=y,
    samples=40]
   {sin(deg(x))};

\end{axis}
\end{tikzpicture}
```

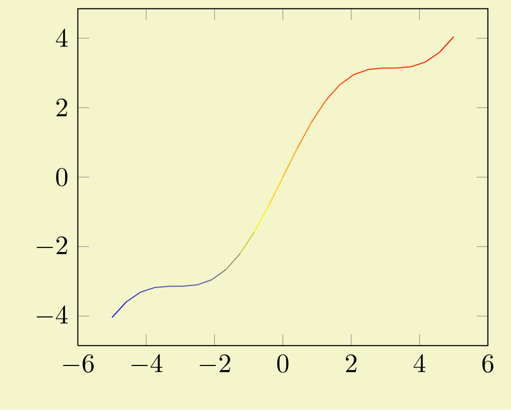
Please note that `\ifdim` compares TEX lengths, so the example employs the suffix `pt` for any number used in this context. That doesn't change the semantics. The two (!) `\expandafter` constructions make sure that `\scope` is invoked with the *content* of `\markopts` instead of the macro name `\markopts`.

4.4.9 1D Colored Mesh Plots

/pgfplots/mesh (no value)

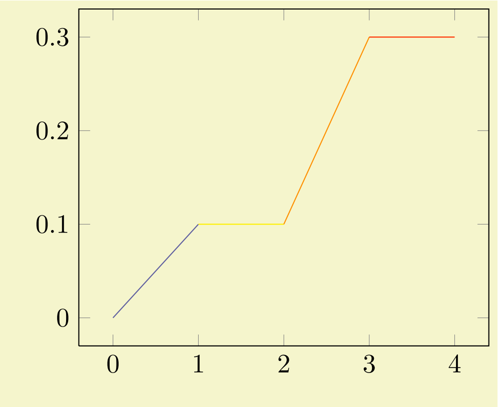
\addplot+[mesh]

Uses the current color map to determine colors for each fixed line segment. Each line segment will get the same color.



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{axis}
    \addplot[mesh] {x+sin(deg(x))};
    \end{axis}
\end{tikzpicture}
```

The color data is per default the y value of the plot. It can be reconfigured using the `point meta` key (which is actually the same as `scatter src`). The following example provides the color data explicitly for `plot coordinates`, using the square bracket notation.



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{axis}
    \addplot[mesh,point meta=explicit]
        coordinates {
            (0,0)   [0]
            (1,0.1) [1]
            (2,0.1) [2]
            (3,0.3) [3]
            (4,0.3) [4]
        };
    \end{axis}
\end{tikzpicture}
```

This onedimensional `mesh` plot is actually a special case of the twodimensional mesh plots, so more detailed configuration, including how to change the color data, can be found in section 4.5.5.

4.4.10 Interrupted Plots

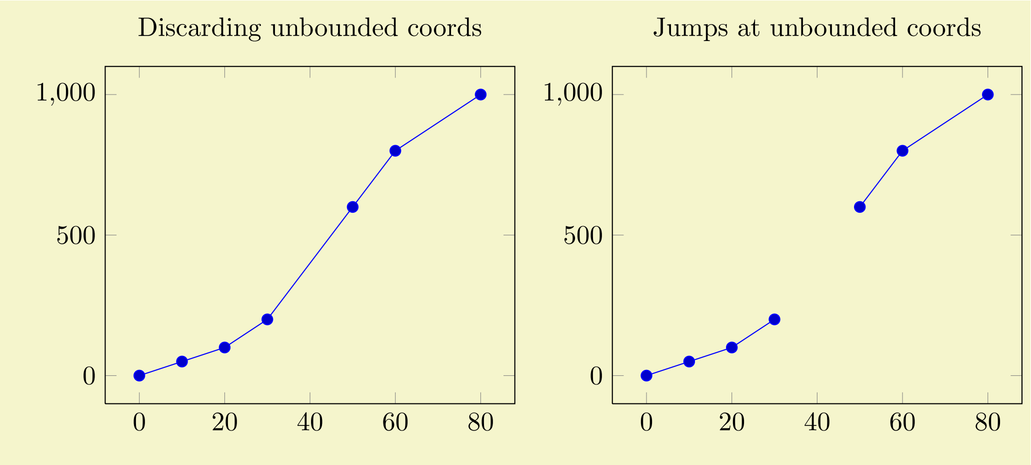
Sometimes it is desirable to draw parts of a single plot separately, without connection between the parts (discontinuities). This can be achieved using the `unbounded coords` key combined with coordinate values `nan`, `inf` or `-inf`.

`/pgfplots/unbounded coords=discard|jump` (initially `discard`)

This key configures what to do if one or more coordinates of a single point are unbounded. Here, unbounded means it is either ±∞ (`+inf` or `-inf`) or it has the special “not–a–number” value `nan`.

The initial setting `discard` discards the complete point and a warning is issued in the log file[17]. This setting has the same effect as if the unbounded point did not occur: PGFPLOTS will interpolate between the bounded adjacent points.

The alternative `jump` allows interrupted plots: it provides extra checking for these coordinates and does not interpolate over them; only those line segments which are adjacent to unbounded coordinates will be skipped.

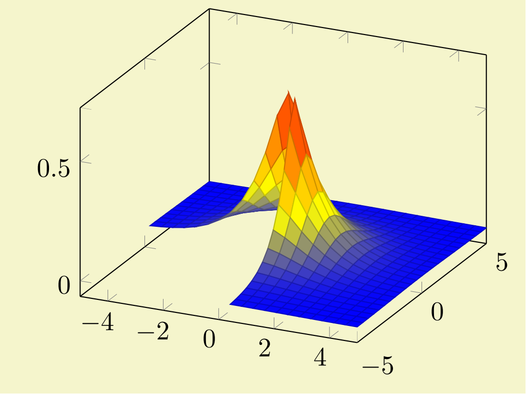


[17] The warning can be disabled with `filter discard warning=false`.

```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}[
    title=Discarding unbounded coords,
    unbounded coords=discard]

    \addplot coordinates {
        (0,0) (10,50) (20,100) (30,200)
        (40,inf) (50,600) (60,800) (80,1000)
    };
\end{axis}
\end{tikzpicture}
\begin{tikzpicture}
\begin{axis}[
    title=Jumps at unbounded coords,
    unbounded coords=jump]
    \addplot coordinates {
        (0,0) (10,50) (20,100) (30,200)
        (40,inf) (50,600) (60,800) (80,1000)
    };
\end{axis}
\end{tikzpicture}
```

For plot expression and its friends, it is more likely to get very large floating point numbers instead of `inf`. In this case, consider using the `restrict x to domain` key described on page 197.

The `unbounded coords=jump` method does also work for mesh/surface plots: every face adjacent to an unbounded coordinate will be discarded in this case. The following example sets up a (cryptic) coordinate filter which cuts out a quarter of the domain and replaces its values with `nan`:



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}[
  unbounded coords=jump,
  %  A technical filter to cut out
  %  the x<0 and y<0 edge.
  filter point/.code={%
    \pgfmathparse
      {\pgfkeysvalueof{/data point/x}<0}%
    \ifpgfmathfloatcomparison
      \pgfmathparse
        {\pgfkeysvalueof{/data point/y}<0}%
      \ifpgfmathfloatcomparison
        \pgfkeyssetvalue{/data point/x}{nan}%
      \fi
    \fi
  },
  ]
  \addplot3[surf] {exp(-sqrt(x^2 + y^2))};
\end{axis}
\end{tikzpicture}
```

4.5 Three Dimensional Plot Types

PGFPLOTS provides three dimensional visualizations like scatter, line, mesh or surface plots. This section explains the methods to provide input coordinates and how to use the different plot types.

4.5.1 Before You Start With 3D

Before we delve into the capabilities of PGFPLOTS for three dimensional visualization, let me start with some preliminary remarks. The reason to use PGFPLOTS for three dimensional plots are similar to those of normal, two dimensional plots: the possibility to get consistent fonts and document consistent styles combined with high–quality output.

While this works very nice for (not too complex) two dimensional plots, it requires considerably more effort than non–graphical documents. This is even more so for three dimensional plots. In other words: PGFPLOTS' three dimensional routines are slow. There are reasons for this and some of them may vanish in future versions. But one of these reasons is, that TeX has never been designed for complex visualisation techniques. Consider the image externalization routines mentioned in section 7.1, in particular the `external` library to reduce typesetting time. Besides the speed limitations, three dimensional plots reach memory limits easily. Therefor, the plot complexity of three dimensional plots is limited to relatively coarse resolutions. Section 7.1 also discusses methods to extend the initial TEX memory limits.

Another issue which arises in three dimensional visualization is depth. PGFPLOTS supports z buffering techniques up to a certain extend. It works pretty well for single scatter plots (`z buffer=sort`), mesh or surface plots (`z buffer=auto`) or parametric mesh and surface plots (`z buffer=sort`). However, it can't combine different `\addplot` commands, those will be drawn in the order of appearance. You may encounter the limitations sometimes. Maybe it will be improved in future versions.

If you decide that you need high complexity, speed and 100% reliable z buffers (depth information), you should consider using other visualization tools and return to PGFPLOTS in several years. If you can wait for a complex picture and you don't even see the limitations arising from z buffering limitations, you should use PGFPLOTS. Again, consider using the automatic picture externalization with the `external` library discussed in section 7.1.

Enough for now, let's continue.

4.5.2 The `\addplot3` Command: Three Dimensional Coordinate Input

`\addplot3[`⟨*options*⟩`]` ⟨*input data*⟩ ⟨*trailing path commands*⟩`;`

The `\addplot3` command is the main interface for any three dimensional plot. It works in the same way as its two dimensional variant `\addplot` which has been described in all detail in section 4.2 on page 20.

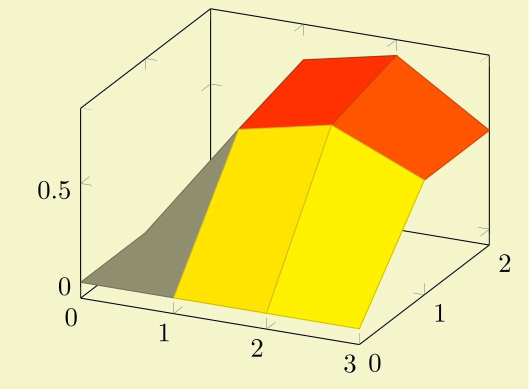
The `\addplot3` command accepts the same input methods as the `\addplot` variant, including expression plotting, coordinates, files and tables. However, a third coordinate is necessary for each of these methods which is usually straight–forward and is explained in all detail in the following.

Furthermore, `\addplot3` has a way to decide whether a *line* visualization or a *mesh* visualization has to be done. The first one is map from one dimension into $\mathbb{R}^3$ and the latter one a map from two dimensions to $\mathbb{R}^3$. Here, the keys `mesh/rows` and `mesh/cols` are used to define mesh sizes (matrix sizes). Usually, you don't have to care about that because the coordinate input routines already allow either one–dimensional or two dimensional structure.

`\addplot3 coordinates {`⟨*coordinate list*⟩`};`
`\addplot3[`⟨*options*⟩`] coordinates {`⟨*coordinate list*⟩`}` ⟨*trailing path commands*⟩`;`

The `\addplot3 coordinates` method works like its two–dimensional variant, `\addplot coordinates` which is described in all detail on page 22:

A long list of coordinates `(`⟨*x*⟩`,`⟨*y*⟩`,`⟨*z*⟩`)` is expected, separated by white spaces. The input list can be either an unordered series of coordinates, for example for scatter or line plots. It can also have matrix structure, in which case an empty input line (which is equivalent to "`\par`") marks the end of one matrix row. Matrix structure can also be provided if one of `mesh/rows` or `mesh/cols` is provided explicitly.



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{axis}
        % this yields a 3x4 matrix:
        \addplot3[surf] coordinates {
            (0,0,0) (1,0,0)   (2,0,0)   (3,0,0)

            (0,1,0) (1,1,0.6) (2,1,0.7) (3,1,0.5)

            (0,2,0) (1,2,0.7) (2,2,0.8) (3,2,0.5)
        };
    \end{axis}
\end{tikzpicture}
```

Here, `\addplot3` reads a matrix with three rows and four columns. The empty lines separate one row from the following.

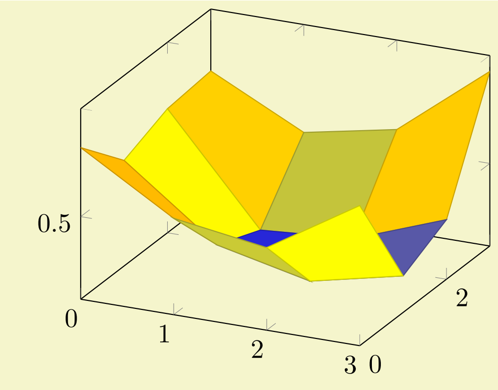
As for the two–dimensional `plot coordinates`, it is possible to provide (constant) mathematical expressions inside of single coordinates. The syntax `(`⟨*x*⟩`,`⟨*y*⟩`,`⟨*z*⟩`) [`⟨*meta*⟩`]` can be used just as for two dimensional `plot coordinates` to provide explicit color data; error bars are also supported.

\addplot3 file {⟨*name*⟩};
\addplot3[⟨*options*⟩] file {⟨*name*⟩} ⟨*trailing path commands*⟩;

The \addplot3 file input method is the same as \addplot file – it only expects one more coordinate. Thus, the input file contains x_i in the first column, y_i in the second column and z_i in the third.

A further column is read after z_i if point meta=explicit has been requested, see the documentation of \addplot file on page 23 for details.

As for \addplot3 coordinates, an empty line in the file marks the end of one matrix row.



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{axis}
        %  We have 'plotdata/first3d.dat' with
        % ---------
        %  0 0 0.8
        %  1 0 0.56
        %  2 0 0.5
        %  3 0 0.75
        %
        %  0 1 0.6
        %  1 1 0.3
        %  2 1 0.21
        %  3 1 0.3
        %
        %  0 2 0.68
        %  1 2 0.22
        %  2 2 0.25
        %  3 2 0.4
        %
        %  0 3 0.7
        %  1 3 0.5
        %  2 3 0.58
        %  3 3 0.9
        %  -> yields a 4x4 matrix:
        \addplot3[surf] file {plotdata/first3d.dat};
    \end{axis}
\end{tikzpicture}
```

For matrix data in files, it is important to specify the ordering in which the matrix entries have been written. The default configuration is mesh/ordering=x varies, so you need to change it to mesh/ordering=y varies in case you have columnwise ordering.

\addplot3 table [⟨*column selection*⟩]{⟨*file*⟩};
\addplot3[⟨*options*⟩] table [⟨*column selection*⟩]{⟨*file*⟩} ⟨*trailing path commands*⟩;

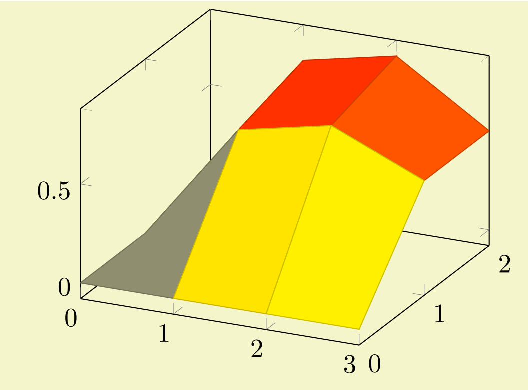
The \addplot3 table input works in the same way as its two dimensional counterpart \addplot table. It only expects a column for the z coordinates. Furthermore, it interprets empty input lines as end–of–row (more generally, end–of–scanline) markers, just as for plot file. The remarks above about the mesh/ordering applies here as well.

/pgfplots/mesh/rows={⟨*integer*⟩}
/pgfplots/mesh/cols={⟨*integer*⟩}

For visualization of mesh or surface plots which need some sort of matrix input, the dimensions of the input matrix need to be known in order to visualize the plots correctly. The matrix structure may be known from end–of–row marks (empty lines as general end–of–scanline markers in the input stream) as has been described above.

If the matrix structure is not yet known, it is necessary to provide at least one of mesh/rows or mesh/cols where mesh/rows indicates the number of samples for y coordinates whereas mesh/cols is the number of samples used for x coordinates (see also mesh/ordering).

Thus, the following example is also a valid method to define an input matrix.



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{axis}
        %  this yields also a 3x4 matrix:
        \addplot3[surf,mesh/rows=3] coordinates {
            (0,0,0) (1,0,0)   (2,0,0)   (3,0,0)
            (0,1,0) (1,1,0.6) (2,1,0.7) (3,1,0.5)
            (0,2,0) (1,2,0.7) (2,2,0.8) (3,2,0.5)
        };
    \end{axis}
\end{tikzpicture}
```

It is enough to supply one of `mesh/rows` or `mesh/cols` – the missing values will be determined automatically.

If you provide one of `mesh/rows` or `mesh/cols`, any end–of–row marker seen inside of input files or coordinate streams will be ignored.

`/pgfplots/mesh/scanline verbose=true|false` (initially `false`)

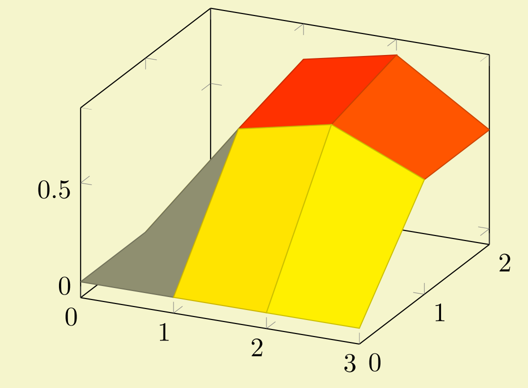
Provides debug messages in the LaTeX output about end–of–scanline markers.

The message will tell whether end–of–scanlines have been found and if they are the same.

`/pgfplots/mesh/ordering=x varies|y varies|rowwise|colwise` (initially `x varies`)

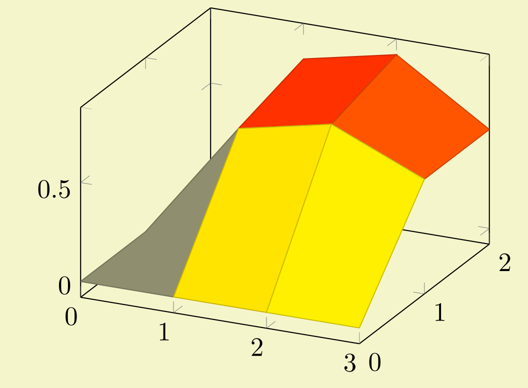
Allows to configure the sequence in which matrices (meshes) are read from `\addplot3 coordinates`, `\addplot3 file` or `\addplot3 table`.

Here, `x varies` means a sequence of points where n=`mesh/cols` successive points have the y coordinate fixed. This is intuitive when you write down a function because x is horizontal and y vertical. Note that in matrix terminology, x refers to *column indices* whereas y refers to *row indices*. Thus, `x varies` is equivalent to `rowwise` ordering in this sense. This is the initial configuration.



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}[mesh/ordering=x varies]
    %  this yields a 3x4 matrix in ‘x varies’
    %  ordering:
    \addplot3[surf] coordinates {
        (0,0,0) (1,0,0)   (2,0,0)   (3,0,0)

        (0,1,0) (1,1,0.6) (2,1,0.7) (3,1,0.5)

        (0,2,0) (1,2,0.7) (2,2,0.8) (3,2,0.5)
    };
\end{axis}
\end{tikzpicture}
```

Consequently, `mesh/ordering=y varies` provides points such that successive m=`mesh/rows` points form a column, i.e. the x coordinate is fixed and the y coordinate changes. In this sense, `y varies` is equivalent to `colwise` ordering, it is actually a matrix transposition.



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}[mesh/ordering=y varies]
    %  this yields a 3x4 matrix in colwise ordering:
    \addplot3[surf] coordinates {
        (0,0,0) (0,1,0)   (0,2,0)

        (1,0,0) (1,1,0.6) (1,2,0.7)

        (2,0,0) (2,1,0.7) (2,2,0.8)

        (3,0,0) (3,1,0.5) (3,2,0.5)
    };
\end{axis}
\end{tikzpicture}
```

Again, note the subtle difference to the common matrix indexing where a column has the second index fixed. PGFPLOTS refers to the way one would write down a function on a sheet of paper (this is consistent with how Matlab (tm) displays discrete functions with matrizes).

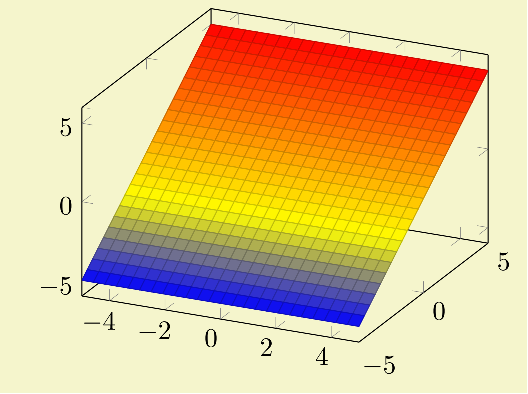
Please note that `shader=interp` relies on low level shadings which need to be given in row wise ordering, so a (potentially expensive) transposition of the data matrix will be performed in this case. If possible, supply your data in row wise ordering for `shader=interp`.

`\addplot3 {`⟨*math expression*⟩`} ;`
`\addplot3[`⟨*options*⟩`] {`⟨*math expression*⟩`}` ⟨*trailing path commands*⟩`;`

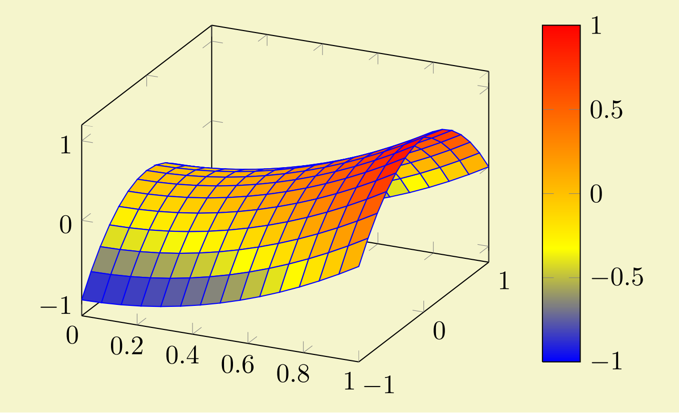
Expression plotting also works in the same way as for two dimensional plots. Now, however, a two dimensional mesh is sampled instead of a single line, which may depend on `x` and `y`.

The method `\addplot3 {`⟨*math expr*⟩`}` visualizes the function $f(x, y) =$⟨*math expr*⟩ where $f\colon [x_1, x_2] \times [y_1, y_2] \to \mathbb{R}$. The interval $[x_1, x_2]$ is determined using the `domain` key, for example using `domain=0:1`. The interval $[y_1, y_2]$ is determined using the `y domain` key. If `y domain` is empty, $[y_1, y_2] = [x_1, x_2]$ will be assumed. If `y domain=0:0` (or any other interval of length zero), it is assumed that the plot does not depend on `y` (thus, it is a line plot).

The number of samples in x direction is set using the `samples` key. The number of samples in y direction is set using the `samples y` key. If `samples y` is not set, the same value as for x is used. If `samples y`≤ 1, it is assumed that the plot does not depend on `y` (meaning it is a line plot).



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{axis}
        \addplot3[surf] {y};
    \end{axis}
\end{tikzpicture}
```



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{axis}[colorbar]
        \addplot3
            [surf,faceted color=blue,
             samples=15,
             domain=0:1,y domain=-1:1]
            {x^2 - y^2};
    \end{axis}
\end{tikzpicture}
```

Expression plotting sets `mesh/rows` and `mesh/cols` automatically; these settings don't have any effect for expression plotting.

```
\addplot3 expression {⟨math expr⟩};
\addplot3[⟨options⟩] expression {⟨math expr⟩} ⟨trailing path commands⟩;
```

The syntax

`\addplot3 {⟨math expression⟩};`

as short-hand equivalent for

`\addplot3 expression {⟨math expression⟩};`

```
\addplot3 (⟨x expression⟩,⟨y expression⟩,⟨z expression⟩) ;
\addplot3[⟨options⟩] (⟨x expression⟩,⟨y expression⟩,⟨z expression⟩) ⟨trailing path commands⟩;
```

A variant of `\addplot3 expression` which allows to provide different coordinate expressions for the x, y and z coordinates. This can be used to generate parameterized plots.

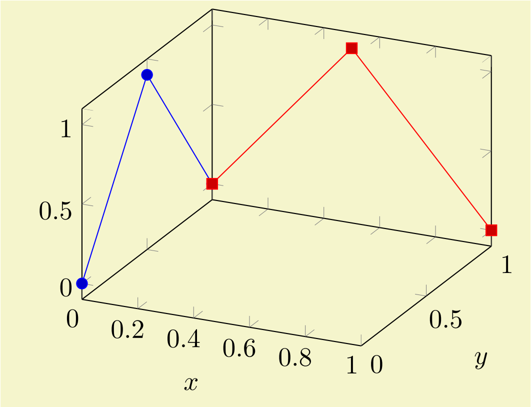
Please note that `\addplot (x,y,x^2)` is equivalent to `\addplot expression {x^2}`.

Note further that since the complete point expression is surrounded by round braces, round braces inside of ⟨*x expression*⟩, ⟨*y expression*⟩ or ⟨*z expression*⟩ need to be treated specially. Surround the expressions (which contain round braces) with curly braces:

`\addplot3 ({⟨x expr⟩}, {⟨y expr⟩}, {⟨z expr⟩});`

4.5.3 Line Plots

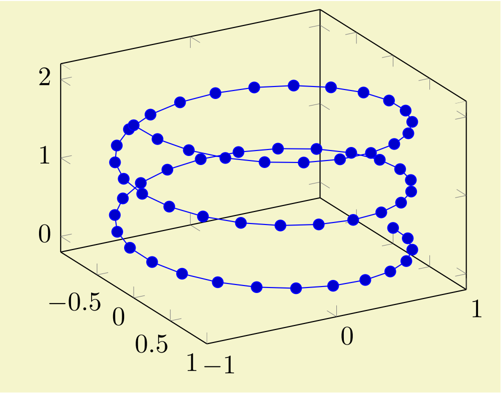
Three dimensional line plots are generated if the input source has no matrix structure. Line plots take the input coordinates and connect them in the order of appearance.



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{axis}[xlabel=$x$,ylabel=$y$]
    \addplot3 coordinates {(0,0,0) (0,0.5,1) (0,1,0)};
    \addplot3 coordinates {(0,1,0) (0.5,1,1) (1,1,0)};
    \end{axis}
\end{tikzpicture}
```

If there is no value for both, `mesh/rows` and `mesh/cols` or if one of them is `1`, PGFPLOTS will draw a line plot. This is also the case if there is no end–of–scanline marker (empty line) in the input stream.

For `\addplot3 expression`, this requires to set `samples y=0` to disable the generation of a mesh.



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}[view={60}{30}]
\addplot3+[domain=0:5*pi,samples=60,samples y=0]
    ({sin(deg(x))},
     {cos(deg(x))},
     {2*x/(5*pi)});
\end{axis}
\end{tikzpicture}
```

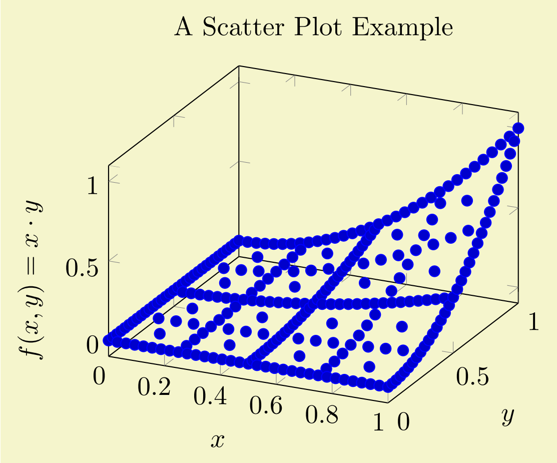
Three dimensional line plots will usually employ lines to connect points (i.e. the initial `sharp plot` handler of Ti*k*Z). The `smooth` method of Ti*k*Z might also prove be an option. Note that no piecewise constant plot, comb or bar plot handler is supported for three dimensional axes.

4.5.4 Scatter Plots

Three dimensional scatter plots have the same interface as for two dimensional scatter plots, so all examples of section 4.4.8 can be used for the three dimensional case as well. The key features are to use `only marks` and/or `scatter` as plot styles.

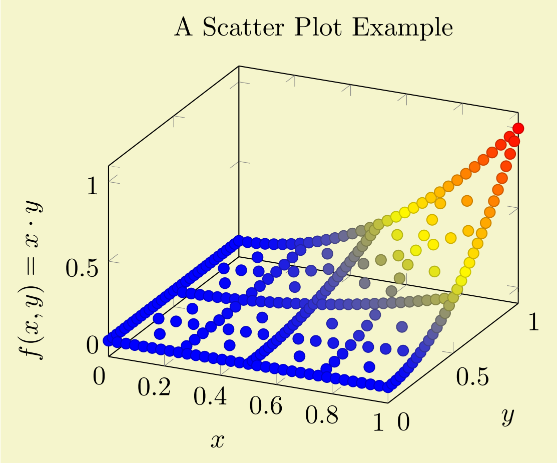
We provide some more examples which are specific for the three dimensional case.

Our first example uses `only marks` to place the current plot `mark` at each input position:



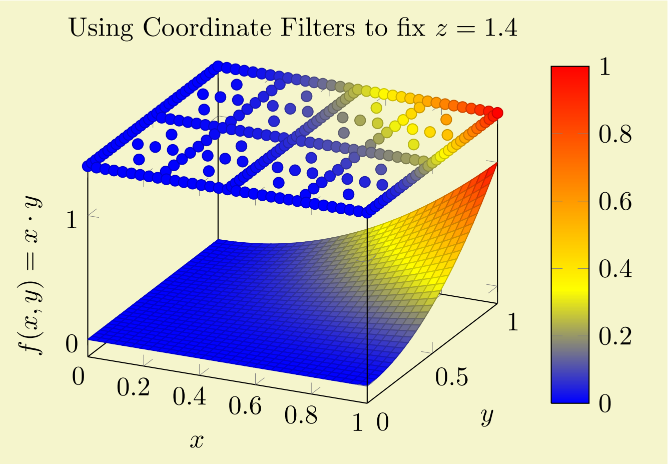
```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{axis}[
        xlabel=$x$,
        ylabel=$y$,
        zlabel={$f(x,y) = x\cdot y$},
        title=A Scatter Plot Example]
    %  'pgfplotsexample4_grid.dat' contains a
    %  large sequence of input points of the form
    %  x_0   x_1     f(x)
    %  0     0       0
    %  0     0.03125 0
    %  0     0.0625  0
    %  0     0.09375 0
    %  0     0.125   0
    %  0     0.15625 0
    \addplot3+[only marks] table
        {plotdata/pgfplotsexample4_grid.dat};
    \end{axis}
\end{tikzpicture}
```

If we add the key `scatter`, the plot mark will also use the colors of the current `colormap`:



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{axis}[
        xlabel=$x$,
        ylabel=$y$,
        zlabel={$f(x,y) = x\cdot y$},
        title=A Scatter Plot Example]
    \addplot3+[only marks,scatter] table
        {plotdata/pgfplotsexample4_grid.dat};
    \end{axis}
\end{tikzpicture}
```

A more sophisticated example is to draw the approximated function as a `surf` plot (which requires matrix data) and the underlying grid (which is `scatter`ed data) somewhere into the same axis. We choose to place the (x, y) grid points at $z = 1.4$. Furthermore, we want the grid points to be colored according to the value of column `f(x)` in the input table:



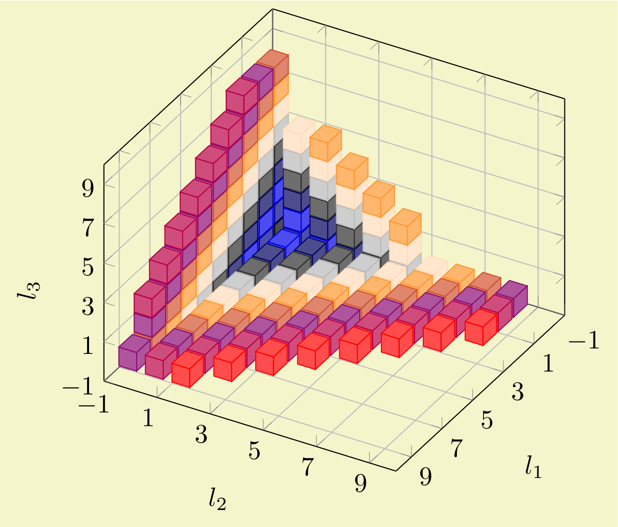
```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{axis}[
        3d box,
        zmax=1.4,
        colorbar,
        xlabel=$x$,
        ylabel=$y$,
        zlabel={$f(x,y) = x\cdot y$},
        title={Using Coordinate Filters to fix $z=1.4$}]
    %  ‘pgfplotsexample4.dat’ contains similar data as in
    %  ‘pgfplotsexample4_grid.dat’, but it uses a uniform
    %  matrix structure (same number of points in every scanline).
    \addplot3[surf,mesh/ordering=y varies]
        table {plotdata/pgfplotsexample4.dat};
    \addplot3[scatter,scatter src=\thisrow{f(x)},only marks, z filter/.code={\def\pgfmathresult{1.4}}]
        table {plotdata/pgfplotsexample4_grid.dat};
    \end{axis}
\end{tikzpicture}
```

We used `z filter` to fix the z coordinate to 1.4. We could also have used the `table/z expr=1.4` feature

```
    \addplot3[scatter,scatter src=\thisrow{f(x)},only marks]
        table[z expr=1.4] {plotdata/pgfplotsexample4_grid.dat};
```

to get exactly the same effect. Choose whatever you like best. The `z filter` works for every coordinate input routine, the `z expr` feature is only available for `plot table`.

The following example uses `mark=cube*` and `z buffer=sort` to place boxes at each input coordinate. Each boxes color is determines by `point meta={x+y+3}`, so the sum of the input coordinates determines the colors. The remaining keys are just for pretty printing.



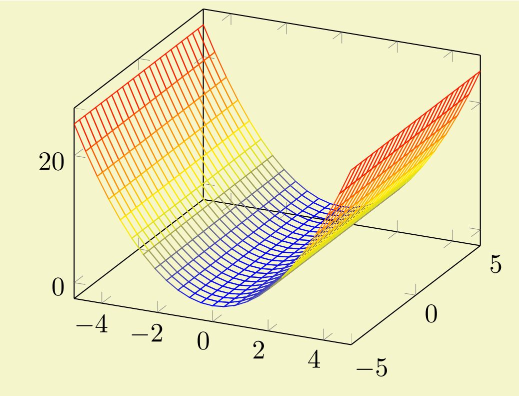
```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}[
    view={120}{40},
    width=220pt,
    height=220pt,
    grid=major,
    z buffer=sort,
    xmin=-1,xmax=9,
    ymin=-1,ymax=9,
    zmin=-1,zmax=9,
    enlargelimits=upper,
    xtick={-1,1,...,19},
    ytick={-1,1,...,19},
    ztick={-1,1,...,19},
    xlabel={$l_1$},
    ylabel={$l_2$},
    zlabel={$l_3$},
    point meta={x+y+z+3},
    colormap={summap}{
        color=(black); color=(blue);
        color=(black); color=(white)
        color=(orange) color=(violet)
        color=(red)
    },
    scatter/use mapped color={
        draw=mapped color,fill=mapped color!70},
    ]
    % 'pgfplots_scatter4.dat' contains a large sequence of
    % the form
    % l_0   l_1     l_2
    % 1     6       -1
    % -1    -1      -1
    % 0     -1      -1
    % -1    0       -1
    % -1    -1      0
    % 1     -1      -1
    % 0     0       -1
    % 0     -1      0
    \addplot3[only marks,scatter,mark=cube*,mark size=7]
        table {plotdata/pgfplots_scatterdata4.dat};

\end{axis}
\end{tikzpicture}
```

4.5.5 Mesh Plots

/pgfplots/mesh (no value)

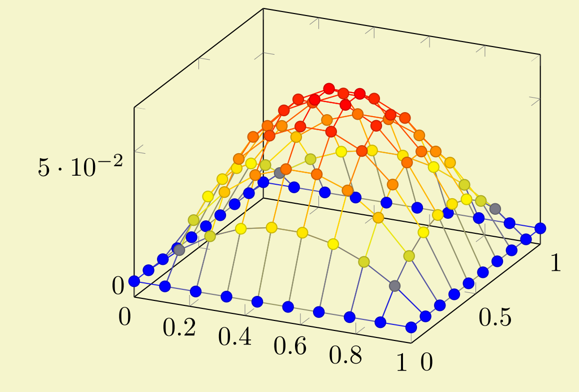
\addplot+[mesh]

A mesh plot uses different colors for each mesh segment. Each mesh segment gets the same color. The color is determined using a "color coordinate" which is also called "meta data" throughout this document. It is the same data which is used for surface and scatter plots as well, see section 4.7. In the initial configuration, the "color coordinate" is the z axis (or the y axis for two dimensional plots). This color coordinate is mapped linearly into the current color map to determine the color for each mesh segment. Thus, if the smallest occurring color data is, say, -1 and the largest is 42, points with color data -1 will get the color at the lower end of the color map and points with color data 42 the color of the upper end of the color map.

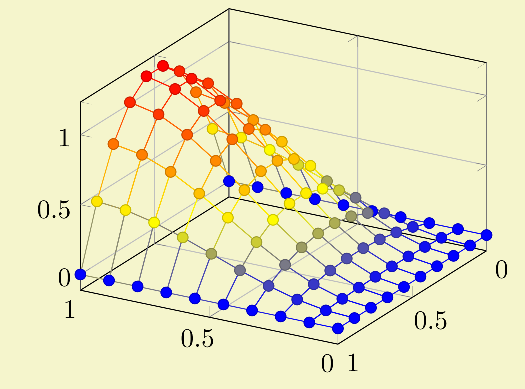


```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{axis}
        \addplot3[mesh] {x^2};
    \end{axis}
\end{tikzpicture}
```

A mesh plot can be combined with markers or with the `scatter` key which does also draw markers in different colors.



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{axis}
    \addplot3+[mesh,scatter,samples=10,domain=0:1]
        {x*(1-x)*y*(1-y)};
    \end{axis}
\end{tikzpicture}
```

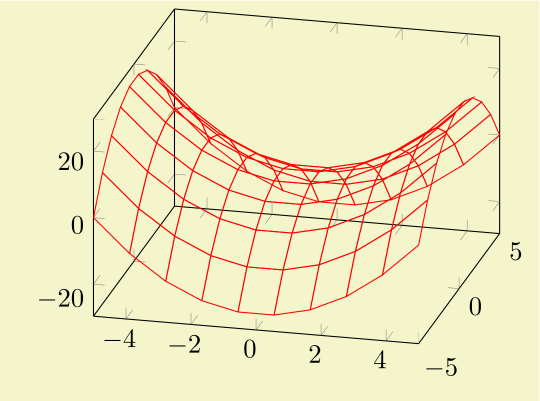


```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{axis}[grid=major,view={210}{30}]
    \addplot3+[mesh,scatter,samples=10,domain=0:1]
        {5*x*sin(2*deg(x)) * y*(1-y)};
    \end{axis}
\end{tikzpicture}
```

Details:

- A mesh plot uses the same implementation as `shader=flat` to get one color for each single segment. Thus, if `shader=flat mean`, the color for a segment is determined using the *mean* of the color data of adjacent vertices. If `shader=flat corner`, the color of a segment is the color of *one* adjacent vertex.
- As soon as `mesh` is activated, `color=mapped color` is installed. This is *necessary* unless one needs a different color – but `mapped color` is the only color which reflects the color data.

  It is possible to use a different color using the `color=`⟨*color name*⟩ as for any other plot.
- It is easily possible to add `mark=`⟨*marker name*⟩ to mesh plots, `scatter` is also possible. Scatter plots will use the same color data as for the mesh.



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{axis}[view/az=14]
    \addplot3[mesh,draw=red,samples=10] {x^2-y^2};
    \end{axis}
\end{tikzpicture}
```

Mesh plots use the mesh legend style to typeset legend images.

/pgfplots/mesh/check=false|warning|error (initially error)

Allows to configure whether an error is generated if mesh/rows × mesh/cols does not equal the total number of coordinates.

If you know exactly what you are doing, it may be useful to disable the check. If you are unsure, it is best to leave the initial setting.

/pgfplots/z buffer=default|none|auto|sort|reverse x seq|reverse y seq|reverse xy seq (initially default)

This key allows to choose between different z buffering strategies. A z buffer determines which parts of an image should be drawn in front of other parts. Since both, the graphics packages PGF and the final document format .pdf are inherently two dimensional, this work has to be done in TEX. Currently, several (fast) heuristics can be used which work reasonably well for simple mesh- and surface plots. Furthermore, there is a (time consuming) sorting method which does also work if the fast heuristics fails.

The z buffering algorithms of PGFPLOTS apply only to a single \addplot command. Different \addplot commands will be drawn on top of each other, in the order of appearance.

The choice default checks if we are currently working with a mesh or surface plot and uses auto in this case. If not, it sets z buffer=none.

The choice none disables z buffering. This is also the case for two dimensional axes which don't need z buffering.

The choice auto is the initial value for any mesh- or surface plot: it uses a very fast heuristics to decide how to realize z buffering for mesh and surface plots. The idea is to reverse either the sequence of all x coordinates, or those of all y coordinates, or both. For regular meshes, this suffices to provide z buffering. In other words: the choice auto will use one of the three reverse strategies reverse * seq (or none at all).

The choice sort can be used for scatter, line, mesh and surface plots. It really sorts according to the depth of each point (or mesh segment)[18]. Sorting in TEX uses a slow algorithm and may require a lot of memory (although it has the expected runtime asymptotics $\mathcal{O}(N \log N)$).

The remaining choices apply only to mesh/surface plots and do nothing more then their name indicates: they reverse the coordinate sequences (using quasi linear runtime). They should only be used in conjunction by z buffer=auto.

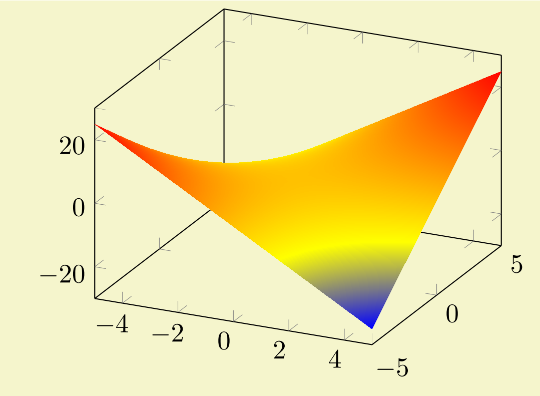
4.5.6 Surface Plots

/pgfplots/surf (no value)

\addplot+[surf]

A surface plot visualizes a two dimensional, single patch using different fill colors for each patch segment. Each patch segment is a (pseudo) rectangle, that means input data is given in form of a data matrix as is discussed in the introductory section about three dimensional coordinates, 4.5.2.

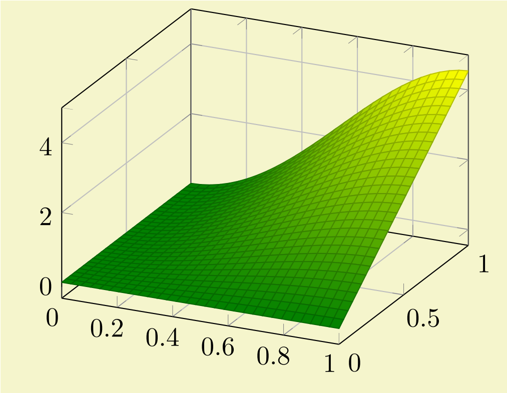
[18] The choice sort is *not* available for surface plots with shader=interp because the low level format doesn't support sorting.



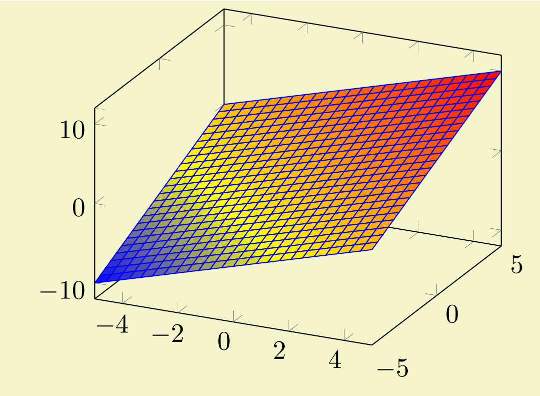
```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{axis}
        \addplot3[surf,shader=interp] {x*y};
    \end{axis}
\end{tikzpicture}
```

The simplest way to generate surface plots is to use the plot expression feature, but – as discussed in section 4.5.2 – other input methods like `\addplot3 table` or `\addplot3 coordinates` are also possible.

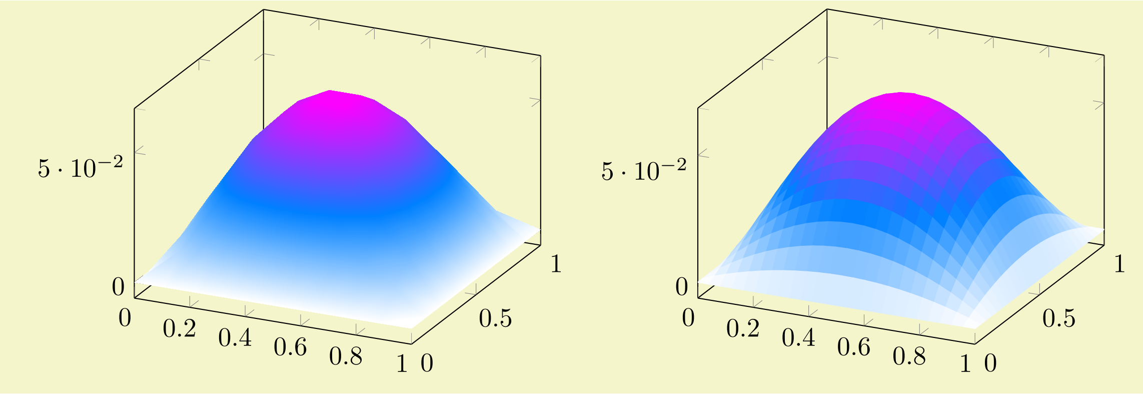
The appearance can be configured using `colormap`s, the value of the `shader`, `faceted color` keys and the current `color` and/or `draw` / `fill` color. As for `mesh` plots, the special `color`=`mapped color` is installed for the faces. The stroking color for faceted plots can be set with `faceted color` (see below for details).



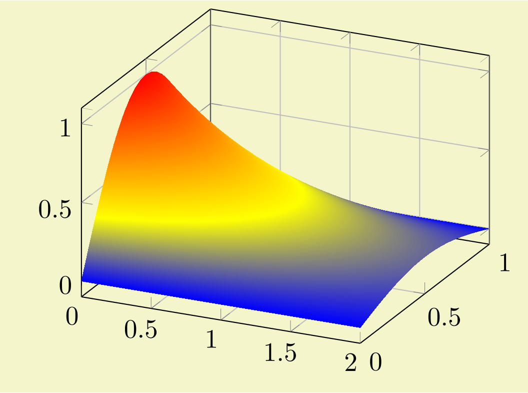
```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{axis}[
        grid=major,
        colormap/greenyellow]
    \addplot3[surf,samples=30,domain=0:1]
        {5*x*sin(2*deg(x)) * y};
    \end{axis}
\end{tikzpicture}
```



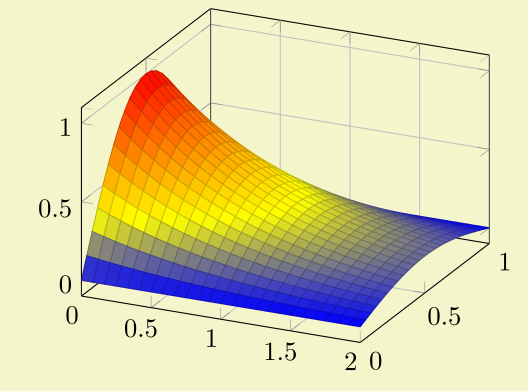
```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{axis}
        \addplot3[surf,faceted color=blue] {x+y};
    \end{axis}
\end{tikzpicture}
```



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{axis}[colormap/cool]
    \addplot3[surf,samples=10,domain=0:1,
        shader=interp]
        {x*(1-x)*y*(1-y)};
    \end{axis}
\end{tikzpicture}
\begin{tikzpicture}
    \begin{axis}[colormap/cool]
    \addplot3[surf,samples=25,domain=0:1,
        shader=flat]
        {x*(1-x)*y*(1-y)};
    \end{axis}
\end{tikzpicture}
```



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{axis}[grid=major]
        \addplot3[surf,shader=interp,
            samples=25,domain=0:2,y domain=0:1]
            {exp(-x) * sin(pi*deg(y))};
    \end{axis}
\end{tikzpicture}
```



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{axis}[grid=major]
        \addplot3[surf,shader=faceted,
            samples=25,domain=0:2,y domain=0:1]
            {exp(-x) * sin(pi*deg(y))};
    \end{axis}
\end{tikzpicture}
```

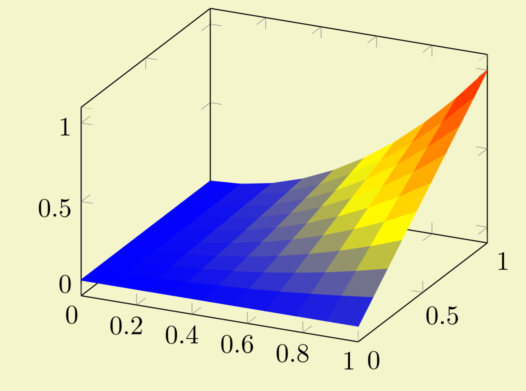
Details about the shading algorithm are provided below in the documentation of `shader`.

Surface plots use the `mesh legend` style to create legend images.

`/pgfplots/shader=flat|interp|faceted|flat corner|flat mean` (initially `faceted`)

Configures the shader used for surface plots. The shader determines how the color data available at each single vertex is used to fill the surface patch.

The simplest choice is to use one fill color for each segment, the choice flat.



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{axis}
    \addplot3[surf,shader=flat,
        samples=10,domain=0:1]
        {x^2*y};
    \end{axis}
\end{tikzpicture}
```

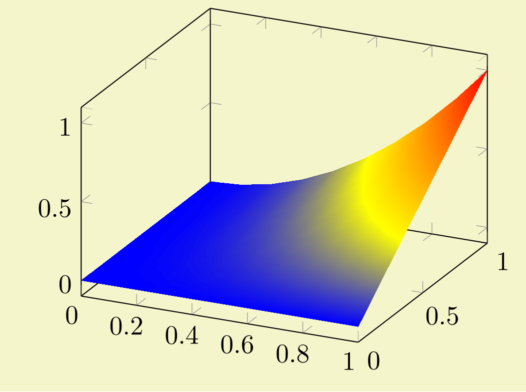
The flat shader provides full support of z buffering, that means it does also support the choice z buffer=sort. There are (currently) two possibilities to determine the single color for every segment:

flat corner Uses the color data of one vertex to color the segment. It is not defined which vertex is used here[19].

flat mean Uses the mean of all four color data values as segment color. This is the initial value as it provides symmetric colors for symmetric functions.

The choice flat is actually the same as flat mean. Please note that shader=flat mean and shader=flat corner also influence mesh plots – the choices determine the mesh segment color.

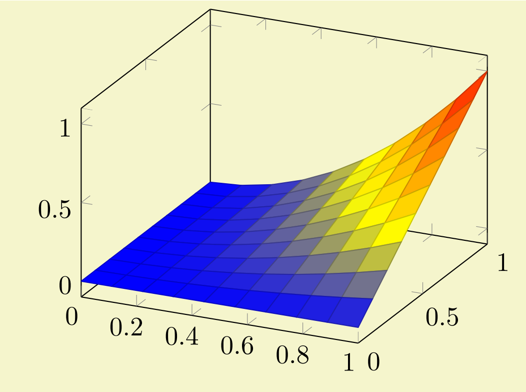
Another choice is shader=interp which uses Goraud shading (bilinear interpolation) to fill the segments.



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{axis}
    \addplot3[surf,shader=interp,
        samples=10,domain=0:1]
        {x^2*y};
    \end{axis}
\end{tikzpicture}
```

The shader=interp setting requires a special low–level shading implementation which is currently (only) available for the postscript driver pgfsys-dvips.def and the pdflatex driver pgfsys-pdftex.def. For other drivers, the choice shader=interp will result in a warning and is equivalent to shader=flat mean.

Finally, the choice shader=faceted uses a constant fill color for every mesh segment (as for flat) and the value of the key /pgfplots/faceted color to draw the connecting mesh elements:



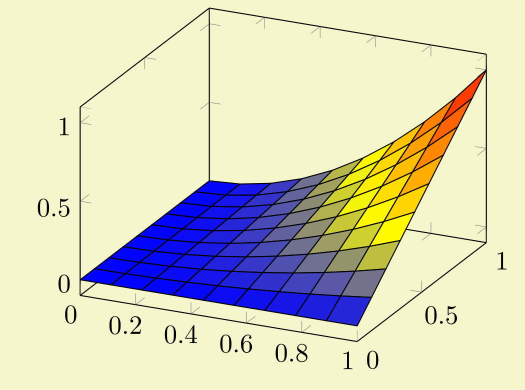
```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{axis}
    \addplot3[surf,shader=faceted,
        samples=10,domain=0:1]
        {x^2*y};
    \end{axis}
\end{tikzpicture}
```

[19] PGFPLOTS just uses the last vertex encountered in its internal processings – but after any z buffer re-orderings.

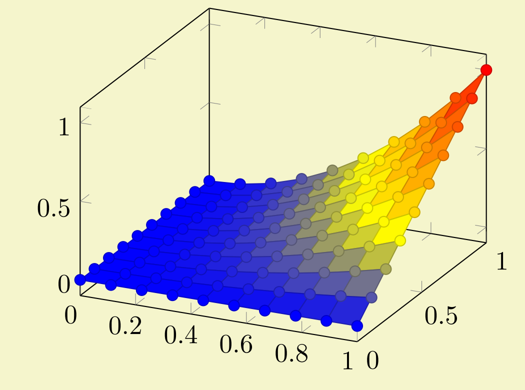
Details:

- The choice shader=faceted is the same as shader=flat – except that it uses a special draw color. So, shader=faceted has the same effect as
  shader=flat,draw=\pgfkeysvalueof{/pgfplots/faceted color}.
- The flat shader uses the current draw and fill colors. They are set with color=mapped color and can be overruled with draw=⟨*draw color*⟩ and fill=⟨*fill color*⟩. The mapped color always contains the color of the color map.
- The interp shader does not support mesh colors and it uses the current color map in any case (it simply ignores the values of draw and fill).
- You easily add mark=⟨*plot mark*⟩ to mesh and/or surface plots or even colored plot marks with scatter. The scatter plot feature will use the same color data as for the surface.

  But: Markers and surfaces do not share the same depth information. They are drawn on top of each other.
- For surface plots with lots of points, shader=interp produces smaller pdf documents, requires less compilation time in TEX and requires less time to display in Acrobat Reader.
- The postscript driver did not work when I tried to write hex encoded 32 bit binary coordinates into the shading. So, the postscript driver *truncates* coordinates to 24 bit – which might result in a loss of precision (the truncation is not very intelligent). See the surf shading/precision key for details. To improve compatibility, this 24 bit truncation algorithm is enabled by default.



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{axis}
    \addplot3[surf,shader=flat,
        draw=black,
        samples=10,domain=0:1]
        {x^2*y};
    \end{axis}
\end{tikzpicture}
```



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{axis}
    \addplot3[surf,shader=faceted,
        scatter,mark=*,
        samples=10,domain=0:1]
        {x^2*y};
    \end{axis}
\end{tikzpicture}
```

/pgfplots/faceted color={⟨*color name*⟩} (initially mapped color!80!black)

Defines the color to be used for meshes of faceted surface plots.

/pgfplots/surf shading/precision=pdf|postscript|ps (initially postscript)

A key to configure how the low level driver for shader=interp writes its data. The choice pdf uses 32 bit binary coordinates (which is lossless). The resulting .pdf files appear to be correct, but they can't be converted to postscript – the converter software always complaints about an error.

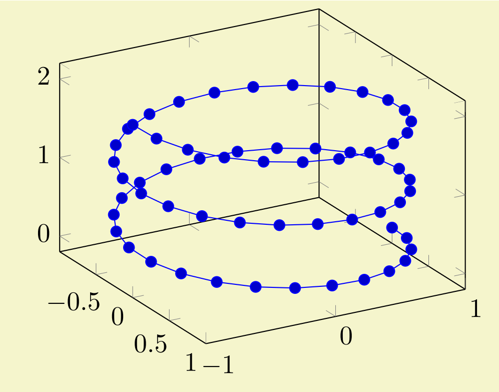
The choice `postscript` (or, in short, `ps`) uses 24 bit truncated binary coordinates. This results in both, readable `.ps` and `.pdf` files. However, the truncation is lossy.

If anyone has ideas how to fix this problem: let me know. As far as I know, postscript should accept 32 bit coordinates, so it might be a mistake in the shading driver.

4.5.7 Parameterized Plots

Parameterized plots use the same plot types as documented in the preceding sections: both, mesh and surface plots are actually special parameterized plots where x and y are on cartesian grid points.

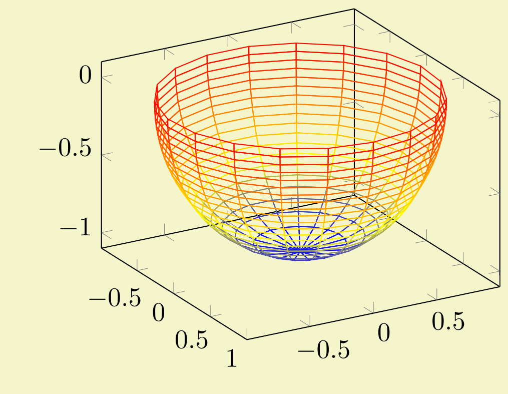
Parameterized plots just need a special way to provide the coordinates:



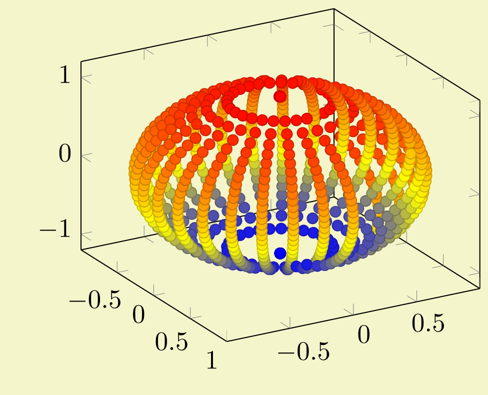
```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{axis}[view={60}{30}]
    \addplot3+[domain=0:5*pi,samples=60,samples y=0]
        ({sin(deg(x))},
         {cos(deg(x))},
         {2*x/(5*pi)});
    \end{axis}
\end{tikzpicture}
```

The preceding example uses `samples y=0` to indicate that a line shall be samples instead of a matrix. The curly braces are necessary because TEX can't nest round braces. The single expressions here are used to parameterize the helix.

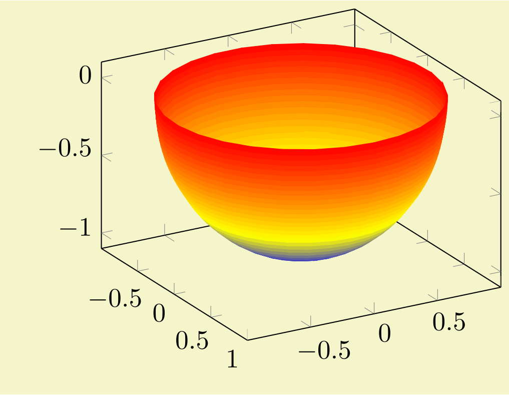
Another example follows. Note that `z buffer=sort` is a necessary method here.



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}[view={60}{30}]
    \addplot3[mesh,z buffer=sort,
        samples=20,domain=-1:0,y domain=0:2*pi]
        ({sqrt(1-x^2) * cos(deg(y))},
         {sqrt( 1-x^2 ) * sin(deg(y))},
         x);
\end{axis}
\end{tikzpicture}
```



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}[view={60}{30}]
    \addplot3[mesh,z buffer=sort,
        scatter,only marks,scatter src=z,
        samples=30,domain=-1:1,y domain=0:2*pi]
        ({sqrt(1-x^2) * cos(deg(y))},
         {sqrt( 1-x^2 ) * sin(deg(y))},
         x);
\end{axis}
\end{tikzpicture}
```



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}[view={60}{30}]
    \addplot3[surf,shader=flat,z buffer=sort,
        samples=30,domain=-1:0,y domain=0:2*pi]
        ({sqrt(1-x^2) * cos(deg(y))},
         {sqrt( 1-x^2 ) * sin(deg(y))},
         x);
\end{axis}
\end{tikzpicture}
```

4.5.8 About 3D Const Plots and 3D Bar Plots

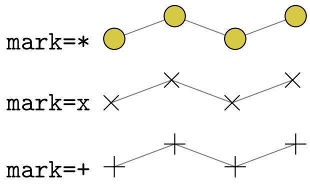
There are currently *no* equivalents of `const plot` and its variants or the bar plot types like `ybar` for three dimensional axes, sorry.

4.6 Markers, Linestyles, (Background-) Colors and Colormaps

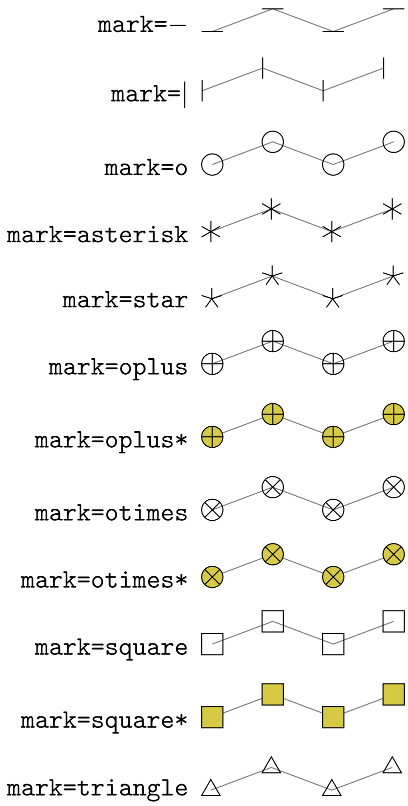
The following options of Ti*k*Z are available to plots.

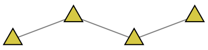
4.6.1 Markers

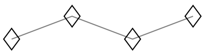
This list is copied from [5, section 29]:

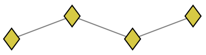


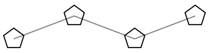
And with `\usetikzlibrary{plotmarks}`:

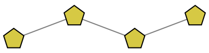


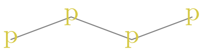
`mark=triangle*` 

`mark=diamond` 

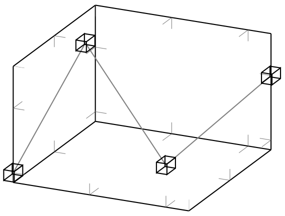
`mark=diamond*` 

`mark=pentagon` 

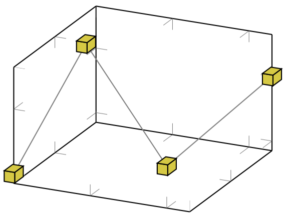
`mark=pentagon*` 

`mark=text` 

This marker is special as it can be configured freely. The character (or even text) used is configured by a set of variables, see below.

`mark=cube` 

This marker is only available inside of a PGFPLOTS axis, it draws a cube with axis parallel faces. Its sizes can be configured separately, see below.

`mark=cube*` 

`User defined` It is possible to define new markers with `\pgfdeclareplotmark`, see below.

All these options have been drawn with the additional options

```
\draw[
    gray,
    thin,
    mark options={%
        scale=2,fill=yellow!80!black,draw=black
    }
]
```

Please see section 4.6.4 for how to change draw- and fill colors.

`/tikz/mark size={⟨dimension⟩}`

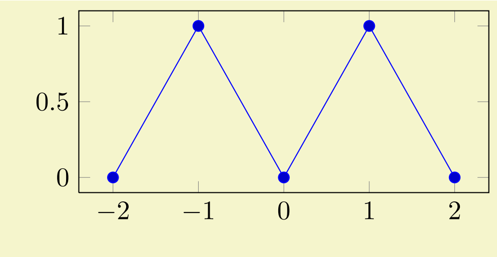
This TikZ option allows to set marker sizes to `{`⟨*dimension*⟩`}`. For circular markers, `{`⟨*dimension*⟩`}` is the radius, for other plot marks it is about half the width and height.

`/pgfplots/cube/size x={⟨dimension⟩}` (initially `\pgfplotmarksize`)
`/pgfplots/cube/size y={⟨dimension⟩}` (initially `\pgfplotmarksize`)
`/pgfplots/cube/size z={⟨dimension⟩}` (initially `\pgfplotmarksize`)

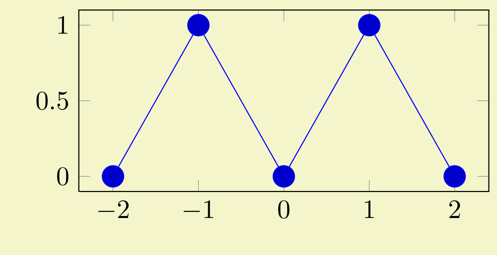
Sets the size for `mark=cube` separately for every axis.

`/tikz/every mark` (no value)

This TikZ style can be reconfigured to set marker appearance options like colors or transformations like scaling or rotation. PGFPLOTS appends its `cycle list` options to this style.



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}[y=2cm]
    \addplot coordinates
        {(-2,0) (-1,1) (0,0) (1,1) (2,0)};
\end{axis}
\end{tikzpicture}
```



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\tikzset{every mark/.append style={scale=2}}
\begin{tikzpicture}
\begin{axis}[y=2cm]
    \addplot coordinates
        {(-2,0) (-1,1) (0,0) (1,1) (2,0)};
\end{axis}
\end{tikzpicture}
```

`/pgfplots/no markers` (style, no value)

Disables plot marks.

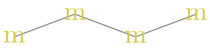
If this style is provided as argument to a complete axis, it is appended to `every axis plot post` such that it disables markers even for `cycle lists` which contain markers.

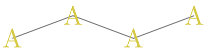
`/tikz/mark options={⟨options⟩}`

Resets `every mark` to `{⟨options⟩}`.

`/pgf/text mark={⟨text⟩}` (initially `p`)

Changes the text shown by `mark=text`.

With `/pgf/text mark=m`: 

With `/pgf/text mark=A`: 

There is no limitation about the number of characters or whatever. In fact, any TEX material can be inserted as `{⟨text⟩}`, including images.

`/pgf/text mark style={⟨options for mark=text⟩}`

Defines a set of options which control the appearance of `mark=text`.

If `/pgf/text mark as node=false` (the default), `{⟨options⟩}` is provided as argument to `\pgftext` – which provides only some basic keys like `left`, `right`, `top`, `bottom`, `base` and `rotate`.

If `/pgf/text mark as node=true`, `{⟨options⟩}` is provided as argument to `\node`. This means you can provide a very powerful set of options including `anchor`, `scale`, `fill`, `draw`, `rounded corners` etc.

`/pgf/text mark as node=true|false` (initially `false`)

Configures how `mark=text` will be drawn: either as `\node` or as `\pgftext`.

The first choice is highly flexible and possibly slow, the second is very fast and usually enough.

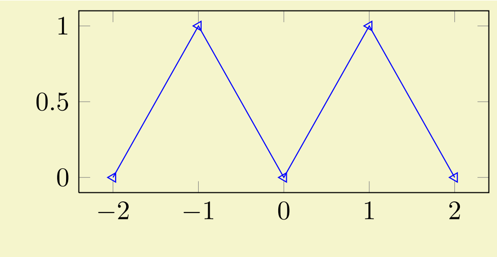
`\pgfdeclareplotmark{⟨plot mark name⟩}{⟨code⟩}`

Defines a new marker named `{⟨plot mark name⟩}`. Whenever it is used, `{⟨code⟩}` will be invoked. It is suppose to contain (preferrable PGF basic level) drawing commands. During `{⟨code⟩}`, the coordinate system's origin denotes the coordinate where the marker shall be placed.

Please refer to [5] section "Mark Plot Handler" for more detailed information.

`/pgfplots/every axis plot post` (style, initially)

The `every axis plot post` style can be used to overwrite parts (or all) of the drawing styles which are assigned for plots.



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
%  Overwrite any cycle list:
\pgfplotsset{
  every axis plot post/.append style={
   mark=triangle,
   every mark/.append style={rotate=90}}}
\begin{tikzpicture}
\begin{axis}[y=2cm]
    \addplot coordinates
        {(-2,0) (-1,1) (0,0) (1,1) (2,0)};
\end{axis}
\end{tikzpicture}
```

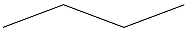
Markers paths are not subjected to clipping as other parts of the figure. Markers are either drawn completely or not at all.

Ti*k*Z offers more options for marker fine tuning, please refer to [5] for details.

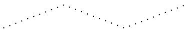
4.6.2 Line Styles

The following line styles are predefined in Ti*k*Z.

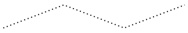
`/tikz/solid` (style, no value)



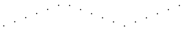
`/tikz/dotted` (style, no value)



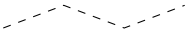
`/tikz/densely dotted` (style, no value)



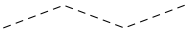
`/tikz/loosely dotted` (style, no value)



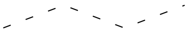
`/tikz/dashed` (style, no value)



`/tikz/densely dashed` (style, no value)



`/tikz/loosely dashed` (style, no value)



since these styles apply to markers as well, you may want to consider using

```
\pgfplotsset{
    every mark/.append style={solid}
}
```

in marker styles.

Besides linestyles, PGF also offers (a lot of) arrow heads. Please refer to [5] for details.

4.6.3 Font Size and Line Width

Often, one wants to change line width and font sizes for plots. This can be done using the following options of Ti*k*Z.

`/tikz/font={⟨font name⟩}` (initially `\normalfont`)

Sets the font which is to be used for text in nodes (like tick labels, legends or descriptions).

A font can be any LaTeX argument like `\footnotesize` or `\small\bfseries`[20].

It may be useful to change fonts only for specific axis descriptions, for example using

[20] ConTeXt and plain TeX users need to provide other statements, of course.

```
\pgfplotsset{
    tick label style={font=\small},
    label style={font=\small},
    legend style={font=\footnotesize}
}
```

See also the predefined styles `normalsize`, `small` and `footnotesize` in section 4.8.11.

/tikz/`line width`={⟨*dimension*⟩} (initially `0.4pt`)

Sets the line width. Please note that line widths for tick lines and grid lines are predefined, so it may be necessary to override the styles `every tick` and `every axis grid`.

The `line width` key is changed quite often in Ti*k*Z. You should use

```
\pgfplotsset{every axis/.append style={line width=1pt}}
```

or

```
\pgfplotsset{every axis/.append style={thick}}
```

to change the overall line width. To also adjust ticks and grid lines, one can use

```
\pgfplotsset{every axis/.append style={
    line width=1pt,
    tick style={line width=0.6pt}}}
```

or styles like

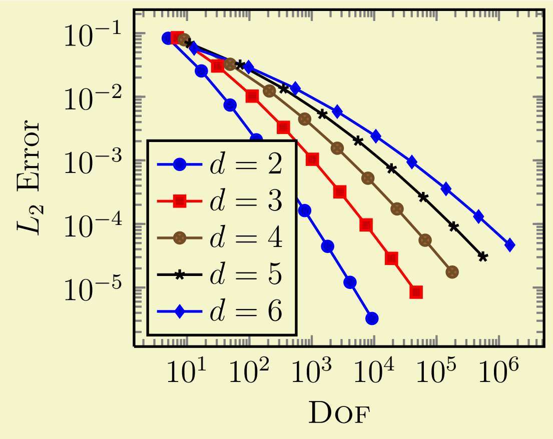
```
\pgfplotsset{every axis/.append style={
    thick,
    tick style={semithick}}}
```

The '`every axis plot`' style can be used to change line widths for plots only.

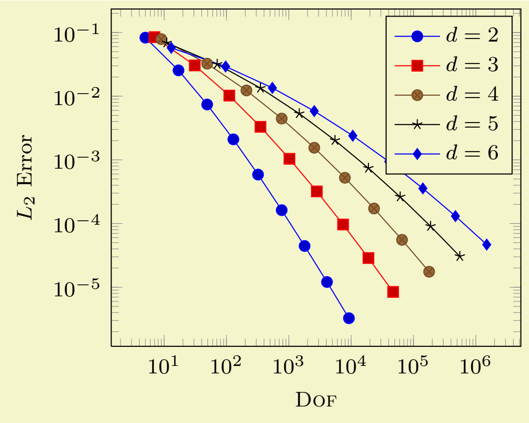
/tikz/`thin` (no value)
/tikz/`ultra thin` (no value)
/tikz/`very thin` (no value)
/tikz/`semithick` (no value)
/tikz/`thick` (no value)
/tikz/`very thick` (no value)
/tikz/`ultra thick` (no value)

These Ti*k*Z styles provide different predefined line widths.

This example shows the same plots as on page 14 (using `\plotcoords` as place holder for the commands on page 14), with different line width and font size.



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\pgfplotsset{every axis/.append style={
    font=\large,
    line width=1pt,
    tick style={line width=0.8pt}}}
\begin{tikzpicture}
    \begin{loglogaxis}[
        legend style={at={(0.03,0.03)},
            anchor=south west},
        xlabel=\textsc{Dof},
        ylabel=$L_2$ Error
    ]
    %  see above for this macro:
    \plotcoords
    \legend{$d=2$,$d=3$,$d=4$,$d=5$,$d=6$}
    \end{loglogaxis}
\end{tikzpicture}
```

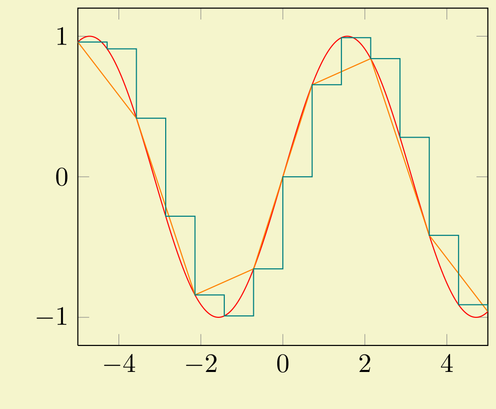


```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\pgfplotsset{every axis/.append style={
    font=\footnotesize,
    thin,
    tick style={ultra thin}}}
\begin{tikzpicture}
    \begin{loglogaxis}[
        xlabel=\textsc{Dof},
        ylabel=$L_2$ Error
    ]
    %  see above for this macro:
    \plotcoords
    \legend{$d=2$,$d=3$,$d=4$,$d=5$,$d=6$}
    \end{loglogaxis}
\end{tikzpicture}
```

4.6.4 Colors

PGF uses the color support of `xcolor`. Therefore, the main reference for how to specify colors is the `xcolor` manual [3]. The PGF manual [5] is the reference for how to select colors for specific purposes like drawing, filling, shading, patterns etc. This section contains a short overview over the specification of colors in [3] (which is not limited to PGFPLOTS).

The package `xcolor` defines a set of predefined colors, namely 🟥 `red`, 🟩 `green`, 🟦 `blue`, 🟦 `cyan`, 🟪 `magenta`, 🟨 `yellow`, ⬛ `black`, ■ `gray`, □ `white`, ■ `darkgray`, ■ `lightgray`, 🟫 `brown`, 🟩 `lime`, ■ `olive`, 🟧 `orange`, ■ `pink`, ■ `purple`, ■ `teal`, 🟪 `violet`.



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{axis}[enlarge x limits=false]
    \addplot[red,samples=500] {sin(deg(x))};

    \addplot[orange,samples=7] {sin(deg(x))};

    \addplot[teal,const plot,
        samples=14] {sin(deg(x))};
    \end{axis}
\end{tikzpicture}
```

Besides predefined colors, it is possible to *mix* two (or more) colors. For example, ■ `red!30!white` contains 30% of 🟥 `red` and 70% of □ `white`. Consequently, one can build ■ `red!70!white` to get 70% red and 30% white or □ `red!10!white` for 10% red and 90% white. This mixing can be done with any color, for example ■ `red!50!green`, ■ `blue!50!yellow` or 🟩 `green!60!black`.

A different type of color mixing is supported, which allows to take 100% of *each* component. For example, ■ `rgb,2:red,1;green,1` will add 1/2 part 🟥 `red` and 1/2 part 🟩 `green` and we reproduced the example from above. Using the denominator 1 instead of 2 leads to 🟨 `rgb,1:red,1;green,1` which uses 1 part 🟥 `red` and 1 part 🟩 `green`. Many programs allow to select pieces between $0, \ldots, 255$, so a denominator of 255 is useful. Consequently, ■ `rgb,255:red,231;green,84;blue,121` uses 231/255 red, 84/255 green and 121/255. This corresponds to the standard RGB color $(231, 84, 121)$. Other examples are 🟩 `rgb,255:red,32;green,127;blue,43`, 🟫 `rgb,255:red,178;green,127;blue,43`, ■ `rgb,255:red,169;green,178;blue,43`.

It is also possible to use RGB values, the HSV color model or the HTML color syntax directly. However, this requires some more programming. I suppose this is the fastest (and probably the most uncomfortable) method to use colors. For example,

```
\definecolor{color1}{rgb}{1,1,0}
\tikz \fill[color1]
    (0,0) rectangle (1em,0.6em);
```

creates the color with 100% red, 100% green and 0% blue;

```
\definecolor{color1}{HTML}{D0B22B}
\tikz \fill[color1]
    (0,0) rectangle (1em,0.6em);
```

creates the color with 208/255 pieces red, 178/255 pieces green and 43 pieces blue, specified in standard HTML notation. Please refer to the `xcolor` manual [3] for more details and color models.

The `xcolor` package provides even more methods to combine colors, among them the prefix '-' (minus) which changes the color into its complementary color (□ `-black`, ■ `-white`, 🟦 `-red`) or color wheel calculations. Please refer to the `xcolor` manual [3].

`/tikz/color={⟨a color⟩}`
`/tikz/draw={⟨stroke color⟩}`
`/tikz/fill={⟨fill color⟩}`

These keys are (generally) used to set colors. Use `color` to set the color for both, drawing and filling. Instead of `color={⟨color name⟩}` you can simply write `{⟨color name⟩}`. The `draw` and `fill` keys only set colors for stroking and filling, respectively.

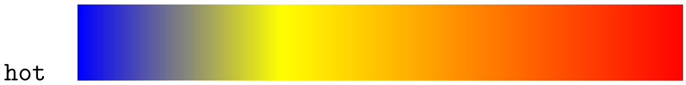
Use `draw=none` to disable drawing and `fill=none` to disable filling[21].

Since these keys belong to Ti*k*Z, the complete documentation can be found in the Ti*k*Z manual [5, Section "Specifying a Color"].

4.6.5 Color Maps

`/pgfplots/colormap name={⟨color map name⟩}` (initially `hot`)

Changes the current color map to the already defined map named `{⟨color map name⟩}`. The predefined color map is



Further color maps are described below.

Colormaps can be used, for example, in scatter plots (see section 4.4.8).

You can use `colormap` to create new color maps (see below).

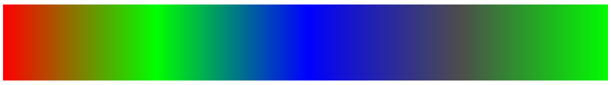
`/pgfplots/colormap={⟨name⟩}{⟨color specification⟩}`

Defines a new colormap named `{⟨name⟩}` according to `{⟨color specification⟩}` and activates it using `colormap name={⟨name⟩}`.

The `{⟨color specification⟩}` is a sequence of positions and associated colors where linear interpolation is applied in-between. The syntax is very similar as the one used for PGF shadings described in [5, VIII – Shadings]: it is a semicolon–separated series of

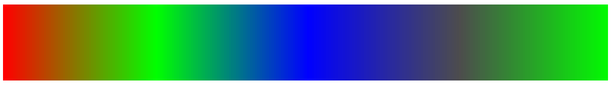
`⟨color type⟩(⟨offset⟩)=(⟨color value⟩);` :

```
%  possibility 1: like PGF shadings:
rgb(0cm)=(1,0,0); rgb(1cm)=(0,1,0); rgb255(2cm)=(0,0,255); gray(3cm)=(0.3);  color(4cm)=(green)
```



If the distance between successive colors is the same anyway, one can skip the ⟨*offset*⟩. The '`;`' separators are not necessary as well:

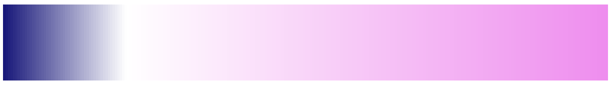
```
%  (simplified) possibility 2: skip ‘;’ and length arguments:
rgb=(1,0,0) rgb=(0,1,0) rgb255=(0,0,255) gray=(0.3) color=(green)
```



It is also possible to provide non-uniform distances between the different colors – if all single positions can be projected onto a uniform grid. PGFPLOTS will perform this interpolation automatically:

[21] Up to now, plot marks always have a stroke color (some also have a fill color). This restriction may be lifted in upcoming versions.

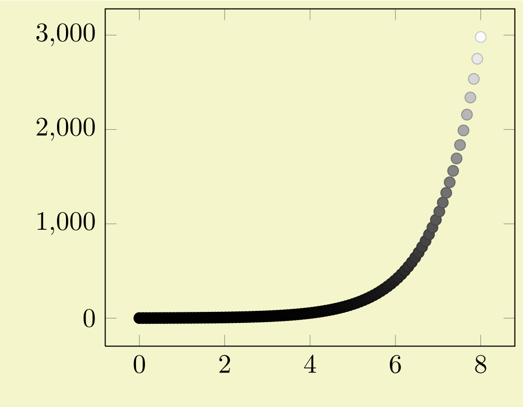
```
%  non uniform spacing example: the mesh width is provided as first
%  part of the specification.
\pgfplotsset{colormap={violetnew}
    {[1cm] rgb255(0cm)=(25,25,122) color(1cm)=(white) rgb255(5cm)=(238,140,238)}}
```



In this last example, the mesh width has been provided explicitly and PGFPLOTS interpolates the missing grid points on its own. It is an error if the provided positions are no multiple of the mesh width. The `\pgfplotsset` employs the public user interface to create a new color map named '`violetnew`'.

The single colors can be separated by semicolons '`;`'. The (optional) length describes how much of the bar is occupied by the interval, it is interpreted relative to the complete length. If the length argument is missing, it is taken to be the last specified length plus the last length difference (the first color defaults to `1cm` in this case).

Each entry has the form ⟨*color model*⟩`(`⟨*length*⟩`)=(`⟨*arguments*⟩`)` where the ⟨*length*⟩ argument is optional as discussed. The example above means that the left end of the color map shall have RGB components $1, 0, 0$, indicating 100% red and 0% green and blue. The next entity starts at `1cm` and describes a color with 100% green. The `rgb255` also expects three RGB components, but in the range $[0, 255]$. Finally, `gray` specifies a color in parenthesis with the same value for each, R G and B and `color` accesses predefined colors.



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{axis}[
        colormap={bw}{gray(0cm)=(0); gray(1cm)=(1)}]
    \addplot+[scatter,only marks,
         domain=0:8,samples=100]
        {exp(x)};
    \end{axis}
\end{tikzpicture}
```

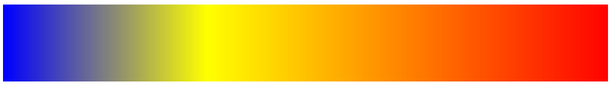
The complete length of a color map is irrelevant: it will be mapped linearly to an internal range anyway (for efficient interpolation). The only requirement is that the left end must be at `0`.

Available color maps are shown below.

/pgfplots/`colormap/hot` (style, no value)

A style which installs the colormap

`{color(0cm)=(blue); color(1cm)=(yellow); color(2cm)=(orange); color(3cm)=(red)}`

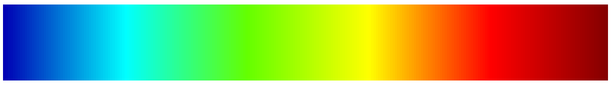


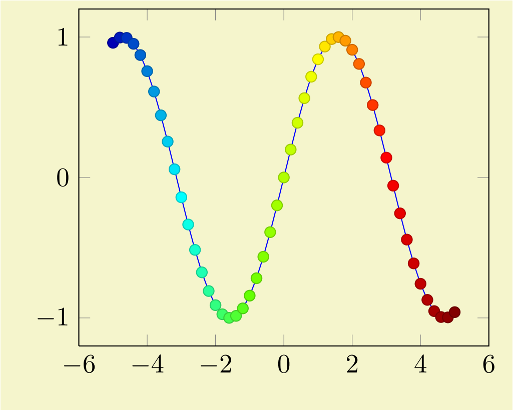
This is the preconfigured color map.

/pgfplots/`colormap/bluered` (style, no value)

A style which installs the colormap

`{rgb255(0cm)=(0,0,180); rgb255(1cm)=(0,255,255); rgb255(2cm)=(100,255,0);`
`rgb255(3cm)=(255,255,0); rgb255(4cm)=(255,0,0); rgb255(5cm)=(128,0,0)}`,





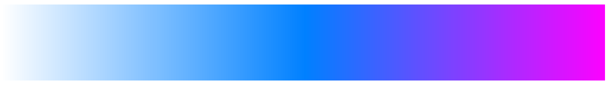
```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{axis}[colormap/bluered]
    \addplot+[scatter,
        scatter src=x,samples=50]
        {sin(deg(x))};
    \end{axis}
\end{tikzpicture}
```

Remark: The style `bluered` (re-)defines the color map and activates it. TEX will be slightly faster if you call `\pgfplotsset{colormap/bluered}` in the preamble (to create the color map once) and use `colormap name=bluered` whenever you need it. This remark holds for every color map style which follows. But you can simply ignore this remark.

`/pgfplots/colormap/cool` (style, no value)

A style which installs the colormap

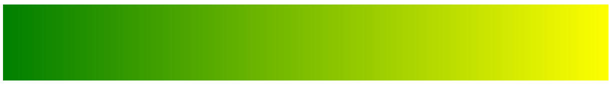
`{rgb255(0cm)=(255,255,255); rgb255(1cm)=(0,128,255); rgb255(2cm)=(255,0,255)}`



`/pgfplots/colormap/greenyellow` (style, no value)

A style which installs the colormap

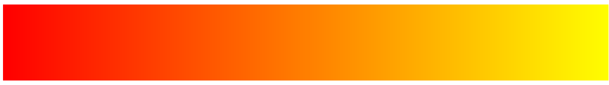
`{rgb255(0cm)=(0,128,0); rgb255(1cm)=(255,255,0)}`



`/pgfplots/colormap/redyellow` (style, no value)

A style which installs the colormap

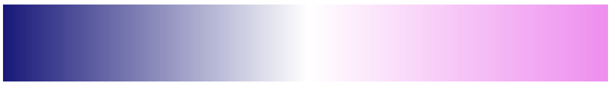
`{rgb255(0cm)=(255,0,0); rgb255(1cm)=(255,255,0)}`



`/pgfplots/colormap/violet` (style, no value)

A style which installs the colormap

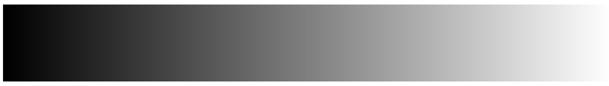
`{rgb255=(25,25,122) color=(white) rgb255=(238,140,238)}`



`/pgfplots/colormap/blackwhite` (style, no value)

A style which installs the colormap

`{gray(0cm)=(0); gray(1cm)=(1)}`



`\pgfplotscolormaptoshadingspec{`⟨*colormap name*⟩`}{`⟨*right end size*⟩`}{`⟨*\macro*⟩`}`

A command which converts a colormap into a PGF shading's color specification. It can be used in commands like `\pgfdeclare*shading` (see the PGF manual [5] for details).

The first argument is the name of a (defined) colormap, the second the rightmost dimension of the specification. The result will be stored in ⟨*macro*⟩.

```
% convert ‘hot’ -> \result
\pgfplotscolormaptoshadingspec{hot}{8cm}\result
% define and use a shading in pgf:
\def\tempb{\pgfdeclarehorizontalshading{tempshading}{1cm}}%
% where ‘\result’ is inserted as last argument:
\expandafter\tempb\expandafter{\result}%
\pgfuseshading{tempshading}%
```

The usage of the result ⟨*macro*⟩ is a little bit low–level.

4.6.6 Cycle Lists – Options Controlling Line Styles

/pgfplots/cycle list={⟨*list*⟩}
/pgfplots/cycle list name={⟨*macro*⟩}

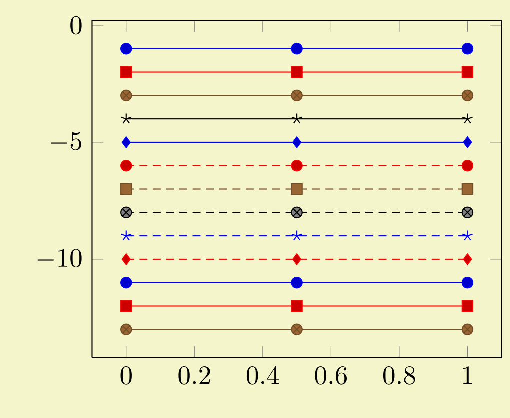
Allows to specify a list of plot specifications which will be used for each `\addplot`-command without explicit plot specification. Thus, the currently active `cycle list` will be used if you write either `\addplot+[`⟨*keys*⟩`] ...;` or if you *don’t* use square brackets as in `\addplot[`⟨*explicit plot specification*⟩`] ...;`.

The list element with index i will be chosen where i is the index of the current `\addplot` command. This indexing does also include plot commands which don’t use the `cycle list`.

There are several possibilities to change the currently active `cycle list`:

1. Use one of the predefined lists[22],

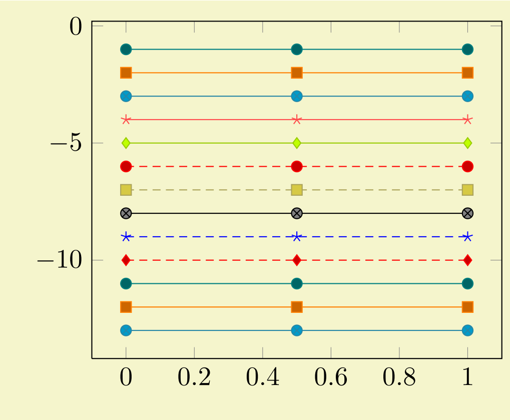
   - `color` (from top to bottom)



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}[
    stack plots=y,stack dir=minus,
    cycle list name=color]
\addplot coordinates {(0,1) (0.5,1) (1,1)};
\addplot coordinates {(0,1) (0.5,1) (1,1)};
\addplot coordinates {(0,1) (0.5,1) (1,1)};
\addplot coordinates {(0,1) (0.5,1) (1,1)};
\addplot coordinates {(0,1) (0.5,1) (1,1)};
\addplot coordinates {(0,1) (0.5,1) (1,1)};
\addplot coordinates {(0,1) (0.5,1) (1,1)};
\addplot coordinates {(0,1) (0.5,1) (1,1)};
\addplot coordinates {(0,1) (0.5,1) (1,1)};
\addplot coordinates {(0,1) (0.5,1) (1,1)};
\addplot coordinates {(0,1) (0.5,1) (1,1)};
\addplot coordinates {(0,1) (0.5,1) (1,1)};
\addplot coordinates {(0,1) (0.5,1) (1,1)};
\end{axis}
\end{tikzpicture}
```

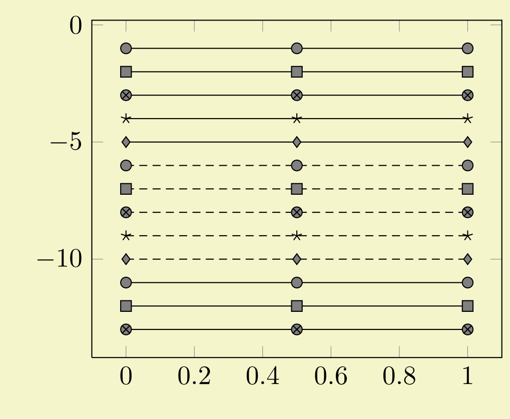
   - `exotic` (from top to bottom)

[22] In an early version, these lists were called `\coloredplotspeclist` and `\blackwhiteplotspeclist` which appeared to be unnecessarily long, so they have been renamed. The old names are still accepted, however.



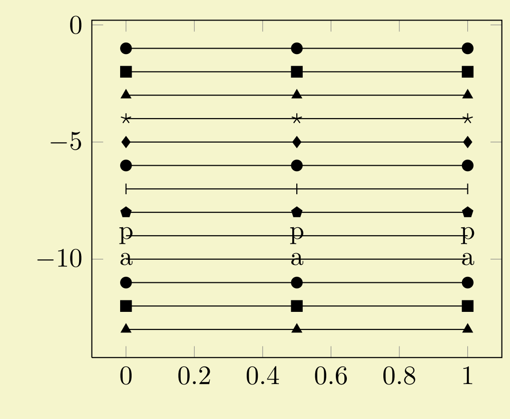
```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}[
    stack plots=y,stack dir=minus,
    cycle list name=exotic]
\addplot coordinates {(0,1) (0.5,1) (1,1)};
\addplot coordinates {(0,1) (0.5,1) (1,1)};
\addplot coordinates {(0,1) (0.5,1) (1,1)};
\addplot coordinates {(0,1) (0.5,1) (1,1)};
\addplot coordinates {(0,1) (0.5,1) (1,1)};
\addplot coordinates {(0,1) (0.5,1) (1,1)};
\addplot coordinates {(0,1) (0.5,1) (1,1)};
\addplot coordinates {(0,1) (0.5,1) (1,1)};
\addplot coordinates {(0,1) (0.5,1) (1,1)};
\addplot coordinates {(0,1) (0.5,1) (1,1)};
\addplot coordinates {(0,1) (0.5,1) (1,1)};
\addplot coordinates {(0,1) (0.5,1) (1,1)};
\addplot coordinates {(0,1) (0.5,1) (1,1)};
\end{axis}
\end{tikzpicture}
```

- black white (from top to bottom)



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}[
    stack plots=y,stack dir=minus,
    cycle list name=black white]
\addplot coordinates {(0,1) (0.5,1) (1,1)};
\addplot coordinates {(0,1) (0.5,1) (1,1)};
\addplot coordinates {(0,1) (0.5,1) (1,1)};
\addplot coordinates {(0,1) (0.5,1) (1,1)};
\addplot coordinates {(0,1) (0.5,1) (1,1)};
\addplot coordinates {(0,1) (0.5,1) (1,1)};
\addplot coordinates {(0,1) (0.5,1) (1,1)};
\addplot coordinates {(0,1) (0.5,1) (1,1)};
\addplot coordinates {(0,1) (0.5,1) (1,1)};
\addplot coordinates {(0,1) (0.5,1) (1,1)};
\addplot coordinates {(0,1) (0.5,1) (1,1)};
\addplot coordinates {(0,1) (0.5,1) (1,1)};
\addplot coordinates {(0,1) (0.5,1) (1,1)};
\end{axis}
\end{tikzpicture}
```

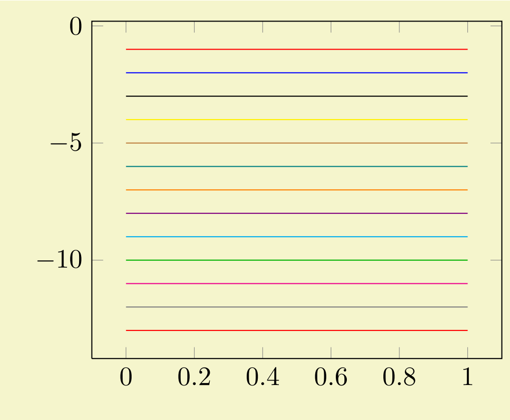
- mark list (from top to bottom)



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}[
    stack plots=y,stack dir=minus,
    cycle list name=mark list]
\addplot coordinates {(0,1) (0.5,1) (1,1)};
\addplot coordinates {(0,1) (0.5,1) (1,1)};
\addplot coordinates {(0,1) (0.5,1) (1,1)};
\addplot coordinates {(0,1) (0.5,1) (1,1)};
\addplot coordinates {(0,1) (0.5,1) (1,1)};
\addplot coordinates {(0,1) (0.5,1) (1,1)};
\addplot coordinates {(0,1) (0.5,1) (1,1)};
\addplot coordinates {(0,1) (0.5,1) (1,1)};
\addplot coordinates {(0,1) (0.5,1) (1,1)};
\addplot coordinates {(0,1) (0.5,1) (1,1)};
\addplot coordinates {(0,1) (0.5,1) (1,1)};
\addplot coordinates {(0,1) (0.5,1) (1,1)};
\addplot coordinates {(0,1) (0.5,1) (1,1)};
\end{axis}
\end{tikzpicture}
```

The mark list always employs the current color, but it doesn't define one.

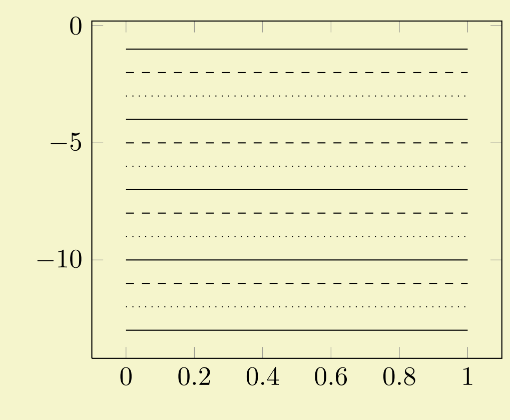
- color list (from top to bottom)



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}[
    stack plots=y,stack dir=minus,
    cycle list name=color list]
\addplot coordinates {(0,1) (0.5,1) (1,1)};
\addplot coordinates {(0,1) (0.5,1) (1,1)};
\addplot coordinates {(0,1) (0.5,1) (1,1)};
\addplot coordinates {(0,1) (0.5,1) (1,1)};
\addplot coordinates {(0,1) (0.5,1) (1,1)};
\addplot coordinates {(0,1) (0.5,1) (1,1)};
\addplot coordinates {(0,1) (0.5,1) (1,1)};
\addplot coordinates {(0,1) (0.5,1) (1,1)};
\addplot coordinates {(0,1) (0.5,1) (1,1)};
\addplot coordinates {(0,1) (0.5,1) (1,1)};
\addplot coordinates {(0,1) (0.5,1) (1,1)};
\addplot coordinates {(0,1) (0.5,1) (1,1)};
\addplot coordinates {(0,1) (0.5,1) (1,1)};
\end{axis}
\end{tikzpicture}
```

The `cycle list name=color` choice also employs markers whereas `color list` uses *only* colors.

- `linestyles` (from top to bottom)



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}[
    stack plots=y,stack dir=minus,
    cycle list name=linestyles]
\addplot coordinates {(0,1) (0.5,1) (1,1)};
\addplot coordinates {(0,1) (0.5,1) (1,1)};
\addplot coordinates {(0,1) (0.5,1) (1,1)};
\addplot coordinates {(0,1) (0.5,1) (1,1)};
\addplot coordinates {(0,1) (0.5,1) (1,1)};
\addplot coordinates {(0,1) (0.5,1) (1,1)};
\addplot coordinates {(0,1) (0.5,1) (1,1)};
\addplot coordinates {(0,1) (0.5,1) (1,1)};
\addplot coordinates {(0,1) (0.5,1) (1,1)};
\addplot coordinates {(0,1) (0.5,1) (1,1)};
\addplot coordinates {(0,1) (0.5,1) (1,1)};
\addplot coordinates {(0,1) (0.5,1) (1,1)};
\addplot coordinates {(0,1) (0.5,1) (1,1)};
\end{axis}
\end{tikzpicture}
```

- `auto` The `cycle list name=auto` always denotes the most recently used cycle list activated by `cycle list` or `cycle list name`.

The definitions of all predefined cycle lists follow (see the end of this paragraph for a syntax description).

```
\pgfplotscreateplotcyclelist{color}{%
    blue,every mark/.append style={fill=blue!80!black},mark=*\\%
    red,every mark/.append style={fill=red!80!black},mark=square*\\%
    brown!60!black,every mark/.append style={fill=brown!80!black},mark=otimes*\\%
    black,mark=star\\%
    blue,every mark/.append style={fill=blue!80!black},mark=diamond*\\%
    red,densely dashed,every mark/.append style={solid,fill=red!80!black},mark=*\\%
    brown!60!black,densely dashed,every mark/.append style={
        solid,fill=brown!80!black},mark=square*\\%
    black,densely dashed,every mark/.append style={solid,fill=gray},mark=otimes*\\%
    blue,densely dashed,mark=star,every mark/.append style=solid\\%
    red,densely dashed,every mark/.append style={solid,fill=red!80!black},mark=diamond*\\%
}
```

```
\pgfplotscreateplotcyclelist{black white}{%
    every mark/.append style={fill=gray},mark=*\\%
    every mark/.append style={fill=gray},mark=square*\\%
    every mark/.append style={fill=gray},mark=otimes*\\%
    mark=star\\%
    every mark/.append style={fill=gray},mark=diamond*\\%
    densely dashed,every mark/.append style={solid,fill=gray},mark=*\\%
    densely dashed,every mark/.append style={solid,fill=gray},mark=square*\\%
    densely dashed,every mark/.append style={solid,fill=gray},mark=otimes*\\%
    densely dashed,every mark/.append style={solid},mark=star\\%
    densely dashed,every mark/.append style={solid,fill=gray},mark=diamond*\\%
}
```

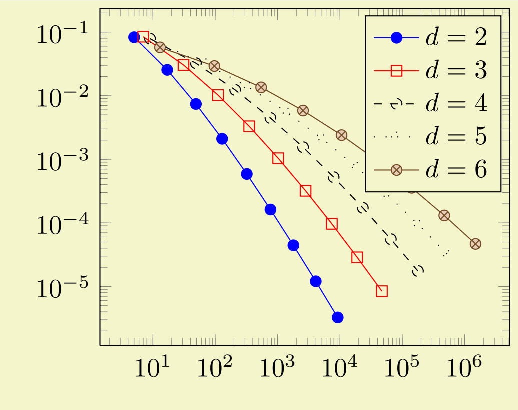
```
\pgfplotscreateplotcyclelist{exotic}{%
    teal,every mark/.append style={fill=teal!80!black},mark=*\\%
    orange,every mark/.append style={fill=orange!80!black},mark=square*\\%
    cyan!60!black,every mark/.append style={fill=cyan!80!black},mark=otimes*\\%
    red!70!white,mark=star\\%
    lime!80!black,every mark/.append style={fill=lime},mark=diamond*\\%
    red,densely dashed,every mark/.append style={solid,fill=red!80!black},mark=*\\%
    yellow!60!black,densely dashed,
        every mark/.append style={solid,fill=yellow!80!black},mark=square*\\%
    black,every mark/.append style={solid,fill=gray},mark=otimes*\\%
    blue,densely dashed,mark=star,every mark/.append style=solid\\%
    red,densely dashed,every mark/.append style={solid,fill=red!80!black},mark=diamond*\\%
}
```

```
%  note that "." is the currently defined Tikz color.
\pgfplotscreateplotcyclelist{mark list}{%
    every mark/.append style={fill=.!80!black},mark=*\\%
    every mark/.append style={fill=.!80!black},mark=square*\\%
    every mark/.append style={fill=.!80!black},mark=triangle*\\%
    mark=star\\%
    every mark/.append style={fill=.!80!black},mark=diamond*\\%
    every mark/.append style={fill=.!80!black},mark=otimes*\\%
    mark=|\\%
    every mark/.append style={fill=.!80!black},mark=pentagon*\\%
    mark=text,text mark=p\\%
    mark=text,text mark=a\\%
}
```

```
\pgfplotscreateplotcyclelist{color list}{%
    red,blue,black,yellow,brown,teal,orange,violet,cyan,green!70!black,magenta,gray}
```

```
\pgfplotscreateplotcyclelist{linestyles}{solid,dashed,dotted}
```

2. The second choice for cycle lists is to provide each entry directly as argument to `cycle list`,



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{loglogaxis}[cycle list={%
    {blue,mark=*},
    {red,mark=square},
    {dashed,mark=o},
    {loosely dotted,mark=+},
    {brown!60!black,
        mark options={fill=brown!40},
        mark=otimes*}}
]
\plotcoords
\legend{$d=2$,$d=3$,$d=4$,$d=5$,$d=6$}
\end{loglogaxis}
\end{tikzpicture}
```

(This example list requires `\usetikzlibrary{plotmarks}`).
The input format is described below in more details.

3. The last method is to combine 1. and 2.: Define named cycle lists in the preamble and use them with ‘`cycle list name`’:

\pgfplotscreateplotcyclelist{⟨*name*⟩}{⟨*list*⟩}

```
\pgfplotscreateplotcyclelist{mylist}{%
    {blue,mark=*},
    {red,mark=square},
    {dashed,mark=o},
    {loosely dotted,mark=+},
    {brown!60!black,mark options={fill=brown!40},mark=otimes*}}
...
\begin{axis}[cycle list name=mylist]
    ...
\end{axis}
```

The format of {⟨*list*⟩}: The argument ⟨*list*⟩ is usually a comma separated list of lists of style keys like colors, line styles, marker types and marker styles. This "comma list of comma lists" structure requires to encapsulate the inner list using curly braces:

```
\pgfplotscreateplotcyclelist{mylist}{%
    {blue,mark=*},
    {red,mark=square},
    {dashed,mark=o},
    {loosely dotted,mark=+},
    {brown!60!black,mark options={fill=brown!40},mark=otimes*}}
```

Alternatively, one can terminate the inner lists (i.e. those for one single plot) with '\\':

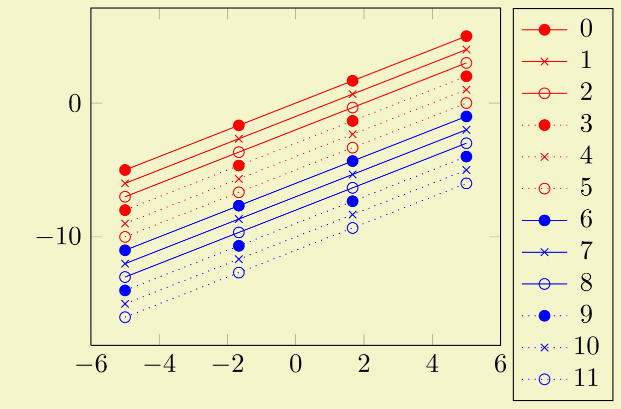
```
\begin{axis}[cycle list={%
    blue,mark=*\\%
    red,mark=square\\%
    dashed,mark=o\\%
    loosely dotted,mark=+\\%
    brown!60!black,mark options={fill=brown!40},mark=otimes*\\%
}
]
...
\end{axis}
```

In this case, the *last* entry also needs a terminating '\\', but one can omit braces around the single entries.

Remark: It is possible to call \pgfplotsset{cycle list={⟨*a list*⟩}} or cycle list name *between* plots. Such a setting remains effective until the end of the current TEX group (that means curly braces). Every \addplot command queries the cycle list using the plot index; it doesn't hurt if cycle lists have changed in the meantime.

/pgfplots/cycle multi list=⟨*list 1*⟩\nextlist⟨*list 2*⟩\nextlist···

Allows to supply more than one cycle list in a way such that each one contributes to the plot style. This is probably best explained using an example:



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}[
    cycle multi list={
      red,blue\nextlist
      solid,{dotted,mark options={solid}}\nextlist
      mark=*,mark=x,mark=o
    },
    samples=3,
    legend entries={0,...,20},
    legend pos=outer north east
]
    \addplot {x};
    \addplot {x-1};
    \addplot {x-2};
    \addplot {x-3};
    \addplot {x-4};
    \addplot {x-5};
    \addplot {x-6};
    \addplot {x-7};
    \addplot {x-8};
    \addplot {x-9};
    \addplot {x-10};
    \addplot {x-11};
\end{axis}
\end{tikzpicture}
```

The provided `cycle multi list` consists of three lists. The style for a single plot is made up using elements of each of the three lists: the first plot has style `red,solid,mark=*`, the second has `red,solid,mark=x`, the third has `red,solid,mark=o`. The fourth plot restarts the third list and uses the next one of list 2: it has `red,dotted,mark options={solid},mark=*` and so on.

The last list will always be advanced for a new plot. The list before the last (in our case the second list) will be advanced after the last one has been reset. In other words: `cycle multi list` allows a composition of different `cycle list` in a lexicographical way[23].

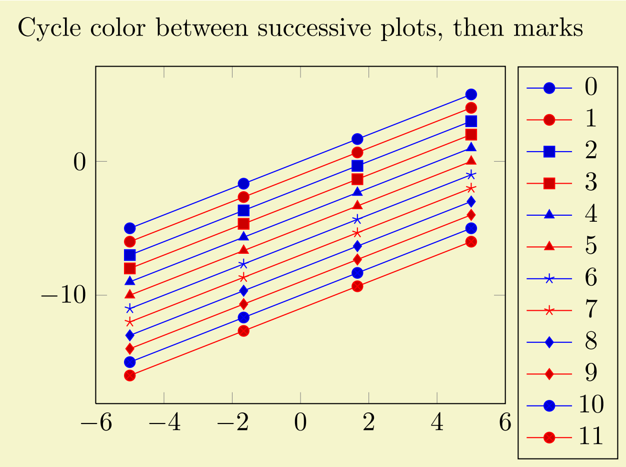
The argument for `cycle multi list` is a sequence of arguments as they would have been provided for `cycle list`, separated by `\nextlist`. In addition to providing a new cycle list, the ⟨*list i*⟩ elements can also denote `cycle list name` values (including the special `auto` cycle list which is the most recently assigned `cycle list` or `cycle list name`). The final `\nextlist` is optional.

The list in our example above could have been written as

```
\begin{axis}[
    cycle multi list={
        red\\blue\\\nextlist
        solid\\dotted,mark options={solid}\\\nextlist
        mark=*\\mark=x\\mark=o\\
    }]
```

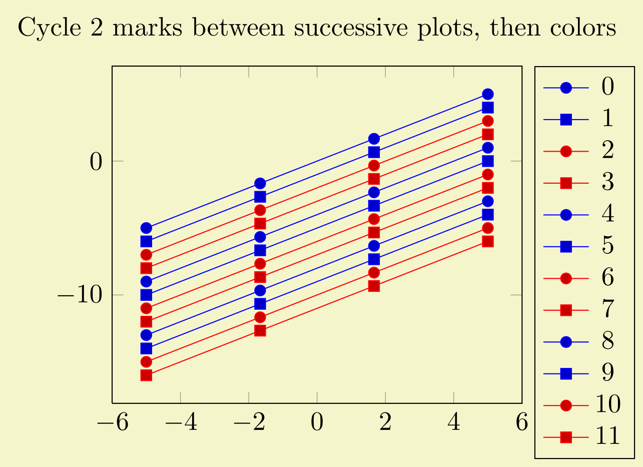
as well (note the terminating `\\` commands!).

[23] For those who prefer formulas: The plot with index $0 \leq i < N$ will use cycle list offsets $i_0, i_1, \ldots, i_k$, $0 \leq i_m < N_m$ where k is the number of arguments provided to `cycle multi list` and N_m is the number of elements in the mth cycle list. The offsets i_m are computed in a loop `{ int tmp=i; for( int m=k-1; m>=0; m=m-1 ) { i_m = tmp%N_m; tmp = tmp/N_m; }}`.



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}[
    title={Cycle color between successive plots, then marks},
    cycle multi list={
        mark list\nextlist
        blue,red%
    },
    samples=3,
    legend entries={0,...,20},
    legend pos=outer north east
]
    \addplot {x};
    \addplot {x-1};
    \addplot {x-2};
    \addplot {x-3};
    \addplot {x-4};
    \addplot {x-5};
    \addplot {x-6};
    \addplot {x-7};
    \addplot {x-8};
    \addplot {x-9};
    \addplot {x-10};
    \addplot {x-11};
\end{axis}
\end{tikzpicture}
```

Using Sub–Lists The ⟨*list i*⟩ entry can also contain just the first n elements of an already known cycle list name using the syntax `[`⟨*number*⟩ `of]`⟨*cycle list name*⟩. For example `[2 of]mark list` will use the first 2 elements of `mark list`:

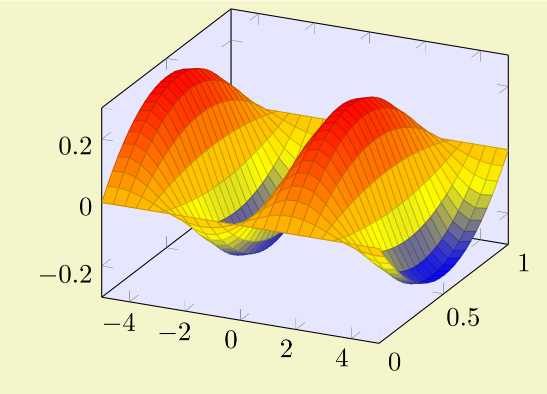


```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}[
    title={Cycle 2 marks between successive plots, then colors},
    cycle multi list={%
        blue,red\nextlist
        [2 of]mark list
    },
    samples=3,
    legend entries={0,...,20},
    legend pos=outer north east
]
    \addplot {x};
    \addplot {x-1};
    \addplot {x-2};
    \addplot {x-3};
    \addplot {x-4};
    \addplot {x-5};
    \addplot {x-6};
    \addplot {x-7};
    \addplot {x-8};
    \addplot {x-9};
    \addplot {x-10};
    \addplot {x-11};
\end{axis}
\end{tikzpicture}
```

4.6.7 Axis Background

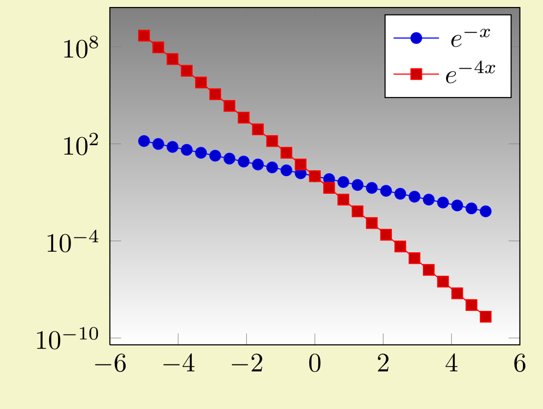
/pgfplots/axis background (initially empty)

This is a style to configure the appearance of the axis as such. It can be defined and/or changed using the axis background/.style={⟨*options*⟩} method. A background path will be generated with ⟨*options*⟩, which may contain fill colors or shadings.



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{axis}[
        axis background/.style={fill=blue!10}]

    \addplot3[surf,y domain=0:1]
        {sin(deg(x)) * y*(1-y)};

    \end{axis}
\end{tikzpicture}
```

Please note that legends are filled with white in the default configuration.



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{semilogyaxis}[
        axis background/.style={
            shade,top color=gray,bottom color=white},
        legend style={fill=white}]

    \addplot {exp(-x)};
    \addplot {exp(-4*x)};
    \legend{$e^{-x}$,$e^{-4x}$}
    \end{semilogyaxis}
\end{tikzpicture}
```

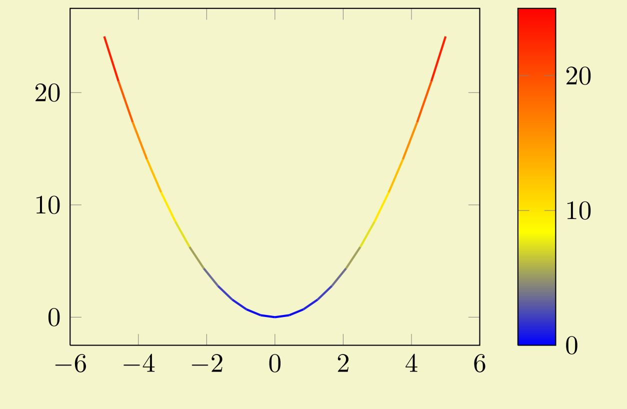
Details about fill and shade can be found in the Ti*k*Z manual, [5].

4.7 Providing Color Data - Point Meta

PGFPLOTS provides features which modify plots depending on a special coordinate, the "point meta data". For example, scatter plots may vary marker colors, size or appearance depending on this special data. Surface and mesh plots are another example: here, the color of a surface patch (or mesh part) depends on "point meta".

The common idea idea is to tell PGFPLOTS how to get this data. It is not necessary to provide data explicitly – in many cases, the data which is used to color surface patches or marker colors is the plot's y or z coordinate. The method used to tell PGFPLOTS where to find "point meta data" is the `point meta` key.

A further common idea is the use of color maps: if the point meta data is in the interval $[m_{\min}, m_{\max}]$, the point meta coordinate $m = m_{\min}$ will get the lowest color provided by the color map while $m = m_{\max}$ will get the highest color provided by the color map. Any coordinate between this values will be mapped linearly: for example, the mean $m = 1/2(m_{\max} + m_{\min})$ will get the middle color of the color map. This is why "point meta" is sometimes called "color data".



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{axis}[colorbar]
        \addplot[mesh,point meta=y,thick] {x^2};
    \end{axis}
\end{tikzpicture}
```

/pgfplots/**point meta**=none|⟨*expression*⟩|x|y|z|f(x)|explicit|explicit symbolic (initially none)

The `point meta` key tells PGFPLOTS where to get the special point meta data. Please note that `point meta` and `scatter src` is actually the same – `scatter src` is an alias for `point meta`. Thus, the summary provided for `scatter src` on page 55 covers the same topics. However, the main reference for `point meta` is here.

`none` The initial choice `none` disables point meta data, resulting in no computational work. Any other choice will activate the computation of upper and lower ranges for point meta data, i.e. the computation of $[m_{\min}, m_{\max}]$.

`x` The choice `x` uses the already available x coordinates as point meta data. This does always refer to the *final* x coordinates after any user transformations, logarithms, stacked plot computations etc. have been applied. Consider using `rawx` if you need the unprocessed coordinate value here.

`y`

`z` The choices `y` and `z` are similar: they use the y or z coordinates respectively as point meta data. Consequently, these three choices do *not* need any extra data. As for `x`, there are math constants `rawy` and `rawz` which yield the unprocessed y and z value, respectively.

`f(x)` This will use the last available coordinate, in other words: it is the same as `y` for two dimensional plots and `z` for three dimensional ones.

`explicit` This choice tells PGFPLOTS to expect *numerical* point meta data which is provided explicitly in the coordinate input streams. This data will be transformed linearly into the current color map as it has been motivated above.

How point meta data is provided for `plot coordinates`, `plot table` and the other input methods is described in all detail in section 4.2.1 – but we provide small examples here to summarize the possibilities:

```
%  for ’coordinates’:
%  provide color data explicitly using [<data>]
%  behind coordinates:
\addplot+[point meta=explicit]
    coordinates {
        (0,0) [1.0e10]
        (1,2) [1.1e10]
        (2,3) [1.2e10]
        (3,4) [1.3e10]
        %  ...
    };
```

```
%  for ’table’:
%  Assumes a datafile.dat like
%  xcolname  ycolname    colordata
%  0         0           0.001
%  1         2           0.3
%  2         2.1         0.4
%  3         3           0.5
%  ...
%  the file may have more columns.
\addplot+[point meta=explicit]
    table[x=xcolname,y=ycolname,meta=colordata]
        {datafile.dat};
%  or, equivalently (perhaps a little bit slower):
\addplot+[point meta=\thisrow{colordata}]
    table[x=xcolname,y=ycolname]
        {datafile.dat};
```

```
%  for ’file’:
%  Assumes a datafile.dat like
%  0         0           0.001
%  1         2           0.3
%  2         2.1         0.4
%  3         3           0.5
%  ...
%  the first three columns will be used here as x,y and meta,
%  resp.
\addplot+[point meta=explicit]
    file {datafile.dat};
```

```
%  ’table’ using expressions which may depend on all
%  columns:
%  Assumes a datafile.dat like
%  xcolname  ycolname    anything    othercol
%  0         0           4           15
%  1         2           5           20
%  2         2.1         8           30
%  3         3           42          40
%  ...
%  the file may have more columns.
\addplot+[point meta={0.5*(\thisrow{anything} + sqrt(\thisrow{othercol}))}]
    table[x=xcolname,y=ycolname]
        {datafile.dat};
```

Thus, there are several methods to provide point meta (color data). The key for the choice `explicit` is that some data is provided explicitly – although `point meta` doesn’t know how. The data is expected to be of numerical type and is mapped linearly into the range $[0, 1000]$ (maybe for use in the current color map).

`explicit symbolic` The choice `explicit symbolic` is very similar to `explicit` in that it expects extra data by the coordinate input routines. However, `explicit symbolic` does not necessarily expect numerical data: you can provide any sort of symbols. One might provide a set of styles, one for each class in a scatter plot. This is realised using `scatter/classes`, see page 57. Input data is provided in the same fashion as mentioned above for the choice `explicit`.

Currently, this choice can only be used for scatter plots.

⟨*expression*⟩ This choice finally allows to compute point meta data using a mathematical expression. The ⟨*expression*⟩ may depend on `x`, `y`, `z` which yield the current x, y or z coordinate, respectively.

The coordinates are completely processed (transformations, logs) as mentioned above for the choice `x`. Furthermore, the ⟨*expression*⟩ may depend on commands which are valid during `\addplot` like `\plotnum` or `\coordindex` (see section 4.22 for details). Computations are performed using the floating point unit of PGF, and all supported arithmetical operations can be used.

In essence, the ⟨*expression*⟩ may depend on everything which is known to all `\addplot` commands: the x, y and (if any) z coordinates. In addition, it may depend upon `rawx`, `rawy` or `rawz`. These three expressions yield the unprocessed x, y or z value as it has been found in the input stream (no logs, no user transformations). If used together with `plot table`, you may also access other table columns (for example with `\thisrow{`⟨*colname*⟩`}`).

The ⟨*expression*⟩ is checked after the other possible choices have already been evaluated. In other words, the statement

```
point meta=explicit, point meta=meta^2+3
```

will evaluate the expression with `meta` set to whatever data has been provided explicitly.

It has been mentioned several times that one application of point meta data is to determine (marker/-face/edge) colors using a linear map into the range $[0, 1000]$ (maybe for use in the current color map). This map works as follows: it is a function

$$\phi\colon [m_{\min}, m_{\max}] \to [0, 1000]$$

with

$$\phi(m) = \frac{m - m_{\min}}{1000}$$

such that $\phi(m_{\min}) = 0$ and $\phi(m_{\max}) = 1000$. The value 1000 is – per convention – the upper limit of all color maps. Now, if a coordinate (or edge/face) has the point meta data m, its color will be determined using $\phi(m)$: it is the color at $\phi(m)$‰ of the current color map.

This transformation depends on the interval $[m_{\min}, m_{\max}]$ which, in turn, can be modified using the keys `point meta rel`, `point meta min` and `point meta max` described below.

The untransformed point meta data is available in the macro `\pgfplotspointmeta` (only in the correct context, for example the scatter plot styles or the `scatter/@pre marker code` interface). This macro contains a low level floating point number (unless it is non-parsed string data). The transformed data will be available in the macro `\pgfplotspointmetatransformed` and is in fixed point representation. It is expected to be in the range $[0, 1000]$.

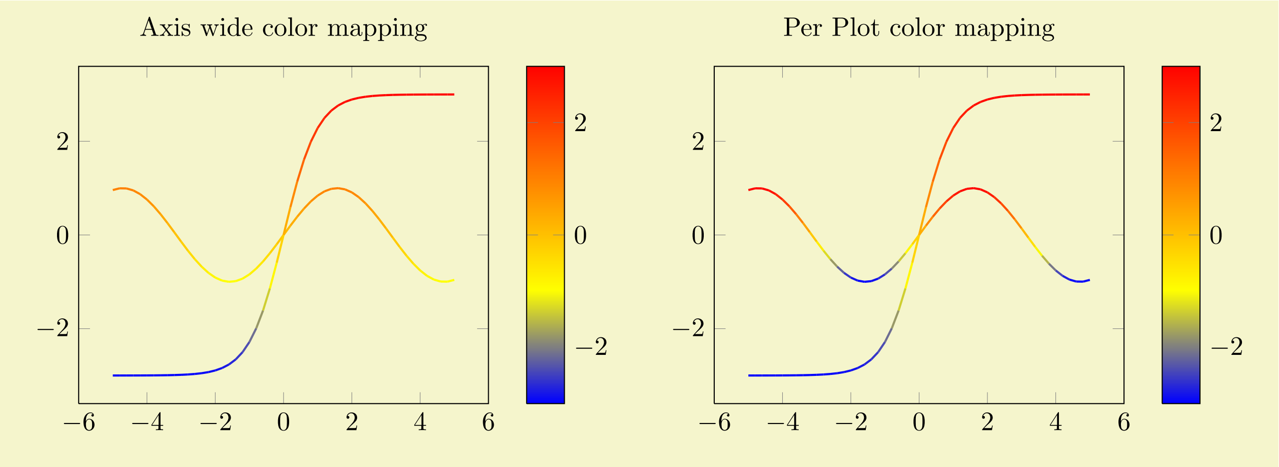
/pgfplots/`set point meta if empty={`⟨*point meta source*⟩`}`

Sets `point meta=`⟨*point meta source*⟩, but only if `point meta=none` currently. This is used for `scatter`, `mesh` and `surf` with `set point meta if empty=f(x)`.

/pgfplots/`point meta rel=axis wide|per plot` (initially `axis wide`)

As already explained in the documentation for `point meta`, one application for point meta data is to determine colors using the current color map and a linear map from point meta data into the current color map. The question is how this linear map is computed.

The key `point meta rel` configures whether the interval of all point meta coordinates, $[m_{\min}, m_{\max}]$ is computed as maximum over all plots in the complete axis (the choice `axis wide`) or only for one particular plot (the choice `per plot`).



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
   \begin{axis}[
      title=Axis wide color mapping,
      colorbar,
      samples=50,point meta rel=axis wide,
      point meta=y]

      \addplot[mesh,thick] {sin(deg(x))};
      \addplot[mesh,thick] {3*tanh(x)};
   \end{axis}
\end{tikzpicture}
~
\begin{tikzpicture}
   \begin{axis}[
      title=Per Plot color mapping,
      colorbar,
      samples=50,
      point meta rel=per plot,
      point meta=y]

      \addplot[mesh,thick] {sin(deg(x))};
      \addplot[mesh,thick] {3*tanh(x)};
   \end{axis}
\end{tikzpicture}
```

/pgfplots/point meta min={⟨*number*⟩}
/pgfplots/point meta max={⟨*number*⟩}

These keys allow to define the range required for the linear map of point meta data into the range $[0, 1000]$ (for example, for current maps) explicitly. This is necessary if the same mapping shall be used for more than one axis.

Remarks about special cases:

- It is possible to provide limits partially; in this case, only the missing limit will be computed.
- If point meta data falls outside of these limits, the linear transformation is still well defined which is acceptable (unless the interval is of zero length). However, color data can't be outside of these limits, so color bars perform a truncation.
- This key can be provided for single plots as well as for the complete axis (or for both).
- If meta limits are provided for a single plot, these limits may also contribute to the axis wide meta interval.

/pgfplots/colormap access=map|direct (initially map)

This key configures how point meta data is used to determine colors from a color map. The initial configuration map performs the linear mapping operation explained above. The choice direct does not perform any transformation; it takes the point meta as integer indizes into the current color map.

Consequently, there is no interpolation between colors in the color map, there will only be as many colors as the color map contains explicitly.

Some more details:

- If there are m colors in the color map and the color data falls outside of $[0, m-1]$, it will be pruned to either the first or the last color.
- If color data is a real number, it will be truncated to the next smaller integer.
- This key does not work for `shader=interp` (note that this shader will always interpolate in the color map).

Attention: This feature is experimental, I did not have time to test it.

4.8 Axis Descriptions

Axis descriptions are labels for x and y axis, titles, legends and the like. Axis descriptions are drawn after the plot is finished and they are not subjected to clipping.

4.8.1 Placement of Axis Descriptions

This section describes how to *modify* the placement of titles, labels, legends and other axis descriptions. It may be skipped at first reading.

There are different methods to place axis descriptions. One of them is to provide coordinates relative to the axis' rectangle such that `(0,0)` is the lower left corner and `(1,1)` is the upper right corner – this is very useful for figure titles or legends. Coordinates of this type, i.e. without unit like `(0,0)` or `(1.03,1)`, are called `axis description cs` (the `cs` stands for "coordinate system"). One other method is of primary interest for axis labels – they should be placed near the tick labels, but it a way that they don't overlap or obscure tick labels. Furthermore, axis labels shall be placed such that they are automatically moved if the axis is rotated (or tick labels are moved to the right side of the figure). There is a special coordinate system to realize these two demands, the `ticklabel cs`.

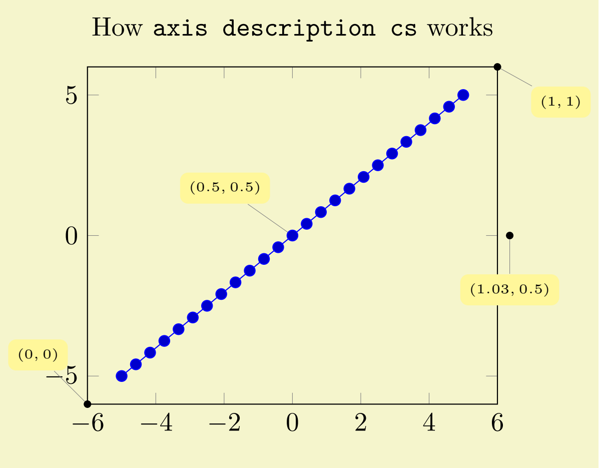
In the following, the two coordinate systems `axis description cs` and `ticklabel cs` are described in more details. It should be noted that `axis description cs` is used automatically, so it might never be necessary to use it explicitly.

Coordinate system `axis description cs`

A coordinate system which is used to place axis descriptions. Whenever the option '`at={(`$\langle x\rangle$`,`$\langle y\rangle$`)}`' occurs in `label style`, `legend style` or any other axis description, `(`$\langle x\rangle$`,`$\langle y\rangle$`)` is interpreted to be a coordinate in `axis description cs`.

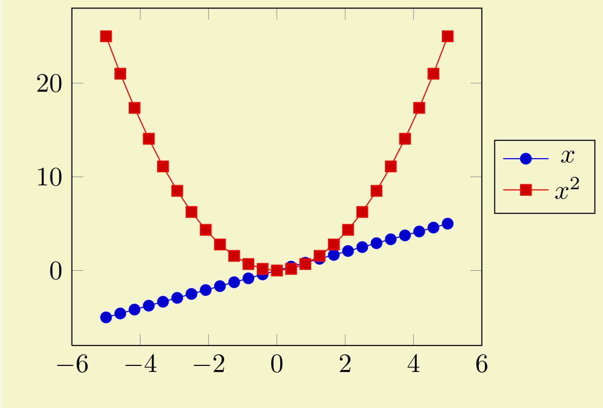
The point $(0, 0)$ is always the lower left corner of the tightest bounding box around the axis (without any descriptions or ticks) while the point $(1, 1)$ is the upper right corner of this bounding box.

In most cases, it is *not* necessary to explicitly write `axis description cs` as it is the default coordinate system for any axis description. An example for how coordinates are placed is shown below.



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
%  [See the TikZ manual if you'd like to learn about nodes and pins]
\begin{tikzpicture}
    \tikzset{
        every pin/.style={fill=yellow!50!white,rectangle,rounded corners=3pt,font=\tiny},
        small dot/.style={fill=black,circle,scale=0.3}
    }
    \begin{axis}[
        clip=false,
        title=How \texttt{axis description cs} works
    ]
    \addplot {x};

    \node[small dot,pin=120:{$(0,0)$}]      at (axis description cs:0,0) {};
    \node[small dot,pin=-30:{$(1,1)$}]      at (axis description cs:1,1) {};
    \node[small dot,pin=-90:{$(1.03,0.5)$}] at (axis description cs:1.03,0.5) {};
    \node[small dot,pin=125:{$(0.5,0.5)$}]  at (axis description cs:0.5,0.5) {};
    \end{axis}
\end{tikzpicture}
```

Axis descriptions are Ti*k*Z nodes, that means all placement and detail options of [5] apply. The point on the node's boundary which is actually shifted to the at coordinate needs to be provided with an anchor (cf [5, Nodes and Edges]):



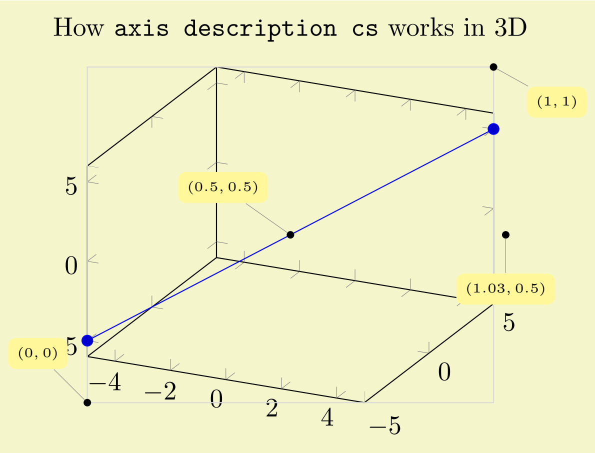
```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{axis}[
        legend entries={$x$,$x^2$},
        legend style={
            at={(1.03,0.5)},
            anchor=west
        }
    ]
    \addplot {x};
    \addplot {x^2};
    \end{axis}
\end{tikzpicture}
```

Standard anchors of nodes are `north`, `east`, `south`, `west` and mixed components like `north east`. Please refer to [5] for a complete documentation of anchors.

Remarks:

- Each of the anchors described in section 4.18 can be described by `axis description cs` as well.
- The `axis description cs` is independent of axis reversals or skewed axes. Only for the default configuration of boxed axes is it the same as `rel axis cs`, i.e. `(0,0)` is the same as the smallest axis coordinate and `(1,1)` is the largest one in case of standard boxed axes[24].
- Even for three dimensional axes, the `axis description cs` is still two-dimensional: it always refers to coordinates relative to the tightest bounding box around the axis (without any descriptions or ticks).

[24] This was different in versions before 1.3: earlier versions did not have the distinction between `axis description cs` and `rel axis cs`.



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
%  the same as above for 3D ...
%  [See the TikZ manual if you'd like to learn about nodes and pins]
\begin{tikzpicture}
    \tikzset{
        every pin/.style={fill=yellow!50!white,rectangle,rounded corners=3pt,font=\tiny},
        small dot/.style={fill=black,circle,scale=0.3}
    }
    \begin{axis}[
        clip=false,
        title=How \texttt{axis description cs} works in 3D
    ]
    \addplot3 coordinates {(-5,-5,-5) (5,5,5)};

    \draw[black!15] (axis description cs:0,0) rectangle (axis description cs:1,1);

    \node[small dot,pin=120:{$(0,0)$}]      at (axis description cs:0,0) {};
    \node[small dot,pin=-30:{$(1,1)$}]      at (axis description cs:1,1) {};
    \node[small dot,pin=-90:{$(1.03,0.5)$}] at (axis description cs:1.03,0.5) {};
    \node[small dot,pin=125:{$(0.5,0.5)$}]  at (axis description cs:0.5,0.5) {};
    \end{axis}
\end{tikzpicture}
```

- Since the view does not influence these positions, axis description cs might not be a good choice for axis labels in 3D. The ticklabel cs is used in this case.

Coordinate system xticklabel cs
Coordinate system yticklabel cs
Coordinate system zticklabel cs
Coordinate system ticklabel cs

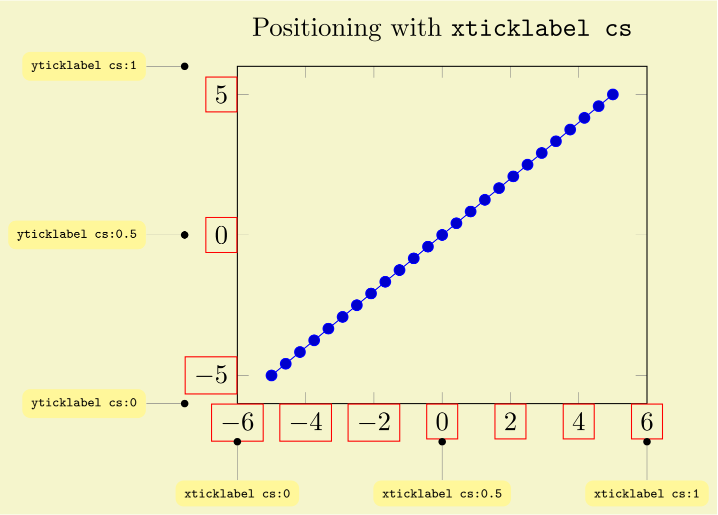
A set of special coordinate systems intended to place axis descriptions (or any other drawing operation) besides tick labels, in a way such that neither tick labels nor the axis as such are obscured.

See also xlabel near ticks as one main application of ticklabel cs.

The xticklabel cs (and its variants) always refer to one, uniquely identified axis: the one which is (or would be) annotated with tick labels.

The ticklabel cs (without explicit x, y or z) can only be used in contexts where the axis character is known from context (for example, inside of xlabel style – there, the ticklabel cs is equivalent to xticklabel cs).

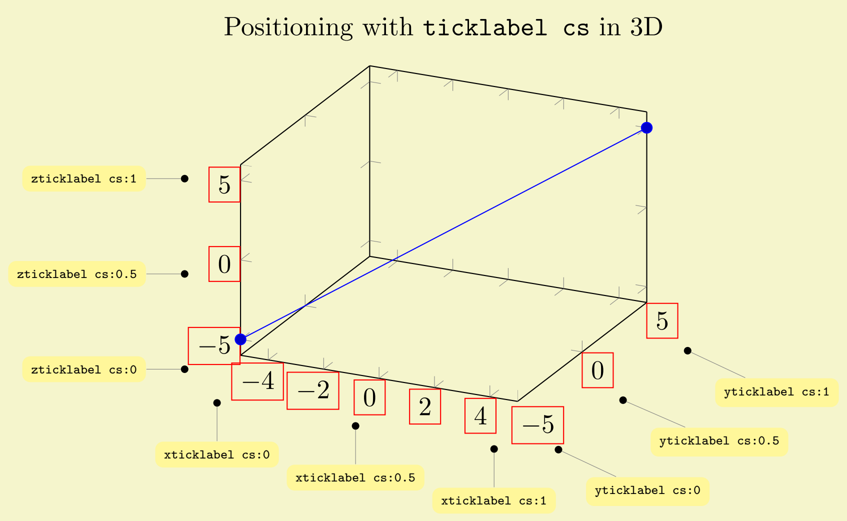
Each of these coordinate systems allows to specify points on a straight line which is placed parallel to an axis containing tick labels, moved away just far enough to avoid overlaps with the tick labels:



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\tikzset{
    every pin/.style={fill=yellow!50!white,rectangle,rounded corners=3pt,font=\tiny},
    small dot/.style={fill=black,circle,scale=0.3}
}
\begin{tikzpicture}
\begin{axis}[
    clip=false,
    ticklabel style={draw=red},
    title=Positioning with \texttt{xticklabel cs}]
    \addplot {x};
    \node[small dot,pin=-90:{\texttt{xticklabel cs:0}}]     at (xticklabel cs:0) {};
    \node[small dot,pin=-90:{\texttt{xticklabel cs:0.5}}]   at (xticklabel cs:0.5) {};
    \node[small dot,pin=-90:{\texttt{xticklabel cs:1}}]     at (xticklabel cs:1) {};


    \node[small dot,pin=180:{\texttt{yticklabel cs:0}}]     at (yticklabel cs:0) {};
    \node[small dot,pin=180:{\texttt{yticklabel cs:0.5}}]   at (yticklabel cs:0.5) {};
    \node[small dot,pin=180:{\texttt{yticklabel cs:1}}]     at (yticklabel cs:1) {};
\end{axis}
\end{tikzpicture}
```

The basic idea is to place coordinates on a straight line which is parallel to the axis containing tick labels – but shifted such that the line does not cut through tick labels.

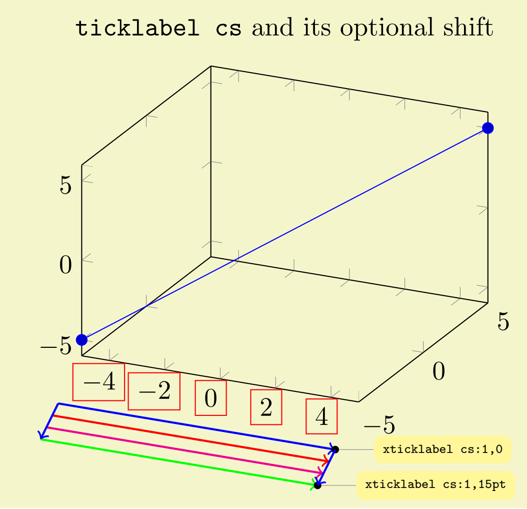
Of course, it is relatively simple to get the same coordinates as in the two dimensional example above with axis description cs, except that ticklabel cs always respects the tick label sizes appropriately. However, ticklabel cs becomes far superior when it comes to three dimensional positioning:



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
%  the same as above for 3D ...
\begin{tikzpicture}
    \tikzset{
        every pin/.style={fill=yellow!50!white,rectangle,rounded corners=3pt,font=\tiny},
        small dot/.style={fill=black,circle,scale=0.3}
    }
    \begin{axis}[
        ticklabel style={draw=red},
        clip=false,
        title=Positioning with \texttt{ticklabel cs} in 3D
    ]
    \addplot3 coordinates {(-5,-5,-5) (5,5,5)};

    \node[small dot,pin=-90:{\texttt{xticklabel cs:0}}]     at (xticklabel cs:0) {};
    \node[small dot,pin=-90:{\texttt{xticklabel cs:0.5}}]   at (xticklabel cs:0.5) {};
    \node[small dot,pin=-90:{\texttt{xticklabel cs:1}}]     at (xticklabel cs:1) {};

    \node[small dot,pin=-45:{\texttt{yticklabel cs:0}}]     at (yticklabel cs:0) {};
    \node[small dot,pin=-45:{\texttt{yticklabel cs:0.5}}]   at (yticklabel cs:0.5) {};
    \node[small dot,pin=-45:{\texttt{yticklabel cs:1}}]     at (yticklabel cs:1) {};

    \node[small dot,pin=180:{\texttt{zticklabel cs:0}}]     at (zticklabel cs:0) {};
    \node[small dot,pin=180:{\texttt{zticklabel cs:0.5}}]   at (zticklabel cs:0.5) {};
    \node[small dot,pin=180:{\texttt{zticklabel cs:1}}]     at (zticklabel cs:1) {};
    \end{axis}
\end{tikzpicture}
```

The coordinate `ticklabel cs:0` is associated to the lower axis limit while `ticklabel cs:1` is near the upper axis limit. The value `0.5` is in the middle of the axis, any other values (including negative values or values beyond 1) are linearly interpolated inbetween.

The `ticklabel cs` also accepts a second (optional) argument: a shift "away" from the tick labels. The shift points to a vector which is orthogonal to the associated axis, away from the tick labels. A shift of `0pt` is directly at the edge of the tick labels in direction of the normal vector, positive values move the position away and negative closer to the tick labels.



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\tikzset{
    every pin/.style={fill=yellow!50!white,rectangle,rounded corners=3pt,font=\tiny},
    small dot/.style={fill=black,circle,scale=0.3}
}
\begin{tikzpicture}
    \begin{axis}[
        xticklabel style={draw=red},
        clip=false,
        title=\texttt{ticklabel cs} and its optional shift
    ]
    \addplot3 coordinates {(-5,-5,-5) (5,5,5)};

    \draw[blue,thick,->]      (xticklabel cs:0,0)     -- (xticklabel cs:1,0);
    \draw[red,thick,->]       (xticklabel cs:0,5pt)   -- (xticklabel cs:1,5pt);
    \draw[magenta,thick,->]   (xticklabel cs:0,10pt)  -- (xticklabel cs:1,10pt);
    \draw[green,thick,->]     (xticklabel cs:0,15pt)  -- (xticklabel cs:1,15pt);
    \node[small dot,pin=0:{\texttt{xticklabel cs:1,0}}]      at (xticklabel cs:1,0) {};
    \node[small dot,pin=0:{\texttt{xticklabel cs:1,15pt}}]   at (xticklabel cs:1,15pt) {};

    \draw[blue,thick,->]      (xticklabel cs:0,0)     -- (xticklabel cs:0,15pt);
    \draw[blue,thick,->]      (xticklabel cs:1,0)     -- (xticklabel cs:1,15pt);
    \end{axis}
\end{tikzpicture}
```

Whenever the `ticklabel cs` is used, the anchor should be set to `anchor=near ticklabel` (see below).

There is one speciality: if you reverse an axis (with `x dir=reverse`), points provided by `ticklabel cs` will be *unaffected* by the axis reversal. This is intented to provide consistent placement even for reversed axes. Use `allow reversal of rel axis cs=false` to disable this feature.

Besides the mentioned positioning methods, there is also the predefined node `current axis`. The anchors of `current axis` can also be used to place descriptions: At the time when axis descriptions are drawn, all anchors which refer to the axis origin (that means the "real" point $(0,0)$) or any of the axis corners can be referenced using `current axis.`⟨*anchor name*⟩. Please see section 4.18, Alignment, for further details.

4.8.2 Alignment of Axis Descriptions

This section describes how to modify the default alignment of axis descriptions. It can be skipped at first reading.

The two topics positioning and alignment always work together: *positioning* means to select an appropriate coordinate and *alignment* means to select an anchor inside of the description which will actually be moved to the desired position.

Ti*k*Z uses many anchors to provide alignment; most of them are named like `north`, `north east` etc. These names hold for any axis description as well (as axis description are Ti*k*Z nodes). Readers can learn details about this topic in the Ti*k*Z manual [5] or some more advice in section 4.18.

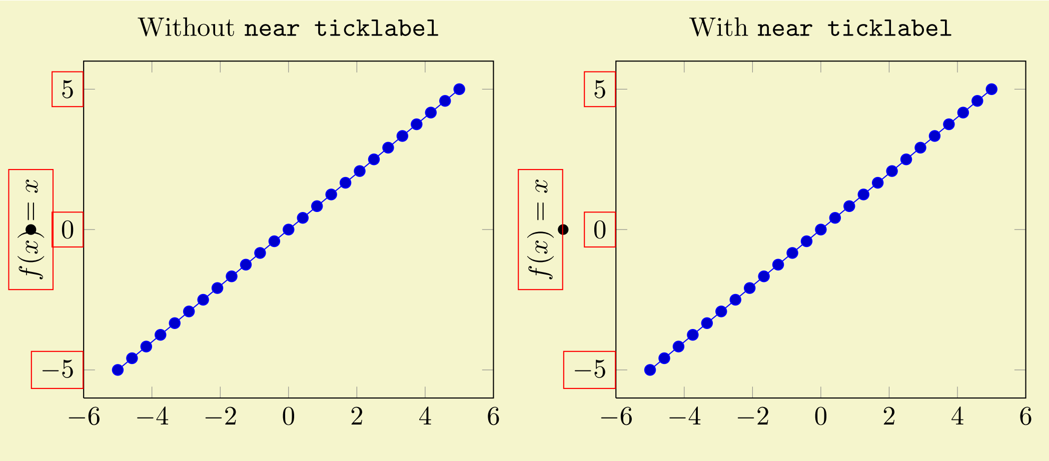
When it comes to axis descriptions, PGFPLOTS offers some specialized anchors and alignment methods which are described below.

Anchor `near xticklabel`
Anchor `near yticklabel`
Anchor `near zticklabel`
Anchor `near ticklabel`

These anchors can be used to align at the part of a node (for example, an axis description) which is *nearest* to the tick labels of a particular axis (or nearest to the position where tick labels would have been drawn if there were any).

These anchors are used for axis labels, especially for three dimensional axes. Furthermore, they are used for every tick label.

Maybe it is best to demonstrate it by example:



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{axis}[
        title=Without \texttt{near ticklabel},
        ylabel={$f(x)=x$},
        every axis y label/.style=
            {at={(ticklabel cs:0.5)},rotate=90,anchor=center},
        clip=false,%  to display the \path below
        ylabel style={draw=red},
        yticklabel style={draw=red}
    ]

        \addplot {x};

        %  visualize the position:
        \fill (yticklabel cs:0.5) circle(2pt);
    \end{axis}
\end{tikzpicture}%
~
\begin{tikzpicture}
    \begin{axis}[
        title=With \texttt{near ticklabel},
        ylabel={$f(x)=x$},
        every axis y label/.style=
            {at={(ticklabel cs:0.5)},rotate=90,anchor=near ticklabel},
        clip=false,
        ylabel style={draw=red},
        yticklabel style={draw=red}
    ]

        \addplot {x};
        \fill (yticklabel cs:0.5) circle(2pt);
    \end{axis}
\end{tikzpicture}
```

The motivation is to place nodes such that they are anchored next to the tick label, regardless of the node's rotation or the position of ticks. The special anchor `near ticklabel` is only available for axis labels (as they have a uniquely identified axis, either x, y or z).

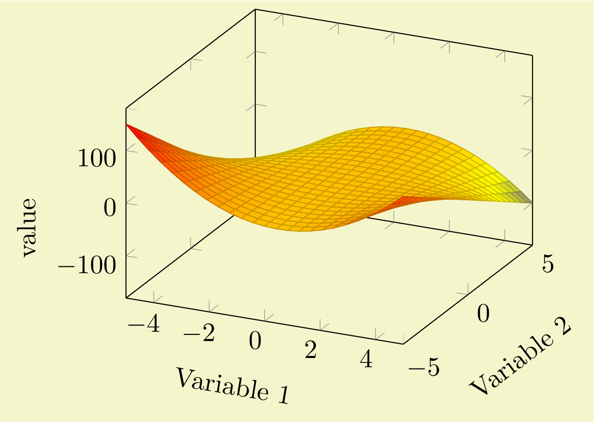
In more detail, the anchor is placed such that first, the node's center is on a line starting in the node's `at` position going in direction of the inwards normal vector of the axis line which contains the tick labels and second, the node does not intrude the axis. This normal vector is the same which is used for the shift argument in `ticklabel cs`: it is orthogonal to the tick label axis. Furthermore, `near ticklabel` inverts the transformation matrix before it computes this intersection point.

The `near ticklabel` anchor and its friends will be added temporarily to any shape used inside of an axis. This includes axis description, but it is not limited to them: it applies to every Ti*k*Z `\node[anchor=near xticklabel] ...` setting.

Note that it is not necessary at all to *have* tick labels in an axis. The anchor will be placed such that it is near the axis on which tick labels *would* be drawn. In fact, every tick label uses `anchor=near ticklabel` as initial configuration.

/tikz/sloped like x axis (no value)
/tikz/sloped like y axis (no value)
/tikz/sloped like z axis (no value)

A key which replaces the rotational / scaling parts of the transformation matrix such that the node is sloped like the provided axis. For two dimensional plots, `sloped like y axis` is effectively the same as `rotate=90`. For a three dimensional axis, this will lead to a larger difference:



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{axis}[
        xlabel=Variable 1,
        ylabel=Variable 2,
        zlabel=value,
        xlabel style={sloped like x axis},
        ylabel style={sloped}
    ]

    \addplot3[surf] {y*x*(1-x)};
    \end{axis}
\end{tikzpicture}
```

Inside of axis labels, `sloped` is an alias `sloped like` ⟨*char*⟩ `axis` with the correct ⟨*char*⟩ chosen automatically.

Please note that rotated text might not look very good (neither on screen nor printed).

4.8.3 Labels

/pgfplots/xlabel={⟨*text*⟩}
/pgfplots/ylabel={⟨*text*⟩}
/pgfplots/zlabel={⟨*text*⟩}

These options set axis labels to {⟨*text*⟩} which is any TEX text. Use curly braces to include special characters, for example "`xlabel={, = characters}`" if characters like '=' or ',' need to be included literally.

Use `xlabel/.add={`⟨*prefix*⟩`}{`⟨*suffix*⟩`}` to modify an already assigned label.

Labels are Ti*k*Z-Nodes which are placed with

```
%  for x:
\node
    [style=every axis label,
    style=every axis x label]

%  for y:
\node
    [style=every axis label,
    style=every axis y label]
```

so their position and appearance can be customized.

Upgrade notice: Since version 1.3, label placement *can* respect the size of adjacent tick labels. Use `\pgfplotsset{compat=1.3}` in the preamble to activate this feature. See `xlabel near ticks` for details.

/pgfplots/xlabel shift={⟨*dimension*⟩} (initially `0pt`)
/pgfplots/ylabel shift={⟨*dimension*⟩} (initially `0pt`)
/pgfplots/zlabel shift={⟨*dimension*⟩} (initially `0pt`)
/pgfplots/label shift={⟨*dimension*⟩}

Shifts labels in direction of the outer normal vector of the axis by an amount of {⟨*dimension*⟩}. The `label shift` sets all three label shifts to the same value.

Attention: This does only work if `\pgfplotsset{compat=1.3}` has been called (More precisely: if `xlabel near ticks` is active for the respective axis).

/pgfplots/**xlabel near ticks** (no value)
/pgfplots/**ylabel near ticks** (no value)
/pgfplots/**zlabel near ticks** (no value)
/pgfplots/**compat**=1.3

These keys place axis labels (like `xlabel`) near the tick labels. If tick labels are small, labels will move closer to the axis. If tick labels are large, axis labels will move away from the axis. This is the default for every three dimensional plot, but it *won't* be used initially for two–dimensional plots for backwards compatibility. Take a look at the definition of `near ticklabel` on page 106 for an example.

The definition of these styles is

```
\pgfplotsset{
    /pgfplots/xlabel near ticks/.style={
        /pgfplots/every axis x label/.style={
            at={(ticklabel cs:0.5)},anchor=near ticklabel
        }
    },
    /pgfplots/ylabel near ticks/.style={
        /pgfplots/every axis y label/.style={
            at={(ticklabel cs:0.5)},rotate=90,anchor=near ticklabel
        }
    }
}
```

It is encouraged to write

```
\pgfplotsset{compat=1.3}
```

in your preamble to install the styles document-wide – it leads to the best output (it avoids unnecessary space). It is not activated initially for backwards compatibility with older versions which used fixed distances from the tick labels.

/pgfplots/**xlabel absolute** (no value)
/pgfplots/**ylabel absolute** (no value)
/pgfplots/**zlabel absolute** (no value)
/pgfplots/**compat**=pre 1.3

Installs placement styles for axis labels such that `xlabel` yields a description of absolute, fixed distance to the axis. This is the initial configuration (for backwards compatibility with versions before 1.3). Use `compat=1.3` to get the most recent, more flexible configuration. Take a look at the definition of `near ticklabel` on page 106 for an example.

These styles are defined by

```
\pgfplotsset{
    /pgfplots/xlabel absolute/.style={%
        /pgfplots/every axis x label/.style={at={(0.5,0)},below,yshift=-15pt},%
        /pgfplots/every x tick scale label/.style={
            at={(1,0)},yshift=-2em,left,inner sep=0pt
        },
    },
    /pgfplots/ylabel absolute/.style={%
        /pgfplots/every axis y label/.style={at={(0,0.5)},xshift=-35pt,rotate=90},
        /pgfplots/every y tick scale label/.style={
            at={(0,1)},above right,inner sep=0pt,yshift=0.3em
        },
    }
}
```

There is no predefined absolute placement style for three dimensional axes.

Whenever possible, consider using `/.append style` instead of overwriting the default styles to ensure compatibility with future versions.

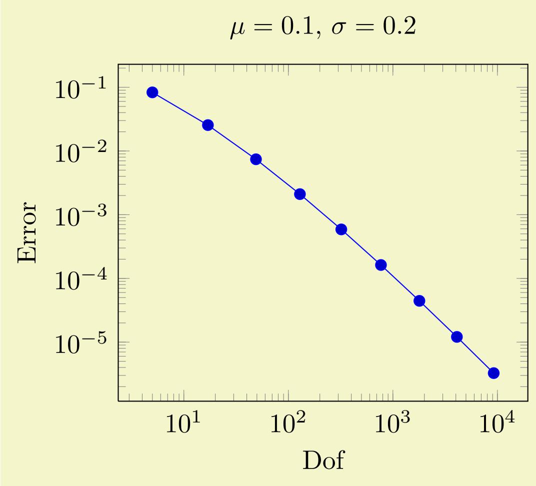
```
\pgfplotsset{every axis label/.append style={...}}
\pgfplotsset{every axis x label/.append style={...}}
\pgfplotsset{every axis y label/.append style={...}}
```

/pgfplots/title={⟨*text*⟩}

Adds a caption to the plot. This will place a Ti*k*Z-Node with

```
\node[every axis title] {text};
```

to the current axis.



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{loglogaxis}[
    xlabel=Dof,ylabel=Error,
    title={$\mu=0.1$, $\sigma=0.2$}]

    \addplot coordinates {
        (5,    8.312e-02)
        (17,   2.547e-02)
        (49,   7.407e-03)
        (129,  2.102e-03)
        (321,  5.874e-04)
        (769,  1.623e-04)
        (1793, 4.442e-05)
        (4097, 1.207e-05)
        (9217, 3.261e-06)
    };
\end{loglogaxis}
\end{tikzpicture}%
```

The title's appearance and/or placing can be reconfigured with

```
\pgfplotsset{title style={at={(0.75,1)}}}
%  or, equivalently,
\pgfplotsset{every axis title/.append style={at={(0.75,1)}}}
```

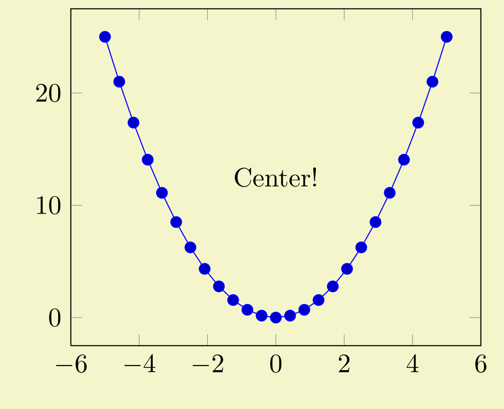
This will place the title at 75% of the x-axis. The coordinate $(0, 0)$ is the lower left corner and $(1, 1)$ the upper right one (see `axis description cs` for details).

Use `title/.add={`⟨*prefix*⟩`}{`⟨*suffix*⟩`}` to modify an already assigned title.

/pgfplots/extra description/.code={⟨...⟩}

Allows to insert `{`⟨*commands*⟩`}` after axis labels, titles and legends have been typeset.

As all other axis descriptions, the code can use $(0, 0)$ to access the lower left corner and $(1, 1)$ to access the upper right one. It won't be clipped.



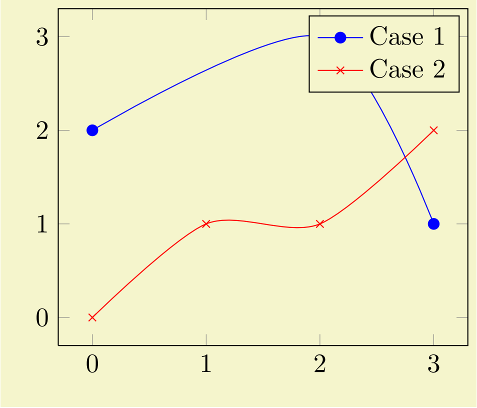
```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\pgfplotsset{every axis/.append style={
    extra description/.code={
        \node at (0.5,0.5) {Center!};
    }}}
\begin{tikzpicture}
    \begin{axis}
    \addplot {x^2};
    \end{axis}
\end{tikzpicture}
```

4.8.4 Legends

Legends can be generated in two ways: the first is to use `\addlegendentry` or `\legend` inside of an axis. The other method is to use the key `legend entries`.

`\addlegendentry{⟨name⟩}`

Adds a single legend entry to the legend list. This will also enable legend drawing.



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}
\addplot[smooth,mark=*,blue] coordinates {
    (0,2)
    (2,3)
    (3,1)
};
\addlegendentry{Case 1}

\addplot[smooth,color=red,mark=x]
    coordinates {
        (0,0)
        (1,1)
        (2,1)
        (3,2)
    };
\addlegendentry{Case 2}
\end{axis}
\end{tikzpicture}
```

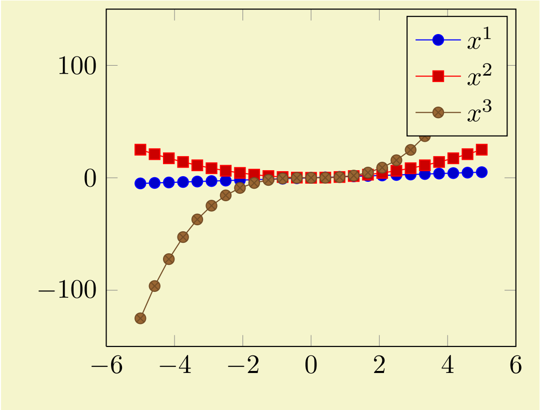
It does not matter where `\addlegendentry` commands are placed, only the sequence matters. You will need one `\addlegendentry` for every `\addplot` command (unless you prefer an empty legend).

Using `\addlegendentry` disables the key `legend entries`.

`\addlegendentryexpanded{⟨TEX text⟩}`

A variant of `\addlegendentry` which provides a method to deal with macros inside of ⟨TEX text⟩.

Suppose ⟨TEX text⟩ contains some sort of parameter which varies *for every plot*. Moreover, you like to use a loop to generate the plots. Then, it is simpler to use `\addlegendentryexpanded`:



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}
    \foreach \p in {1,2,3} {
      \addplot {x^\p};
      \addlegendentryexpanded{$x^\p$}
    }
\end{axis}
\end{tikzpicture}
```

Note that this example wouldn't have worked with `\addlegendentry{$x^\p$}` because the macro `\p` is no longer defined when PGFPLOTS attempts to draw the legend.

The invocation `\addlegendentryexpanded{$x^\p$}` is equivalent to calling `\addlegendentry{$x^2$}` if `\p` expands to 2.

The argument ⟨TEX text⟩ is expanded until nothing but un-expandable material remains (i.e. it uses the TEX primitive `\edef`). Occasionally, `{`⟨TEX text⟩`}` contains parts which should be expanded (like `\p`) and other parts which should be left unexpanded (for example `\pgfmathprintnumber{\p}`). Then, use

`\noexpand\pgfmathprintnumber{\p}`

or, equivalently

`\protect\pgfmathprintnumber{\p}`

to avoid expansion of the macro which follows the `\protect` immediately.

`\legend{⟨list⟩}`

You can use \legend{⟨*list*⟩} to assign a complete legend.

```
\legend{$d=2$,$d=3$,$d=4$,$d=5$,$d=6$}
```

The argument of \legend is list of entries, one for each plot.

Two different delimiters are supported:

1. There are comma–separated lists like

   ```
   \legend{$d=2$,$d=3$,$d=4$,$d=5$,$d=6$}
   ```

   These lists are processed using the PGF \foreach command.

2. It is also possible to delimit the list by '\\'. In this case, the *last element must be terminated* by \\ as well:

   ```
   \legend{$a=1, b=2$\\,$a=2, b=3$\\$a=3, b=5$\\}
   ```

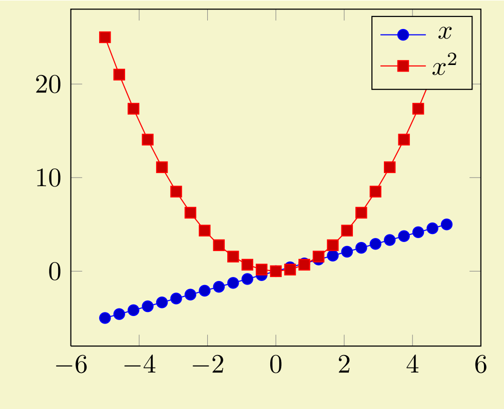
   This syntax simplifies the use of ',' inside of legend entries.

The short marker/line combination shown in legends is acquired from the {⟨*style options*⟩} argument of \addplot.

Using \legend overwrites any other existing legend entries.

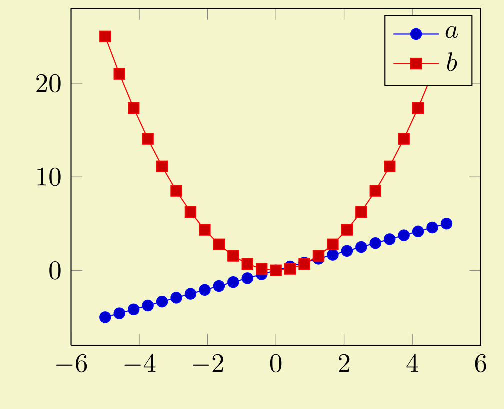
/pgfplots/**legend entries**={⟨*comma separated list*⟩}

This key can be used to assign legend entries just like the commands \addlegendentry and \legend. Again, the positioning is relative to the axis rectangle (unless units like cm or pt are specified explicitly).



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{axis}[legend entries={$x$,$x^2$}]
    \addplot {x};
    \addplot {x^2};
    \end{axis}
\end{tikzpicture}
```

The commands for legend creation take precedence: the key legend entries is only considered if there is no legend command in the current axis.



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{axis}[legend entries={$x$,$x^2$}]
    \addplot {x};
    \addplot {x^2};
    \legend{$a$,$b$}%  overrides the option
    \end{axis}
\end{tikzpicture}
```

Please be careful with whitespaces in {⟨*comma separated list*⟩}: they will contribute to legend entries. Consider using '%' at the end of each line in multiline arguments (the end of line character is also a whitespace in TeX).

\addlegendimage{⟨*options*⟩}

Adds a further "plot style" for legend creation.

Each \addplot command appends its plot style options to a list, and \addlegendimage adds ⟨*options*⟩ to the very same list.

Thus, the effect is as if you had provided \addplot[⟨*options*⟩], but \addlegendimage bypasses all the logic usually associated with a plot. In other words: except for the legend, the state of the axis remains as if the command would not have been drawn. Not even the current plot's index is advanced, making \addlegendimage compatible with any plot styles like ybar.

Use \addlegendimage to provide custom styles into legends, for example to document custom \draw commands inside of an axis.

You can call \label after \addlegendimage just as for a normal style.

4.8.5 Legend Appearance

/pgfplots/every axis legend (style, no value)

The style "every axis legend" determines the legend's position and outer appearance:

```
\pgfplotsset{every axis legend/.append style={
        at={(0,0)},
        anchor=south west}}
```

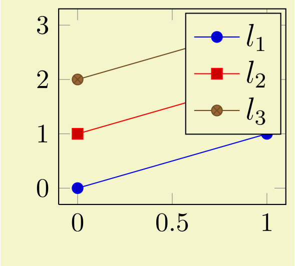
will draw it at the lower left corner of the axis while

```
\pgfplotsset{every axis legend/.append style={
        at={(1,1)},
        anchor=north east}}
```

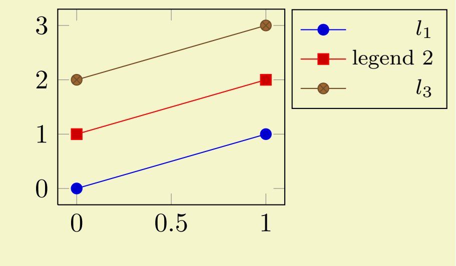
means the upper right corner. The 'anchor' option determines which point *of the legend* will be placed at $(0,0)$ or $(1,1)$.

The legend is a Ti*k*Z-matrix, so one can use any Ti*k*Z option which affects nodes and matrizes (see [5, section 13 and 14]). The matrix is created by something like

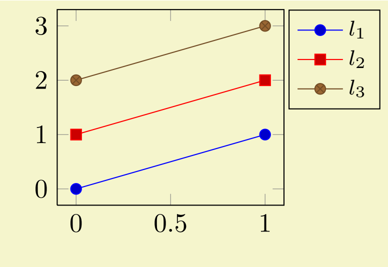
```
\matrix[style=every axis legend] {
    draw plot specification 1 & \node{legend 1}\\
    draw plot specification 2 & \node{legend 2}\\
    ...
};
```



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}[
    %  this modifies 'every axis legend':
    legend style={font=\large}
]
\addplot coordinates {(0,0) (1,1)};
\addplot coordinates {(0,1) (1,2)};
\addplot coordinates {(0,2) (1,3)};
\legend{$l_1$,$l_2$,$l_3$}
\end{axis}
\end{tikzpicture}
```

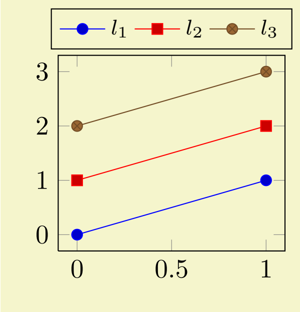


```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}[
    %  align right:
    legend style={
        cells={anchor=east},
        legend pos=outer north east,
    }
]
\addplot coordinates {(0,0) (1,1)};
\addplot coordinates {(0,1) (1,2)};
\addplot coordinates {(0,2) (1,3)};
\legend{$l_1$, legend $2$,$l_3$}
\end{axis}
\end{tikzpicture}
```



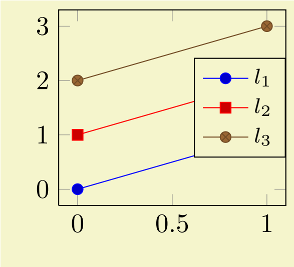
```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
%  similar placement as previous example:
\pgfplotsset{every axis legend/.append style={
        at={(1.02,1)},
        anchor=north west}}
\begin{tikzpicture}
\begin{axis}
\addplot coordinates {(0,0) (1,1)};
\addplot coordinates {(0,1) (1,2)};
\addplot coordinates {(0,2) (1,3)};
\legend{$l_1$,$l_2$,$l_3$}
\end{axis}
\end{tikzpicture}
```

Use `legend columns={⟨number⟩}` to configure the number of horizontal legend entries.



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\pgfplotsset{every axis legend/.append style={
        at={(0.5,1.03)},
        anchor=south}}
\begin{axis}[legend columns=4]
\addplot coordinates {(0,0) (1,1)};
\addplot coordinates {(0,1) (1,2)};
\addplot coordinates {(0,2) (1,3)};
\legend{$l_1$,$l_2$,$l_3$}
\end{axis}
\end{tikzpicture}
```

Instead of the `/.append style`, it is possible to use `legend style` as in the following example. It has the same effect.



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}[
    legend style={
        at={(1,0.5)},
        anchor=east}]
\addplot coordinates {(0,0) (1,1)};
\addplot coordinates {(0,1) (1,2)};
\addplot coordinates {(0,2) (1,3)};
\legend{$l_1$,$l_2$,$l_3$}
\end{axis}
\end{tikzpicture}
```

The default `every axis legend` style is

```
\pgfplotsset{every axis legend/.style={
    cells={anchor=center},%  Centered entries
    inner xsep=3pt,inner ysep=2pt,nodes={inner sep=2pt,text depth=0.15em},
    anchor=north east,
    shape=rectangle,
    fill=white,
    draw=black,
    at={(0.98,0.98)}
    }
}
```

Whenever possible, consider using `/.append style` to keep the default styles active. This ensures compatibility with future versions.

```
\pgfplotsset{every axis legend/.append style={...}}
```

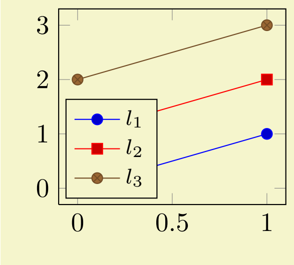
/pgfplots/**legend style**={⟨*key-value-list*⟩}

An abbreviation for `every axis legend/.append style={⟨key-value-list⟩}`.

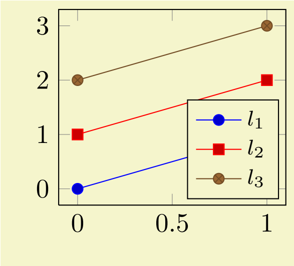
/pgfplots/**legend pos**=south west|south east|north west|north east|outer north east

A style which provides shorthand access to some commonly used legend positions.

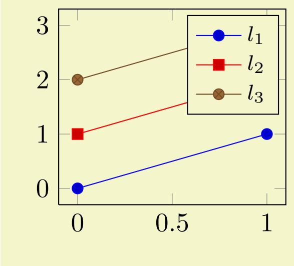
Each of these styles appends `at={(⟨x⟩,⟨y⟩)},anchor=⟨name⟩` values to `every axis legend`.



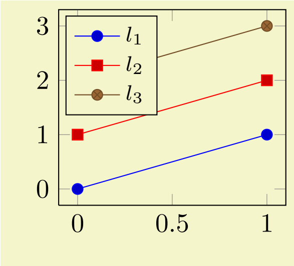
```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}[legend pos=south west]
\addplot coordinates {(0,0) (1,1)};
\addplot coordinates {(0,1) (1,2)};
\addplot coordinates {(0,2) (1,3)};
\legend{$l_1$,$l_2$,$l_3$}
\end{axis}
\end{tikzpicture}
```



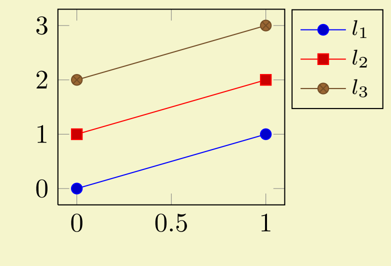
```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}[legend pos=south east]
\addplot coordinates {(0,0) (1,1)};
\addplot coordinates {(0,1) (1,2)};
\addplot coordinates {(0,2) (1,3)};
\legend{$l_1$,$l_2$,$l_3$}
\end{axis}
\end{tikzpicture}
```



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}[legend pos=north east]
\addplot coordinates {(0,0) (1,1)};
\addplot coordinates {(0,1) (1,2)};
\addplot coordinates {(0,2) (1,3)};
\legend{$l_1$,$l_2$,$l_3$}
\end{axis}
\end{tikzpicture}
```



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}[legend pos=north west]
\addplot coordinates {(0,0) (1,1)};
\addplot coordinates {(0,1) (1,2)};
\addplot coordinates {(0,2) (1,3)};
\legend{$l_1$,$l_2$,$l_3$}
\end{axis}
\end{tikzpicture}
```



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}[legend pos=outer north east]
\addplot coordinates {(0,0) (1,1)};
\addplot coordinates {(0,1) (1,2)};
\addplot coordinates {(0,2) (1,3)};
\legend{$l_1$,$l_2$,$l_3$}
\end{axis}
\end{tikzpicture}
```

/pgfplots/**legend columns**={⟨*number*⟩} (default `1`)

Allows to configure the maximum number of adjacent legend entries. The default value `1` places legend entries vertically below each other.

Use `legend columns=-1` to draw all entries horizontally.

/pgfplots/**legend plot pos**=`left|right|none` (initially `left`)

Configures where the small line specifications will be drawn: left of the description, right of the description or not at all.

/pgfplots/every legend image post (style, no value)

A style which can be used to provide drawing options to every small legend image. These options apply after current plot style has been set, allowing users different line styles for legends than for plots.

/pgfplots/legend image code/.code={⟨...⟩}

Allows to replace the default images which are drawn inside of legends. When this key is evaluated, the current plot specification has already been activated (using \begin{scope}[current plot style])[25], so any drawing operations use the same styles as the \addplot command.

The default is the style line legend.

/pgfplots/line legend (style, no value)

A style which sets legend image code (back) to its initial value.

Its initial value is

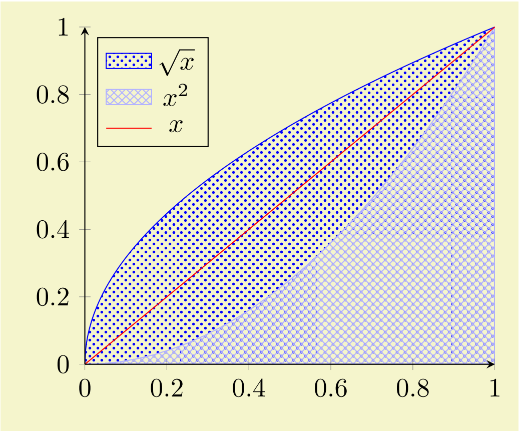
```
\pgfplotsset{
    /pgfplots/line legend/.style={
        legend image code/.code={
            \draw[mark repeat=2,mark phase=2,##1]
                plot coordinates {
                    (0cm,0cm)
                    (0.3cm,0cm)
                    (0.6cm,0cm)
                };%
        }
    }
}
```

The style line legend can also be used to use a different legend style for one particular plot (see the docs for area legend for an example).

/pgfplots/area legend (style, no value)

A style which sets legend image code to

```
\pgfplotsset{
    legend image code/.code={%
        \draw[#1] (0cm,-0.1cm) rectangle (0.6cm,0.1cm);
    }
}
```



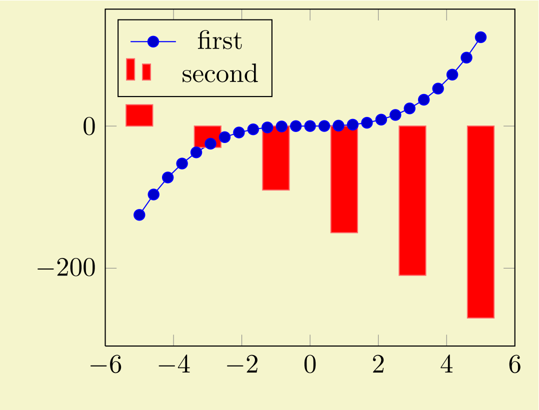
```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
%  \usetikzlibrary{patterns}
\begin{tikzpicture}
\begin{axis}[area legend,
    axis x line=bottom,
    axis y line=left,
    domain=0:1,
    legend style={at={(0.03,0.97)},
        anchor=north west},
    axis on top,xmin=0]
\addplot[pattern=crosshatch dots,
    pattern color=blue,draw=blue,
    samples=500]
    {sqrt(x)}    \closedcycle;

\addplot[pattern=crosshatch,
    pattern color=blue!30!white,
    draw=blue!30!white]
    {x^2} \closedcycle;

\addplot[red,line legend] coordinates {(0,0) (1,1)};
\legend{$\sqrt x$,$x^2$,$x$}
\end{axis}
\end{tikzpicture}
```

/pgfplots/xbar legend (no value)

[25] This was different in versions before 1.3. The new scope features allows plot styles to change legendimagecode.

/pgfplots/ybar legend (no value)
/pgfplots/zbar legend (no value)
/pgfplots/xbar interval legend (no value)
/pgfplots/ybar interval legend (no value)
/pgfplots/zbar interval legend (no value)

These style keys redefine `legend image code` such that legends use `xbar`, `ybar` or the `xbar interval` and `ybar interval` handlers.



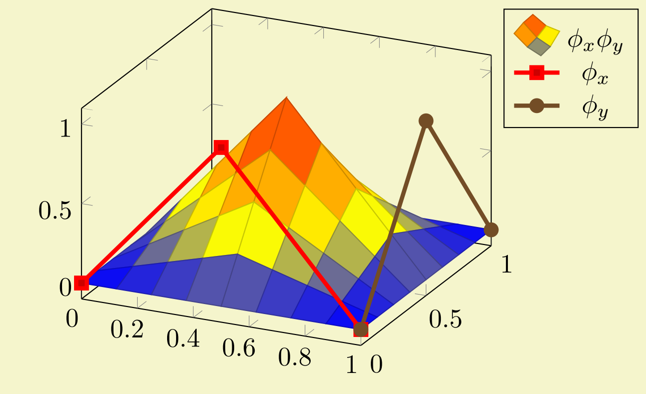
```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{axis}[legend pos=north west]
    \addplot {x^3};
    \addplot[ybar,fill=red,draw=red!60,
        ybar legend,mark=none,samples=5]
        {-30*(x +4)};
    \legend{first,second}
    \end{axis}
\end{tikzpicture}
```

The initial values for these styles might be interesting if someone wants to modify them. Here are they:

```
\pgfplotsset{
    /pgfplots/xbar legend/.style={
        /pgfplots/legend image code/.code={%
            \draw[##1,/tikz/.cd,bar width=3pt,yshift=-0.2em,bar shift=0pt]
            plot coordinates {(0cm,0.8em) (2*\pgfplotbarwidth,0.6em)};},
    },
    /pgfplots/ybar legend/.style={
        /pgfplots/legend image code/.code={%
            \draw[##1,/tikz/.cd,bar width=3pt,yshift=-0.2em,bar shift=0pt]
            plot coordinates {(0cm,0.8em) (2*\pgfplotbarwidth,0.6em)};},
    },
    /pgfplots/xbar interval legend/.style={%
        /pgfplots/legend image code/.code={%
            \draw[##1,/tikz/.cd,yshift=-0.2em,bar interval width=0.7,bar interval shift=0.5]
            plot coordinates {(0cm,0.8em) (5pt,0.6em) (10pt,0.6em)};},
    },
    /pgfplots/ybar interval legend/.style={
        /pgfplots/legend image code/.code={%
            \draw[##1,/tikz/.cd,yshift=-0.2em,bar interval width=0.7,bar interval shift=0.5]
            plot coordinates {(0cm,0.8em) (5pt,0.6em) (10pt,0.6em)};},
    },
}
```

/pgfplots/mesh legend (no value)

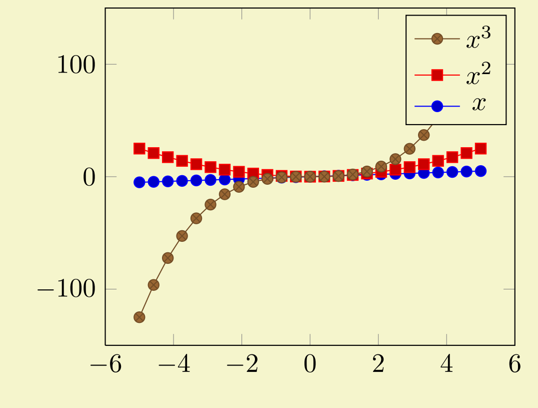
Redefines `legend image code` such that it is compatible with `mesh` and `surf` plot handlers (for three dimensional visualization mainly).



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{axis}[legend pos=outer north east]
    \addplot3[surf,samples=9,domain=0:1]
        {(1-abs(2*(x-0.5))) * (1-abs(2*(y-0.5)))};
    \addlegendentry{$\phi_x \phi_y$}

    \addplot3+[ultra thick] coordinates {(0,0,0) (0.5,0,1) (1,0,0)};
    \addlegendentry{$\phi_x $}

    \addplot3+[ultra thick] coordinates {(1,0,0) (1,0.5,1) (1,1,0)};
    \addlegendentry{$\phi_y $}
    \end{axis}
\end{tikzpicture}
```

/pgfplots/reverse legend=true|false (initially false)

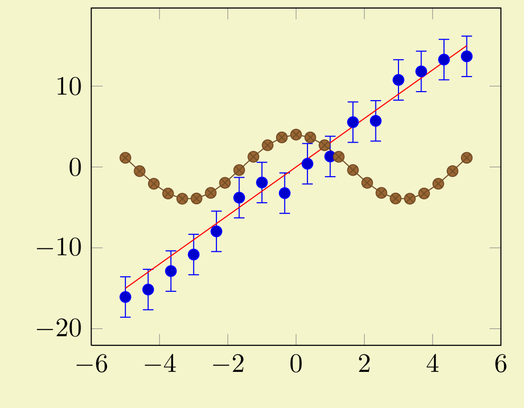
Allows to reverse the order in which the pairs (legend entry, plot style) are drawn.



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{axis}[reverse legend]
    \addplot {x};
    \addlegendentry{$x$}
    \addplot {x^2};
    \addlegendentry{$x^2$}
    \addplot {x^3};
    \addlegendentry{$x^3$}
    \end{axis}
\end{tikzpicture}
```

4.8.6 Legends with \label and \ref

PGFPLOTS offers a \label and \ref feature for LaTeX to assemble a legend manually, for example as part of the figure caption. These references work as usual LaTeX references: a \label remembers where and what needs to be referenced and a \ref expands to proper text. In context of plots, a \label remembers the plot specification of one plot and a \ref expands to the small image which would also be used inside of legends.



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}[baseline]
\begin{axis}
    \addplot+[only marks,
            samples=15,
            error bars/y dir=both,
            error bars/y fixed=2.5]
        {3*x+2.5*rand};
    \label{pgfplots:label1}

    \addplot+[mark=none] {3*x};
    \label{pgfplots:label2}

    \addplot {4*cos(deg(x))};
    \label{pgfplots:label3}
\end{axis}
\end{tikzpicture}
```

```
The picture shows the estimations \ref{pgfplots:label1} which are subjected to noise.
It appears the model \ref{pgfplots:label2} fits the data appropriately.
Finally, \ref{pgfplots:label3} is only here to get three examples.
```

The picture shows the estimations ● which are subjected to noise. It appears the model —— fits the data appropriately. Finally, —●— is only here to get three examples.

\label{⟨*label name*⟩}

\label[⟨*reference*⟩]{⟨*label name*⟩}

When used after \addplot, this command creates a LaTeX label named {⟨*label name*⟩}[26]. If this label is cross-referenced with \ref{⟨*label name*⟩} somewhere, the associated plot specification will be inserted.

Label3 = —●—;Label2 = ——

```
Label3 = \ref{pgfplots:label3};
Label2 = \ref{pgfplots:label2}
```

The label is assembled using legend image code and the plot style of the last plot. Any PGFPLOTS option is expanded until only Ti*k*Z (or PGF) options remain; these options are used to get an independent label[27].

More precisely, the small image generated by \ref{⟨*label name*⟩} is

```
\tikz[/pgfplots/every crossref picture] {...}
```

where the contents is determined by legend image code and the plot style.

The second syntax, \label[⟨*reference*⟩]{⟨*label name*⟩} allows to label particular pieces of an \addplot command. It is (currently) only interesting for scatter/classes: there, it allows to reference particular classes of the scatter plot. See page 57 for more details.

\ref{⟨*label name*⟩}

Can be used to reference a labeled, single plot. See the example above.

This will also work together with hyperref links and \pageref[28].

/pgfplots/refstyle={⟨*label name*⟩}

Can be used to set the *styles* of a labeled, single plot. This allows to write

```
\addplot[/pgfplots/refstyle={pgfplots:label2}]
```

somewhere. Please note that it may be easier to define a style with .style.

/pgfplots/every crossref picture (style, no value)

A style which will be used by the cross-referencing feature for plots. The default is

```
\pgfplotsset{every crossref picture/.style={baseline,yshift=0.3em}}
```

4.8.7 Axis Lines

An extension by Pascal Wolkotte

By default the axis lines are drawn as a box, but it is possible to change the appearance of the x and y axis lines.

/pgfplots/axis x line=box|top|middle|center|bottom|none (initially box)
/pgfplots/axis x line*=box|top|middle|center|bottom|none (initially box)
/pgfplots/axis y line=box|left|middle|center|right|none (initially box)
/pgfplots/axis y line*=box|left|middle|center|right|none (initially box)
/pgfplots/axis lines=box|left|middle|center|right|none
/pgfplots/axis lines*=box|left|middle|center|right|none

These keys allow to choose the locations of the axis lines. The last one, axis lines sets the same value for every axis.

Ticks and tick labels are placed according to the chosen value as well. The choice bottom will draw the x line at $y = y_{\min}$, middle will draw the x line at $y = 0$, and top will draw it at $y = y_{\max}$. Finally, box is a combination of options top and bottom. The y variant works similarly.

The case center is a synonym for middle, both draw the line through the respective coordinate 0. If this coordinate is not part of the axis limit, the lower axis limit is chosen instead.

[26] This feature is *only* available in LaTeX, sorry.

[27] Please note that you can't use the label/ref mechanism in conjunction with image externalization as this will (naturally) lead to undefined references.

[28] Older versions of PGFPLOTS required the use of \protect\ref when used inside of captions or section headings. This is no longer necessary.

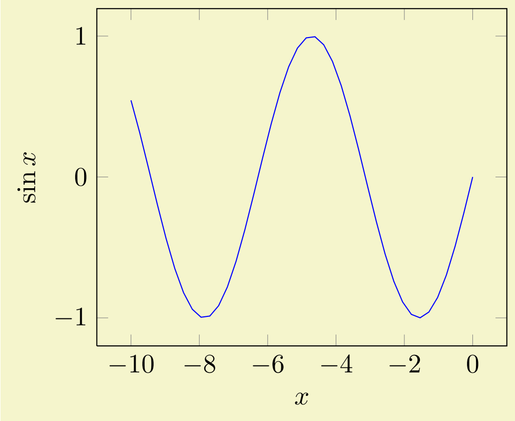
The starred versions ...line* *only* affect the axis lines, without correcting the positions of axis labels, tick lines or other keys which are (possibly) affected by a changed axis line. The non-starred versions are actually styles which set the starred key *and* some other keys which also affect the figure layout:

- In case axis x line=box, the style every boxed x axis will be installed immediately.
- In case axis x line≠box, the style every non boxed x axis will be installed immediately. Furthermore, axis labels positions will be adjusted to fit the choosen value.

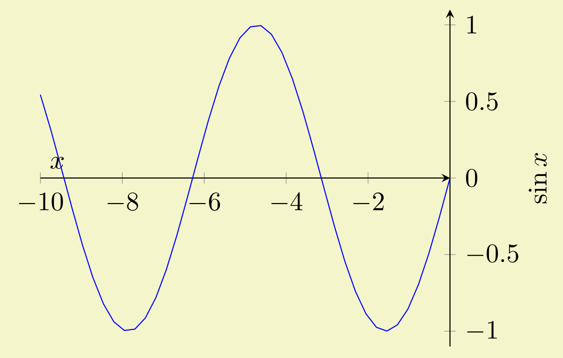
The same holds true for the y-variants. The default styles are defined as

```
\pgfplotsset{
    every non boxed x axis/.style={
        xtick align=center,
        enlarge x limits=false,
        x axis line style={-stealth}
    },
    every boxed x axis/.style={}
}
```

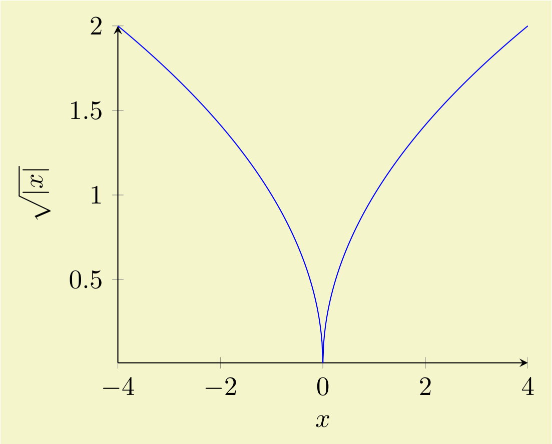
Feel free to overwrite these styles if the default doesn't fit your needs or taste. Again, these styles will *not* be used for axis line*.



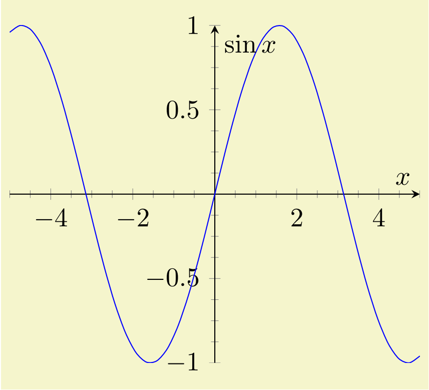
```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}[
    xlabel=$x$,ylabel=$\sin x$]

    \addplot[blue,mark=none,
         domain=-10:0,samples=40]
        {sin(deg(x))};
\end{axis}
\end{tikzpicture}
```



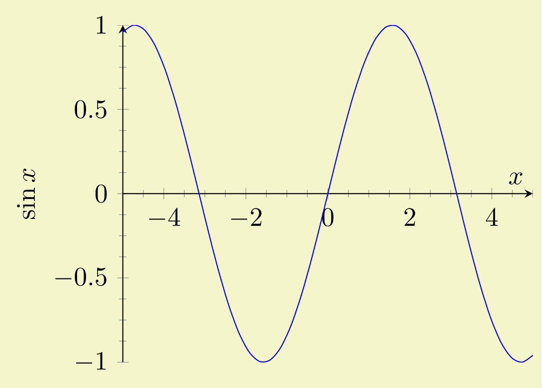
```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}[
    axis x line=middle,
    axis y line=right,
    ymax=1.1, ymin=-1.1,
    xlabel=$x$,ylabel=$\sin x$
]
    \addplot[blue,mark=none,
         domain=-10:0,samples=40]
        {sin(deg(x))};
\end{axis}
\end{tikzpicture}
```



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}[
    axis x line=bottom,
    axis y line=left,
    xlabel=$x$,ylabel=$\sqrt{|x|}$
]
\addplot[blue,mark=none,
     domain=-4:4,samples=501]
    {sqrt(abs(x))};
\end{axis}
\end{tikzpicture}
```



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}[
    minor tick num=3,
    axis y line=center,
    axis x line=middle,
    xlabel=$x$,ylabel=$\sin x$
    ]
    \addplot[smooth,blue,mark=none,
        domain=-5:5,samples=40]
       {sin(deg(x))};
\end{axis}
\end{tikzpicture}
```



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}[
    minor tick num=3,
    axis y line=left,
    axis x line=middle,
    xlabel=$x$,ylabel=$\sin x$
    ]
    \addplot[smooth,blue,mark=none,
        domain=-5:5,samples=40]
       {sin(deg(x))};
\end{axis}
\end{tikzpicture}
```

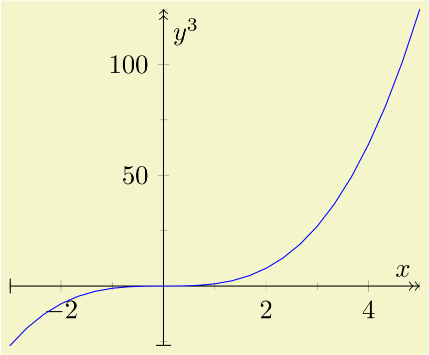
In case `middle`, the style `every inner axis x line` allows to adjust the appearance.

Note that three dimensional axes only support to use the same value for every axis, i.e. three dimensional axes support only the `axis lines` key (or, preferably for 3D axes, the `axis lines*` key – check what looks best). See section 4.10.4 for examples of three dimensional axis line variations.

/pgfplots/`every inner x axis line` (no value)
/pgfplots/`every inner y axis line` (no value)
/pgfplots/`every inner z axis line` (no value)

A style key which can be redefined to customize the appearance of *inner* axis lines. Inner axis lines are those drawn by the `middle` (or `center`) choice of `axis x line`, see above.

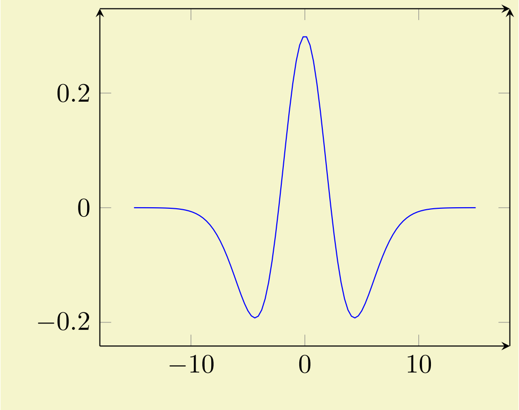
This style affects *only* the line as such.



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}[
    minor tick num=1,
    axis x line=middle,
    axis y line=middle,
    every inner x axis line/.append style=
        {|->>},
    every inner y axis line/.append style=
        {|->>},
    xlabel=$x$,ylabel=$y^3$
]
\addplot[blue,domain=-3:5] {x^3};
\end{axis}
\end{tikzpicture}
```

/pgfplots/every outer x axis line (no value)
/pgfplots/every outer y axis line (no value)
/pgfplots/every outer z axis line (no value)

Similar to every inner x axis line, this style configures the appearance of all axis lines which are part of the outer box.



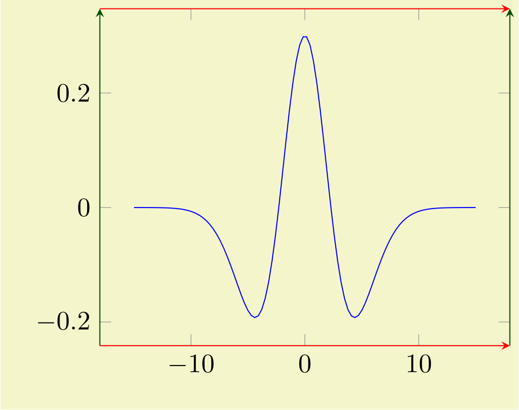
```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}[
    separate axis lines, %  important !
    every outer x axis line/.append style=
        {-stealth},
    every outer y axis line/.append style=
        {-stealth},
]
\addplot[blue,id=DoG,
        samples=100,
        domain=-15:15]
  gnuplot{1.3*exp(-x**2/10) - exp(-x**2/20)};
\end{axis}
\end{tikzpicture}
```

/pgfplots/separate axis lines={⟨*true,false*⟩} (default true)

Enables or disables separate path commands for every axis line. This option affects *only* the case if axis lines are drawn as a *box*.

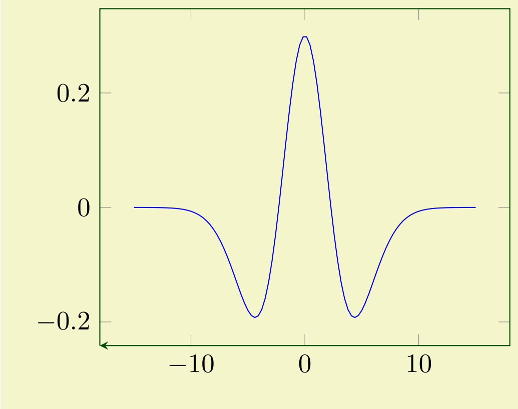
Both cases have their advantages and disadvantages, I fear there is no reasonable default (suggestions are welcome).

The case separate axis lines=true allows to draw arrow heads on each single axis line, but it can't close edges very well – in case of thick lines, unsatisfactory edges occur.



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}[
    separate axis lines,
    every outer x axis line/.append style=
        {-stealth,red},
    every outer y axis line/.append style=
        {-stealth,green!30!black},
]
\addplot[blue,
        samples=100,
        domain=-15:15]
    {1.3*exp(0-x^2/10) - exp(0-x^2/20)};
  %  Unfortunately, there is a bug in PGF 2.00
  %  something like exp(-10^2)
  %  must be written as exp(0-10^2) :-(
\end{axis}
\end{tikzpicture}
```

The case separate axis lines=false issues just *one* path for all axis lines. It draws a kind of rectangle, where some parts of the rectangle may be skipped over if they are not wanted. The advantage is that edges are closed properly. The disadvantage is that at most one arrow head is added to the path (and yes, only one drawing color is possible).



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}[
    separate axis lines=false,
    every outer x axis line/.append style=
        {-stealth,red},
    every outer y axis line/.append style=
        {-stealth,green!30!black},
]
\addplot[blue,id=DoG,
        samples=100,
        domain=-15:15]
  gnuplot{1.3*exp(-x**2/10) - exp(-x**2/20)};
\end{axis}
\end{tikzpicture}
```

/pgfplots/axis line style={⟨*key-value-list*⟩}

A command which appends {⟨*key-value-list*⟩} to *all* axis line appearance styles.

/pgfplots/inner axis line style={⟨*key-value-list*⟩}

A command which appends {⟨*key-value-list*⟩} to both, every inner x axis line and the y variant.

/pgfplots/outer axis line style={⟨*key-value-list*⟩}

A command which appends {⟨*key-value-list*⟩} to both, every outer x axis line and the y variant.

/pgfplots/x axis line style={⟨*key-value-list*⟩}
/pgfplots/y axis line style={⟨*key-value-list*⟩}
/pgfplots/z axis line style={⟨*key-value-list*⟩}

A command which appends {⟨*key-value-list*⟩} to all axis lines styles for either x or y axis.

/pgfplots/every boxed x axis (no value)
/pgfplots/every boxed y axis (no value)
/pgfplots/every boxed z axis (no value)

A style which will be installed as soon as axis x line=box (y) is set.

The default is simply empty.

/pgfplots/every non boxed x axis (no value)
/pgfplots/every non boxed y axis (no value)
/pgfplots/every non boxed z axis (no value)

A style which will be installed as soon as axis x line (y) will be set to something different than box.

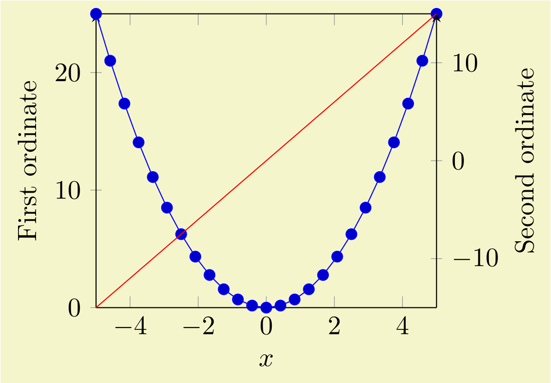
The default is

```
\pgfplotsset{
    every non boxed x axis/.style={
        xtick align=center,
        enlarge x limits=false,
        x axis line style={-stealth}}}
```

with similar values for the y-variant. Feel free to redefine this style to your needs / taste.

4.8.8 Two Ordinates (y axis)

In some applications, more than one y axis is used if the x range is the same. This section demonstrates how to create them.

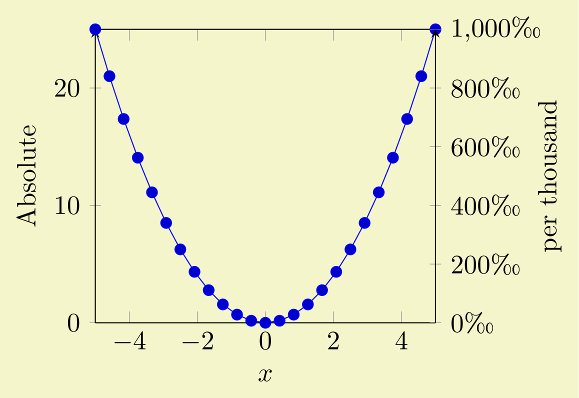


```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
  \begin{axis}[
    scale only axis,
    xmin=-5,xmax=5,
    axis y line=left,
    xlabel=$x$,
    ylabel=First ordinate]
  \addplot {x^2};
  \end{axis}

  \begin{axis}[
    scale only axis,
    xmin=-5,xmax=5,
    axis y line=right,
    axis x line=none,
    ylabel=Second ordinate]
  \addplot[red] {3*x};
  \end{axis}
\end{tikzpicture}
```

The basic idea is to draw two axis “on top” of each other – one, which contains the x axis and the left y axis, and one which has *only* the right y axis. Since PGFPLOTS does not really know what it’s doing here, user attention in the following possibly non-obvious aspects is required:

1. Scaling. You should set `scale only axis` because this forces equal dimensions for both axis, without respecting any labels.
2. Same x limits. You should set those limits explicitly.

You may want to consider different legend styles. It is also possible to use only the axis, without any plots:



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
%  \usepackage{textcomp}
\begin{tikzpicture}
  \begin{axis}[
    scale only axis,
    xmin=-5,xmax=5,
    axis y line=left,
    xlabel=$x$,
    ylabel=Absolute]
  \addplot {x^2};
  \end{axis}

  \begin{axis}[
    scale only axis,
    xmin=-5,xmax=5,
    ymin=0,ymax=1000,
    yticklabel=
{$\pgfmathprintnumber{\tick}$\textperthousand},
    axis y line=right,
    axis x line=none,
    y label style={yshift=-10pt},
    ylabel=per thousand]
  \end{axis}
\end{tikzpicture}
```

4.8.9 Axis Discontinuities

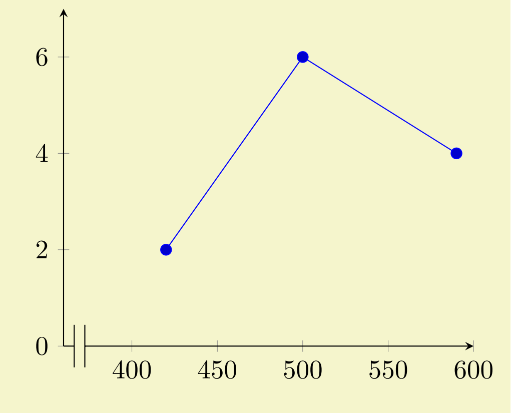
An extension by Pascal Wolkotte

In case the range of either of the axis do not include the zero value, it is possible to visualize this with a discontinuity decoration on the corresponding axis line.

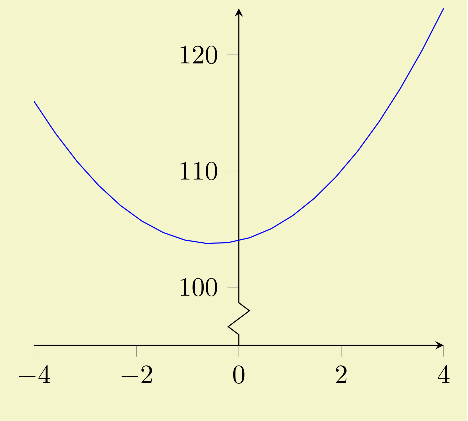
/pgfplots/axis x discontinuity=crunch|parallel|none (initially none)
/pgfplots/axis y discontinuity=crunch|parallel|none (initially none)
/pgfplots/axis z discontinuity=crunch|parallel|none (initially none)

Insert a discontinuity decoration on the x (or y, respectively) axis. This is to visualize that the y axis does cross the x axis at its 0 value, because the minimum x axis value is positive or the maximum value is negative.

The description applies `axis y discontinuity` as well, with interchanged meanings of x and y.



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}[
    axis x line=bottom,
    axis x discontinuity=parallel,
    axis y line=left,
    xmin=360, xmax=600,
    ymin=0, ymax=7,
     enlargelimits=false
]
    \addplot coordinates {
        (420,2)
        (500,6)
        (590,4)
    };
\end{axis}
\end{tikzpicture}
```

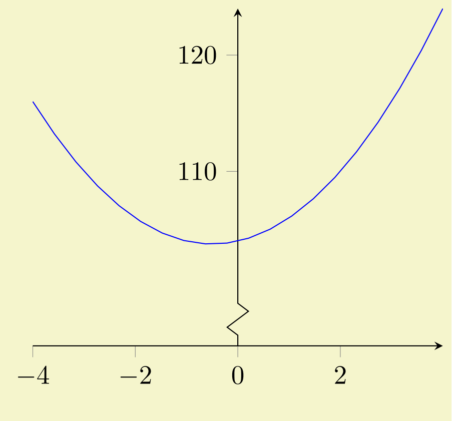


```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}[
    axis x line=bottom,
    axis y line=center,
    tick align=outside,
    axis y discontinuity=crunch,
    ymin=95, enlargelimits=false
]
    \addplot[blue,mark=none,
         domain=-4:4,samples=20]
        {x*x+x+104};
\end{axis}
\end{tikzpicture}
```

A problem might occur with the placement of the ticks on the axis. This can be solved by specifying the minimum or maximum axis value for which a tick will be placed.

/pgfplots/**xtickmin**={⟨*coord*⟩} (default `axis limits`)
/pgfplots/**ytickmin**={⟨*coord*⟩} (default `axis limits`)
/pgfplots/**ztickmin**={⟨*coord*⟩} (default `axis limits`)
/pgfplots/**xtickmax**={⟨*coord*⟩} (default `axis limits`)
/pgfplots/**ytickmax**={⟨*coord*⟩} (default `axis limits`)
/pgfplots/**ztickmax**={⟨*coord*⟩} (default `axis limits`)

The options `xtickmin`, `xtickmax` and `ytickmin`, `ytickmax` allow to define the axis tick limits, i.e. the axis values before respectively after no ticks will be placed. Everything outside of the axis tick limits will be not drawn. Their default values are equal to the axis limits.

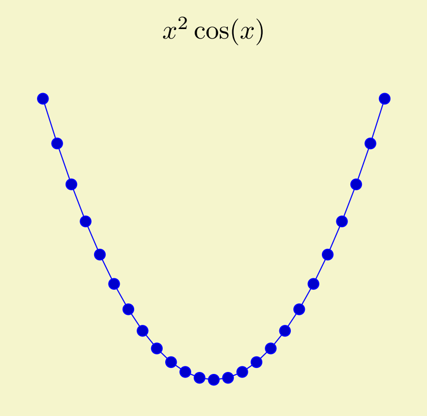


```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}[
    axis x line=bottom,
    axis y line=center,
    tick align=outside,
    axis y discontinuity=crunch,
    xtickmax=3,
    ytickmin=110,
    ymin=95, enlargelimits=false
]
    \addplot[blue,mark=none,
         domain=-4:4,samples=20]
        {x*x+x+104};
\end{axis}
\end{tikzpicture}
```

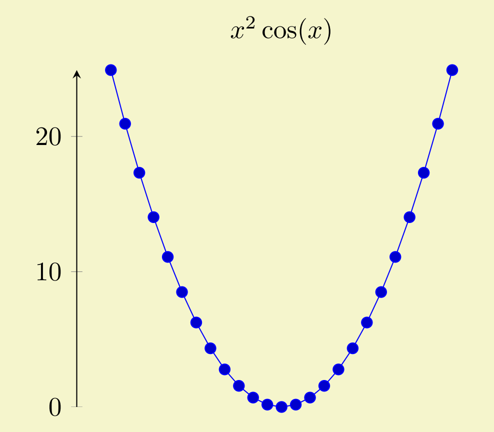
/pgfplots/hide x axis=true|false (initially false)
/pgfplots/hide y axis=true|false (initially false)
/pgfplots/hide z axis=true|false (initially false)
/pgfplots/hide axis=true|false (initially false)

Allows to hide either a selected axis or all of them. No outer rectangle, no tick marks and no labels will be drawn. Only titles and legends will be processed as usual.

Axis scaling and clipping will be done as if you did not use hide axis.



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{axis}[
        hide x axis,
        hide y axis,
        title={$x^2\cos(x)$}]
    \addplot {cos(x)*x^2};
    \end{axis}
\end{tikzpicture}
```



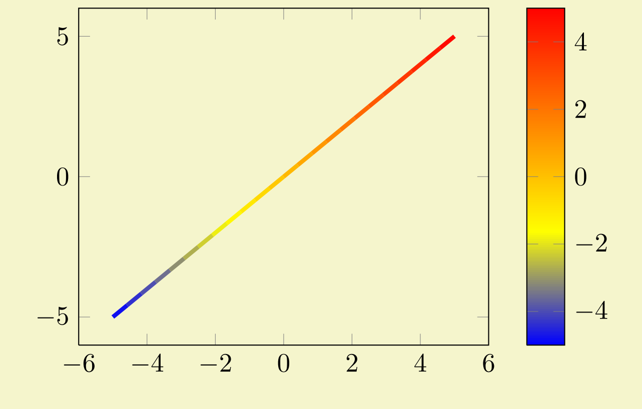
```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{axis}[
        hide x axis,
        axis y line=left,
        title={$x^2\cos(x)$}]
    \addplot {cos(x)*x^2};
    \end{axis}
\end{tikzpicture}
```

4.8.10 Color Bars

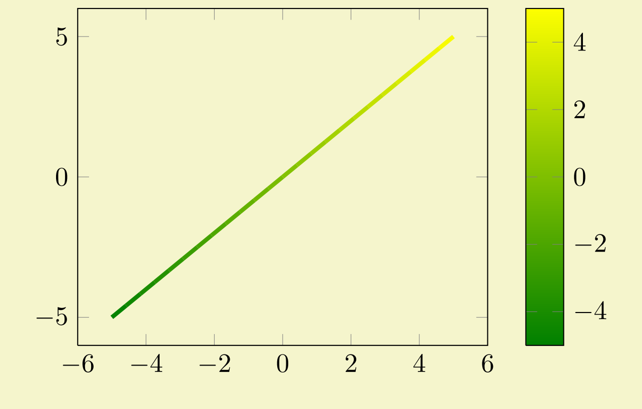
PGFPLOTS supports mesh, surface and scatter plots which can use color maps. While color maps can be chosen as described in section 4.6.5, they can be visualized using color bars.

/pgfplots/colorbar=true|false (initially false)

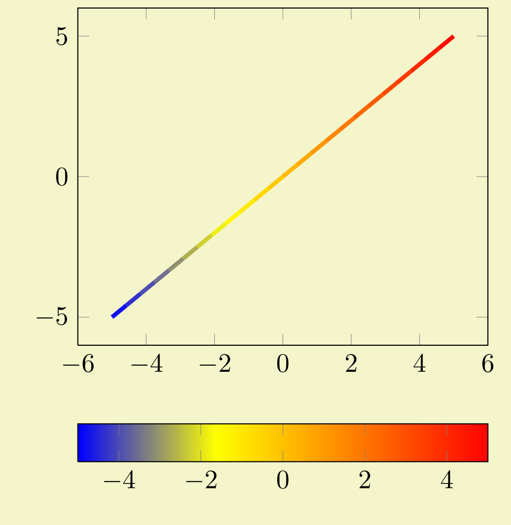
Activates or deactivates color bars.



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{axis}[colorbar]
        \addplot[mesh,ultra thick] {x};
    \end{axis}
\end{tikzpicture}
```



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{axis}[colorbar,colormap/greenyellow]
        \addplot[mesh,ultra thick] {x};
    \end{axis}
\end{tikzpicture}
```



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{axis}[colorbar horizontal]
        \addplot[mesh,ultra thick] {x};
    \end{axis}
\end{tikzpicture}
```

A color bar is only useful for plots which actually use color data – that is, the special data which is called “point meta” in PGFPLOTS (see section 4.7). Activating color bars for a plot without color map results in an empty color bar.

Color bars are just normal axes which are placed right besides their parent axes. The only difference is that they inherit several styles such as line width and fonts and they contain a bar shaded with the color map of the current axis.

Color bars are drawn internally with

```
\axis[every colorbar,colorbar shift,colorbar=false]
    \addplot graphics {};
\endaxis
```

where the placement, alignment, appearance and other options are done by the two styles `every colorbar` and `colorbar shift`. These styles and the possible placement and alignment options are described below.

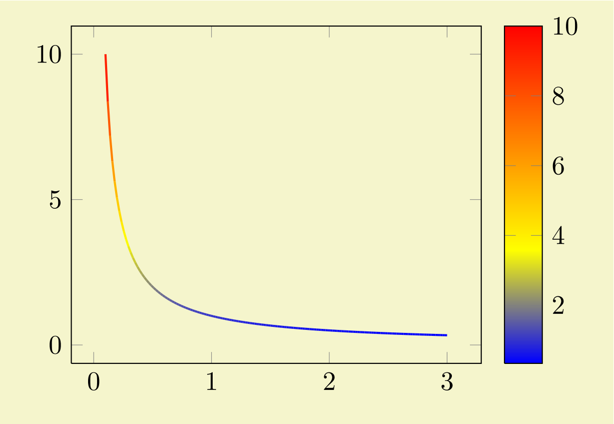
Remarks for special cases:

- Since there is always only one color bar per plot, this color bar uses the axis wide configurations of color map and color data.
- If someone needs more than one color bar, the draw command above needs to be updated. See the key `colorbar/draw/.code` for this special case.

`/pgfplots/colorbar right` (style, no value)

A style which re-defines `every colorbar` and `colorbar shift` such that color bars are placed right of their parent axis.

This is the initial configuration.



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{axis}[colorbar right]
    \addplot[mesh,thick,samples=150,domain=0.1:3]
        {1/x};
    \end{axis}
\end{tikzpicture}
```

The style `colorbar right` is defined to be

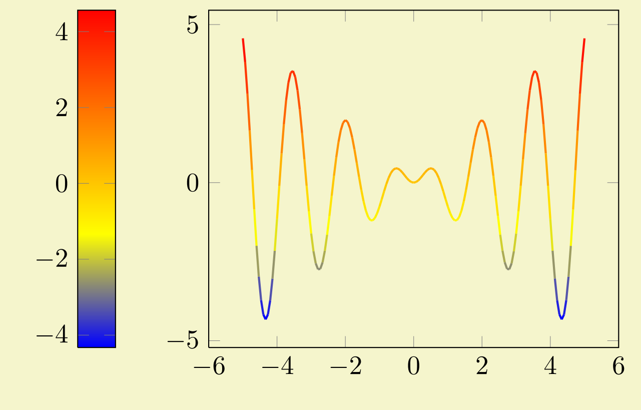
```
\pgfplotsset{
    colorbar right/.style={
        /pgfplots/colorbar=true,
        /pgfplots/colorbar shift/.style={xshift=0.3cm},
        /pgfplots/every colorbar/.style={
            title=,
            xlabel=,
            ylabel=,
            zlabel=,
            legend entries=,
            axis on top,
            at={(parent axis.right of north east)},
            anchor=north west,
            xmin=0,
            xmax=1,
            ymin=\pgfkeysvalueof{/pgfplots/point meta min},
            ymax=\pgfkeysvalueof{/pgfplots/point meta max},
            plot graphics/xmin=0,
            plot graphics/xmax=1,
            plot graphics/ymin=\pgfkeysvalueof{/pgfplots/point meta min},
            plot graphics/ymax=\pgfkeysvalueof{/pgfplots/point meta max},
            enlargelimits=false,
            scale only axis,
            height=\pgfkeysvalueof{/pgfplots/parent axis height},
            x=\pgfkeysvalueof{/pgfplots/colorbar/width},
            yticklabel pos=right,
            xtick=\empty,
            colorbar vertical/lowlevel,
        }
    },
    /pgfplots/colorbar vertical/lowlevel/.style={
        plot graphics/lowlevel draw/.code 2 args={%
            \pgfuseshading{...} %  some advanced basic level shading operations
        }
    },
}
```

Attention: colorbar right *re*-defines every colorbar. That means any user customization must take place *after* colorbar right:

```
%  correct:
\begin{axis}[colorbar right, colorbar style={<some customization>}]
%  wrong, colorbar right resets the customization:
\begin{axis}[colorbar style={<some customization>}, colorbar right]
```

/pgfplots/colorbar left (style, no value)

A style which re-defines every colorbar and colorbar shift such that color bars are placed left of their parent axis.



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{axis}[colorbar left]
    \addplot[mesh,thick,samples=150]
        {x*sin(deg(4*x))};
    \end{axis}
\end{tikzpicture}
```

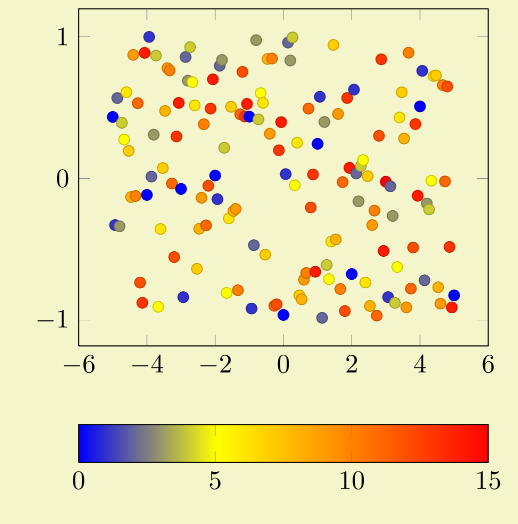
The style `colorbar left` is defined to be

```
\pgfplotsset{
    colorbar left/.style={
        /pgfplots/colorbar right,
        /pgfplots/colorbar shift/.style={xshift=-0.3cm},
        /pgfplots/every colorbar/.append style={
            at={(parent axis.left of north west)},
            anchor=north east,
            yticklabel pos=left,
        }
    }
}
```

Attention: `colorbar left` *re*-defines `every colorbar`. That means any user customization must take place *after* `colorbar left` (see also the documentation for `colorbar right`).

/pgfplots/`colorbar horizontal` (style, no value)

A style which re-defines `every colorbar` and `colorbar shift` such that color bars are placed below their parent axis, with a horizontal bar.



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{axis}[colorbar horizontal]
    \addplot[only marks,scatter,
        scatter src={mod(\coordindex,15)},samples=150]
        {rand};
    \end{axis}
\end{tikzpicture}
```

This style is defined to be

```
\pgfplotsset{
    colorbar horizontal/.style={
        /pgfplots/colorbar=true,
        /pgfplots/colorbar shift/.style={yshift=-0.3cm},
        /pgfplots/every colorbar/.style={
            title=,
            xlabel=,
            ylabel=,
            zlabel=,
            legend entries=,
            axis on top,
            at={(parent axis.below south west)},
            anchor=north west,
            ymin=0,
            ymax=1,
            xmin=\pgfkeysvalueof{/pgfplots/point meta min},
            xmax=\pgfkeysvalueof{/pgfplots/point meta max},
            plot graphics/ymin=0,
            plot graphics/ymax=1,
            plot graphics/xmin=\pgfkeysvalueof{/pgfplots/point meta min},
            plot graphics/xmax=\pgfkeysvalueof{/pgfplots/point meta max},
            enlargelimits=false,
            scale only axis,
            width=\pgfkeysvalueof{/pgfplots/parent axis width},
            y=\pgfkeysvalueof{/pgfplots/colorbar/width},
            xticklabel pos=left,
            ytick=\empty,
            colorbar horizontal/lowlevel,
        }%
    },%
    /pgfplots/colorbar horizontal/lowlevel/.style={%
        plot graphics/lowlevel draw/.code 2 args={%
            \pgfuseshading{...} %  some advanced basic level shading operations
        },%
    },%
}
```

Attention: colorbar horizontal *re*-defines every colorbar. That means any user customization must take place *after* colorbar horizontal:

```
%  correct:
\begin{axis}[colorbar horizontal, colorbar style={<some customization>}]
%  wrong, colorbar horizontal resets the customization:
\begin{axis}[colorbar style={<some customization>}, colorbar horizontal]
```

/pgfplots/every colorbar (style, no value)

This style governs the placement, alignment and appearance of color bars. Any desired detail changes for color bars can be put into this style. Additionally, there is a style colorbar shift which is set after every colorbar. The latter style is intended to contain only shift transformations like xshift or yshift (making it easier to overwrite or deactivate them).

While a color bar is drawn, the predefined node parent axis can be used to align at the parent axis.

Predefined node parent axis

A node for the parent axis of a color bar. It is only valid for color bars.

Thus,

```
\pgfplotsset{
    colorbar style={
        at={(parent axis.right of north east)},
        anchor=north west,
    },
    colorbar shift/.style={xshift=0.3cm}
}
```

places the colorbar in a way that its top-left (north west) corner is aligned right of the top right corner (right of north east) of its parent axis. Combining this with the colorbar shift is actually the same as the initial setting.

Since color bars depend on some of its parent's properties, these properties are available as values of the following keys:

/pgfplots/point meta min (no value)
/pgfplots/point meta max (no value)

The values of these keys contain the lower and upper bound of the color map, i.e. the lower and upper limit for the color bar.

The value is \pgfkeysvalueof{/pgfplots/point meta min} inside of every colorbar.

/pgfplots/parent axis width (no value)
/pgfplots/parent axis height (no value)

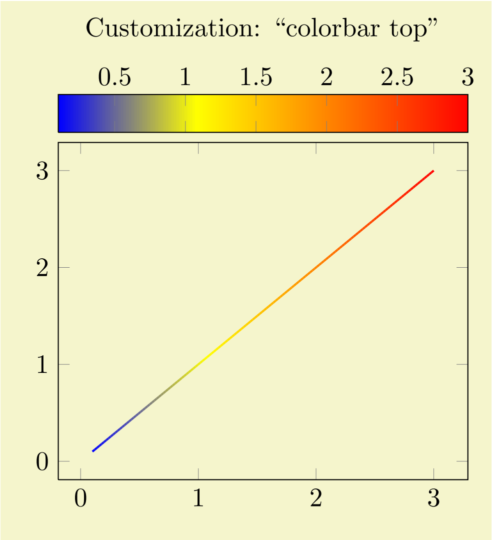
The values of these keys contain the size of the parent axis. They can be used as width and/or height arguments for every colorbar with \pgfkeysvalueof{/pgfplots/parent axis width}.

These values are only valid inside of color bars.

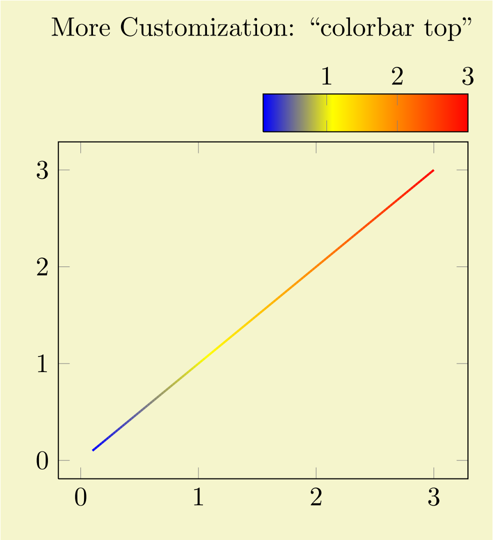
Besides these values, each color bar inherits a list of styles of its parent axis, namely

- every tick,
- every minor tick,
- every major tick,
- every axis grid,
- every minor grid,
- every major grid,
- every tick label.

This can be used to inherit line width and/or fonts.



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}[
    colorbar horizontal,
    colorbar style={
      at={(0.5,1.03)},anchor=south,
      xticklabel pos=upper
    },
    title style={yshift=1cm},
    title=Customization: ‘‘colorbar top’’]

    \addplot[mesh,thick,samples=150,domain=0.1:3]
        {x};
\end{axis}
\end{tikzpicture}
```



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}[
    colorbar horizontal,
    colorbar style={
      at={(1,1.03)},anchor=south east,
      width=0.5*
        \pgfkeysvalueof{/pgfplots/parent axis width},
      xticklabel pos=upper,
    },
    title style={yshift=1cm},
    title=More Customization: ‘‘colorbar top’’]

    \addplot[mesh,thick,samples=150,domain=0.1:3]
        {x};
\end{axis}
\end{tikzpicture}
```

Please take a look at the predefined styles colorbar right, colorbar left and colorbar horizontal for more details about configuration possibilities for every colorbar.

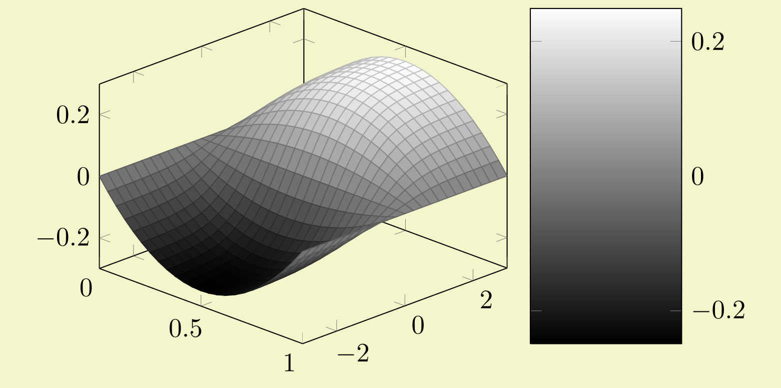
Remark: A color bar is just a normal axis. That means every colorbar can contain specifications where to place tick labels, extra ticks, scalings and most other features of a normal axis as well (except nested color bars).

/pgfplots/colorbar style={⟨*key-value list*⟩}

A shortcut for every colorbar/.append style={⟨*key-value list*⟩}. It appends options to the colorbar style.

/pgfplots/colorbar/width={⟨*dimension*⟩} (initially 0.5cm)

Sets the width of a color bar.



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{axis}[
        view/az=45,
        colorbar,
        colorbar/width=2cm,
        colormap/blackwhite]

    \addplot3[surf,domain=0:1,y domain=-3:3] {x*(1-x)*tanh(y)};
    \end{axis}
\end{tikzpicture}
```

For vertical color bars, this sets the height.

/pgfplots/colorbar shift (style, no value)

This style is installed after `every colorbar`. It is intended to contain only shift transformations like `xshift` and/or `yshift`. The reason to provide two separate styles is to allow easier deactivation of shift transformations.

```
\pgfplotsset{
    colorbar shift/.style={xshift=1cm}
}
```

Predefined node `current colorbar axis`

A predefined node for the color bar of an axis. After `\end{axis}`, this node can be used to align further graphical elements at the color bar. Note that `current axis` refers to the axis as such while `current colorbar axis` refers to the color bar (which is an axis itsself).

`/pgfplots/colorbar/draw/.code={`⟨...⟩`}`

This code key belongs to the low level interface of colorbars. It is invoked whenever a color bar needs to be drawn. Usually, it won't be necessary to use or modify this key explicitly.

In the context when this key is invoked, the styles inherited from the parent axis are already set and the required variables (see the documentation of `every colorbar`) are initialised.

This code key can be replaced if one needs more than one color bar (or other wrinkles).

The initial configuration is

```
\pgfplotsset{colorbar/draw/.code={%
    \axis[every colorbar,colorbar shift,colorbar=false]
    \addplot graphics {};
    \endaxis
    }
}
```

Please note that a color bar axis is nothing special as such – it is just a normal axis with one `plot graphics` command and it is invoked with a special set of options. The only special thing is that a set of styles and some variables are inherited from its parent axis.

4.8.11 Scaling Descriptions: Predefined Styles

It is reasonable to change font sizes, marker sizes etc. together with the overall plot size: Large plots should also have larger fonts and small plots should have small fonts and a smaller distance between ticks.

`/tikz/font=\normalfont|\small|\tiny|`...
`/pgfplots/max space between ticks={`⟨*integer*⟩`}`
`/pgfplots/try min ticks={`⟨*integer*⟩`}`
`/tikz/mark size={`⟨*integer*⟩`}`

These keys should be adjusted to the figure's dimensions. Use

```
\pgfplotsset{tick label style={font=\footnotesize},
    label style={font=\small},
    legend style={font=\small}
}
```

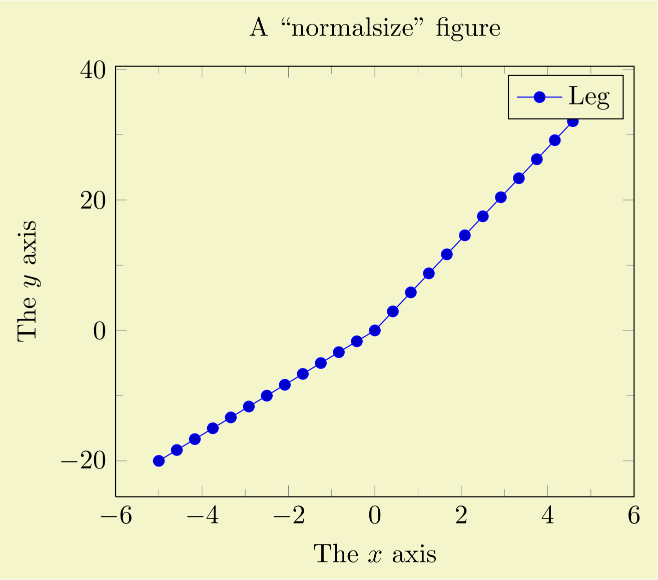
to provide different fonts for different descriptions.

The keys `max space between ticks` and `try min ticks` are described on page 172 and configure the approximate distance and number of successive tick labels (in `pt`). Please omit the `pt` suffix here.

There are a couple of predefined scaling styles which set some of these options:

`/pgfplots/normalsize` (style, no value)

Re-initialises the standard scaling options of PGFPLOTS.



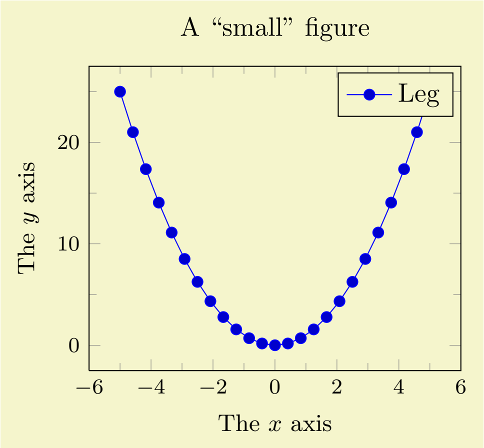
```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{axis}[normalsize,
        title=A ‘‘normalsize’’ figure,
        xlabel=The $x$ axis,
        ylabel=The $y$ axis,
        minor tick num=1,
        legend entries={Leg}]
        \addplot {max(4*x,7*x)};
    \end{axis}
\end{tikzpicture}
```

The initial setting is

```
\pgfplotsset{
    normalsize/.style={
        /pgfplots/width=240pt,
        /pgfplots/height=207pt,
        /pgfplots/max space between ticks=35
    }
}
```

/pgfplots/small (style, no value)

Redefines several keys such that the axis is “smaller”.



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{axis}[small,
        title=A ‘‘small’’ figure,
        xlabel=The $x$ axis,
        ylabel=The $y$ axis,
        minor tick num=1,
        legend entries={Leg}]
        \addplot {x^2};
    \end{axis}
\end{tikzpicture}
```

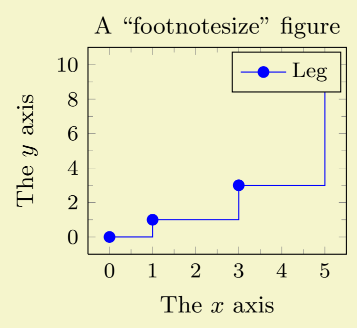
The initial setting is

```
\pgfplotsset{
    small/.style={
        width=6.5cm,
        height=,
        tick label style={font=\footnotesize},
        label style={font=\small},
        max space between ticks=25,
    }
}
```

Feel free to redefine the scaling – the option may still be useful to get more ticks without typing too much. You could, for example, set `small,width=6cm`.

/pgfplots/footnotesize (style, no value)

Redefines several keys such that the axis is even smaller. The tick labels will have `\footnotesize`.



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{axis}[footnotesize,
        title=A ‘‘footnotesize’’ figure,
        xlabel=The $x$ axis,
        ylabel=The $y$ axis,
        minor tick num=1,
        legend entries={Leg}]
        \addplot+[const plot]
            coordinates {
            (0,0) (1,1) (3,3) (5,10)
        };
    \end{axis}
\end{tikzpicture}
```

The initial setting is

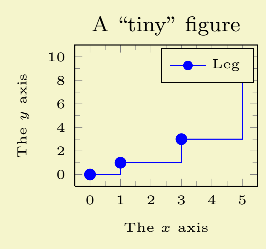
```
\pgfplotsset{
    footnotesize/.style={
        width=5cm,
        height=,
        legend style={font=\footnotesize},
        tick label style={font=\footnotesize},
        label style={font=\small},
        title style={font=\small},
        every axis title shift=0pt,
        max space between ticks=15,
        every mark/.append style={mark size=8},
        major tick length=0.1cm,
        minor tick length=0.066cm,
    },
}
```

As for `small`, it can be convenient to set `footnotesize` and set `width` afterwards.

You will need `compat=1.3` or newer for this to work.

/pgfplots/tiny (style, no value)

Redefines several keys such that the axis is very small. Most descriptions will have `\tiny` as fontsize.



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{axis}[tiny,
        title=A ‘‘tiny’’ figure,
        xlabel=The $x$ axis,
        ylabel=The $y$ axis,
        minor tick num=1,
        legend entries={Leg}]
        \addplot+[const plot]
            coordinates {
            (0,0) (1,1) (3,3) (5,10)
        };
    \end{axis}
\end{tikzpicture}
```

The initial setting is

```
\pgfplotsset{
    tiny/.style={
        width=4cm,
        height=,
        legend style={font=\tiny},
        tick label style={font=\tiny},
        label style={font=\tiny},
        title style={font=\footnotesize},
        every axis title shift=0pt,
        max space between ticks=12,
        every mark/.append style={mark size=6},
        major tick length=0.1cm,
        minor tick length=0.066cm,
        every legend image post/.append style={scale=0.8},
    },
}
```

As for `small`, it can be convenient to use `tiny,width=4.5cm` to adjust the width.

You will need `compat=1.3` or newer for this to work.

4.9 Scaling Options

/pgfplots/`width={⟨dimen⟩}`

Sets the width of the final picture to `{`⟨*dimen*⟩`}`. If no `height` is specified, scaling will respect aspect ratios.

Remarks:

- The scaling only affects the width of one unit in x-direction or the height for one unit in y-direction. Axis labels and tick labels won't be resized, but their size is used to determine the axis scaling.
- You can use the `scale={`⟨*number*⟩`}` option,

  ```
  \begin{tikzpicture}[scale=2]
  \begin{axis}
  ...
  \end{axis}
  \end{tikzpicture}
  ```

  to scale the complete picture.
- The Ti*k*Z-options `x` and `y` which set the unit dimensions in x and y directions can be specified as arguments to `\begin{axis}[x=1.5cm,y=2cm]` if needed (see below). These settings override the `width` and `height` options.
- You can also force a fixed width/height of the axis (without looking at labels) with

  ```
  \begin{tikzpicture}
  \begin{axis}[width=5cm,scale only axis]
      ...
  \end{axis}
  \end{tikzpicture}
  ```

- Please note that up to the writing of this manual, PGFPLOTS only estimates the size needed for axis- and tick labels. It does not include legends which have been placed outside of the axis[29]. This may be fixed in future versions.

  Use the `x={`⟨*dimension*⟩`}`, `y={`⟨*dimension*⟩`}` and `scale only axis` options if the scaling happens to be wrong.

/pgfplots/`height={⟨dimen⟩}`

See `width`.

/pgfplots/`scale only axis=true|false` (initially `false`)

If `scale only axis` is enabled, label, tick and legend dimensions won't influence the size of the axis rectangle, that means `width` and `height` apply only to the axis rectangle

[29] I.e. the '`width`' option will not work as expected, but the bounding box is still ok.

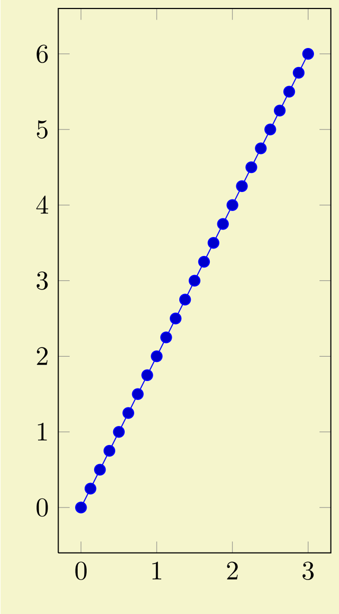
If `scale only axis=false` (the default), PGFPLOTS will try to produce the desired width *including* labels, titles and ticks.

/pgfplots/x={⟨*dimen*⟩}
/pgfplots/y={⟨*dimen*⟩}
/pgfplots/z={⟨*dimen*⟩}
/pgfplots/x={(⟨*x*⟩,⟨*y*⟩)}
/pgfplots/y={(⟨*x*⟩,⟨*y*⟩)}
/pgfplots/z={(⟨*x*⟩,⟨*y*⟩)}

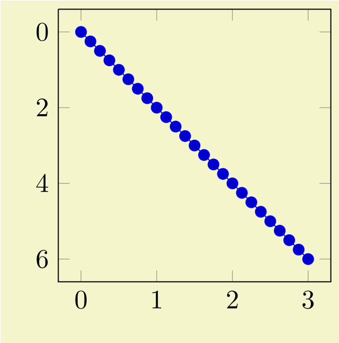
Sets the unit vectors for x (or y). Every logical plot coordinate (x, y) is drawn at the position

$$x \cdot \begin{bmatrix} e_{xx} \\ e_{xy} \end{bmatrix} + y \cdot \begin{bmatrix} e_{yx} \\ e_{yy} \end{bmatrix}.$$

The unit vectors e_x and e_y determine the paper position in the current (always two dimensional) image. The key `x={`⟨*dimen*⟩`}` simply sets $e_x = (\langle dimen \rangle, 0)^T$ while `y={`⟨*dimen*⟩`}` sets $e_y = (0, \langle dimen \rangle)^T$. Here, `{`⟨*dimen*⟩`}` is any TEX size like `1mm`, `2cm` or `5pt`. It is allowed to specify a negative `{`⟨*dimen*⟩`}`.



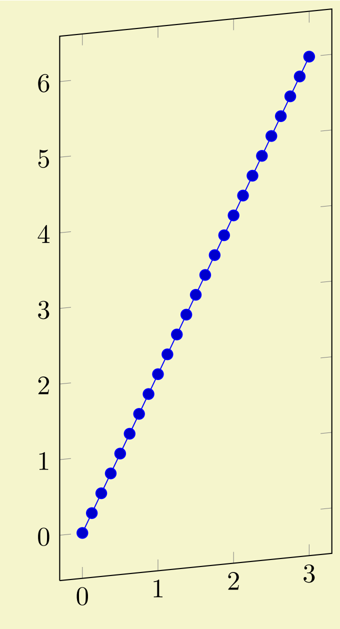
```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}[x=1cm,y=1cm]
\addplot expression[domain=0:3] {2*x};
\end{axis}
\end{tikzpicture}
```



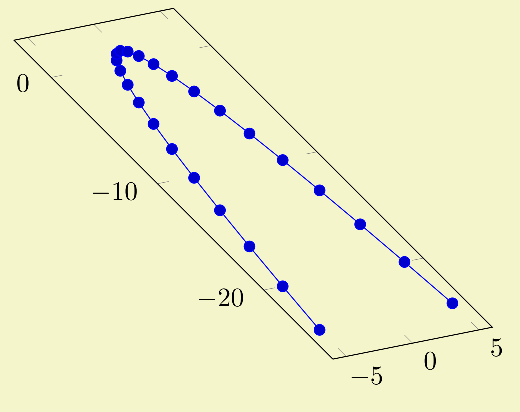
```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}[x=1cm,y=-0.5cm]
\addplot expression[domain=0:3] {2*x};
\end{axis}
\end{tikzpicture}
```

The second syntax, `x={(`⟨*x*⟩`,`⟨*y*⟩`)}` sets $e_x = (\langle x \rangle, \langle y \rangle)^T$ explicitly[30]; the corresponding `y` key works similarly. This allows to define skewed or rotated axes.

[30] Please note that you need extra curly braces around the vector. Otherwise, the comma will be interpreted as separator for the next key-value pair.



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}[x={(1cm,0.1cm)},y=1cm]
\addplot expression[domain=0:3] {2*x};
\end{axis}
\end{tikzpicture}
```



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}[
        x={(5pt,1pt)},
        y={(-4pt,4pt)}]
\addplot {1-x^2};
\end{axis}
\end{tikzpicture}
```

Setting x explicitly overrides the `width` option. Setting y explicitly overrides the `height` option.

Setting `x` and/or `y` for logarithmic axis will set the dimension used for $1 \cdot e \approx 2.71828$.

Please note that it is *not* possible to specify `x` as argument to `tikzpicture`. The option

```
\begin{tikzpicture}[x=1.5cm]
\begin{axis}
    ...
\end{axis}
\end{tikzpicture}
```

won't have any effect because an axis rescales its coordinates (see the `width` option).

Limitations: Unfortunately, skewed axes are **not available for bar plots**. Furthermore, support for custom vectors in 3D plots is currently **experimental** and may not work at all, sorry.

`/pgfplots/xmode=normal|linear|log` (initially `normal`)
`/pgfplots/ymode=normal|linear|log` (initially `normal`)
`/pgfplots/zmode=normal|linear|log` (initially `normal`)

Allows to choose between linear (=normal) or logarithmic axis scaling or logplots for each x, y, z-combination.

Logarithmic plots use the current setting of `log basis x` and its variants to determine the basis (default is e).

`/pgfplots/x dir=normal|reverse` (initially `normal`)

/pgfplots/y dir=normal|reverse (initially normal)
/pgfplots/z dir=normal|reverse (initially normal)

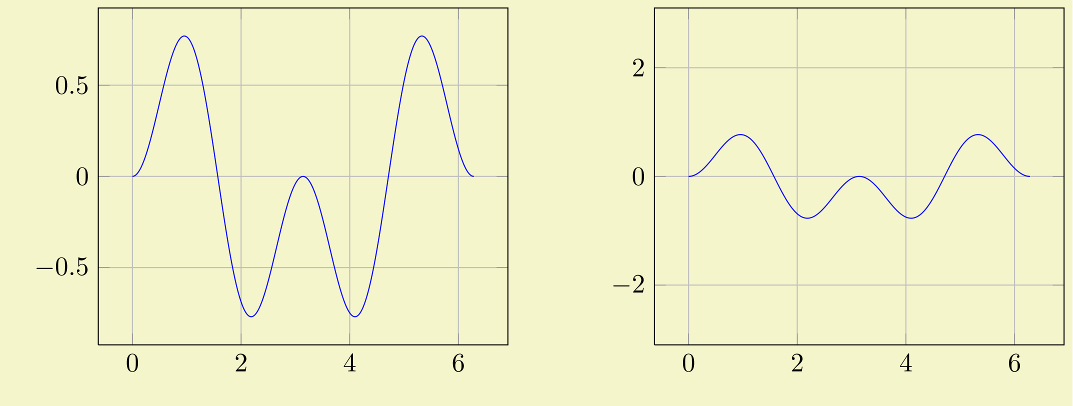
Allows to revert axis directions such that values are given in decreasing order.

This key is documented in all detail on page 154.

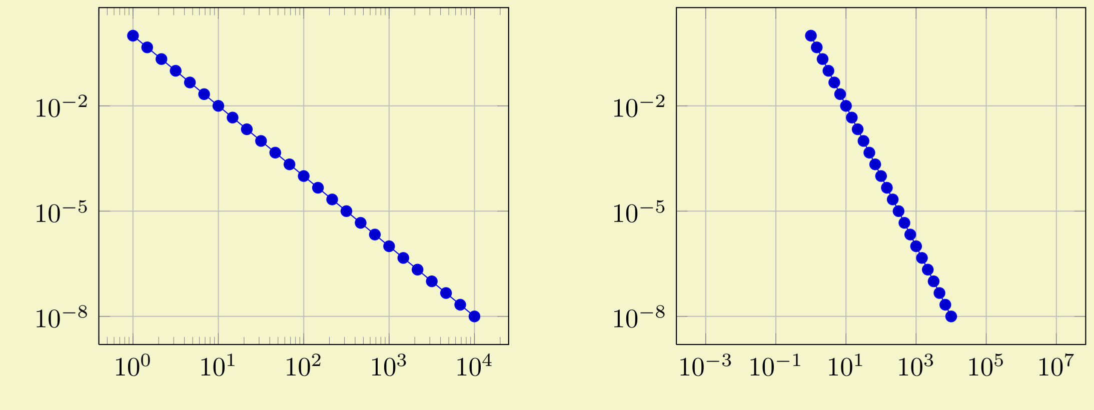
/pgfplots/axis equal={⟨*true,false*⟩} (initially false)

Each unit vector is set to the same length while the axis dimensions stay constant. Afterwards, the size ratios for each unit in x and y will be the same.

Axis limits will be enlarge to compensate for the scaling effect.



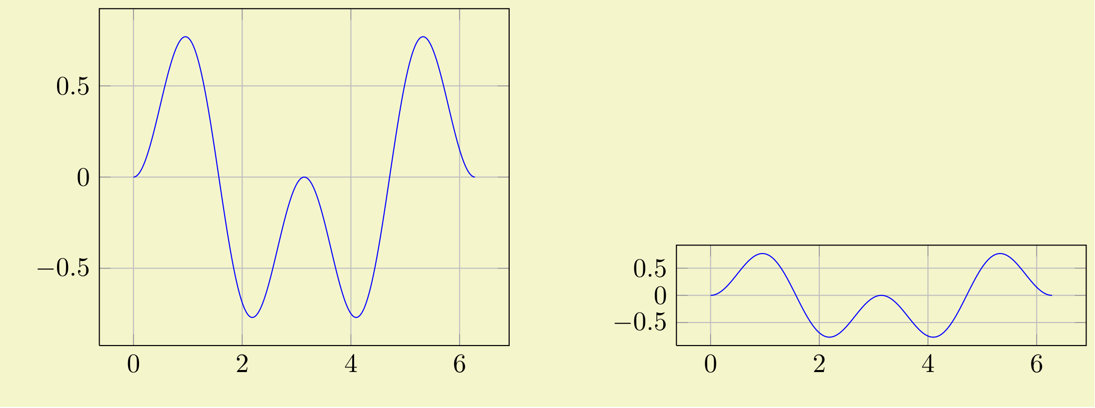
```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{axis}[axis equal=false,grid=major]
        \addplot[blue] expression[domain=0:2*pi,samples=300] {sin(deg(x))*sin(2*deg(x))};
    \end{axis}
\end{tikzpicture}
\hspace{1cm}
\begin{tikzpicture}
    \begin{axis}[axis equal=true,grid=major]
        \addplot[blue] expression[domain=0:2*pi,samples=300] {sin(deg(x))*sin(2*deg(x))};
    \end{axis}
\end{tikzpicture}
```



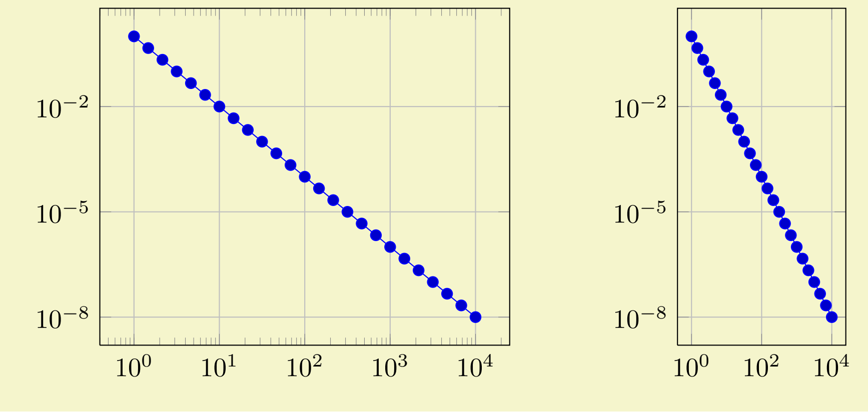
```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{loglogaxis}[axis equal=false,grid=major]
        \addplot expression[domain=1:10000] {x^-2};
    \end{loglogaxis}
\end{tikzpicture}
\hspace{1cm}
\begin{tikzpicture}
    \begin{loglogaxis}[axis equal=true,grid=major]
        \addplot expression[domain=1:10000] {x^-2};
    \end{loglogaxis}
\end{tikzpicture}
```

/pgfplots/axis equal image={⟨*true,false*⟩} (initially `false`)

Similar to `axis equal`, but the axis limits will stay constant as well (leading to smaller images).



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{axis}[axis equal image=false,grid=major]
        \addplot[blue] expression[domain=0:2*pi,samples=300] {sin(deg(x))*sin(2*deg(x))};
    \end{axis}
\end{tikzpicture}
\hspace{1cm}
\begin{tikzpicture}
    \begin{axis}[axis equal image=true,grid=major]
        \addplot[blue] expression[domain=0:2*pi,samples=300] {sin(deg(x))*sin(2*deg(x))};
    \end{axis}
\end{tikzpicture}
```



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{loglogaxis}[axis equal image=false,grid=major]
        \addplot expression[domain=1:10000] {x^-2};
    \end{loglogaxis}
\end{tikzpicture}
\hspace{1cm}
\begin{tikzpicture}
    \begin{loglogaxis}[axis equal image=true,grid=major]
        \addplot expression[domain=1:10000] {x^-2};
    \end{loglogaxis}
\end{tikzpicture}
```

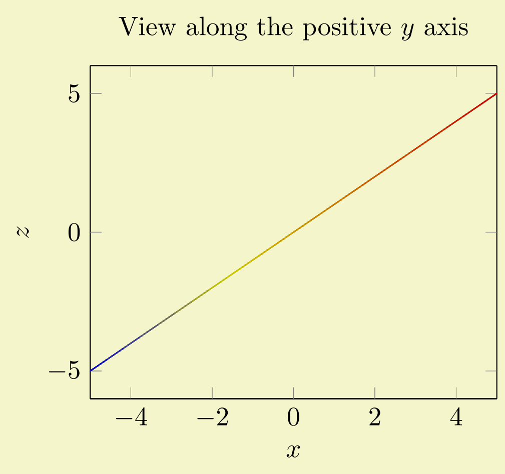
4.10 3D Axis Configuration

This section described keys which are used to configure the appearance of three dimensional figures. Some of them apply for two–dimensional plots as special case as well, and they will also be discussed in the respective sections of this manual.

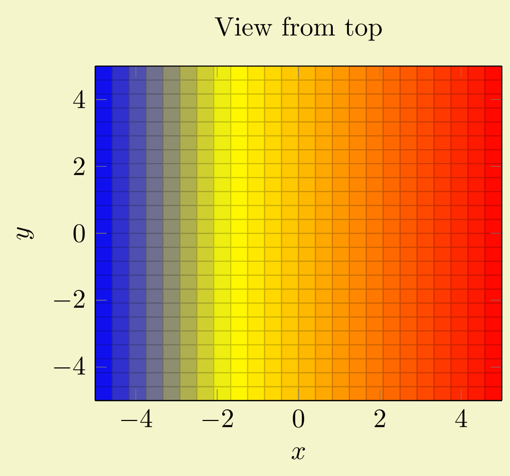
4.10.1 View Configuration

/pgfplots/view={⟨*azimuth*⟩}{⟨*elevation*⟩} (initially {25}{30})

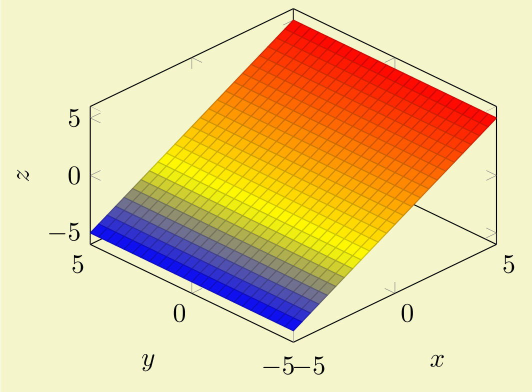
Changes both view angles of a 3D axis. The azimuth (first argument) is the horizontal angle which is rotated around the z axis. For a 3D plot, the z axis always points to the top. The elevation (second argument) is the vertical rotation around the (rotated) x axis. Positive elevation values indicate a view from above, negative a view from below. All values are measured in degree.



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{axis}[view={0}{0},
        xlabel=$x$,
        zlabel=$z$,
        title=View along the positive $y$ axis]
        \addplot3[surf] {x};
    \end{axis}
\end{tikzpicture}
```



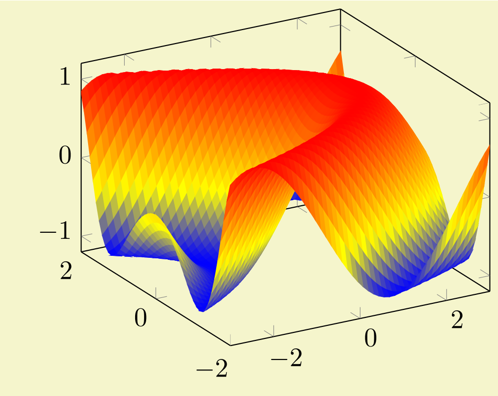
```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{axis}[view={0}{90},
        xlabel=$x$,
        ylabel=$y$,
        title=View from top]
        \addplot3[surf] {x};
    \end{axis}
\end{tikzpicture}
```



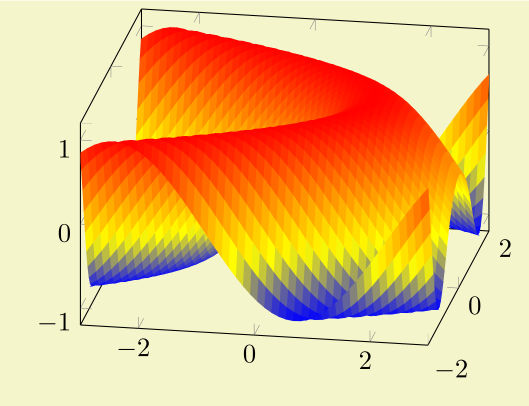
```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{axis}[view={-45}{45},
        xlabel=$x$,ylabel=$y$,zlabel=$z$]
        \addplot3[surf] {x};
    \end{axis}
\end{tikzpicture}
```

/pgfplots/view/az={⟨*azimuth*⟩}
/pgfplots/view/h={⟨*azimuth*⟩} (initially 25)

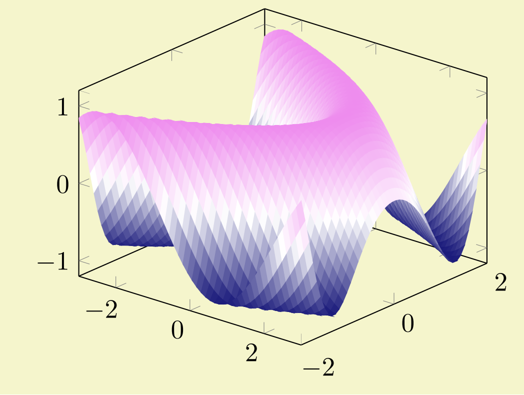
Changes only the azimuth view angle, i.e. the horizontal (first) view angle which is rotated around the z axis.



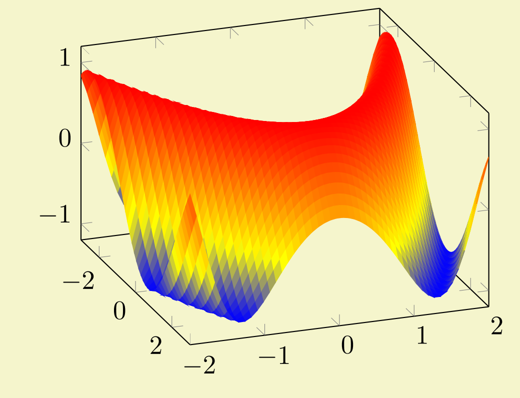
```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{axis}[view/h=-30]
    \addplot3[
        surf,
        % shader=interp,
        shader=flat,
        samples=50,
        domain=-3:3,y domain=-2:2]
        {sin(deg(x+y^2))};
    \end{axis}
\end{tikzpicture}
```



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{axis}[view/h=10]
    \addplot3[
        surf,
        % shader=interp,
        shader=flat,
        samples=50,
        domain=-3:3,y domain=-2:2]
        {sin(deg(x+y^2))};
    \end{axis}
\end{tikzpicture}
```



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{axis}[view/h=40,colormap/violet]
    \addplot3[
        surf,
        % shader=interp,
        shader=flat,
        samples=50,
        domain=-3:3,y domain=-2:2]
        {sin(deg(x+y^2))};
    \end{axis}
\end{tikzpicture}
```



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{axis}[view/h=70]
    \addplot3[
        surf,
        % shader=interp,
        shader=flat,
        samples=50,
        domain=-3:3,y domain=-2:2]
        {sin(deg(x+y^2))};
    \end{axis}
\end{tikzpicture}
```

/pgfplots/view/el={⟨*elevation*⟩}
/pgfplots/view/v={⟨*elevation*⟩} (initially 30)

Changes only the vertical elevation, i.e. the second argument to view. Positive values view from above, negative values from below.

4.10.2 Styles Used Only For 3D Axes

/pgfplots/every 3d description (style, no value)

This style allows to change the appearance of *descriptions* for three dimensional axes. Naturally, a three dimensional axis will display axis labels for x and y differently than a two dimensional axis (for example, the y axis label won't be rotated by 90 degrees). The every 3d description style installs the necessary display options for three dimensional axis descriptions.

The initial value is:

```
\pgfkeys{
    /pgfplots/every 3d description/.style={
        %  Only these description styles can be changed here:
        every axis x label/.style={at={(ticklabel cs:0.5)},
            anchor=near ticklabel},
        every axis y label/.style={at={(ticklabel cs:0.5)},
            anchor=near ticklabel},
        every x tick scale label/.style={
            at={(xticklabel cs:0.95,5pt)},
            anchor=near xticklabel,inner sep=0pt},
        every y tick scale label/.style={
            at={(yticklabel cs:0.95,5pt)},
            anchor=near yticklabel,inner sep=0pt},
        try min ticks=3,
    }%
}
```

As the name suggests, every 3d description can only be used to set styles for axis labels, tick labels and titles. It has *not* been designed to reset other styles, you will need to change these options either for each axis separately or by means of user defined styles. The reason for this limitation is: other options can (and, in many cases) need to be set before the axis is processed. However, the decision whether we have a two dimensional or a three dimensional axis has to be postponed until the processing is more or less complete – so only some remaining keys can be set.

/pgfplots/every 3d view {⟨*h*⟩}{⟨*v*⟩} (style, no value)

A style which can be used for fine tuning of the output for specific views.

This style will be installed right after every 3d description, but before other axis description related keys are set (in other words: it has higher precedence than every 3d description but less precedence than keys provided to the axis directly).

One example is preconfigured for view={0}{90} (from top):

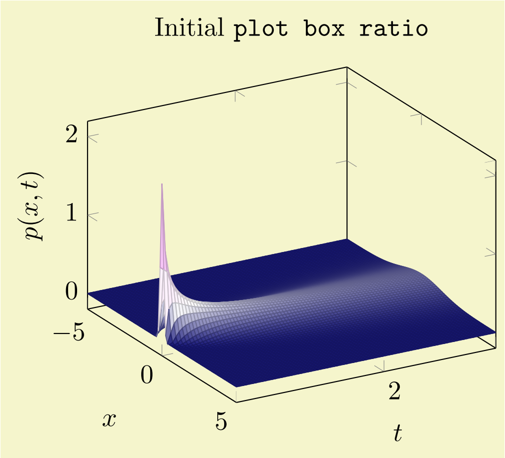
```
\pgfplotsset{
    /pgfplots/every 3d view {0}{90}/.style={
        xlabel near ticks,
        ylabel near ticks,
        axis on top=true
    }
}
```

4.10.3 Appearance Of The 3D Box

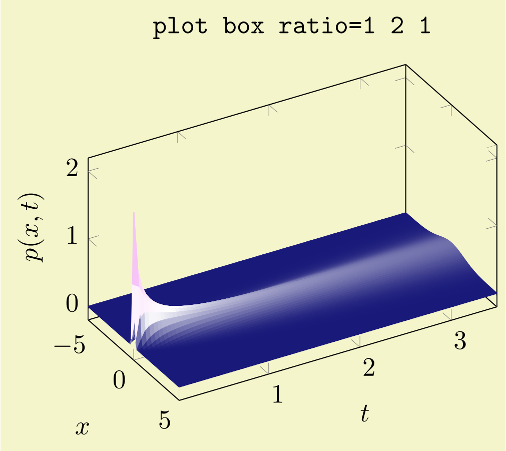
/pgfplots/plot box ratio={⟨*x stretch*⟩}{⟨*y stretch*⟩}{⟨*z stretch*⟩} (initially 111)

Allows to customize the aspect ratio between the three different axes in a three dimensional plot.

The plot box ratio is applied before any rotations and stretch–to–fill routines have been invoked. Thus, the initial setting {1}{1}{1} makes all axes equally long.



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}[
    view/h=60,
    plot box ratio={1}{1}{1},
    colormap={violet}{[1cm] rgb255(0cm)=(25,25,122)
        color(1cm)=(white) rgb255(5cm)=(238,140,238)},
    xlabel=$x$,
    ylabel=$t$,
    zlabel={$p(x,t)$},
    shader=faceted,
    title=Initial \texttt{plot box ratio},
]
    \addplot3[surf,y domain=0.02:3.5,samples=81]
        {1/(2*sqrt(pi*y)) * exp(0-x^2/y)};
    %  the '0' is a work-around for a bug in PGF 2.00
\end{axis}
\end{tikzpicture}
```



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}[
    view/h=60,
    plot box ratio={1}{2}{1},
    colormap={violet}{[1cm] rgb255(0cm)=(25,25,122)
        color(1cm)=(white) rgb255(5cm)=(238,140,238)},
    xlabel=$x$,
    ylabel=$t$,
    zlabel={$p(x,t)$},
    shader=flat,
    title=\texttt{plot box ratio=1 2 1},
]
    \addplot3[surf,y domain=0.02:3.5,samples=81]
        {1/(2*sqrt(pi*y)) * exp(0-x^2/y)};
    %  the '0' is a work-around for a bug in PGF 2.00
\end{axis}
\end{tikzpicture}
```

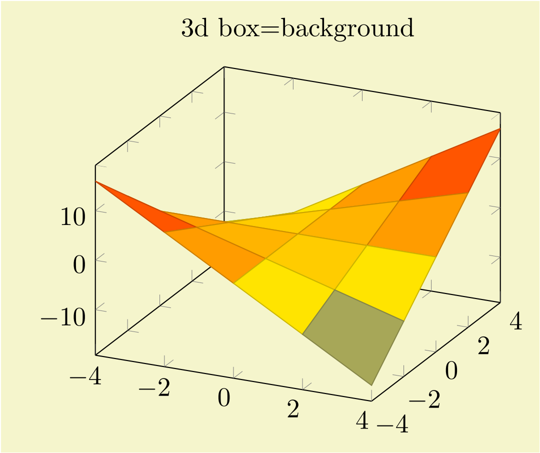
This key applies only to three dimensional axes. After the scaling, the axes will be stretched to fill the width and height for this plot. Thus, the effects of plot box ratio might be undone by this stretching for particular views.

/pgfplots/3d box=background|complete|complete* (initially background)

Allows to configure the appearance of boxed three dimensional axes.

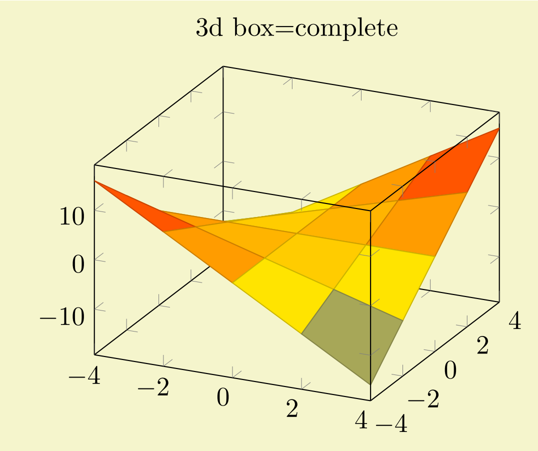
Type only 3d box (without value) as alias for 3d box=complete.

The choice background is the initial setting, it does not draw axis lines (and grid lines) which are in the foreground.



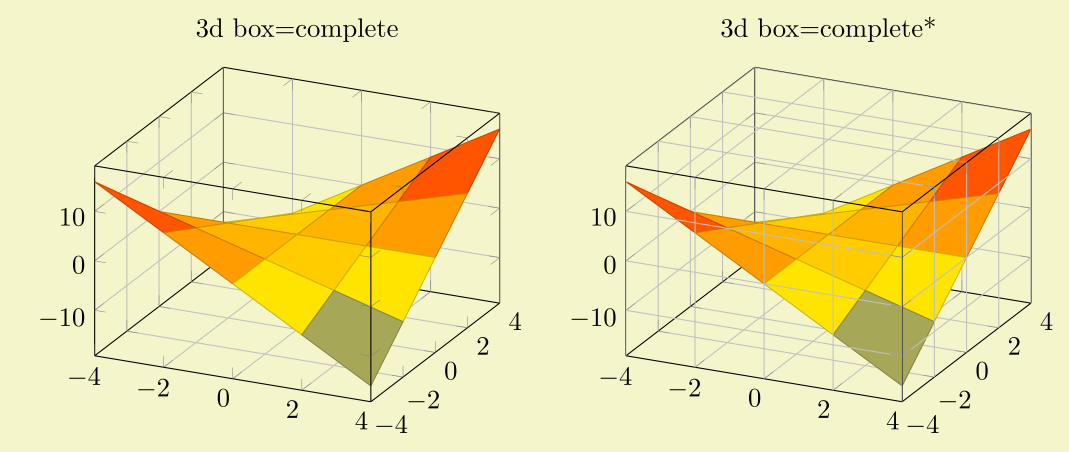
```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{axis}[
        3d box=background,
        %  pretty printing, but irrelevant:
        title={3d box=background},
        samples=5,
        domain=-4:4,
        xtick=data,
        ytick=data,
    ]
        \addplot3[surf] {x*y};
    \end{axis}
\end{tikzpicture}
```

The choice complete also draws axis lines and tick lines in the foreground, but it doesn't draw grid lines in the foreground. The result yields a complete box:



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{axis}[
        3d box,%  same as 3d box=complete
        %  pretty printing, but irrelevant:
        title={3d box=complete},
        samples=5,
        domain=-4:4,
        xtick=data,
        ytick=data,
    ]
        \addplot3[surf] {x*y};
    \end{axis}
\end{tikzpicture}
```

Finally, the choice `complete*` is the same as `complete`, but it also draws grid lines.



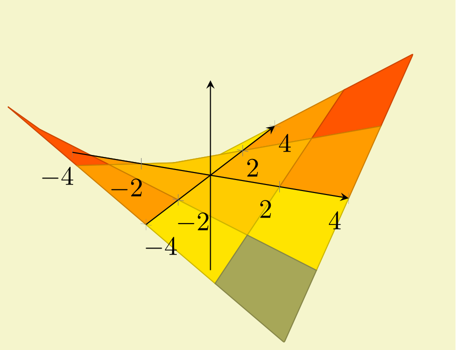
```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{axis}[
        3d box=complete,
        grid=major,
        title={3d box=complete},
        samples=5, domain=-4:4,
        xtick=data, ytick=data,
    ]
        \addplot3[surf] {x*y};
    \end{axis}
\end{tikzpicture}%
~
\begin{tikzpicture}
    \begin{axis}[
        3d box=complete*,
        grid=major,
        title={3d box=complete*},
        samples=5, domain=-4:4,
        xtick=data, ytick=data,
    ]
        \addplot3[surf] {x*y};
    \end{axis}
\end{tikzpicture}
```

Before any foreground parts are actually processed, the style `every 3d box foreground` will be installed. This allows to change the appearance of foreground axis components like `tick style` or `axis line style` separately from the background components.

Note that `3d box=complete` is *only* available for boxed axes, i.e. together with `axis lines=box`. It is an error to use a different combination.

4.10.4 Axis Line Variants

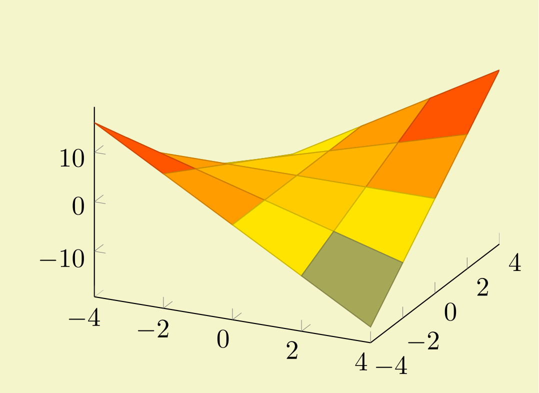
Three dimensional axes also benefit from the axis lines=box or axis lines=center styles discussed in section 4.8.7. The choice axis lines=box is standard, it draws a box (probably affected by the 3d box=complete key). The choice axis lines=center draws all three axes such that they pass through the origin. It might be necessary to combine this key with axis on top as there is no depth information.



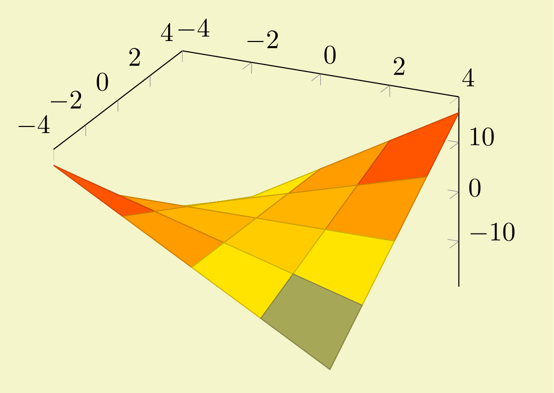
```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{axis}[
        axis lines=center,
        axis on top,
        samples=5, domain=-4:4,
        xtick=data, ytick=data,
        ztick=\empty, %  no z ticks here
    ]
        \addplot3[surf] {x*y};
    \end{axis}
\end{tikzpicture}
```

The remaining choices axis lines*=left and axis lines*=right select different sets of axes in a way such that tick labels and axis label won't disturb the plot's content. The '*' suppress the use of special styles which are mainly adequate for 2d axes, see the documentation of axis lines. Such a set of axes is always on the boundary of the two–dimensional projection.

The choice axis lines*=left chooses a set of axes which are left (or bottom, respectively) whereas the choice axis lines*=right chooses a set of axes which are on the right (or top, respectively):



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{axis}[
        axis lines*=left,
        samples=5, domain=-4:4,
        xtick=data, ytick=data,
    ]
        \addplot3[surf] {x*y};
    \end{axis}
\end{tikzpicture}
```



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{axis}[
        axis lines*=right,
        samples=5, domain=-4:4,
        xtick=data, ytick=data,
    ]
        \addplot3[surf] {x*y};
    \end{axis}
\end{tikzpicture}
```

It is not (yet) possible to mix different styles like axis x line=center,axis z line=top.

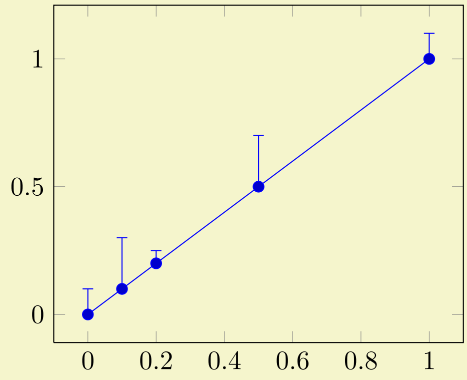
4.11 Error Bars

PGFPLOTS supports error bars for normal and logarithmic plots.

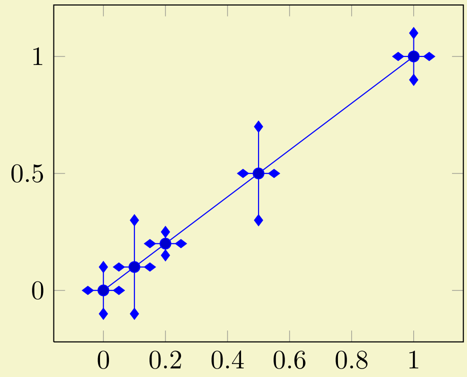
Error bars are enabled for each plot separately, using ⟨*options*⟩ after \addplot:

```
\addplot+[error bars/.cd,x dir=both,y dir=both] ...
```

Error bars inherit all drawing options of the associated plot, but they use their own marker and style arguments additionally.

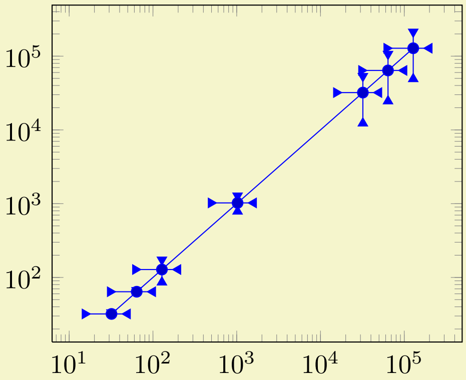


```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}
\addplot+[error bars/.cd,
    y dir=plus,y explicit]
coordinates {
    (0,0)     +- (0.5,0.1)
    (0.1,0.1) +- (0.05,0.2)
    (0.2,0.2) +- (0,0.05)
    (0.5,0.5) +- (0.1,0.2)
    (1,1)     +- (0.3,0.1)};
\end{axis}
\end{tikzpicture}
```

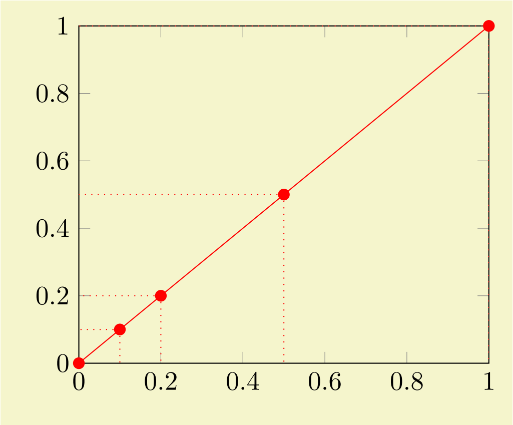


```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}
\addplot+[error bars/.cd,
    y dir=both,y explicit,
    x dir=both,x fixed=0.05,
    error mark=diamond*]
coordinates {
    (0,0)     +- (0.5,0.1)
    (0.1,0.1) +- (0.05,0.2)
    (0.2,0.2) +- (0,0.05)
    (0.5,0.5) +- (0.1,0.2)
    (1,1)     +- (0.3,0.1)};
\end{axis}
\end{tikzpicture}
```

x	y	errorx	errory
32	32	0	0
64	64	0	0
128	128	0	0.3
1,024	1,024	0	0.2
32,068	32,068	0	0.6
64,000	64,000	0	0.6
$1.28 \cdot 10^5$	$1.28 \cdot 10^5$	0	0.6



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\pgfplotstabletypeset{pgfplots.testtable2.dat}

\begin{tikzpicture}
\begin{loglogaxis}
\addplot+[error bars/.cd,
    x dir=both,x fixed relative=0.5,
    y dir=both,y explicit relative,
    error mark=triangle*]
    table[x=x,y=y,y error=errory]
    {pgfplots.testtable2.dat};
\end{loglogaxis}
\end{tikzpicture}
```



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}[enlargelimits=false]
\addplot[red,mark=*]
    plot[error bars/.cd,
    y dir=minus,y fixed relative=1,
    x dir=minus,x fixed relative=1,
    error mark=none,
    error bar style={dotted}]
coordinates
    {(0,0) (0.1,0.1) (0.2,0.2)
     (0.5,0.5) (1,1)};
\end{axis}
\end{tikzpicture}
```

/pgfplots/**error bars/x dir**=none|plus|minus|both (initially none)
/pgfplots/**error bars/y dir**=none|plus|minus|both (initially none)
/pgfplots/**error bars/z dir**=none|plus|minus|both (initially none)

Draws either no error bars at all, only marks at $x + \epsilon_x$, only marks at $x - \epsilon_x$ or marks at both, $x + \epsilon_x$ and $x - \epsilon_x$. The x-error ϵ_x is acquired using one of the following options.

The same holds for the `y dir` option.

/pgfplots/**error bars/x fixed**={⟨*value*⟩} (initially 0)
/pgfplots/**error bars/y fixed**={⟨*value*⟩} (initially 0)
/pgfplots/**error bars/z fixed**={⟨*value*⟩} (initially 0)

Provides a common, absolute error $\epsilon_x = \langle value \rangle$ for all input coordinates.

For linear x axes, the error mark is drawn at $x \pm \epsilon_x$ while for logarithmic x axes, it is drawn at $\log(x \pm \epsilon_x)$. Computations are performed in PGF's floating point arithmetics.

/pgfplots/**error bars/x fixed relative**={⟨*percent*⟩} (initially 0)
/pgfplots/**error bars/y fixed relative**={⟨*percent*⟩} (initially 0)
/pgfplots/**error bars/z fixed relative**={⟨*percent*⟩} (initially 0)

Provides a common, relative error $\epsilon_x = \langle percent \rangle \cdot x$ for all input coordinates. The argument ⟨*percent*⟩ is thus given relatively to input x coordinates such that $\langle percent \rangle = 1$ means 100%.

Error marks are thus placed at $x \cdot (1 \pm \epsilon_x)$ for linear axes and at $\log(x \cdot (1 \pm \epsilon_x))$ for logarithmic axes. Computations are performed in floating point for linear axis and using the identity $\log(x \cdot (1 \pm \epsilon_x)) = \log(x) + \log(1 \pm \epsilon_x)$ for logarithmic scales.

/pgfplots/**error bars/x explicit** (no value)
/pgfplots/**error bars/y explicit** (no value)
/pgfplots/**error bars/z explicit** (no value)

Configures the error bar algorithm to draw x-error bars at any input coordinate for which user-specified errors are available. Each error is interpreted as absolute error, see `x fixed` for details.

The different input formats of errors are described in section 4.11.1.

/pgfplots/**error bars/x explicit relative** (no value)
/pgfplots/**error bars/y explicit relative** (no value)
/pgfplots/**error bars/z explicit relative** (no value)

Configures the error bar algorithm to draw x-error bars at any input coordinate for which user-specified errors are available. Each error is interpreted as relative error, that means error marks are placed at $x(1 \pm \langle value \rangle(x))$ (works as for `error bars/x fixed relative`).

/pgfplots/**error bars/error mark**=⟨*marker*⟩

Sets an error marker for any error bar. {⟨*marker*⟩} is expected to be a valid plot mark, see section 4.6.

/pgfplots/**error bars/error mark options**={⟨*key-value-list*⟩}

Sets a key-value list of options for any error mark. This option works similary to the Ti*k*Z '`mark options`' key.

/pgfplots/error bars/error bar style={⟨*key-value-list*⟩}

Appends the argument to '/pgfplots/every error bar' which is installed at the beginning of every error bar.

/pgfplots/error bars/draw error bar/.code 2 args={⟨...⟩}

Allows to change the default drawing commands for error bars. The two arguments are

- the source point, (x, y) and
- the target point, $(\tilde{x}, \tilde{y})$.

Both are determined by PGFPLOTS according to the options described above. The default code is

```
\pgfplotsset{
    /pgfplots/error bars/draw error bar/.code 2 args={%
        \pgfkeysgetvalue{/pgfplots/error bars/error mark}%
            {\pgfplotserrorbarsmark}%
        \pgfkeysgetvalue{/pgfplots/error bars/error mark options}%
            {\pgfplotserrorbarsmarkopts}%
        \draw #1 -- #2 node[pos=1,sloped,allow upside down] {%
            \expandafter\tikz\expandafter[\pgfplotserrorbarsmarkopts]{%
                \expandafter\pgfuseplotmark\expandafter{\pgfplotserrorbarsmark}%
                \pgfusepath{stroke}}%
        };
    }
}
```

4.11.1 Input Formats of Error Coordinates

Error bars with explicit error estimations for single data points require some sort of input format. This applies to 'error bars/⟨*[xy]*⟩ explicit' and 'error bars/⟨*[xy]*⟩ explicit relative'.

Error bar coordinates can be read from 'plot coordinates' or from 'plot table'. The inline plot coordinates format is

```
\addplot coordinates {
    (1,2) +- (0.4,0.2)
    (2,4) +- (1,0)
    (3,5)
    (4,6) +- (0.3,0.001)
}
```

where $(1, 2) \pm (0.4, 0.2)$ is the first coordinate, $(2, 4) \pm (1, 0)$ the second and so forth. The point $(3, 5)$ has no error coordinate.

The 'plot table' format is

```
\addplot table[x error=COLNAME,y error=COLNAME]
```

or

```
\addplot table[x error index=COLINDEX,y error index=COLINDEX]
```

These options are used as the 'x' and 'x index' options.

You can supply error coordinates even if they are not used at all; they will be ignored silently in this case.

4.12 Number Formatting Options

PGFPLOTS typeset tick labels rounded to given precision and in configurable number formats. The command to do so is \pgfmathprintnumber; it uses the current set of number formatting options.

These options are described in all detail in the manual for PGFPLOTSTABLE, which comes with PGFPLOTS. Please refer to that manual.

\pgfmathprintnumber{⟨*x*⟩}

Generates pretty-printed output for the (real) number {⟨*x*⟩}. The input number {⟨*x*⟩} is parsed using \pgfmathfloatparsenumber which allows arbitrary precision.

Numbers are typeset in math mode using the current set of number printing options, see below. Optional arguments can also be provided using \pgfmathprintnumber[⟨*options*⟩]{⟨*x*⟩}.

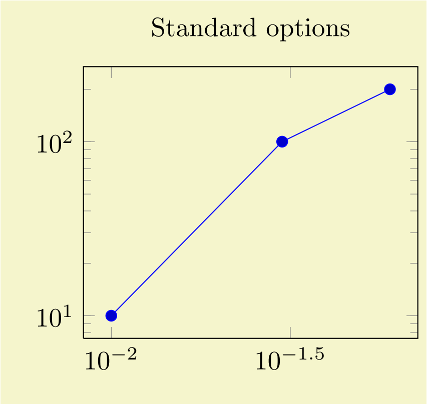
Please refer to the manual of PgfplotsTable (shipped with this package) for details about the number options.

/pgfplots/log identify minor tick positions=true|false (initially false)

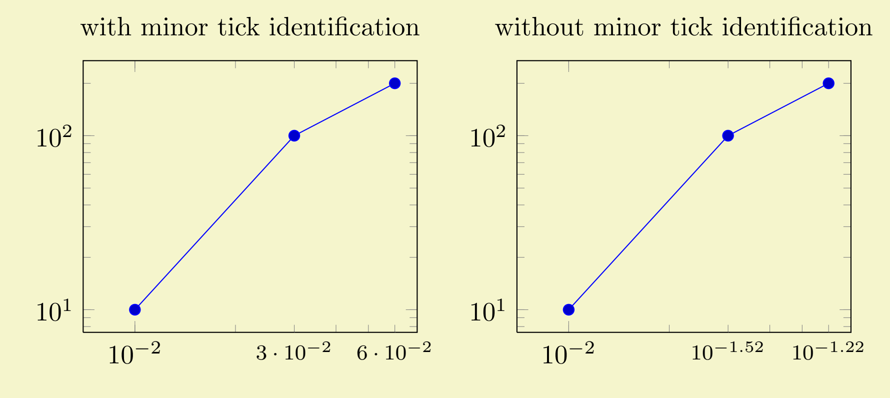
Set this to true if you want to identify log–plot tick labels at positions

$$i \cdot 10^j$$

with $i \in \{2,3,4,5,6,7,8,9\}$, $j \in \mathbb{Z}$. This may be valuable in conjunction with the 'extra x ticks' and 'extra y ticks' options.



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}%
\begin{loglogaxis}
    [title=Standard options,
    width=6cm]
\addplot coordinates {
    (1e-2,10)
    (3e-2,100)
    (6e-2,200)
};
\end{loglogaxis}
\end{tikzpicture}%
```



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\pgfplotsset{every axis/.append style={%
    width=6cm,
    xmin=7e-3,xmax=7e-2,
    extra x ticks={3e-2,6e-2},
    extra x tick style={major tick length=0pt,font=\footnotesize}
}}%

\begin{tikzpicture}%
    \begin{loglogaxis}[
        xtick={1e-2},
        title=with minor tick identification,
        extra x tick style={
            log identify minor tick positions=true}]
    \addplot coordinates {
        (1e-2,10)
        (3e-2,100)
        (6e-2,200)
    };
    \end{loglogaxis}
\end{tikzpicture}%

\begin{tikzpicture}%
    \begin{loglogaxis}[
        xtick={1e-2},
        title=without minor tick identification,
        extra x tick style={
            log identify minor tick positions=false}]
    \addplot coordinates {
        (1e-2,10)
        (3e-2,100)
        (6e-2,200)
    };
    \end{loglogaxis}%
\end{tikzpicture}%
```

This key is set by the default styles for extra ticks.

/pgfplots/log number format code/.code={⟨...⟩}

Provides TEX-code to generate log plot tick labels. Argument '#1' is the (natural) logarithm of the tick position. The default implementation invokes `log base 10 number format code` after it changed the log basis to 10. It also checks the other log plot options.

This key will have a different meaning when the log basis has been chosen explicitly, see the `log basis x` key.

/pgfplots/log base 10 number format code/.code={⟨...⟩}

Allows to change the overall appearance of base 10 log plot tick labels. The default is

```
\pgfplotsset{
    log base 10 number format code/.code={$10^{\pgfmathprintnumber{#1}}$}
}
```

where the '`log plot exponent style`' allows to change number formatting options.

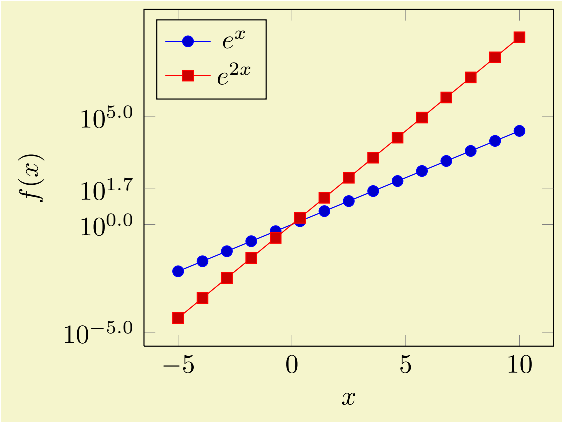
/pgfplots/log number format basis/.code={⟨...⟩}

This part of the representation routines for log ticks in *arbitrary* basis (see the `log basis x` key). It is used instead of the key above if the log basis has been changed. The first supplied argument is the log basis, the second the exponent. The initial configuration is

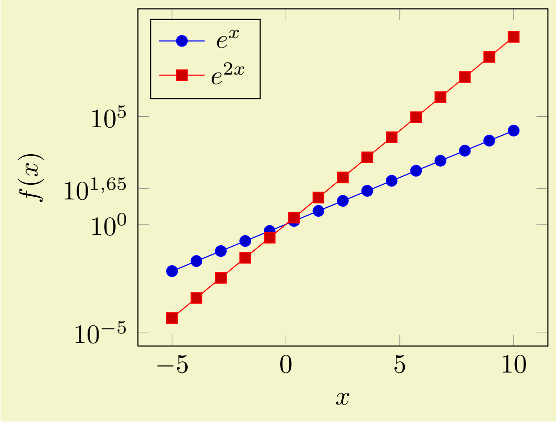
```
\pgfplotsset{
    /pgfplots/log number format basis/.code 2 args={$#1^{\pgfmathprintnumber{#2}}$}
}
```

/pgfplots/log plot exponent style={⟨*key-value-list*⟩}

Allows to configure the number format of log plot exponents. This style is installed just before '`log base 10 number format code`' will be invoked. Please note that this style will be installed within the default code for '`log number format code`'.



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\pgfplotsset{
    samples=15,
    width=7cm,
    xlabel=$x$,
    ylabel=$f(x)$,
    extra y ticks={45},
    legend style={at={(0.03,0.97)},
        anchor=north west}}

\begin{tikzpicture}
\begin{semilogyaxis}[
    log plot exponent style/.style={
        /pgf/number format/fixed zerofill,
        /pgf/number format/precision=1},
    domain=-5:10]

    \addplot {exp(x)};
    \addplot {exp(2*x)};

    \legend{$e^x$,$e^{2x}$}
\end{semilogyaxis}
\end{tikzpicture}
```



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\pgfplotsset{
    samples=15,
    width=7cm,
    xlabel=$x$,
    ylabel=$f(x)$,
    extra y ticks={45},
    legend style={at={(0.03,0.97)},
        anchor=north west}}

\begin{tikzpicture}
\begin{semilogyaxis}[
    log plot exponent style/.style={
        /pgf/number format/fixed,
        /pgf/number format/use comma,
        /pgf/number format/precision=2},
    domain=-5:10]

    \addplot {exp(x)};
    \addplot {exp(2*x)};

    \legend{$e^x$,$e^{2x}$}
\end{semilogyaxis}
\end{tikzpicture}
```

4.13 Specifying the Plotted Range

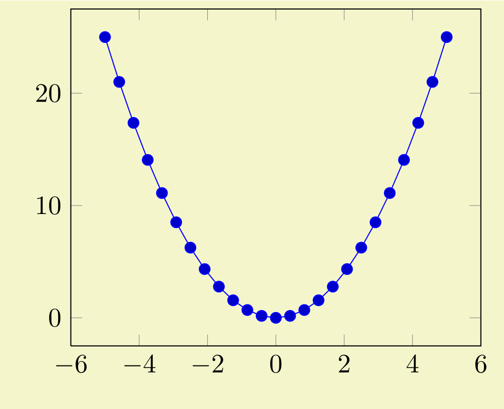
/pgfplots/xmin={⟨*coord*⟩}
/pgfplots/ymin={⟨*coord*⟩}
/pgfplots/zmin={⟨*coord*⟩}
/pgfplots/xmax={⟨*coord*⟩}
/pgfplots/ymax={⟨*coord*⟩}
/pgfplots/zmax={⟨*coord*⟩}

The options xmin, xmax and ymin, ymax allow to define the axis limits, i.e. the lower left and the upper right corner. Everything outside of the axis limits will be clipped away.

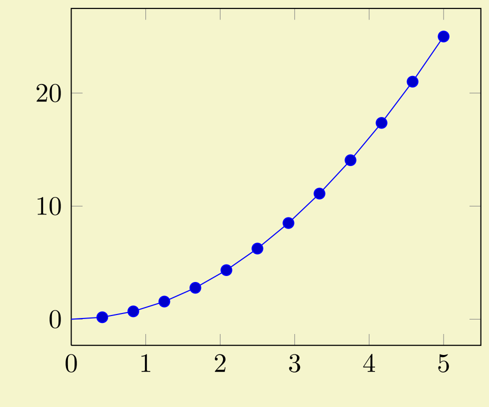
Each missing limit will be determined automatically.

If x-limits have been specified explicitly and y-limits are computed automatically, the automatic computation of y-limits will only considers points which fall into the specified x-range (and vice–versa). The same holds true if, for example, only xmin has been provided explicitly: in that case, xmax will be updated only for points for which $x \geq$xmin holds. This feature can be disabled using clip limits=false.

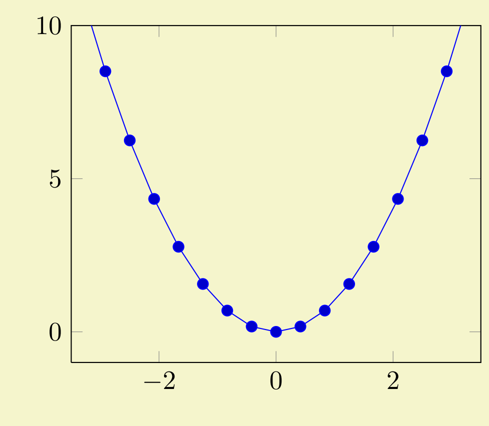
Axis limits can be increased automatically using the enlargelimits option.



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{axis}
    \addplot {x^2};
    \end{axis}
\end{tikzpicture}
```



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{axis}[xmin=0]
    \addplot {x^2};
    \end{axis}
\end{tikzpicture}
```



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{axis}[ymax=10]
    \addplot {x^2};
    \end{axis}
\end{tikzpicture}
```

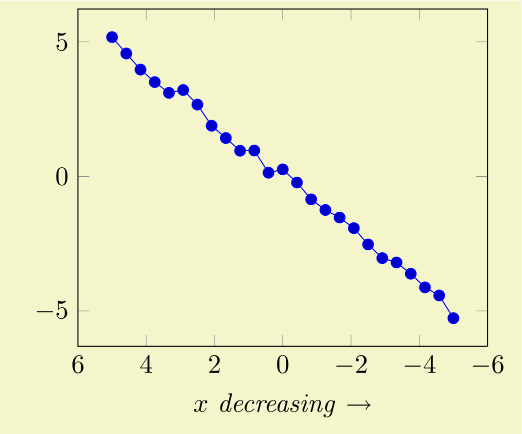
`/pgfplots/xmode=normal|linear|log` (initially `normal`)
`/pgfplots/ymode=normal|linear|log` (initially `normal`)
`/pgfplots/zmode=normal|linear|log` (initially `normal`)

Allows to choose between linear (=normal) or logarithmic axis scaling or logplots for each x, y, z-combination.

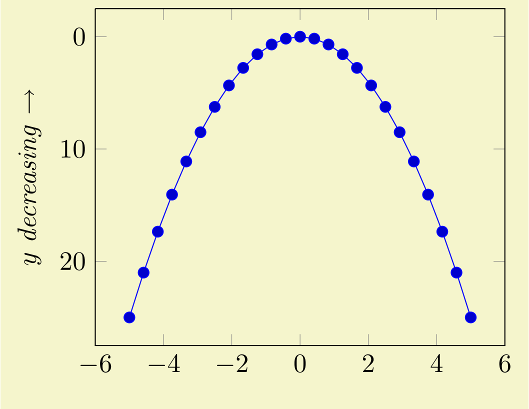
Logarithmic plots use the current setting of `log basis x` and its variants to determine the basis (default is e).

`/pgfplots/x dir=normal|reverse` (initially `normal`)
`/pgfplots/y dir=normal|reverse` (initially `normal`)
`/pgfplots/z dir=normal|reverse` (initially `normal`)

Allows to revert axis directions such that values are given in decreasing order.

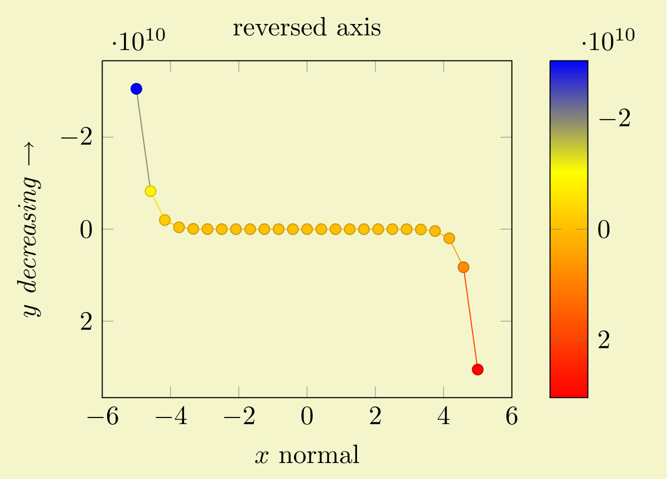


```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}[
    xlabel=$x$ \emph{decreasing} $\to$,
    x dir=reverse]
    \addplot {x+rand*0.3};
\end{axis}
\end{tikzpicture}
```



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}[
    ylabel=$y$ \emph{decreasing} $\to$,
    y dir=reverse]
    \addplot {x^2};
\end{axis}
\end{tikzpicture}
```

Note, that axis descriptions and relative positioning macros will stay at the same place as they would for non–reversed axes.



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}[
    ylabel=$y$ \emph{decreasing} $\to$,
    xlabel=$x$ normal,
    title=reversed axis,
    y dir=reverse,
    colorbar,
    colorbar style={y dir=reverse}]
    \addplot+[mesh,scatter] {x^15};
\end{axis}
\end{tikzpicture}
```

Note that colorbars won't be reversed automatically, you will have to reverse the sequence of color bars manually in case this is required as in the preceding example.

/pgfplots/clip limits=true|false (initially true)

Configures what to do if some, but not all axis limits have been specified explicitly. In case clip limits=true, the automatic limit computation will *only* consider points which do not contradict the explicitly set limits.

This option has nothing to do with path clipping, it only affects how the axis limits are computed.

/pgfplots/enlarge x limits=auto|true|false|upper|lower|⟨*val*⟩|value=⟨*val*⟩|abs value=⟨*val*⟩|abs=⟨*val*⟩|rel=⟨*val*⟩ (initially auto)

/pgfplots/enlarge y limits=auto|true|false|upper|lower|⟨*val*⟩|value=⟨*val*⟩|abs value=⟨*val*⟩|abs=⟨*val*⟩|rel=⟨*val*⟩ (initially auto)

/pgfplots/enlarge z limits=auto|true|false|upper|lower|⟨*val*⟩|value=⟨*val*⟩|abs value=⟨*val*⟩|abs=⟨*val*⟩|rel=⟨*val*⟩ (initially auto)

/pgfplots/enlargelimits=⟨*common value*⟩

Enlarges the axis size for one axis (or all of them for enlargelimits) somewhat if enabled.

You can set xmin, xmax and ymin, ymax to the minimum/maximum values of your data and enlarge x limits will enlarge the canvas such that the axis doesn't touch the plots.

- The value true enlarges the lower and upper limit.
- The value false uses tight axis limits as specified by the user (or read from input coordinates).
- The value auto will enlarge limits only for axis for which axis limits have been determined automatically.

  For three–dimensional figures, the auto mechanism applies only for the z axis. The x and y axis won't be enlarged.
- The value upper enlarges only the upper axis limit while lower enlarges only the lower axis limit.
- Values like 'enlarge x limits=0.1' will enlarge lower and upper axis limit relatively (in this example, 10% of the axis limits will be added on both sides).
- It is also possible to change just the relative threshold using the value={⟨*val*⟩} key. It can be combined with any of the other possible values. For example,

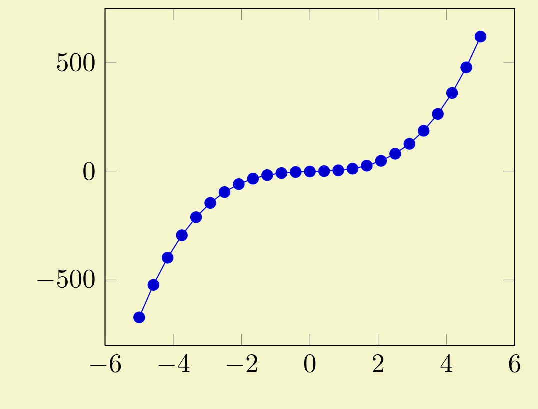
  \pgfplotsset{enlarge x limits={value=0.2,upper}}

  will enlarge (only) the upper axis limit by 20% of the axis range. Another example is

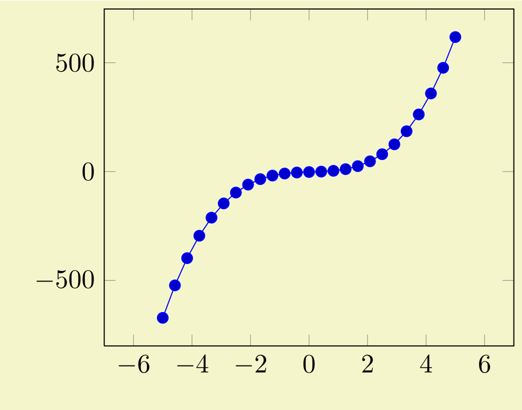
  \pgfplotsset{enlarge x limits={value=0.2,auto}}

  which changes the default threshold of the auto value to 20%.
- While value uses relative thresholds, abs value is used in the same way with absolute values.

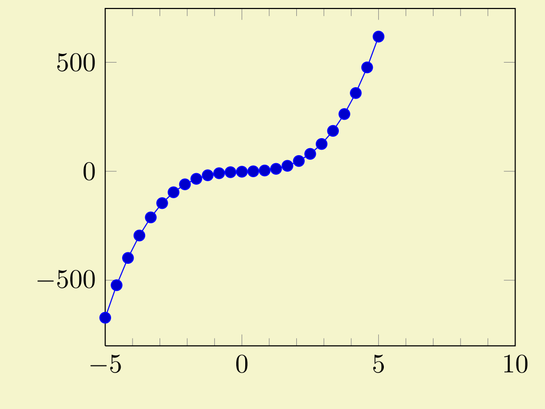
  Attention: abs value is applied *multiplicative* for logarithmic axes! That means abs value=10 for a logarithmic axis adds $\log 10$ to upper and/or lower axis limits.
- Finally, abs={⟨*value*⟩} is the same as true,abs value={⟨*value*⟩} and rel={⟨*value*⟩} is the same as true,value={⟨*value*⟩}.



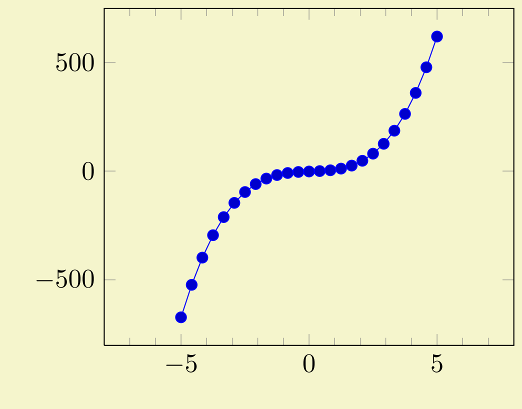
```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{axis}
        \addplot {5 * x^3 - x^2 + 4*x -2};
    \end{axis}
\end{tikzpicture}
```



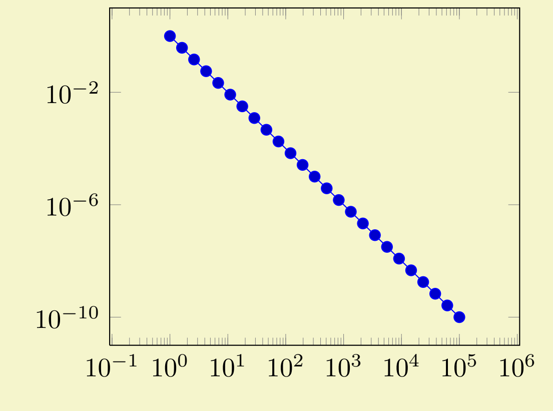
```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{axis}[enlarge x limits=0.2]
        \addplot {5 * x^3 - x^2 + 4*x -2};
    \end{axis}
\end{tikzpicture}
```



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{axis}[minor x tick num=4,
        enlarge x limits={rel=0.5,upper}
    ]
        \addplot {5 * x^3 - x^2 + 4*x -2};
    \end{axis}
\end{tikzpicture}
```



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{axis}[minor x tick num=4,
        enlarge x limits={abs=3}
    ]
        \addplot {5 * x^3 - x^2 + 4*x -2};
    \end{axis}
\end{tikzpicture}
```



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{loglogaxis}[enlarge x limits={abs=11}]
        \addplot+[domain=1:100000] {x^-2};
    \end{loglogaxis}
\end{tikzpicture}
```

```
\begin{pgfplotsinterruptdatabb}
```

⟨*environment contents*⟩
\end{pgfplotsinterruptdatabb}

Everything in {⟨*environment contents*⟩} will not contribute to the data bounding box.

4.14 Tick Options

4.14.1 Tick Coordinates and Label Texts

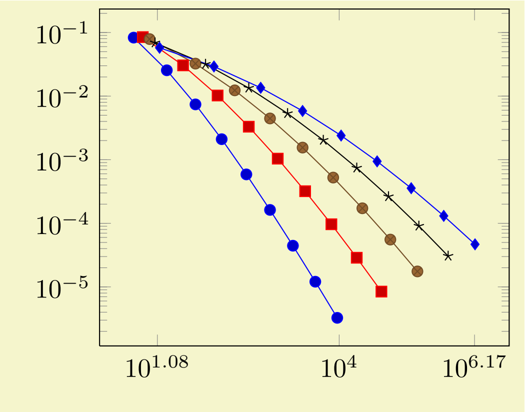
/pgfplots/xtick=\empty|data|{⟨*coordinate list*⟩} (initially {⟨⟩})
/pgfplots/ytick=\empty|data|{⟨*coordinate list*⟩} (initially {⟨⟩})
/pgfplots/ztick=\empty|data|{⟨*coordinate list*⟩} (initially {⟨⟩})

These options assign a list of *Positions* where ticks shall be placed. The argument is either the empty string (which is the initial value), the command `\empty`, `data` or a list of coordinates. The initial configuration of an empty string means to generate these positions automatically. The choice `\empty` will result in no tick at all. The special value `data` will produce tick marks at every coordinate of the first plot. Otherwise, tick marks will be placed at every coordinate in {⟨*coordinate list*⟩}.

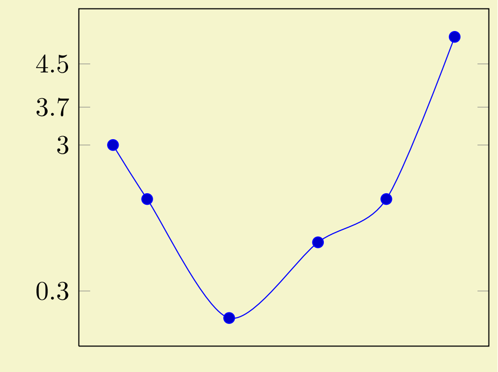
The {⟨*coordinate list*⟩} will be used inside of a `\foreach \x in` {⟨*coordinate list*⟩} statement. The format is as follows:

- `{0,1,2,5,8,1e1,1.5e1}` (a series of coordinates),
- `{0,...,5}` (the same as `{0,1,2,3,4,5}`),
- `{0,2,...,10}` (the same as `{0,2,4,6,8,10}`),
- `{9,...,3.5}` (the same as `{9, 8, 7, 6, 5, 4}`),
- See [5, Section 34] for a more detailed definition of the options.
- Please be careful with white spaces inside of {⟨*coordinate list*⟩} (at least around the dots).

For logplots, PGFPLOTS will apply log(·) to each element in '{⟨*coordinate list*⟩}'.



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{loglogaxis}[xtick={12,9897,1468864}]
    %  see above for this macro:
    \plotcoords
    \end{loglogaxis}
\end{tikzpicture}
```



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}[
    xtick=\empty,
    ytick={-2,0.3,3,3.7,4.5}]
\addplot+[smooth] coordinates {
    (-2,3) (-1.5,2) (-0.3,-0.2)
    (1,1.2) (2,2) (3,5)};
\end{axis}
\end{tikzpicture}
```

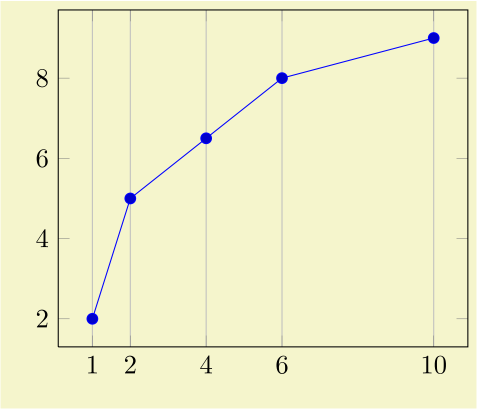
Attention: You can't use the '`...`' syntax if the elements are too large for TEX! For example, '`xtick=1.5e5,2e7,3e8`' will work (because the elements are interpreted as strings, not as numbers), but '`xtick=1.5,3e5,...,1e10`' will fail because it involves real number arithmetics beyond TEX's capacities.

The default choice for tick *positions* in normal plots is to place a tick at each coordinate $i \cdot h$. The step size h depends on the axis scaling and the axis limits. It is chosen from a list a "feasible" step sizes such that neither too much nor too few ticks will be generated. The default for logplots is to place ticks at positions 10^i in the axis' range. The positions depend on the axis scaling and the dimensions of the picture. If log plots contain just one (or two) positions 10^i in their limits, ticks will be placed at positions $10^{i \cdot h}$ with "feasible" step sizes h as in the case of linear axis.

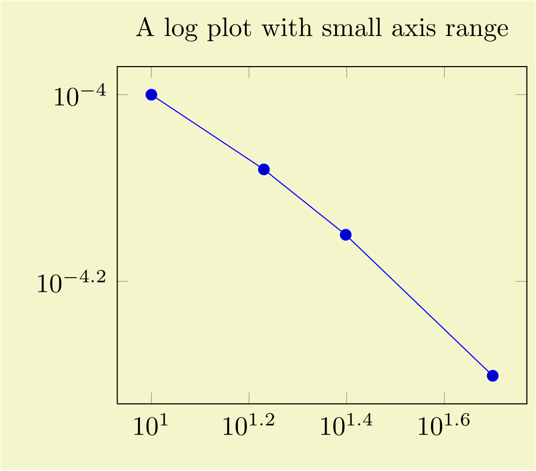
The tick *appearance* can be (re-)configured with

```
\pgfplotsset{tick style={very thin,gray}}%  modifies the style ‘every tick’
\pgfplotsset{minor tick style={black}}   %  modifies the style ‘every minor tick’
```

These style commands can be used at any time. The tick line width can be configured with '`major tick length`' and '`minor tick length`'.



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}[xtick=data,xmajorgrids]
    \addplot coordinates {
        (1,2)
        (2,5)
        (4,6.5)
        (6,8)
        (10,9)
    };
\end{axis}
\end{tikzpicture}
```

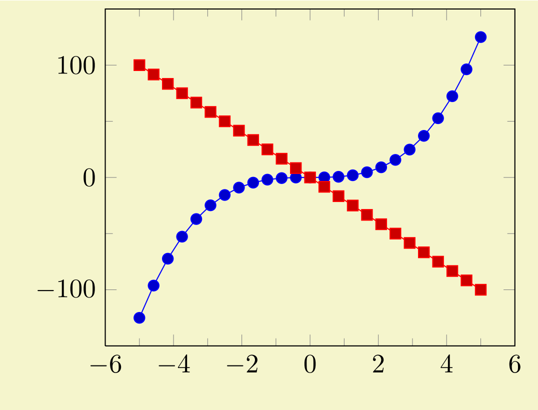


```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{loglogaxis}[
    title=A log plot with small axis range]

    \addplot coordinates {
        (10,1e-4)
        (17,8.3176e-05)
        (25,7.0794e-05)
        (50,5e-5)
    };
\end{loglogaxis}
\end{tikzpicture}
```

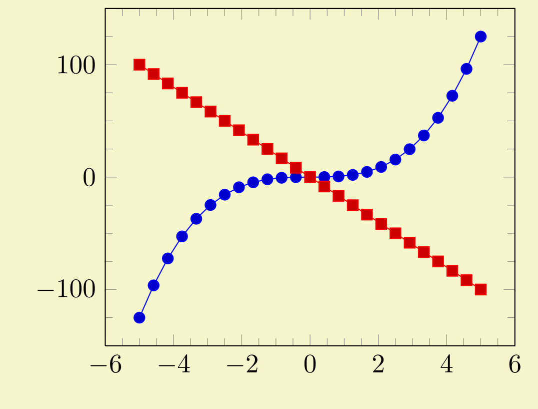
/pgfplots/**minor x tick num**={⟨*number*⟩} (initially 0)
/pgfplots/**minor y tick num**={⟨*number*⟩} (initially 0)
/pgfplots/**minor z tick num**={⟨*number*⟩} (initially 0)
/pgfplots/**minor tick num**={⟨*number*⟩}

Sets the number of minor tick lines used either for single axes or for all of them.

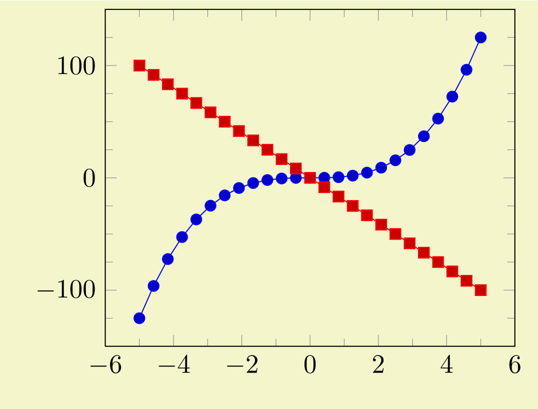
Minor ticks will be disabled if the major ticks don't have the same distance and they are currently only available for linear axes (not for logarithmic ones).



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{axis}[minor tick num=1]
    \addplot {x^3};
    \addplot {-20*x};
    \end{axis}
\end{tikzpicture}
```



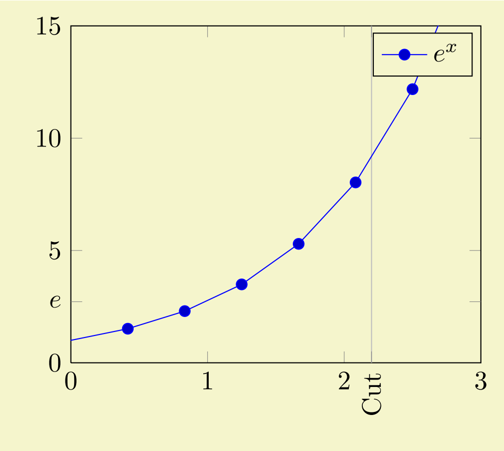
```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{axis}[minor tick num=3]
    \addplot {x^3};
    \addplot {-20*x};
    \end{axis}
\end{tikzpicture}
```



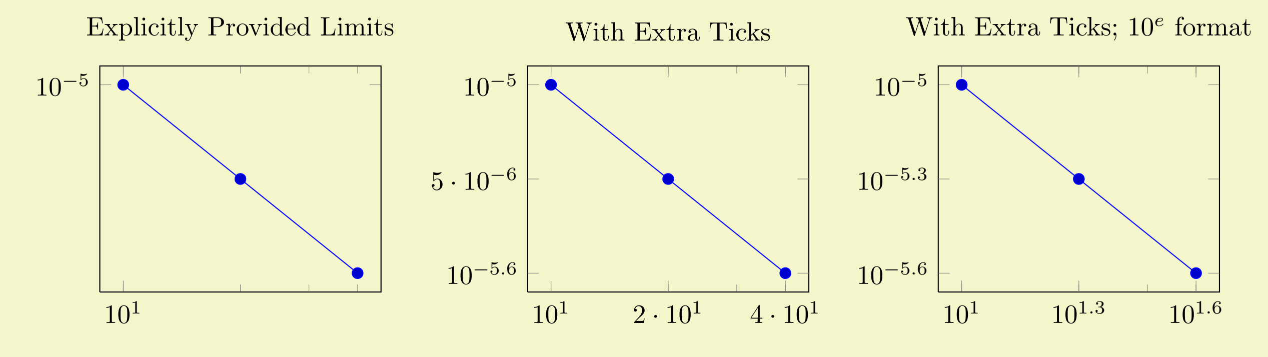
```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{axis}[minor x tick num=1,
                 minor y tick num=3]
    \addplot {x^3};
    \addplot {-20*x};
    \end{axis}
\end{tikzpicture}
```

/pgfplots/**extra x ticks**={⟨*coordinate list*⟩}
/pgfplots/**extra y ticks**={⟨*coordinate list*⟩}
/pgfplots/**extra z ticks**={⟨*coordinate list*⟩}

Adds *additional* tick positions and tick labels to the x or y axis. ‘Additional’ tick positions do not affect the normal tick placement algorithms, they are drawn after the normal ticks. This has two benefits: first, you can add single, important tick positions without disabling the default tick label generation and second, you can draw tick labels ‘on top’ of others, possibly using different style flags.



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}[
    xmin=0,xmax=3,ymin=0,ymax=15,
    extra y ticks={2.71828},
    extra y tick labels={$e$},
    extra x ticks={2.2},
    extra x tick style={grid=major,
        tick label style={
            rotate=90,anchor=east}},
    extra x tick labels={Cut},
]
    \addplot {exp(x)};
    \addlegendentry{$e^x$}
\end{axis}
\end{tikzpicture}
```



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\pgfplotsset{every axis/.append style={width=5.3cm}}
\begin{tikzpicture}
\begin{loglogaxis}[
    title=Explicitly Provided Limits,
    xtickten={1,2},
    ytickten={-5,-6}]
\addplot coordinates
    {(10,1e-5) (20,5e-6) (40,2.5e-6)};
\end{loglogaxis}
\end{tikzpicture}

\begin{tikzpicture}
\begin{loglogaxis}[
    title=With Extra Ticks,
    xtickten={1,2},
    ytickten={-5,-6},
    extra x ticks={20,40},
    extra y ticks={5e-6,2.5e-6}]
\addplot coordinates
    {(10,1e-5) (20,5e-6) (40,2.5e-6)};
\end{loglogaxis}
\end{tikzpicture}

\begin{tikzpicture}
\begin{loglogaxis}[
    title=With Extra Ticks; $10^e$ format,
    extra tick style={log identify minor tick positions=false},
    xtickten={1,2},
    ytickten={-5,-6},
    extra x ticks={20,40},
    extra y ticks={5e-6,2.5e-6}]
\addplot coordinates
    {(10,1e-5) (20,5e-6) (40,2.5e-6)};
\end{loglogaxis}
\end{tikzpicture}
```

Remarks:

- Use `extra x ticks` to highlight special tick positions. The use of `extra x ticks` does not affect minor tick/grid line generation, so you can place extra ticks at positions $j \cdot 10^i$ in log–plots.

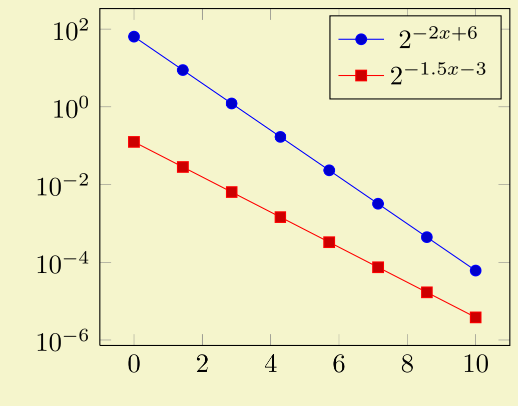
- Extra ticks are always typeset as major ticks.

  They are affected by `major tick length` or options like `grid=major`.
- Use the style `every extra x tick` (`every extra y tick`) to configure the appearance.
- You can also use '`extra x tick style={`⟨...⟩`}`' which has the same effect.

/pgfplots/`xtickten={`⟨*exponent base 10 list*⟩`}`
/pgfplots/`ytickten={`⟨*exponent base 10 list*⟩`}`
/pgfplots/`ztickten={`⟨*exponent base 10 list*⟩`}`

These options allow to place ticks at selected positions $10^k, k \in$ `{`⟨*exponent base 10 list*⟩`}`. They are only used for logplots. The syntax for `{`⟨*exponent base 10 list*⟩`}` is the same as above for `xtick={`⟨*list*⟩`}` or `ytick={`⟨*list*⟩`}`.

Using '`xtickten={1,2,3,4}`' is equivalent to '`xtick={1e1,1e2,1e3,1e4}`', but it requires fewer computational time and it allows to use the short syntax '`xtickten={1,...,4}`'.



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{semilogyaxis}[
    samples=8,
    ytickten={-6,-4,...,4},
    domain=0:10]

\addplot {2^(-2*x + 6)};
\addlegendentry{$2^{-2x + 6}$}

%  or invoke gnuplot to generate coordinates:
\addplot gnuplot[id=pow2]
    {2**(-1.5*x -3)};
\addlegendentry{$2^{-1.5x -3}$}
\end{semilogyaxis}
\end{tikzpicture}
```

In case `log basis x`$\neq$ 10, the meaning of `xtickten` changes. In such a case, `xtickten` will still assign the exponent, but for the chosen `log basis x` instead of base 10.

/pgfplots/`xticklabels={`⟨*label list*⟩`}`
/pgfplots/`yticklabels={`⟨*label list*⟩`}`
/pgfplots/`zticklabels={`⟨*label list*⟩`}`

Assigns a *list* of tick *labels* to each tick position. Tick *positions* are assigned using the `xtick` and `ytick`-options.

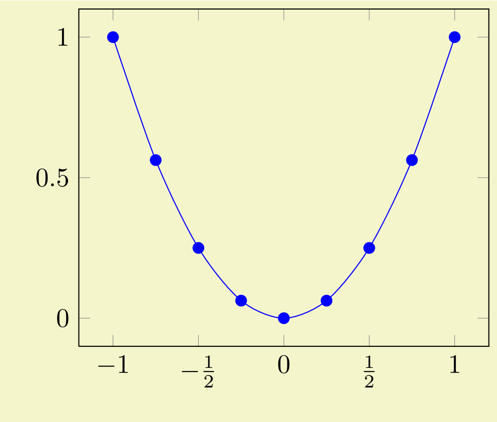
This is one of two options to assign tick labels directly. The other option is `xticklabel={`⟨*command*⟩`}` (or `yticklabel={`⟨*command*⟩`}`). The option '`xticklabel`' offers higher flexibility while '`xticklabels`' is easier to use. See also the variant `xticklabels from table`.

The argument `{`⟨*label list*⟩`}` has the same format as for ticks, that means

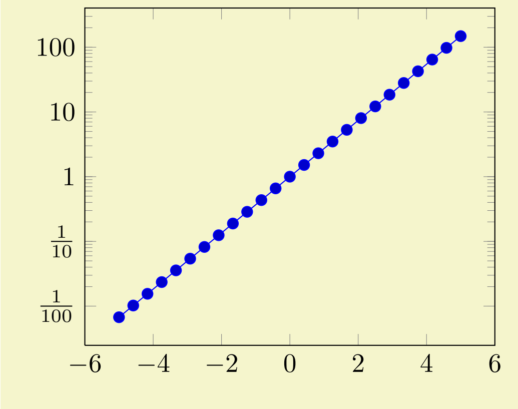
```
xticklabels={$\frac{1}{2}$,$e$}
```

Denotes the two–element–list $\{\frac{1}{2}, e\}$. The list indices match the indices of the tick positions. If you need commas inside of list elements, use

```
xticklabels={{0,5}, $e$}.
```



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}[
    xtick={-1.5,-1,...,1.5},
    xticklabels={%
        $-1\frac 12$,
        $-1$,
        $-\frac 12$,
        $0$,
        $\frac 12$,
        $1$}
]
\addplot[smooth,blue,mark=*]
coordinates {
    (-1,    1)
    (-0.75, 0.5625)
    (-0.5,  0.25)
    (-0.25, 0.0625)
    (0,     0)
    (0.25,  0.0625)
    (0.5,   0.25)
    (0.75,  0.5625)
    (1,     1)
};
\end{axis}
\end{tikzpicture}
```



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{semilogyaxis}[
  ytickten={-2,-1,0,1,2},
  yticklabels={$\frac{1}{100}$,%
      $\frac{1}{10}$,%
    1,10,100},
]
    \addplot {exp(x)};
\end{semilogyaxis}
\end{tikzpicture}
```

Note that it is also possible to terminate list entries with two backslashes, \\. In that case, the last entry needs to be terminated by \\ as well (it is the same alternative syntax which is also accepted for `\legend` and `cycle list`).

/pgfplots/**xticklabel**={⟨*command*⟩}
/pgfplots/**yticklabel**={⟨*command*⟩}
/pgfplots/**zticklabel**={⟨*command*⟩}

These keys change the TEX-command which creates the tick *labels* assigned to each tick position (see options `xtick` and `ytick`).

This is one of the two options to assign tick labels directly. The other option is '`xticklabels`={⟨*label list*⟩}' (or `yticklabels`={⟨*label list*⟩}). The option '`xticklabel`' offers higher flexibility while '`xticklabels`' is easier to use.

The argument {⟨*command*⟩} can be any TEX-text. The following commands are valid inside of {⟨*command*⟩}:

`\tick` The current element of option `xtick` (or `ytick`).

`\ticknum` The current tick number, starting with 0 (it is a macro containing a number).

`\nexttick` This command is only valid in case if the `x tick label as interval` option is set (or the corresponding variable for y). It will contain the position of the next tick position, that means the right boundary of the tick interval.

The default argument is

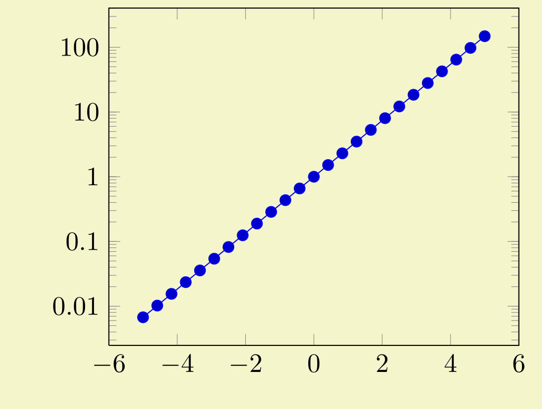
- `\axisdefaultticklabel` for normal plots:

```
\def\axisdefaultticklabel{$\pgfmathprintnumber{\tick}$}
```

- `\axisdefaultticklabellog` for logplots:

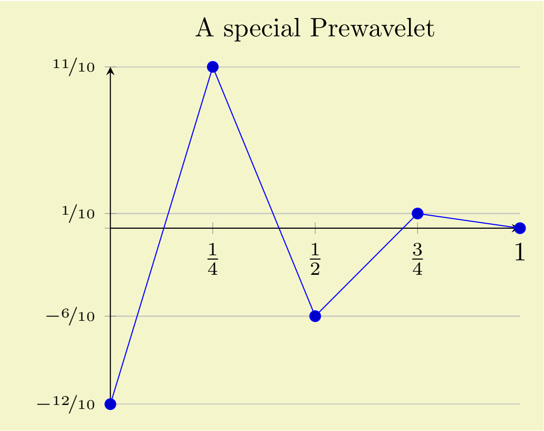
```
\def\axisdefaultticklabellog{%
    \pgfkeysgetvalue{/pgfplots/log number format code/.@cmd}\pgfplots@log@label@style
    \expandafter\pgfplots@log@label@style\tick\pgfeov
}
```

That means you can configure the appearance of linear axis with the number formatting options described in section 4.12 and logarithmic axis with `log number format code`, see below.



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{semilogyaxis}[
        yticklabel style={/pgf/number format/fixed},
        %  changes tick labels to a number instead
        %  of exponential notation:
        yticklabel={%
            \pgfmathfloatparsenumber{\tick}%
            \pgfmathfloatexp{\pgfmathresult}%
            \pgfmathprintnumber{\pgfmathresult}%
        },
    ]
        \addplot {exp(x)};
    \end{semilogyaxis}
\end{tikzpicture}
```

The following example uses explicitly formatted x tick labels and a small TEX script to format y tick labels in the form $\langle \textit{sign}\rangle\langle \textit{number}\rangle$`/10`.



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
%  \usepackage{nicefrace}% required
\begin{tikzpicture}
\begin{axis}[
    %  x ticks explicitly formatted:
    xtick={0,1,0.5,0.25,0.75},
    xticklabels={$0$,$1$,$\frac12$,$\frac14$,$\frac34$},
    %  y ticks automatically by some code fragment:
    ytick=data,
    yticklabel={%
        \scriptsize
        \ifdim\tick pt<0pt %  a TeX \if -- see TeX Book
            \pgfmathparse{-10*\tick}%
            $-\nicefrac{\pgfmathprintnumber{\pgfmathresult}}{10}$%
        \else
            \ifdim\tick pt=0pt
            \else
                \pgfmathparse{10*\tick}%
                $\nicefrac{\pgfmathprintnumber{\pgfmathresult}}{10}$%
            \fi
        \fi
    },
    ymajorgrids,
    title=A special Prewavelet,
    axis x line=center,
    axis y line=left,
    ]
    \addplot coordinates {(0,-1.2) (0.25,1.1)
        (0.5,-0.6) (0.75,0.1) (1,0)};
\end{axis}
\end{tikzpicture}
```

The TEX script takes the `\tick` macro as input and applies some logic. The `\ifdim\tick pt<0pt` means "if dimension `\tick pt` < `0pt`". The `\ifdim` is TEX's only way to compare real fixed point numbers and the author did not want to invoke `\pgfmath` for this simple task. Since `\ifdim` expects a dimension, we have to use the `pt` suffix which is compatible with `\pgfmath`. The result is that negative numbers, zero and positive numbers are typeset differently.

You can change the appearance of tick labels with

```
\pgfplotsset{tick label style={
    font=\tiny,
    /pgf/number format/sci}}%  this modifies the ‘every tick label’ style
```

and/or

```
\pgfplotsset{x tick label style={
    above,
    /pgf/number format/fixed zerofill}}%  this modifies the ‘every x tick label’ style
```

and

```
\pgfplotsset{y tick label style={font=\bfseries}}%  modifies ‘every y tick label’
```

/pgfplots/**xticklabels from table**={⟨\table or filename⟩}{⟨*colname*⟩}
/pgfplots/**yticklabels from table**={⟨\table or filename⟩}{⟨*colname*⟩}
/pgfplots/**zticklabels from table**={⟨\table or filename⟩}{⟨*colname*⟩}

A variant of `xticklabels={⟨list⟩}` which uses each entry in the column named ⟨*colname*⟩ from a table as tick labels.

The first argument ⟨\table or filename⟩ can be either a loaded table macro (i.e. the result of `\pgfplotstableread{⟨file name⟩}{⟨\table⟩}`) or just a file name.

The second argument can be a column name, a column alias or a `create on use` specification (see PgfplotsTable for the latter two). Furthermore, it can be `[index]`⟨*integer*⟩ in which case ⟨*integer*⟩ is a column index.

The behavior of `xticklabels from table` is the same as if the column ⟨*colname*⟩ would have been provided as comma separated list to `xticklabels`. This means the column can contain text, TEX macros or even math mode.

If you have white spaces in your cells, enclose the complete cell in curly braces, `{example cell}`. The detailed input format for tables is discussed in `\addplot table` and in the documentation for PgfplotsTable.

`/pgfplots/extra x tick label={⟨TEX code⟩}`
`/pgfplots/extra y tick label={⟨TEX code⟩}`
`/pgfplots/extra z tick label={⟨TEX code⟩}`

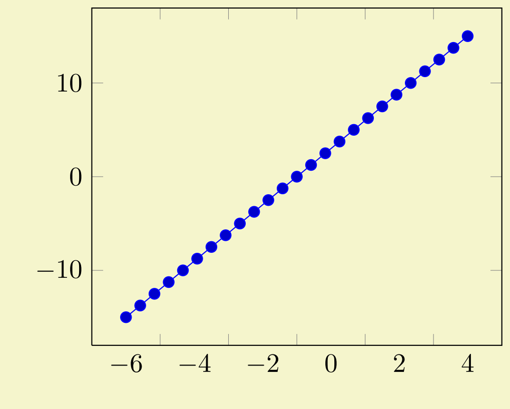
As `xticklabel` provides code to generate tick labels for each `xtick`, the key `extra x tick label` provides code to generate tick labels for every element in `extra x ticks`.

`/pgfplots/extra x tick labels={⟨label list⟩}`
`/pgfplots/extra y tick labels={⟨label list⟩}`
`/pgfplots/extra z tick labels={⟨label list⟩}`

As `xticklabels` provides explicit tick labels for each `xtick`, the key `extra x tick labels` provides explicit tick labels for every element in `extra x ticks`.

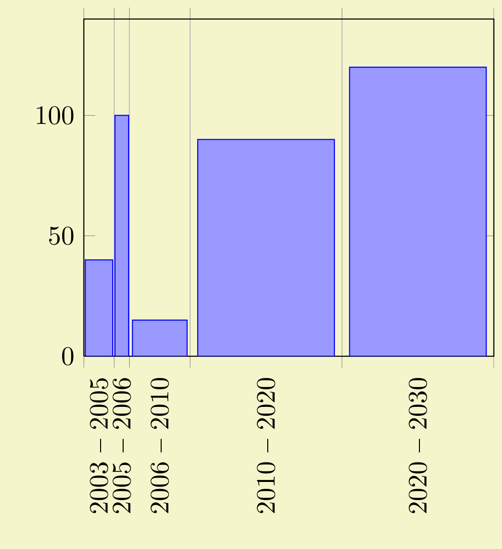
`/pgfplots/x tick label as interval=true|false` (initially `false`)
`/pgfplots/y tick label as interval=true|false` (initially `false`)
`/pgfplots/z tick label as interval=true|false` (initially `false`)

Allows to treat tick labels as intervals; that means the tick positions denote the interval boundaries. If there are n positions, $(n-1)$ tick labels will be generated, one for each interval.



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}[x tick label as interval]
    \addplot {3*x};
\end{axis}
\end{tikzpicture}
```

This mode enables the use of `\nexttick` inside of `xticklabel` (or `yticklabel`). A common application might be a bar plot.



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}[
    ybar interval=0.9,
    x tick label as interval,
    xmin=2003,xmax=2030,
    ymin=0,ymax=140,
    xticklabel={
       $\pgfmathprintnumber{\tick}$
    -- $\pgfmathprintnumber{\nexttick}$},
    xtick=data,
    x tick label style={
        rotate=90,anchor=east,
        /pgf/number format/1000 sep=}
]

    \addplot[draw=blue,fill=blue!40!white]
        coordinates
        {(2003,40) (2005,100) (2006,15)
         (2010,90) (2020,120) (2030,3)};
\end{axis}
\end{tikzpicture}
```

`/pgfplots/xminorticks=true|false` (initially `true`)

/pgfplots/yminorticks=true|false (initially true)
/pgfplots/zminorticks=true|false (initially true)
/pgfplots/xmajorticks=true|false (initially true)
/pgfplots/ymajorticks=true|false (initially true)
/pgfplots/zmajorticks=true|false (initially true)
/pgfplots/ticks=minor|major|both|none (initially both)

Enables/disables the small tick lines either for single axis or for all of them. Major ticks are those placed at the tick positions and minor ticks are between tick positions. Please note that minor ticks are automatically disabled if xtick is not a uniform range[31].

The key minor tick length={⟨*dimen*⟩} configures the tick length for minor ticks while the major variant applies to major ticks. You can configure the appearance using the following styles:

```
\pgfplotsset{every tick/.append style={color=black}} %  applies to major and minor ticks,
\pgfplotsset{every minor tick/.append style={thin}}  %  applies only to minor ticks,
\pgfplotsset{every major tick/.append style={thick}} %  applies only to major ticks.
```

There is also the style "every tick" which applies to both, major and minor ticks.

/pgfplots/xtickmin={⟨*coord*⟩}
/pgfplots/ytickmin={⟨*coord*⟩}
/pgfplots/ztickmin={⟨*coord*⟩}
/pgfplots/xtickmax={⟨*coord*⟩}
/pgfplots/ytickmax={⟨*coord*⟩}
/pgfplots/ztickmax={⟨*coord*⟩}

These keys can be used to modify minimum/maximum values before ticks are drawn. Because this applies to axis discontinuities, it is described on page 125 under section 4.8.9, "Axis Discontinuities"'.

4.14.2 Tick Alignment: Positions and Shifts

/pgfplots/xtick pos=left|right|both (initially both)
/pgfplots/ytick pos=left|right|both (initially both)
/pgfplots/ztick pos=left|right|both (initially both)
/pgfplots/tick pos=left|right|both

Allows to choose where to place the small tick lines. In the default configuration, this does also affect tick *labels*, see below. The tick pos style sets all of them to the same value (aliased by tickpos). This option is only useful for boxed axes.

For x, the additional choices bottom and top can be used which are equivalent to left and right, respectively. Both are accepted for y.

Changing tick pos will also affect the placement of tick labels.

Note that it can also affect axis lines key although not all combinations make sense. Make sure the settings are consistent.

/pgfplots/xticklabel pos=left|right|default (initially default)
/pgfplots/yticklabel pos=left|right|default (initially default)
/pgfplots/zticklabel pos=left|right|default (initially default)
/pgfplots/ticklabel pos=left|right|default (initially default)

Allows to choose where to place tick *labels*. The choices left and right place tick labels either at the left or at the right side of the complete axis. The choice default uses the same setting as xtick pos (or ytick pos). This option is only useful for boxed axes – keep it to default for non-boxed figures. The ticklabel pos style sets all three of them to the same value.

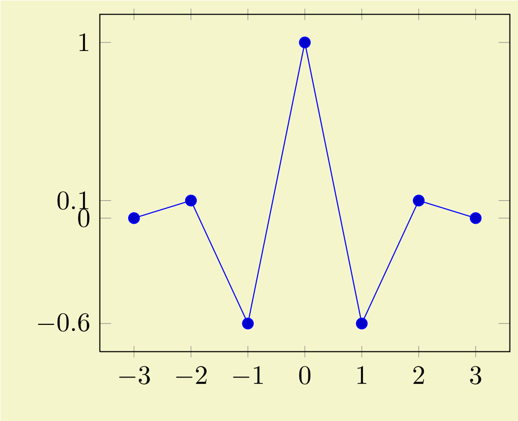
For x, the additional choices bottom and top can be used which are equivalent to left and right, respectively. Both are accepted for x.

/pgfplots/xtick align=inside|center|outside (initially inside)
/pgfplots/ytick align=inside|center|outside (initially inside)
/pgfplots/ztick align=inside|center|outside (initially inside)

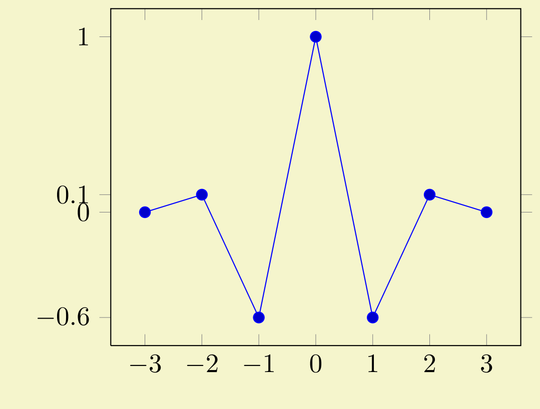
[31] A uniform list means the difference between all elements is the same for linear axis or, for logarithmic axes, log(10).

/pgfplots/tick align=inside|center|outside (initially inside)

Allows to change the location of the ticks relative to the axis lines. The tick align sets all of them to the same value. Default is “inside”.

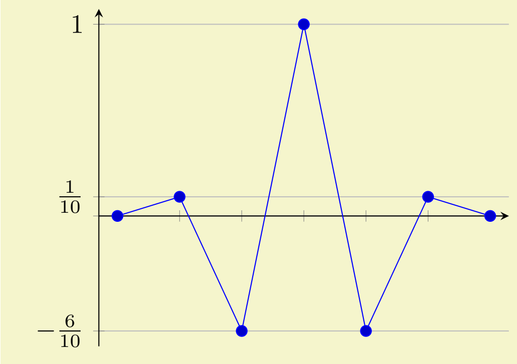


```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}[
    xtick=data,ytick=data,
    xtick align=center]
\addplot coordinates
    {(-3,0) (-2,0.1) (-1,-0.6)
     (0,1)
     (1,-0.6) (2,0.1) (3,0)};
\end{axis}
\end{tikzpicture}
```



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}[
    xtick=data,ytick=data,
    ytick align=outside]
\addplot coordinates
    {(-3,0) (-2,0.1) (-1,-0.6)
     (0,1)
     (1,-0.6) (2,0.1) (3,0)};
\end{axis}
\end{tikzpicture}
```

These tick alignment options are set automatically by the axis x line and axis y line methods (unless one appends an asterisk ‘*’):



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}[
    xtick=data,
    axis x line=center,
    xticklabels={,,},
    ytick={-0.6,0,0.1,1},
    yticklabels={
        $-\frac{6}{10}$,,
        $\frac{1}{10}$,$1$},
    ymajorgrids,
    axis y line=left,
    enlargelimits=0.05]
\addplot coordinates
    {(-3,0) (-2,0.1) (-1,-0.6)
     (0,1)
     (1,-0.6) (2,0.1) (3,0)};
\end{axis}
\end{tikzpicture}
```

/pgfplots/xticklabel shift={⟨*dimension*⟩} (initially empty)
/pgfplots/yticklabel shift={⟨*dimension*⟩} (initially empty)
/pgfplots/zticklabel shift={⟨*dimension*⟩} (initially empty)
/pgfplots/ticklabel shift={⟨*dimension*⟩} (initially empty)

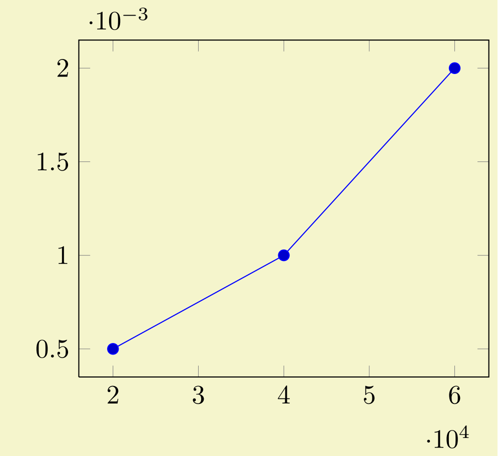
Shifts tick labels in direction of the outer unit normal of the axis by an amount of {⟨*dimension*⟩}. The ticklabel shift sets the same value for all axes.

This is usually unnecessary as the anchor of a tick label already yields enough spacing in most cases.

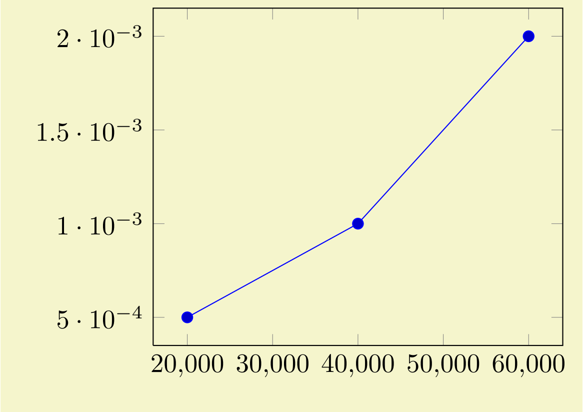
4.14.3 Tick Scaling - Common Factors In Ticks

/pgfplots/scaled ticks=true|false|base 10:⟨*e*⟩|real:⟨*num*⟩|manual:{⟨*label*⟩}{⟨*code*⟩} (initially true)
/pgfplots/scaled x ticks=⟨*one of the values*⟩ (initially true)
/pgfplots/scaled y ticks=⟨*one of the values*⟩ (initially true)
/pgfplots/scaled z ticks=⟨*one of the values*⟩ (initially true)

Allows to factor out common exponents in tick labels for *linear axes*. For example, if you have tick labels $20000, 40000$ and 60000, you may want to save some space and write $2, 4, 6$ with a separate factor '$\cdot 10^4$'. Use 'scaled ticks=true' to enable this feature. In case true, tick scaling will be triggered if the data range is either too large or too small (see below).



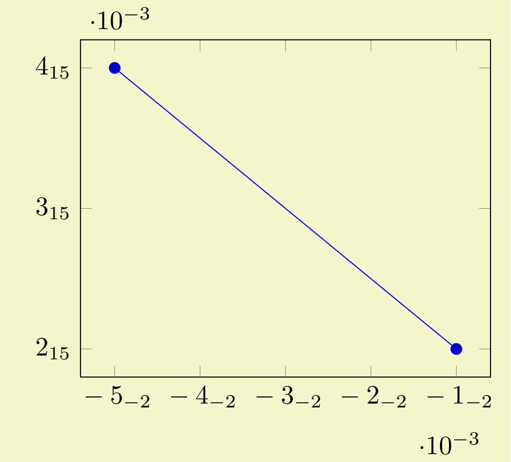
```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}[scaled ticks=true]
    \addplot coordinates {
        (20000,0.0005)
        (40000,0.0010)
        (60000,0.0020)
    };
\end{axis}
\end{tikzpicture}%
```



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}[scaled ticks=false]
    \addplot coordinates {
        (20000,0.0005)
        (40000,0.0010)
        (60000,0.0020)
    };
\end{axis}
\end{tikzpicture}
```

The scaled ticks key is a style which simply sets scaled ticks for both, x and y.

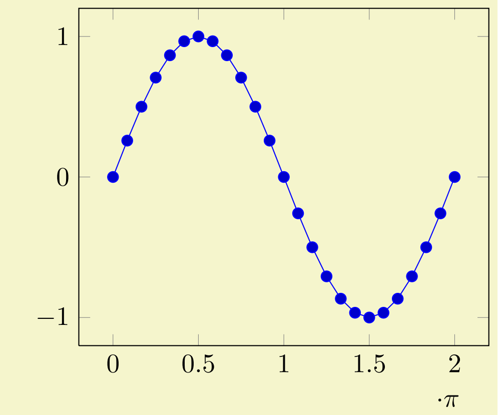
The value base 10:⟨*e*⟩ allows to adjust the algorithm manually. For example, base 10:3 will divide every tick label by 10^3:



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{axis}[scaled ticks=base 10:3,
        /pgf/number format/sci subscript]
    \addplot coordinates
        {(-0.00001,2e12) (-0.00005,4e12) };
    \end{axis}
\end{tikzpicture}
```

Here, the `sci subscript` option simply saves space. In general, `base 10:`e will divide every tick by 10^e. The effect is not limited by the "too large or too small" decisions mentioned above.

The value `real:`$\langle num \rangle$ allows to divide every tick by a fixed $\langle num \rangle$. For example, the following plot is physically ranged from 0 to 2π, but the tick scaling algorithm is configured to divide every tick label by π.



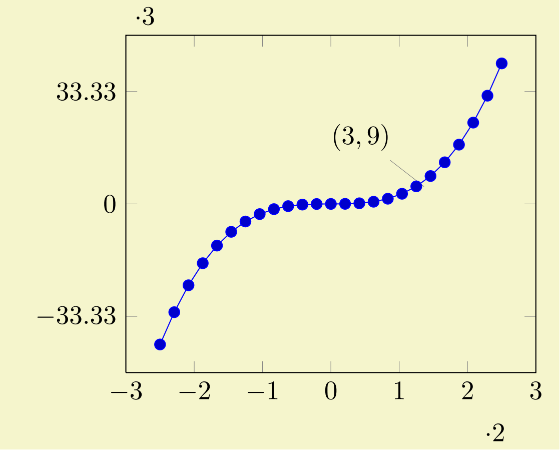
```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{axis}[
        xtick={0,1.5708,...,10},
        domain=0:2*pi,
        scaled x ticks={real:3.1415},
        xtick scale label code/.code={$\cdot \pi$}]
    \addplot {sin(deg(x))};
    \end{axis}
\end{tikzpicture}
```

Setting `scaled ticks=real:`$\langle num \rangle$ also changes the `tick scale label code` to

```
\pgfkeys{/pgfplots/xtick scale label code/.code=
    {$\pgfkeysvalueof{/pgfplots/tick scale binop} \pgfmathprintnumber{#1}$}}.
```

The key `tick scale binop` is described below, it is set initially to `\cdot`.

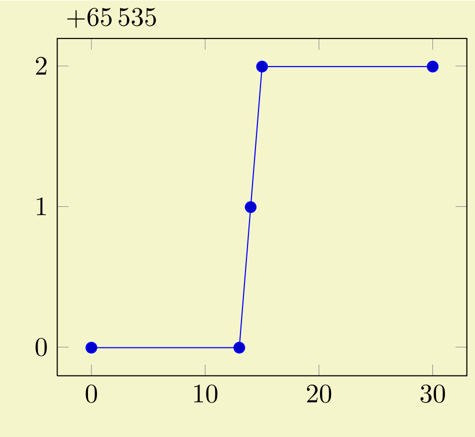
A further – not very useful – example is shown below. Every x tick label has been divided by 2, every y tick label by 3.



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{axis}[
        scaled x ticks=real:2,
        scaled y ticks=real:3]
    \addplot {x^3};
    \node[pin=135:{$(3,9)$}] at (axis cs:3,9) {};
    \end{axis}
\end{tikzpicture}
```

Unfortunately, $\langle num \rangle$ can't be evaluated with PGF's math parser (yet) to maintain the full data range accepted by PGFPLOTS.

The last option, `scaled ticks=manual:{`$\langle label \rangle$`}{`$\langle code \rangle$`}` allows even more customization. It allows *full control* over the displayed scaling label *and* the scaling code: `{`$\langle text \rangle$`}` is used as-is inside of the tick scaling label while `{`$\langle code \rangle$`}` is supposed to be a one-argument-macro which scales each tick. Example:



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}[
    %  warning: the '%' signs are necessary (?)
    scaled y ticks=manual:{$+65\,535$}{%
        \pgfmathfloatcreate{1}{6.5535}{4}%
        \pgfmathfloatsubtract{#1}{\pgfmathresult}%
    },
    yticklabel style={
        /pgf/number format/fixed,
        /pgf/number format/precision=1},
]
\addplot coordinates {
    (0, 65535)
    (13, 65535)
    (14, 65536)
    (15, 65537)
    (30, 65537)
};
\end{axis}
\end{tikzpicture}
```

The example uses `$+65\,535$` as tick scale label content. Furthermore, it defines the customized tick label formula $y - (+6.5535 \cdot 10^4) = y - 65535$ to generate y tick labels.

The {⟨*text*⟩} can be arbitrary. It is completely in user control. The second argument, {⟨*code*⟩} is supposed to be a one-argument-macro in which #1 is the current tick position in floating point representation. The macro is expected to assign `\pgfmathresult` (also in floating point representation). The PGF manual [5] contains detailed documentation about its math engine (including floating point[32]).

This feature may also be used do transform coordinates in case they can't be processed with PGFPLOTS: transform them and supply a proper tick scaling method such that tick labels represent the original range.

If {⟨*text*⟩} is empty, the tick scale label won't be drawn (and no space will be occupied).

Tick scaling does *not* work for logarithmic axes.

/pgfplots/xtick scale label code/.code={⟨...⟩}
/pgfplots/ytick scale label code/.code={⟨...⟩}
/pgfplots/ztick scale label code/.code={⟨...⟩}

Allows to change the default code for scaled tick labels. The default is

```
\pgfplotsset{
    xtick scale label code/.code={$\cdot 10^{#1}$}
}
```

More precisely, it is

```
\pgfplotsset{
    xtick scale label code/.code={$\pgfkeysvalueof{/pgfplots/tick scale binop} 10^{#1}$}
}
```

and the initial value of `tick scale binop` is `\cdot`, but it can be changed to `\times` if desired.

If the code is empty, no tick scale label will be drawn (and no space is consumed).

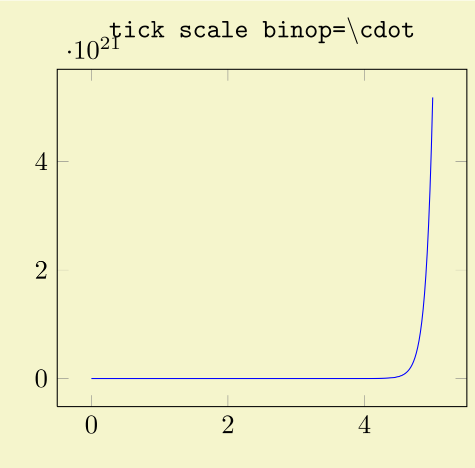
/pgfplots/tick scale label code/.code={⟨...⟩}

A style which sets `xtick scale label code` and those for y and z.

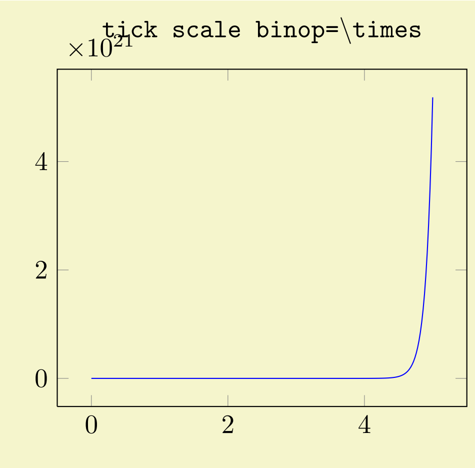
/pgfplots/tick scale binop={⟨*TeX math operator*⟩} (initially `\cdot`)

Sets the binary operator used to display tick scale labels.

[32] However, that particular stuff is newer than PGF 2.00. At the time of this writing, it is only available as (public) CVS version.



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}[
    title=\texttt{tick scale
        binop=\textbackslash cdot}]
\addplot
    [mark=none,blue,samples=250,
     domain=0:5]
    {exp(10*x)};
\end{axis}
\end{tikzpicture}
```



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}[
    title=\texttt{tick scale
        binop=\textbackslash times},
    tick scale binop=\times]
\addplot
    [mark=none,blue,samples=250,
     domain=0:5]
    {exp(10*x)};
\end{axis}
\end{tikzpicture}
```

/pgfplots/scale ticks below={⟨*exponent*⟩}

Allows fine tuning of the 'scaled ticks' algorithm: if the axis limits are of magnitude 10^e and $e <${⟨*exponent*⟩}, the common prefactor 10^e will be factored out. The default is -1.

/pgfplots/scale ticks above={⟨*exponent*⟩}

Allows fine tuning of the 'scaled ticks' algorithm: if the axis limits are of magnitude 10^e and $e >${⟨*exponent*⟩}, the common prefactor 10^e will be factored out. The default is 3.

4.14.4 Tick Fine Tuning

The tick placement algorithm depends on a number of parameters which can be tuned to get better results.

/pgfplots/max space between ticks={⟨*number*⟩} (initially 35)

Configures the maximum space between adjacent ticks in full points. The suffix "pt" has to be omitted and fractional numbers are not supported. The default is 35.

/pgfplots/try min ticks={⟨*number*⟩} (initially 3)

Configures a loose lower bound on the number of ticks. It should be considered as a suggestion, not a tight limit. The default is 4. This number will increase the number of ticks if 'max space between ticks' produces too few of them.

The total number of ticks may still vary because not all fractional numbers in the axis' range are valid tick positions.

/pgfplots/try min ticks log={⟨*number*⟩} (initially 3)

The same as try min ticks, but for logarithmic axis.

/pgfplots/tickwidth={⟨*dimension*⟩} (initially 0.15cm)

/pgfplots/major tick length={⟨*dimension*⟩} (initially 0.15cm)

Sets the width of major tick lines.

/pgfplots/subtickwidth={⟨*dimension*⟩} (initially 0.1cm)
/pgfplots/minor tick length={⟨*dimension*⟩} (initially 0.1cm)

Sets the width of minor tick lines.

/pgfplots/xtick placement tolerance (initially 0.05pt)
/pgfplots/ytick placement tolerance (initially 0.05pt)
/pgfplots/ztick placement tolerance (initially 0.05pt)

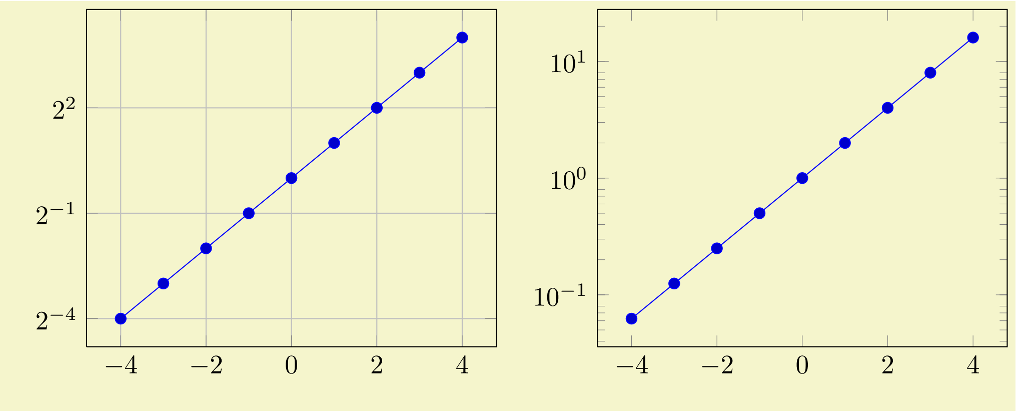
Tick lines and labels will be placed if they are no more than this tolerance beyond the axis limits. This threshold should be chosen such that it does not produce visible differences while still providing fault tolerance.

The threshold is given in paper units of the final figure.

/pgfplots/log basis x={⟨*number*⟩} (initially empty)
/pgfplots/log basis y={⟨*number*⟩} (initially empty)
/pgfplots/log basis z={⟨*number*⟩} (initially empty)

Allows to change the logarithms used for logarithmic axes.

Changing to a different log basis is nothing but a scale. However, it also changes the way tick labels are displayed: they will also be shown in the new basis.



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{semilogyaxis}[log basis y=2,grid=major,samples at={-4,...,4}]
        \addplot {2^x};
    \end{semilogyaxis}
\end{tikzpicture}
~
\begin{tikzpicture}
    \begin{semilogyaxis}[log basis y=10,samples at={-4,...,4}]
        \addplot {2^x};
    \end{semilogyaxis}
\end{tikzpicture}
```

The initial setting is '`log basis x=`' which defaults to: the natural logarithm for any coordinates (basis exp(1)), and the logarithm base 10 for the display of tick labels.

If the log basis is changed to something different than the empty string, the chosen logarithm will be applied to any input coordinate (if the axis scale is log as well) and tick labels will be displayed in this basis.

In other words: usually, you see log axes base 10 and that's it. It is only interesting for coordinate filters: the initial setting (with empty ⟨*number*⟩) uses coordinate lists basis *e* although the display will use basis 10 (i.e. it is rescaled). Any non-empty value ⟨*number*⟩ causes both, coordinate lists *and* display to use ⟨*number*⟩ as basis for the logarithm. The javascript code of the `clickable` library will always use the *display* basis (which is usally 10) when it computes slopes.

Technical remarks. When `log basis x` is used, the style `log basis ticks={`⟨*axis char*⟩`}` will be installed (in this case `log basis ticks=x`). This style in turn will change `log number format code`.

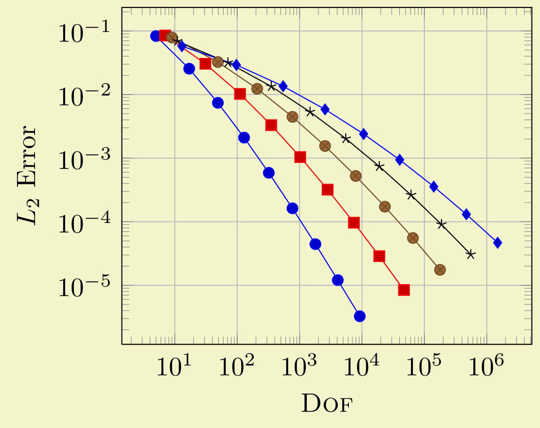
Please note that `xtickten` will be used differently now: it will provide the desired ticks in the new basis! Despite the misleading name "`ten`", `xtickten={1,2,3,4}` will yield ticks at $2^1, 2^2, 2^3, 2^4$ if `log basis x=2` has been set.

4.15 Grid Options

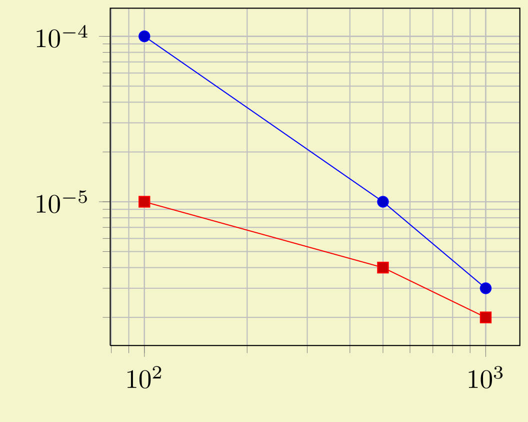
`/pgfplots/xminorgrids=true|false` (initially `false`)
`/pgfplots/yminorgrids=true|false` (initially `false`)
`/pgfplots/zminorgrids=true|false` (initially `false`)
`/pgfplots/xmajorgrids=true|false` (initially `false`)
`/pgfplots/ymajorgrids=true|false` (initially `false`)
`/pgfplots/zmajorgrids=true|false` (initially `false`)
`/pgfplots/grid=minor|major|both|none` (initially `false`)

Enables/disables different grid lines. Major grid lines are placed at the normal tick positions (see `xmajorticks`) while minor grid lines are placed at minor ticks (see `xminorticks`).

This example employs the coordinates defined on page 14.



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{loglogaxis}[
    xlabel={\textsc{Dof}},
    ylabel={$L_2$ Error},
    grid=major
]
%  see above for this macro:
\plotcoords
\end{loglogaxis}
\end{tikzpicture}
```



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{loglogaxis}[
    grid=both,
    tick align=outside,
    tickpos=left]
\addplot coordinates
    {(100,1e-4) (500,1e-5) (1000,3e-6)};
\addplot coordinates
    {(100,1e-5) (500,4e-6) (1000,2e-6)};
\end{loglogaxis}
\end{tikzpicture}
```

Grid lines will be drawn before tick lines are processed, so ticks will be drawn on top of grid lines. You can configure the appearance of grid lines with the styles

```
\pgfplotsset{grid style={help lines}} %  modifies the style 'every axis grid'
\pgfplotsset{minor grid style={color=blue}} %  modifies the style 'every minor grid'
\pgfplotsset{major grid style={thick}} % modifies the style 'every major grid'
```

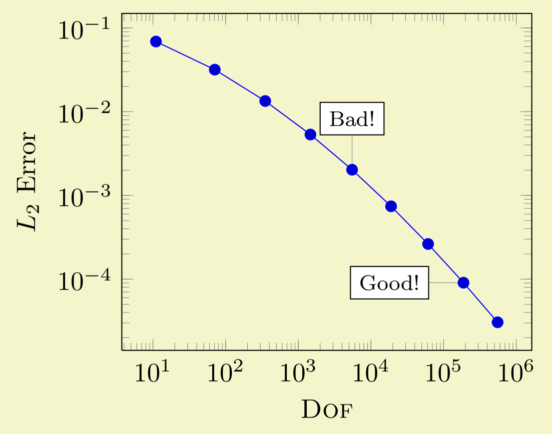
4.16 Accessing Axis Coordinates for Annotations

Coordinate system `axis cs`

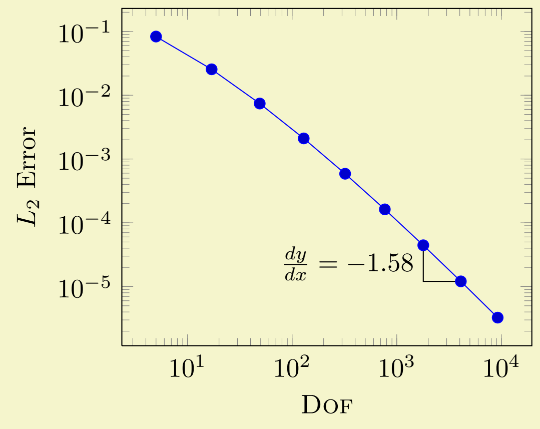
PGFPLOTS provides a new coordinate system for use inside of an axis, the "axis coordinate system", axis cs.

It can be used to draw any Ti*k*Z-graphics at axis coordinates. It is used like

```
\draw
   (axis cs:18943,2.873391e-05)
|- (axis cs:47103,8.437499e-06);
```



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\tikzstyle{every pin}=[fill=white,
    draw=black,
    font=\footnotesize]
\begin{tikzpicture}
    \begin{loglogaxis}[
        xlabel={\textsc{Dof}},
        ylabel={$L_2$ Error}]

    \addplot coordinates {
        (11,     6.887e-02)
        (71,     3.177e-02)
        (351,    1.341e-02)
        (1471,   5.334e-03)
        (5503,   2.027e-03)
        (18943,  7.415e-04)
        (61183,  2.628e-04)
        (187903, 9.063e-05)
        (553983, 3.053e-05)
    };

    \node[coordinate,pin=above:{Bad!}]
        at (axis cs:5503,2.027e-03) {};
    \node[coordinate,pin=left:{Good!}]
        at (axis cs:187903,9.063e-05)    {};
    \end{loglogaxis}
\end{tikzpicture}
```



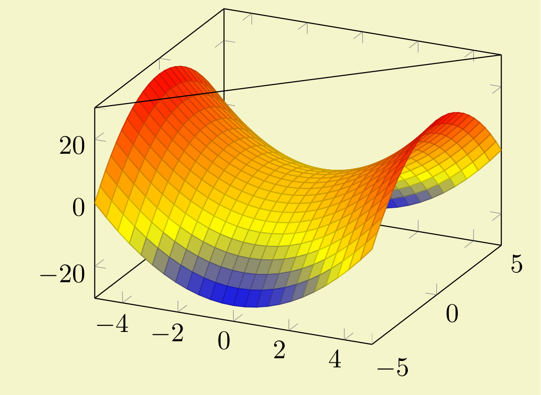
```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{loglogaxis}[
    xlabel=\textsc{Dof},
    ylabel=$L_2$ Error
]
\draw
        (axis cs:1793,4.442e-05)
    |-  (axis cs:4097,1.207e-05)
    node[near start,left]
    {$\frac{dy}{dx} = -1.58$};

\addplot coordinates {
    (5,    8.312e-02)
    (17,   2.547e-02)
    (49,   7.407e-03)
    (129,  2.102e-03)
    (321,  5.874e-04)
    (769,  1.623e-04)
    (1793, 4.442e-05)
    (4097, 1.207e-05)
    (9217, 3.261e-06)
};
\end{loglogaxis}
\end{tikzpicture}
```

Attention: Whenever you draw additional graphics, consider using axis cs! It applies any logarithms, data scaling transformations or whatever PGFPLOTS usually does!

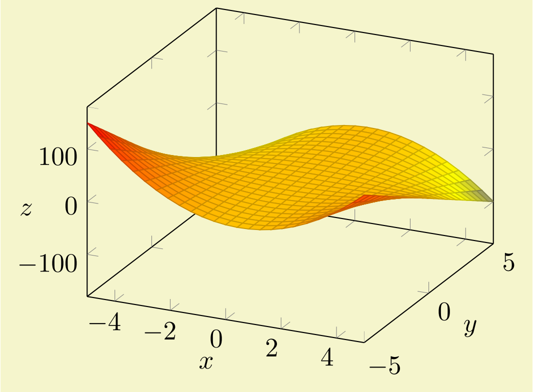
There is also a low–level interface to access the transformations and coordinates, see section 8 on page 224.

Coordinate system rel axis cs

The "relative axis coordinate system", rel axis cs, uses the complete axis vectors as units. That means '$x = 0$' denotes the point on the lower x axis range and '$x = 1$' the point on the upper x axis range (see the remark below for x dir=reverse).



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}

    \addplot3[surf] {x^2 - y^2};
    \draw  (rel axis cs:0,0,1)
        -- (rel axis cs:1,1,1);
\end{axis}
\end{tikzpicture}
```



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}[
    xlabel=$x$,
    ylabel=$y$,
    zlabel=$z$,
    every axis x label/.style={
        at={(rel axis cs:0.5,-0.15,-0.15)}},
    every axis y label/.style={
        at={(rel axis cs:1.15,0.5,-0.15)}},
    every axis z label/.style={
        at={(rel axis cs:-0.15,-0.15,0.5)}},
]

    \addplot3[surf] {x*(1-x)*y};
\end{axis}
\end{tikzpicture}
```

Points identified by rel axis cs use the syntax

(rel axis cs:⟨x⟩,⟨y⟩) or

(rel axis cs:⟨x⟩,⟨y⟩,⟨z⟩)

where ⟨x⟩, ⟨y⟩ and ⟨z⟩ are coordinates or constant mathematical expressions. The second syntax is only available in three dimensional axes.

There is one speciality: if you reverse an axis (with x dir=reverse), points provided by rel axis cs will be *unaffected* by the axis reversal. This is intended to provide consistent placement even for reversed axes. Use allow reversal of rel axis cs=false to disable this feature.

There is also a low–level interface to access the transformations and coordinates, see section 8 on page 224.

Predefined node current plot begin

This coordinate will be defined for every plot and can be used is ⟨*trailing path commands*⟩ or after a plot. It is the first coordinate of the current plot.

Predefined node current plot end

This coordinate will be defined for every plot. It is the last coordinate of the current plot.

/pgfplots/allow reversal of rel axis cs=true|false (initially true)

A fine tuning key which specifies how to deal with x dir=reverse and rel axis cs and ticklabel cs.

The initial configuration true means that points placed with rel axis cs and/or ticklabel cs will be at the same position inside of the axes even if its ordering has been reversed. The choice false will disable the special treatment of x dir=reverse.

4.17 Style Options

4.17.1 All Supported Styles

PGFPLOTS provides many styles to customize its appearance and behavior. They can be defined and changed in any place where keys are allowed. Furthermore, own styles are defined easily.

Key handler ⟨*key*⟩`/.style={`⟨*key-value-list*⟩`}`

Defines or redefines a style ⟨*key*⟩. A style is a normal key which will set all options in `{`⟨*key-value-list*⟩`}` when it is set.
Use `\pgfplotsset{`⟨*key*⟩`/.style={`⟨*key-value-list*⟩`}}` to (re-) define a style ⟨*key*⟩ in the namespace `/pgfplots`.

Key handler ⟨*key*⟩`/.append style={`⟨*key-value-list*⟩`}`

Appends `{`⟨*key-value-list*⟩`}` to an already existing style ⟨*key*⟩. This is the preferred method to change the predefined styles: if you only append, you maintain compatibility with future versions.
Use `\pgfplotsset{`⟨*key*⟩`/.append style={`⟨*key-value-list*⟩`}}` to append `{`⟨*key-value-list*⟩`}` to the style ⟨*key*⟩. This will assume the prefix `/pgfplots`.

Styles installed for linear/logarithmic axis

`/pgfplots/every axis` (style, initially `empty`)

Installed at the beginning of every axis. Ti*k*Z options inside of it will be used for anything inside of the axis rectangle and any axis descriptions.

`/pgfplots/every semilogx axis` (style, initially `empty`)

Installed at the beginning of every plot with linear x axis and logarithmic y axis, but after '`every axis`'.

`/pgfplots/every semilogy axis` (style, initially `empty`)

Likewise, but with interchanged roles for x and y.

`/pgfplots/every loglog axis` (style, initially `empty`)

Installed at the beginning of every double–logarithmic plot.

`/pgfplots/every linear axis` (style, initially `empty`)

Installed at the beginning of every plot with normal axis scaling.

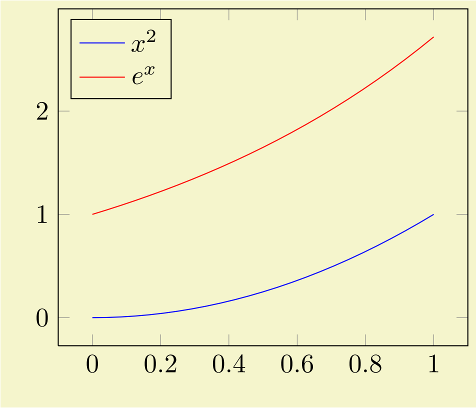
Styles installed for single plots

`/pgfplots/every axis plot` (style, initially `empty`)

Installed for each plot. This style may contain options like samples, gnuplot parameters, error bars and it may contain options which affect the final drawing commands.

`/pgfplots/every axis plot post` (style, initially `empty`)

This style is similar to `every axis plot` in that is applies to any drawing command in `\addplot`. However, it is set *after* any user defined styles or `cycle list` options.



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\pgfplotsset{
    every axis plot post/.append style=
        {mark=none}}

\begin{axis}[
    legend style={
        at={(0.03,0.97)},anchor=north west},
    domain=0:1]
    \addplot {x^2};
    \addplot {exp(x)};
    \legend{$x^2$,$e^x$}
\end{axis}
\end{tikzpicture}
```

/pgfplots/every axis plot no # (style, initially empty)

Used for every #th plot where $\# = 0, 1, 2, 3, 4, \ldots$.

/pgfplots/every forget plot (style, initially empty)

Used for every plot which has forget plot activated.

/pgfplots/forget plot style={⟨*key-value-list*⟩}

An abbreviation for every forget plot/.append style={⟨*key-value-list*⟩}.

Styles for axis descriptions

/pgfplots/every axis label (style, initially empty)

Used for all axis label (like xlabel and ylabel).

/pgfplots/label style={⟨*key-value-list*⟩}

An abbreviation for every axis label/.append style={⟨*key-value-list*⟩}.

/pgfplots/every axis x label (style, no value)
/pgfplots/every axis y label (style, no value)
/pgfplots/every axis z label (style, no value)

Used only x or only for y labels, installed after 'every axis label'.

The initial settings are set by xlabel absolute and its variants (if the initial configuration compat=pre 1.3 is active) or xlabel near ticks which provides the better spacing as it incorporates the tick label sizes to compute the position.

Attention: These styles will be overwritten by axis x line and/or axis y line. Please remember to place your modifications after the axis line variations.

/pgfplots/x label style={⟨*key-value-list*⟩}
/pgfplots/y label style={⟨*key-value-list*⟩}
/pgfplots/z label style={⟨*key-value-list*⟩}
/pgfplots/xlabel style={⟨*key-value-list*⟩}
/pgfplots/ylabel style={⟨*key-value-list*⟩}
/pgfplots/zlabel style={⟨*key-value-list*⟩}

Different abbreviations for every axis x label/.append style={⟨*key-value-list*⟩} (or the respective style for y, every axis y label/.append style=).

/pgfplots/every axis title (style, no value)

Used for any axis title. The at=(⟨*x,y*⟩) syntax will place the title using axis description cs.

The initial setting is

```
\pgfplotsset{every axis title/.style={at={(0.5,1)},above,yshift=6pt}}
```

To be more precise, the yshift doesn't use the hardcoded 6pt: it uses the value of

/pgfplots/every axis title shift={⟨*default shift*⟩} (initially 6pt)

which can be reset if needed.

/pgfplots/title style={⟨*key-value-list*⟩}

An abbreviation for every axis title/.append style={⟨*key-value-list*⟩}.

/pgfplots/every axis legend (style, no value)

Installed for each legend. As described for axis description cs, the legend's position can be placed using coordinates between 0 and 1 (it employs axis description cs automatically).

The initial setting is

```
\pgfplotsset{every axis legend/.style={
        cells={anchor=center},
        inner xsep=3pt,inner ysep=2pt,nodes={inner sep=2pt,text depth=0.15em},
        anchor=north east,
        shape=rectangle,
        fill=white,draw=black,
        at={(0.98,0.98)}}}
```

/pgfplots/legend style={⟨*key-value-list*⟩}

An abbreviation for every axis legend/.append style={⟨*key-value-list*⟩}.

/pgfplots/every legend image post (style, no value)

Allows to change the appearance of the small legend images *after* the options of the plot style have been applied. Thus, legend formatting can be changed independently of the plot style using every legend image post.

This key is also documented on page 116.

Styles for axis lines

/pgfplots/every outer x axis line (style, initially empty)
/pgfplots/every outer y axis line (style, initially empty)
/pgfplots/every outer z axis line (style, initially empty)

Installed for every axis line which lies on the outer box.

If you want arrow heads, you may also need to check the separate axis lines boolean key.

/pgfplots/every inner x axis line (style, initially empty)
/pgfplots/every inner y axis line (style, initially empty)
/pgfplots/every inner z axis line (style, initially empty)

Installed for every axis line which is drawn using the center or middle options.

/pgfplots/axis line style={⟨*key-value-list*⟩}
/pgfplots/inner axis line style={⟨*key-value-list*⟩}
/pgfplots/outer axis line style={⟨*key-value-list*⟩}
/pgfplots/x axis line style={⟨*key-value-list*⟩}
/pgfplots/y axis line style={⟨*key-value-list*⟩}
/pgfplots/z axis line style={⟨*key-value-list*⟩}

These options modify selects parts of the axis line styles. They set every inner x axis line and every outer x axis line and the respective y variants.

Please refer to section 4.8.7 on page 123 for details about styles for axis lines.

/pgfplots/every 3d box foreground (style, no value)

Installed for the parts drawn by 3d box=complete. This affects axis lines, tick lines and grid lines drawn in the *foreground*. The background drawing operations have already been done when this style is evaluated.

/pgfplots/3d box foreground style={⟨*key-value-list*⟩}

An abbreviation for every 3d box foreground/.append style={⟨*key-value-list*⟩}.

Styles for ticks

/pgfplots/every tick (style, initially very thin,gray)

Installed for each of the small tick *lines*.

/pgfplots/tick style={⟨*key-value-list*⟩}

An abbreviation for every tick/.append style={⟨*key-value-list*⟩}.

/pgfplots/every minor tick (style, initially empty)

Used for each minor tick line, installed after ‘every tick’.

/pgfplots/minor tick style={⟨*key-value-list*⟩}

An abbreviation for every minor tick/.append style={⟨*key-value-list*⟩}.

/pgfplots/every major tick (style, initially empty)

Used for each major tick line, installed after 'every tick'.

/pgfplots/major tick style={⟨*key-value-list*⟩}

An abbreviation for every major tick/.append style={⟨*key-value-list*⟩}.

/pgfplots/every tick label (style, initially empty)

Used for each x and y tick labels.

/pgfplots/tick label style={⟨*key-value-list*⟩}
/pgfplots/ticklabel style={⟨*key-value-list*⟩}

Different abbreviations for every tick label/.append style={⟨*key-value-list*⟩} (or the respective style for y, every tick label/.append style=).

/pgfplots/every x tick label (style, initially empty)
/pgfplots/every y tick label (style, initially empty)
/pgfplots/every z tick label (style, initially empty)

Used for each x (or y, respectively) tick label, installed after 'every tick label'.

/pgfplots/x tick label style={⟨*key-value-list*⟩}
/pgfplots/y tick label style={⟨*key-value-list*⟩}
/pgfplots/z tick label style={⟨*key-value-list*⟩}
/pgfplots/xticklabel style={⟨*key-value-list*⟩}
/pgfplots/yticklabel style={⟨*key-value-list*⟩}
/pgfplots/zticklabel style={⟨*key-value-list*⟩}

Different abbreviations for every x tick label/.append style={⟨*key-value-list*⟩} (or the respective style for y, every y tick label/.append style=).

/pgfplots/every x tick scale label (style, no value)
/pgfplots/every y tick scale label (style, no value)
/pgfplots/every z tick scale label (style, no value)

Configures placement and display of the nodes containing the order of magnitude of tick labels, see section 4.14.3 for more information about scaled ticks.

The initial settings are

```
\pgfplotsset{
    every x tick scale label/.style={at={(1,0)},yshift=-2em,left,inner sep=0pt},
    every y tick scale label/.style={at={(0,1)},above right,inner sep=0pt,yshift=0.3em}}
```

/pgfplots/x tick scale label style={⟨*key-value-list*⟩}
/pgfplots/y tick scale label style={⟨*key-value-list*⟩}
/pgfplots/z tick scale label style={⟨*key-value-list*⟩}

An abbreviation for every x tick scale label/.append style={⟨*key-value-list*⟩} (or the respective style for y, every y tick scale label/.append style=).

/pgfplots/every x tick (style, initially empty)
/pgfplots/every y tick (style, initially empty)
/pgfplots/every z tick (style, initially empty)

Installed for tick *lines* on either x or y axis.

/pgfplots/xtick style={⟨*key-value-list*⟩}
/pgfplots/ytick style={⟨*key-value-list*⟩}
/pgfplots/ztick style={⟨*key-value-list*⟩}

An abbreviation for every x tick/.append style={⟨*key-value-list*⟩} (or the respective style for y, every y tick/.append style=).

/pgfplots/every minor x tick (style, initially empty)
/pgfplots/every minor y tick (style, initially empty)
/pgfplots/every minor z tick (style, initially empty)

Installed for minor tick lines on either x or y axis.

/pgfplots/minor x tick style={⟨*key-value-list*⟩}
/pgfplots/minor y tick style={⟨*key-value-list*⟩}
/pgfplots/minor z tick style={⟨*key-value-list*⟩}

An abbreviation for every minor x tick/.append style={⟨*key-value-list*⟩} (or the respective style for y, every minor y tick/.append style=).

/pgfplots/every major x tick (style, initially empty)
/pgfplots/every major y tick (style, initially empty)
/pgfplots/every major z tick (style, initially empty)

Installed for major tick lines on either x or y axis.

/pgfplots/major x tick style={⟨*key-value-list*⟩}
/pgfplots/major y tick style={⟨*key-value-list*⟩}
/pgfplots/major z tick style={⟨*key-value-list*⟩}

An abbreviation for every major x tick/.append style={⟨*key-value-list*⟩} (or the respective style for y, every major y tick/.append style=).

/pgfplots/every extra x tick (style, no value)
/pgfplots/every extra y tick (style, no value)
/pgfplots/every extra z tick (style, no value)

Allows to configure the appearance of 'extra x ticks'. This style is installed before touching the first extra x tick. It is possible to set any option which affects tick or grid line generation.

The initial setting is

```
\pgfplotsset{
    every extra x tick/.style={/pgfplots/log identify minor tick positions=true},
    every extra y tick/.style={/pgfplots/log identify minor tick positions=true}}
```

Useful examples are shown below.

```
\pgfplotsset{every extra x tick/.append style={grid=major}}
\pgfplotsset{every extra x tick/.append style={major tick length=0pt}}
\pgfplotsset{every extra x tick/.append style={/pgf/number format=sci subscript}}
```

/pgfplots/extra x tick style={⟨*key-value-list*⟩}
/pgfplots/extra y tick style={⟨*key-value-list*⟩}
/pgfplots/extra z tick style={⟨*key-value-list*⟩}

An abbreviation for every extra x tick/.append style={⟨*key-value-list*⟩} (or the respective style for y, every extra y tick/.append style=).

/pgfplots/extra tick style={⟨*key-value-list*⟩}

An abbreviation which appends ⟨*key-value-list*⟩ to every extra x tick, every extra y tick and every extra z tick.

Styles for grid lines

/pgfplots/every axis grid (style, initially thin,black!25)

Used for each grid line.

/pgfplots/grid style={⟨*key-value-list*⟩}

An abbreviation for every axis grid/.append style={⟨*key-value-list*⟩}.

/pgfplots/every minor grid (style, initially empty)

Used for each minor grid line, installed after 'every axis grid'.

/pgfplots/minor grid style={⟨*key-value-list*⟩}

An abbreviation for every minor grid/.append style={⟨*key-value-list*⟩}.

/pgfplots/every major grid (style, initially empty)

Likewise, for major grid lines.

/pgfplots/major grid style={⟨*key-value-list*⟩}

An abbreviation for every major grid/.append style={⟨*key-value-list*⟩}.

/pgfplots/every axis x grid (style, initially empty)
/pgfplots/every axis y grid (style, initially empty)
/pgfplots/every axis z grid (style, initially empty)

Used for each grid line in either x or y direction.

/pgfplots/x grid style={⟨*key-value-list*⟩}
/pgfplots/y grid style={⟨*key-value-list*⟩}
/pgfplots/z grid style={⟨*key-value-list*⟩}

An abbreviation for every axis x grid/.append style={⟨*key-value-list*⟩} (or the respective style for y, every axis y grid/.append style=).

/pgfplots/every minor x grid (style, initially empty)
/pgfplots/every minor y grid (style, initially empty)
/pgfplots/every minor z grid (style, initially empty)

Used for each minor grid line in either x or y direction.

/pgfplots/minor x grid style={⟨*key-value-list*⟩}
/pgfplots/minor y grid style={⟨*key-value-list*⟩}
/pgfplots/minor z grid style={⟨*key-value-list*⟩}

An abbreviation for every minor x grid/.append style={⟨*key-value-list*⟩} (or the respective style for y, every minor y grid/.append style=).

/pgfplots/every major x grid (style, initially empty)
/pgfplots/every major y grid (style, initially empty)
/pgfplots/every major z grid (style, initially empty)

Used for each major grid line in either x or y direction.

/pgfplots/major x grid style={⟨*key-value-list*⟩}
/pgfplots/major y grid style={⟨*key-value-list*⟩}
/pgfplots/major z grid style={⟨*key-value-list*⟩}

An abbreviation for every major x grid/.append style={⟨*key-value-list*⟩} (or the respective style for y, every major y grid/.append style=).

Styles for error bars

/pgfplots/every error bar (style, initially thin)

Installed for every error bar.

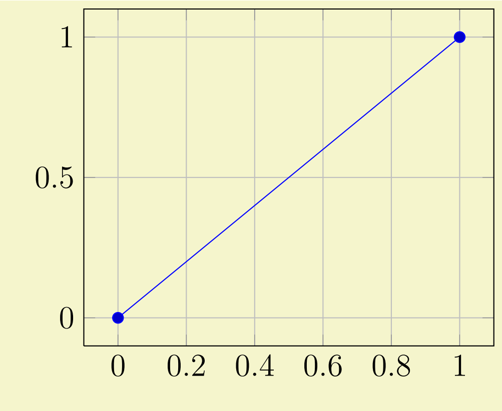
/pgfplots/error bars/error bar style={⟨*key-value-list*⟩}

An abbreviation for every error bar/.append style={⟨*key-value-list*⟩}.

4.17.2 (Re-)Defining Own Styles

Use \pgfplotsset{⟨*style name*⟩/.style={⟨*key-value-list*⟩}} to create own styles. If ⟨*style name*⟩ exists already, it will be replaced. Please note that it is *not* possible to use the Ti*k*Z-command \tikzstyle{⟨*style name*⟩}=[] in this context[33].

[33] This was possible in a previous version and is still supported for backwards compatibility. But in some cases, it may not work as expected.



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\pgfplotsset{my personal style/.style=
    {grid=major,font=\large}}

\begin{tikzpicture}
\begin{axis}[my personal style]
    \addplot coordinates {(0,0) (1,1)};
\end{axis}
\end{tikzpicture}
```

4.18 Alignment Options and Bounding Box Control

4.18.1 Basic Alignment

Alignment works with two main methods: a coordinate where the axis shall be drawn and an "anchor" inside of the axis which shall be drawn at this particular coordinate. This methodology is common for each Ti*k*Z node – and an axis is nothing but a (special) Ti*k*Z node. The coordinate can be specified using the `at` key, while the anchor can be specified with the `anchor` key. In most cases, it is sufficient to provide only an anchor – unless one needs more than one axis in the same picture environment.

/pgfplots/`at`=`{`⟨*coordinate expression*⟩`}`

Assigns a position for the complete axis image. This option works similarly to the `at`-option of `\node[at={`⟨*coordinate expression*⟩`}]`, see [5]. The common syntax is `at={(`⟨x,y⟩`)}`.

The idea is to provide an `{`⟨*coordinate expression*⟩`}` where the axis will be placed. The axis' anchor will be placed at `{`⟨*coordinate expression*⟩`}`.

/pgfplots/`anchor`=`{`⟨*name*⟩`}` (initially `south west`)

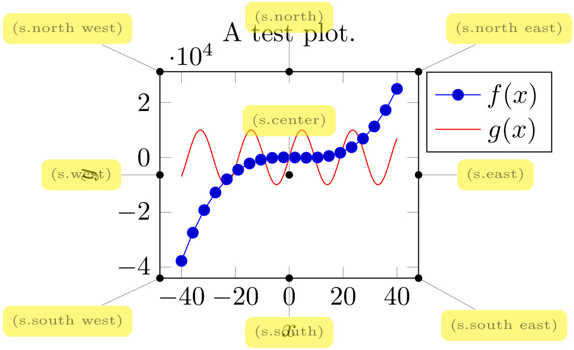
Chooses one of the different possible positions inside of an axis which is placed with `at`. The `at` key defines the position where to place the axis inside of the embedding picture, the `anchor` key defines which point of the axis shall be positioned by '`at`'. The initial configuration assumes `at={(0,0)}`. Thus, `anchor=center` will place the axis' center at the logical picture position $(0,0)$. Similarly, `anchor=south west` will position the lower left corner of the axis at $(0,0)$.

For users who are familiar with Ti*k*Z: an axis is actually a very special node, so anchors work as in [5].

Anchors are useful in conjunction with horizontal or vertical alignment of plots, see the examples below.

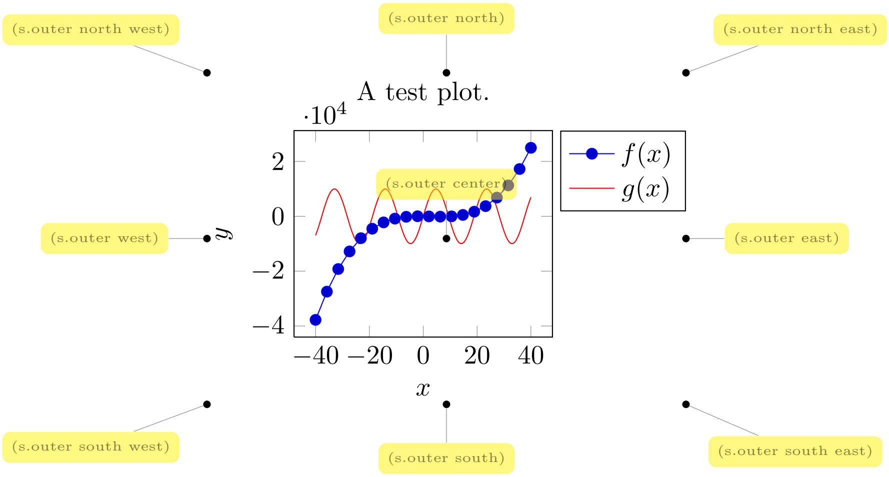
There are four sets of anchors available: anchors positioned on the axis bounding box, anchors on the outer bounding box and anchors which have one coordinate on the outer bounding box and the other one at a position of the axis rectangle. Finally, one can place anchors near the origin.

In more detail, we have Anchors on the axis rectangle (the bounding box around the axis)[34],

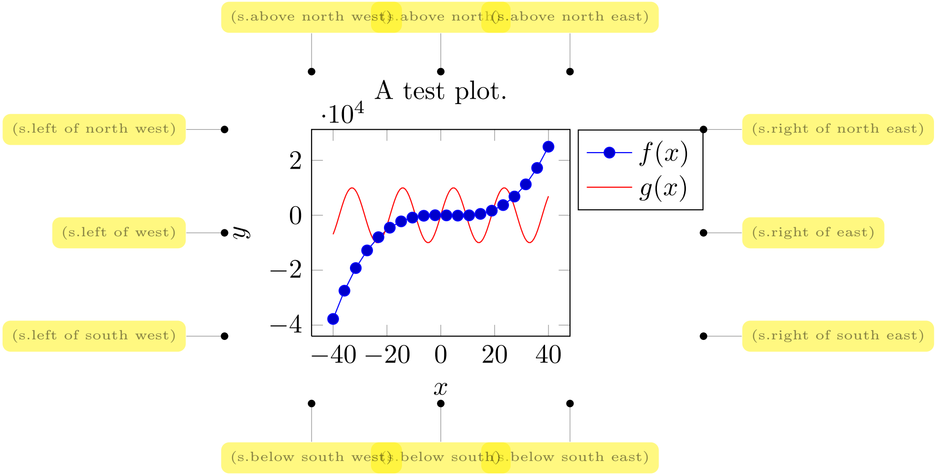


Anchors on the outer bounding box,

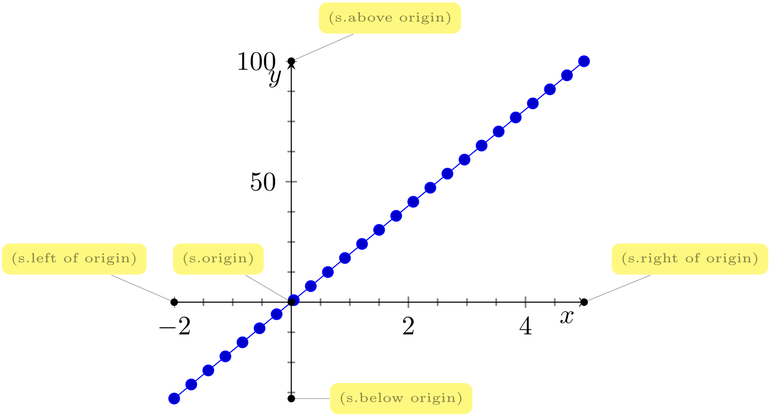
[34] Versions prior to PGFPLOTS v.1.3 did *not* use the bounding box of the axis, they used axis coordinates to orient these anchors. This has been fixed. If you *really* want to undo the bugfix, see `compat/anchors`.



There are anchors which have one coordinate on the outer bounding box, and one on the axis rectangle,



And finally, we have origin anchors which are especially useful when axis lines pass through the origin,



The default value is anchor=south west. You can use anchors in conjunction with the Ti*k*Z baseline option and/or \begin{pgfinterruptboundingbox} to perform alignment.

Remarks: Each of the anchors on the axis rectangle has an equivalent to a coordinate in the axis description cs described in section 4.8.1. That means the first set of anchors actually lives on the *tight bounding box around the axis* (without any ticks or descriptions). The south west anchor will always be the lower left corner of this bounding box, even in case of a rotated or skewed coordinate system[35]. Similar statements hold for the other anchors.

[35] Note that this is only true for versions since 1.3.

4.18.2 Vertical alignment with `baseline`

`/tikz/baseline` (no value)

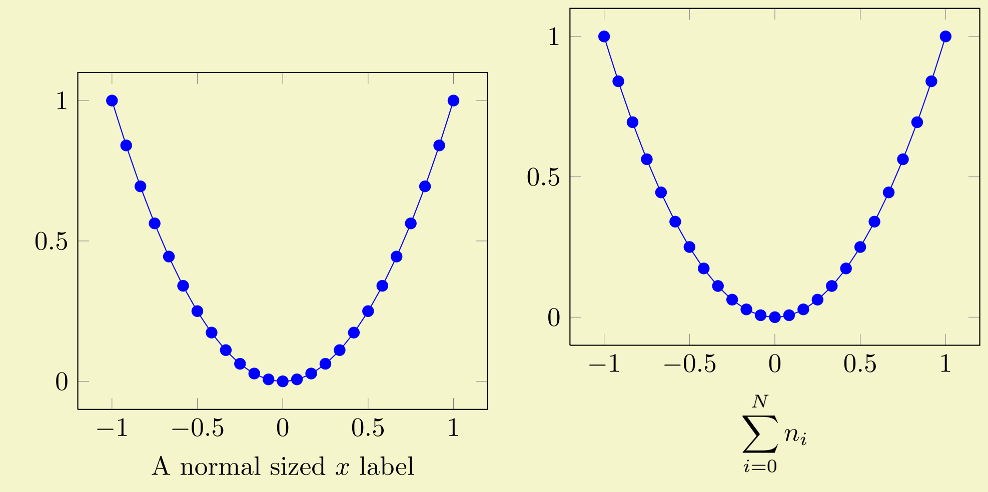
The `baseline` option should be provided as argument to a `tikzpicture`. It configures Ti*k*Z to shift the picture position $y = 0$ to the embedding text's baseline:

This is ● a picture,here ● another one.

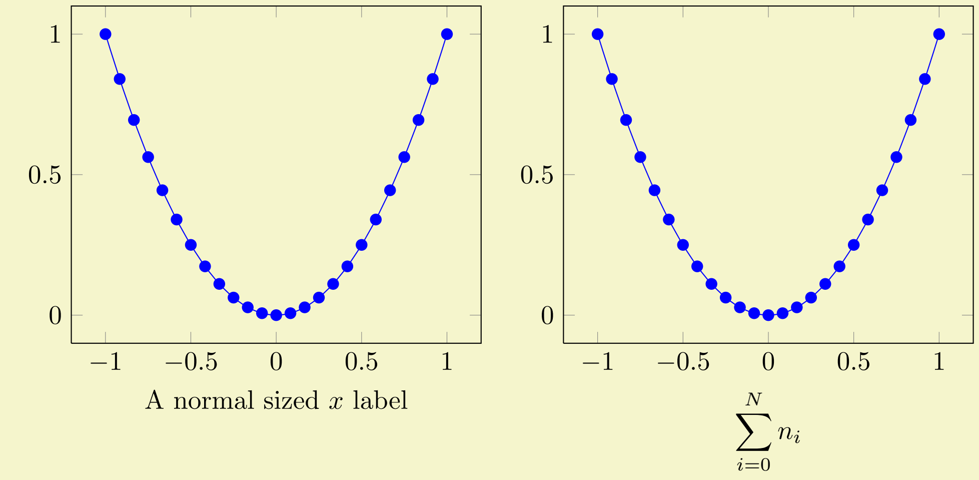
```
This is \tikz[baseline]\fill[red] (0,0) circle(3pt); a picture,
here \tikz[baseline]\fill[red] (0,10pt) circle(3pt); another one.
```

Consequently, the `baseline` option allows to align different `tikzpicture`s. An axis is, by default, placed with `at={(0,0)}`, and the `anchor` key specifies which part of the axis is placed at `(0,0)`. Consequently, the `baseline` option, together with `anchor`, allows to align different axes with the embedding text.

The default axis anchor is `south west`, which means that the picture coordinate $(0, 0)$ is the lower left corner of the axis. As a consequence, the Ti*k*Z option "`baseline`" allows vertical alignment of adjacent plots:



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
%  1. Unaligned:
\pgfplotsset{domain=-1:1}
\begin{tikzpicture}
    \begin{axis}[xlabel=A normal sized $x$ label]
    \addplot[smooth,blue,mark=*] {x^2};
    \end{axis}
\end{tikzpicture}%
\hspace{0.15cm}
\begin{tikzpicture}
    \begin{axis}[xlabel={$\displaystyle \sum_{i=0}^N n_i $ }]
    \addplot[smooth,blue,mark=*] {x^2};
    \end{axis}
\end{tikzpicture}
```

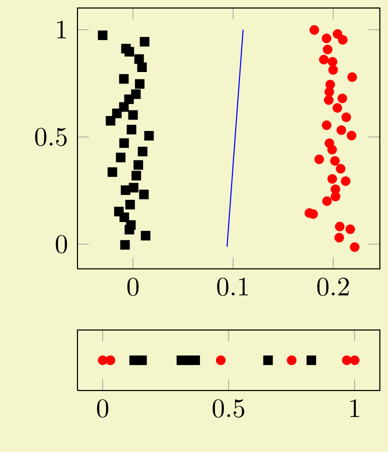


```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
%  2. Aligned:
\pgfplotsset{domain=-1:1}
\begin{tikzpicture}[baseline]
    \begin{axis}[xlabel=A normal sized $x$ label]
    \addplot[smooth,blue,mark=*] {x^2};
    \end{axis}
\end{tikzpicture}%
\hspace{0.15cm}
\begin{tikzpicture}[baseline]
    \begin{axis}[xlabel={$\displaystyle \sum_{i=0}^N n_i $ }]
    \addplot[smooth,blue,mark=*] {x^2};
    \end{axis}
\end{tikzpicture}
```

The `baseline` key is related to `\begin{minipage}[`⟨*align how*⟩`]` or `\begin{tabular}[`⟨*align how*⟩`]`: the ⟨*align how*⟩ tells LaTeX which part of the `minipage` or `tabular` shall be positioned on the baseline. Thus, `baseline` does the some for pictures (with more freedom for ⟨*align how*⟩).

4.18.3 Horizontal Alignment

If you place multiple `axes` into a single `tikzpicture` and use the '`anchor`'-option, you can control horizontal alignment:



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\pgfplotsset{every axis/.append style={
cycle list={
    {red,only marks,mark options={
        fill=red,scale=0.8},mark=*},
    {black,only marks,mark options={
        fill=black,scale=0.8},mark=square*}}}}

\begin{axis}[width=4cm,scale only axis,
    name=main plot]
\addplot file
    {plotdata/pgfplots_scatterdata1.dat};
\addplot file
    {plotdata/pgfplots_scatterdata2.dat};
\addplot[blue] coordinates {
    (0.093947,    -0.011481)
    (0.101957,    0.494273)
    (0.109967,    1.000027)};
\end{axis}

%  introduce named coordinate:
\path (main plot.below south west) ++(0,-0.1cm)
    coordinate (lower plot position);

\begin{axis}[at={(lower plot position)},
    anchor=north west,
    width=4cm,scale only axis,height=0.8cm,
    ytick=\empty]

\addplot file
  {plotdata/pgfplots_scatterdata1_latent.dat};
\addplot file
  {plotdata/pgfplots_scatterdata2_latent.dat};
\end{axis}
\end{tikzpicture}
```

4.18.4 Bounding box restrictions

Bounding box restrictions can be realized with several methods of PGF:

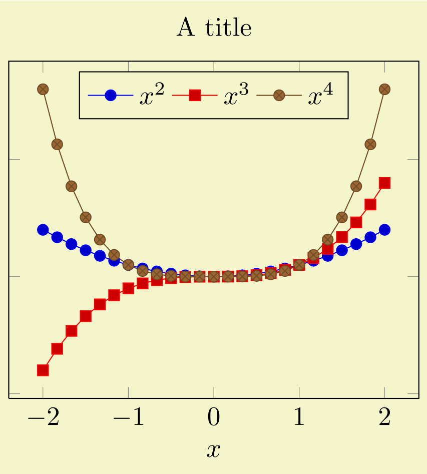
1. The `overlay` option,
2. The `pgfinterruptboundingbox` environment,
3. The `\useasboundingbox` path.

`/tikz/overlay` (no value)

A special key of PGF which disables bounding box updates for (parts of) the image. The effect is that those parts are an "overlay" over the document.

For PGFPLOTS, `overlay` can be useful to position legends or other axis descriptions outside of the axis – without affecting its size (and without affecting alignment).

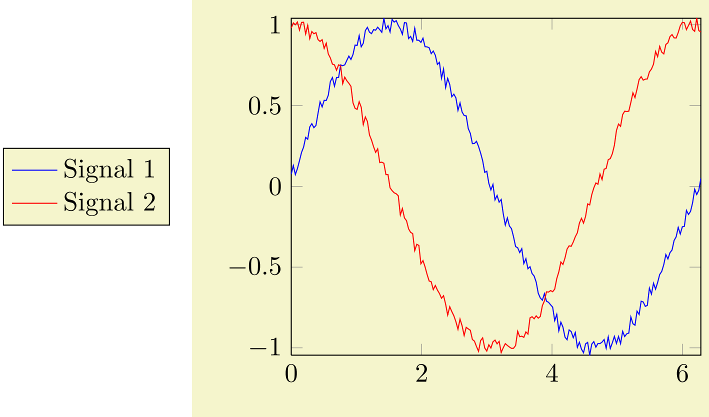
For example, one may want to include only certain parts of the axis into the final bounding box. This would allow horizontal alignment (centering):



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}%
   \begin{axis}[
      title=A title,
      ylabel style={overlay},
      yticklabel style={overlay},
      xlabel={$x$},
      ylabel={$y$},
      legend style={at={(0.5,0.97)},
         anchor=north,legend columns=-1},
      domain=-2:2
   ]
   \addplot {x^2};
   \addplot {x^3};
   \addplot {x^4};
   \legend{$x^2$,$x^3$,$x^4$}
   \end{axis}
\end{tikzpicture}%
```

Now, the left axis descriptions (y label and y ticks) stick out of the bounding box.

The following example places a legend somewhere without affecting the bounding box.

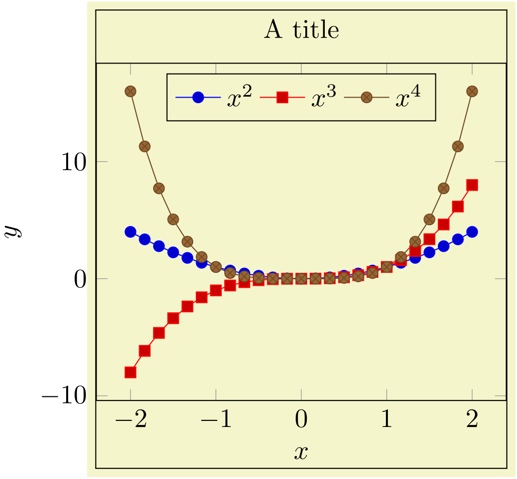


```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
   \begin{axis}[
      domain=0:6.2832,samples=200,
      legend style={
         overlay,
         at={(-0.5,0.5)},
         anchor=center},
      every axis plot post/.append style={mark=none},
      enlargelimits=false]

   \addplot {sin(deg(x)+3)+rand*0.05};
   \addplot {cos(deg(x)+2)+rand*0.05};
   \legend{Signal 1,Signal 2}
   \end{axis}
\end{tikzpicture}
```

More information about the `overlay` option can be found in the PGF manual [5].

```
\begin{pgfinterruptboundingbox}
  ⟨environment contents⟩
\end{pgfinterruptboundingbox}
```

An alternative to `overlay` is shown below: the figure has a truncated bounding box with is shown using `\fbox`.



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\setlength{\fboxsep}{0pt}%
\fbox{%
\begin{tikzpicture}%
    \begin{pgfinterruptboundingbox}
    \begin{axis}[
        name=my plot,
        title=A title,
        xlabel={$x$},
        ylabel={$y$},
        legend style={at={(0.5,0.97)},
            anchor=north,legend columns=-1},
        domain=-2:2
    ]
    \addplot {x^2};
    \addplot {x^3};
    \addplot {x^4};
    \legend{$x^2$,$x^3$,$x^4$}
    \end{axis}
    \end{pgfinterruptboundingbox}

    \useasboundingbox
              (my plot.below south west)
    rectangle (my plot.above north east);
\end{tikzpicture}%
}%
```

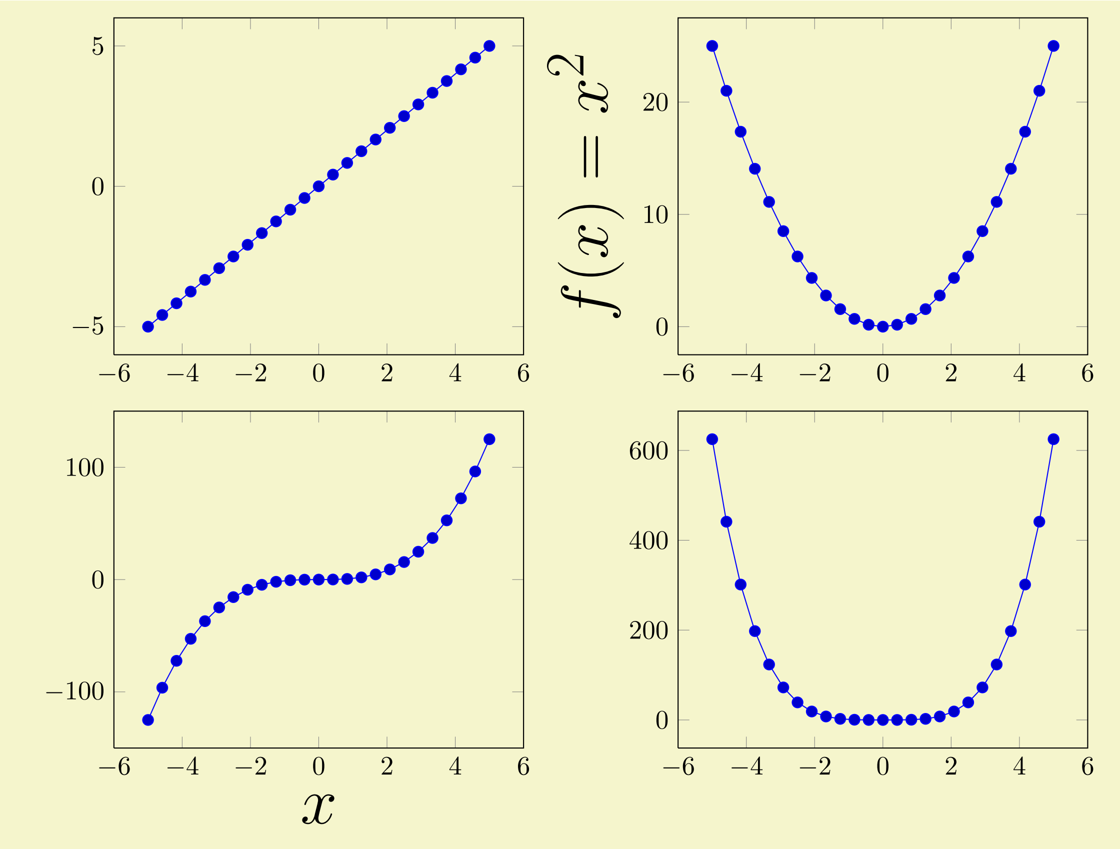
The `pgfinterruptboundingbox` environment does not include its content into the image's bounding box, and `\useasboundingbox` sets the pictures bounding box to the following argument (see [5]).

4.18.5 Alignment In Array Form (Subplots)

Sometimes alignment in array form is desired. One can use the following approach or use the library `groupplots` on page 5.4 which does not utilize Ti*k*Z matrices.

While it is possible to use (for example) `tabular` combined with the vertical and horizontal alignment methods discussed above, it might be better to use a Ti*k*Z `matrix`.

A Ti*k*Z matrix is some sort of "graphical" table. It knows everything about picture alignment and it has more flexibility than `tabular`. The complete documentation of a Ti*k*Z matrix is beyond the scope of this manual, please refer to [5] for details. But we provide an example here:



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \matrix {
        \begin{axis}
            \addplot {x};
        \end{axis}
        &
        %  differently large labels are aligned automatically:
        \begin{axis}[ylabel={$f(x)=x^2$},ylabel style={font=\Huge}]
            \addplot {x^2};
        \end{axis}
        \\
        %
        \begin{axis}[xlabel=$x$,xlabel style={font=\Huge}]
            \addplot {x^3};
        \end{axis}
        &
        \begin{axis}
            \addplot {x^4};
        \end{axis}
        \\
    };
\end{tikzpicture}
```

So, a matrix is a picture element inside of `tikzpicture`. Its cells are separated by '`&`' as in tabular (or, if '`&`' causes problems, with `\pgfmatrixnextcell`). Its rows are separated by '`\\`'. Each cell is aligned using the cells' anchor. Since, by default, the anchor of an axis is placed at the lower left corner, the example above is completely aligned, without the need for any bounding box modifications – even the labels are aligned correctly. If another anchor shall be used, simply place

```
\pgfplotsset{anchor=....}
\matrix {
  ...
};
```

in front of the matrix. This will use the same configuration for every sub-plot.

Attention: Unfortunately, the array alignment with `\matrix` is *incompatible* with legends. A legend is also a matrix and, unfortunately, Ti*k*Z matrizes can't be nested. Of course, this does not affect the manually created legends by means of `\label` and `\ref`, see section 4.8.6 for details.

If one wishes to have an easier approach of aligning one should refer to the library `groupplots` on p. 5.4.

4.18.6 Miscellaneous for Alignment

Predefined node `current axis`

A node which refers to the current axis or the last typeset axis.

You can use this node in axis descriptions, for example to place axis labels or titles.

Remark: If you use `current axis` inside of axis descriptions, the "current axis" is not yet finished. That means you *can't use any outer anchor* inside of axis descriptions.

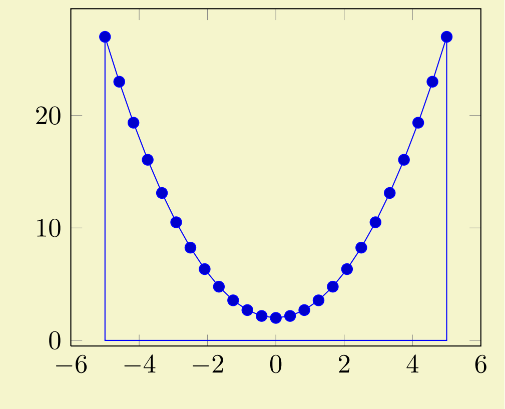
It is also possible to use `current axis` in any drawing or plotting commands inside of an axis (but no outer anchor as these are not defined when drawing commands are processed). This usage is similar to the `axis description cs`.

4.19 Closing Plots (Filling the Area Under Plots)

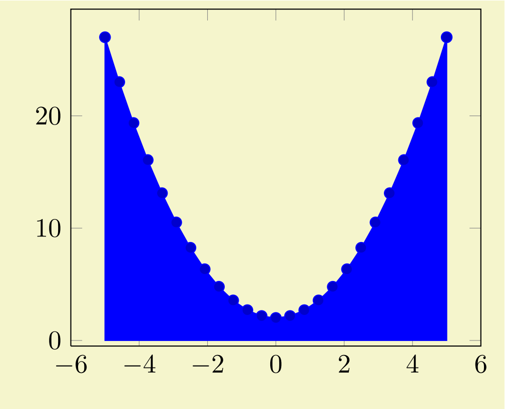
`\closedcycle`

Provide `\closedcycle` as ⟨*trailing path commands*⟩ after `\addplot` to draw a closed line from the last plot coordinate to the first one.

Use `\closedcycle` whenever you intend to fill the area under a plot.



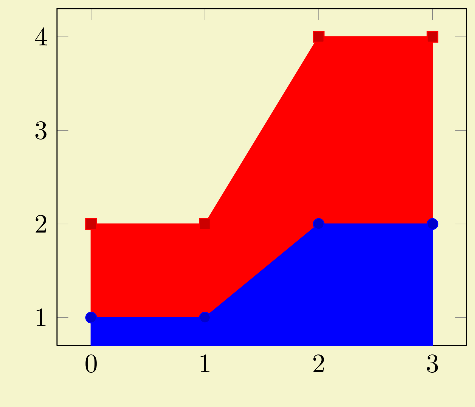
```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{axis}
    \addplot {x^2+2} \closedcycle;
    \end{axis}
\end{tikzpicture}
```



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{axis}
    \addplot+[fill] {x^2+2} \closedcycle;
    \end{axis}
\end{tikzpicture}
```

In case of stacked plots, `\closedcycle` connects the current plot with the previous plot instead of connecting with the x axis[36].

[36] The implementation for stacked plots requires some additional logic to determine the filled area: `\closedcycle` will produce a `plot coordinates` command with *reversed* coordinates of the previous plot. This is usually irrelevant for end users, but it assumes that the plot's type is symmetric. Since constant plots are inherently asymmetric, `\closedcycle` will use `const plot mark right` as reversed sequence for `const plot mark left`.



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{axis}[stack plots=y]
    \addplot+[fill] coordinates
        {(0,1) (1,1) (2,2) (3,2)} \closedcycle;
    \addplot+[fill] coordinates
        {(0,1) (1,1) (2,2) (3,2)} \closedcycle;
    \end{axis}
\end{tikzpicture}
```

4.20 Symbolic Coordinates and User Transformations

PGFPLOTS supports user transformations which can be applied to input and output coordinates. Suppose the plot shall display days versus account statements over time. Then, one wants to visualize date versus credit balance. But: dates need to be transformed to numbers before doing so! Furthermore, tick labels shall be displayed as dates as well. This, and more general transformations, can be realized using the `x coord trafo` and `y coord trafo` keys.

Remark: This section applies to users who want to have non-standard input *coordinates*. If you have normal numbers which don't need to be transformed and you like to have special symbols as tick labels, you should consider using the `xticklabels` (`yticklabels`) key described on page 162.

/pgfplots/x coord trafo/.code={⟨...⟩}
/pgfplots/y coord trafo/.code={⟨...⟩}
/pgfplots/z coord trafo/.code={⟨...⟩}
/pgfplots/x coord inv trafo/.code={⟨...⟩}
/pgfplots/y coord inv trafo/.code={⟨...⟩}
/pgfplots/z coord inv trafo/.code={⟨...⟩}

These code keys allow arbitrary coordinate transformations which are applied to input coordinates and output tick labels.

The `x coord trafo` and `y coord trafo` command keys take one argument which is the input coordinate. They are expected to set `\pgfmathresult` to the final value.

At this level, the input coordinate is provided as it is found in the `\addplot` statement. For example, if x coordinates are actually of the form ⟨*year*⟩-⟨*month*⟩-⟨*day*⟩, for example `2008-01-05`, then a useful coordinate transformation would transform this string into a number (see below for a predefined realization).

In short, *no* numerics has been applied to input coordinates when this transformation is applied[37].

The input coordinate transformation is applied to

- any input coordinates (specified with `\addplot` or `axis cs`),
- any user-specified `xtick` or `ytick` options,
- any user-specified `extra x ticks` and `extra y ticks` options,
- any user-specified axis limits like `xmin` and `xmax`.

The output coordinate transformation `x coord inv trafo` is applied to tick positions just before evaluating the `xticklabel` and `yticklabel` keys. The argument to `x coord inv trafo` is a fixed point number (which may have trailing zeros after the period). The tick label code may use additional macros defined by the inverse transformation.

Remark: PGFPLOTS will continue to produce tick positions as usual, no extra magic is applied. It may be necessary to provide tick positions explicitly if the default doesn't respect the coordinate space properly.

[37] Of course, if coordinates have been generated by gnuplot or PGF, this does no longer hold.

The initial value of these keys is

```
\pgfplotsset{
    x coord trafo/.code={},
    x coord inv trafo/.code={}}
```

which simply disables the transformation (the same for y, of course).

Remark: It might be necessary to set

```
\pgfplotsset{
    xticklabel={\tick},
    scaled x ticks=false,
    plot coordinates/math parser=false,
}
```

in order to avoid number formatting routines on `\tick` or numerics for tick scale methods. This is done automatically by the predefined symbolic coordinate styles (see below).

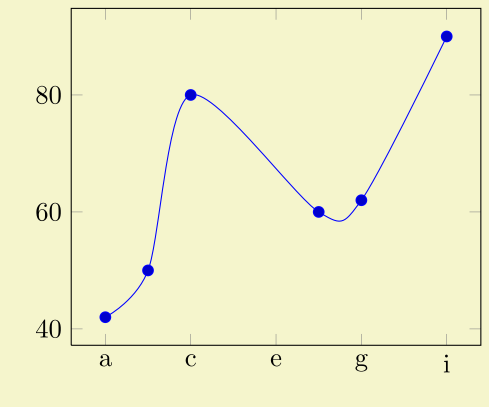
4.20.1 String Symbols as Input Coordinates

It is possible to provide a string dictionary to PGFPLOTS. An input coordinate can then use any symbol provided in that dictionary.

/pgfplots/symbolic x coords={⟨*dictionary*⟩}
/pgfplots/symbolic y coords={⟨*dictionary*⟩}
/pgfplots/symbolic z coords={⟨*dictionary*⟩}

A styles which sets `x coord trafo` and `x coord inv trafo` (or the respective `y` or `z` variants) such that any element in {⟨*dictionary*⟩} is a valid input coordinate. The {⟨*dictionary*⟩} can be a comma separated list or a list terminated with '\\'. In both case, white spaces are considered to be part of the names (use '%' at end of lines).

The dictionary will assign integer numbers to every element. These integers are used internally for arithmetics. Finally, the inverse transformation takes a fixed point number and maps it to the nearest integer, and that integer is mapped into the dictionary.



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}[symbolic x coords={a,b,c,d,e,f,g,h,i}]
    \addplot+[smooth] coordinates {
        (a,42)
        (b,50)
        (c,80)
        (f,60)
        (g,62)
        (i,90)};
\end{axis}
\end{tikzpicture}
```

The effect of the transformation is simply that input coordinates can be elements of the dictionary and tick labels will be chosen out of this dictionary as well.

4.20.2 Dates as Input Coordinates

The already mentioned application of using dates as input coordinates has been predefined, together with support for hours and minutes. It relies on the PGF calendar library which converts dates to numbers in the julian calendar. Then, one coordinate unit is one day.

```
\usepgfplotslibrary{dateplot} % LaTeX and plain TeX
\usepgfplotslibrary[dateplot] % ConTeXt
\usetikzlibrary{pgfplots.dateplot} % LaTeX and plain TeX
```

`\usetikzlibrary[pgfplots.dateplot] % ConTEXt`

Loads the coordinate transformation code.

`/pgfplots/date coordinates in=x|y`

Installs `x coord trafo` and `x coord inv trafo` (or the respective y variant) such that ISO dates of the form ⟨*year*⟩-⟨*month*⟩-⟨*day*⟩ are accepted. For example, `2006-02-28` will be converted to an "appropriate" integer using the julian calender. Input coordinates may be of the form

⟨*year*⟩-⟨*month*⟩-⟨*day*⟩

or they may contain times as

⟨*year*⟩-⟨*month*⟩-⟨*day*⟩ ⟨*hour*⟩:⟨*minute*⟩.

The result of the transformation are numbers where one unit is one day and times are fractional numbers.

The transformation is realized using the PGF-calendar module, see [5, Calendar Library]. This reference also contains more information about extended syntax options for dates.

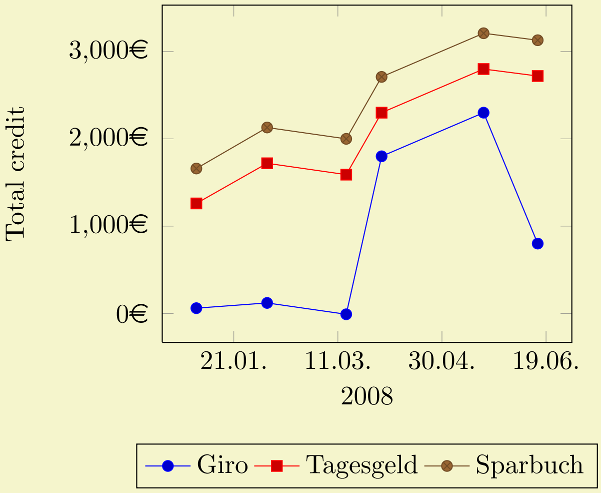
The inverse transformation provides the following macros which are available during tick label evaluation (i.e. when used inside of `xticklabel` or `yticklabel`):

- `\year` expands to the year component,
- `\month` expands to the month component,
- `\day` expands to the day component,
- `\hour` expands to the hour component (using two digits),
- `\Hour` expands to the hour component (but omits leading zeros),
- `\minute` expands to the minute component (two digits),
- `\Minute` expands to the minute component (omits leadings zeros),
- `\lowlevel` expands to the low level number representing the tick,
- `\second` will always be `00`.

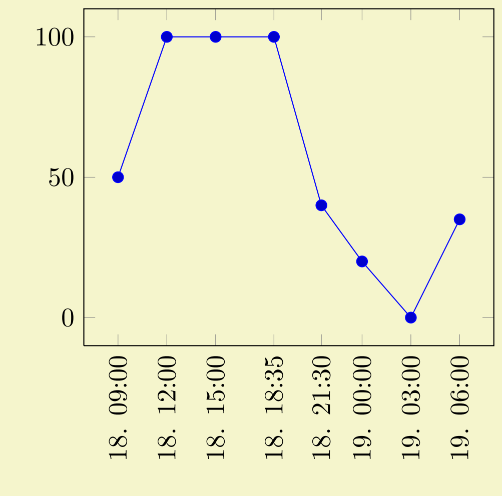
This allows to use `\day.\month.\year` or `\day. \hour:\minute` inside of `xticklabel`, for example.

A complete example (with fictional data) is shown below.

date	account1	account2	account3
2008-01-03	60	1200	400
2008-02-06	120	1600	410
2008-03-15	-10	1600	410
2008-04-01	1800	500	410
2008-05-20	2300	500	410
2008-06-15	800	1920	410



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
%  requires \usepgfplotslibrary{dateplot} !

\pgfplotstabletypeset[string type]{plotdata/accounts.dat}

\begin{tikzpicture}
    \begin{axis}[
        date coordinates in=x,
        xticklabel={\day.\month.},
        xlabel={2008},
        stack plots=y,
        yticklabel={\pgfmathprintnumber{\tick}\EUR{}}, %  <- requires \usepackage{eurosym}
        ylabel=Total credit,
        ylabel style={yshift=10pt},
        legend style={
            at={(0.5,-0.3)},anchor=north,legend columns=-1}]

    \addplot table[x=date,y=account1] {plotdata/accounts.dat};
    \addplot table[x=date,y=account2] {plotdata/accounts.dat};
    \addplot table[x=date,y=account3] {plotdata/accounts.dat};
    \legend{Giro,Tagesgeld,Sparbuch}
    \end{axis}
\end{tikzpicture}
```



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
%  requires \usepgfplotslibrary{dateplot} !
\begin{tikzpicture}
  \begin{axis}[
    date coordinates in=x,
    xtick=data,
    xticklabel style=
        {rotate=90,anchor=near xticklabel},
    xticklabel=\day. \hour:\minute,
    date ZERO=2009-08-18,%  <- improves precision!
  ]
  \addplot coordinates {
    (2009-08-18 09:00,  050)
    (2009-08-18 12:00,  100)
    (2009-08-18 15:00,  100)
    (2009-08-18 18:35,  100)
    (2009-08-18 21:30,  040)
    (2009-08-19,        020)
    (2009-08-19 3:00,   000)
    (2009-08-19 6:0,    035)
  };
  \end{axis}
\end{tikzpicture}
```

Attention: If you intend to use hours and minutes, you should *always* provide the `date ZERO` to maintain adequate precision!

/pgfplots/date ZERO=⟨*year*⟩-⟨*month*⟩-⟨*day*⟩ (initially 2006-01-01)

A technical key which defines the 0 coordinate of `date coordinates in`. Users will never see the resulting numbers, so one probably never needs to change it. However, the resulting numbers may become very large and a mantisse of 6 significant digits may not be enough to get accurate results. In this case, `date ZERO` should be set to a number which falls into the input date range.

4.21 Skipping Or Changing Coordinates – Filters

/pgfplots/x filter/.code={⟨...⟩}
/pgfplots/y filter/.code={⟨...⟩}
/pgfplots/z filter/.code={⟨...⟩}
/pgfplots/filter point/.code={⟨...⟩}

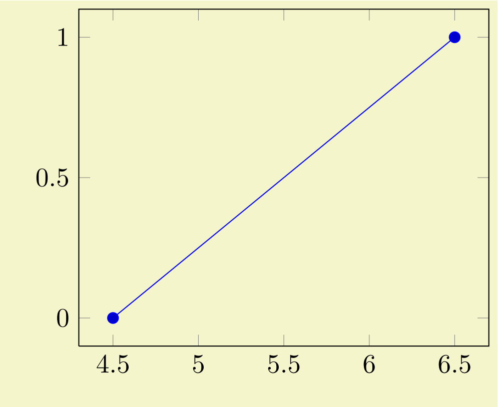
The code keys `x filter` and `y filter` allow coordinate filtering which are based on a *single* coordinate. A coordinate filter gets an input coordinate as `#1`, applies some operation and writes the result into the macro `\pgfmathresult`. If `\pgfmathresult` is empty afterwards, the coordinate is discarded. You can also set \pgfmathresult to nan or inf in which case the coordinate can be either discarded (if unbounded coords=discard is set) or the plot can be interrupted (the case unbounded coords=jump).

The filter point/.code filter allows filtering dependent on all components forming a complete point (x, y and z); it is described below.

It is allowed if filters do not change \pgfmathresult. In this case, the unfiltered coordinate will be used.

Coordinate filters are useful in automatic processing system, where PGFPLOTS is used to display automatically generated plots. You may not want to filter your coordinates by hand, so these options provide a tool to do this automatically.

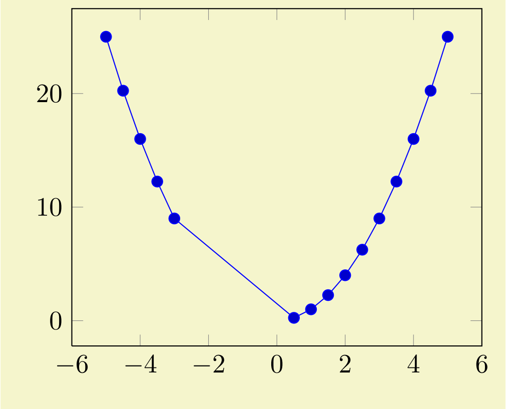
The following filter adds 0.5 to every x coordinate.



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}[x filter/.code=
    {\pgfmathadd{#1}{0.5}}]
\addplot coordinates {
    (4,0)
    (6,1)
};
\end{axis}
\end{tikzpicture}
```

Please refer to [5, pgfmath manual] for details about the math engine of PGF. Please keep in mind that the math engine works with limited TEX precision.

During evaluation of the filter, the macro \coordindex contains the number of the current coordinate (starting with 0). Thus, the following filter discards all coordinates after the 5th and before the 10th.



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}[
    samples=20,
    x filter/.code={
        \ifnum\coordindex>4
            \ifnum\coordindex<11
                \def\pgfmathresult{}
            \fi
        \fi
    }]
\addplot {x^2};
\end{axis}
\end{tikzpicture}
```

There is also a style key which simplifies selection by index, see below.

PGFPLOTS invokes the filter with argument #1 set to the input coordinate. For x-filters, this is the x-coordinate as it is specified to \addplot, for y-filters it is the y-coordinate.

If the corresponding axis is logarithmic, #1 is the *logarithm* (see log basis x and its variants) of the coordinate as a real number, for example #1=4.2341. In case the logarithm was undefined, the argument will be empty.

The arguments to coordinate filters are minimally preprocessed: first, for logarithmic axes, the *log* of the argument is supplied. Second, any high level coordinate maps like x coord trafo (which may be used to map dates to numbers or string to numbers or so) are applied. In consequence, the #1 argument is supposed to be a number. No further transformation has been applied.

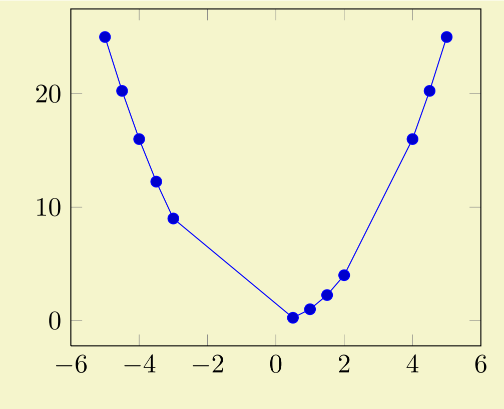
Occasionally, it might be handy to get the “raw”, completely unprocessed input coordinate as it has been reported by the coordinate input routine. This unprocessed data is available in the three math parser constants rawx, rawy and rawz (use \pgfmathrawx, \pgfmathrawy and \pgfmathrawz as a way to assign the value of interest to `\pgfmathresult`). All these values are ready for use in filters (and some other methods influence plots as well).

If key filters are invoked for `plot table`, access to the current row's data can be achieved using `\thisrow{`⟨*column name*⟩`}` (and its variants). This includes all columns of the table.

The `filter point` key is more technical. It doesn't take an argument: its arguments are given in terms of the `pgfkeys` variables `/data point x`, `/data point y` and `/data point z`. It may change its coordinates using `\pgfkeyssetvalue{/data point x}{`⟨*the new value*⟩`}`; access to variables can be get with `\pgfkeysvalueof{/data point/x}` or, if the argument shall be written into a macro, with `\pgfkeysgetvalue`. This filter is evaluated after the other ones.

/pgfplots/`skip coords between index={`⟨*begin*⟩`}{`⟨*end*⟩`}`

A style which appends an `x filter` which discards selected coordinates. The selection is done by index where indexing starts with 0, see `\coordindex`. Every coordinate with index ⟨*begin*⟩ $\leq i <$ ⟨*end*⟩ will be skipped.



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}[
    samples=20,
    skip coords between index={5}{11},
    skip coords between index={15}{18}]

\addplot {x^2};
\end{axis}
\end{tikzpicture}
```

/pgfplots/`each nth point={`⟨*integer*⟩`}`

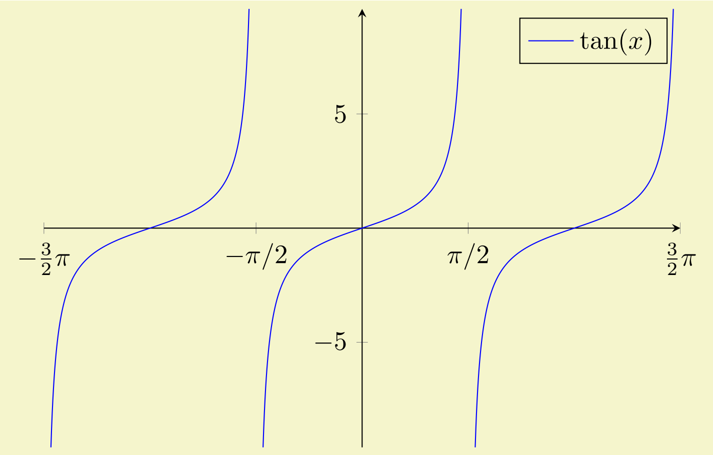
A style which appends an `x filter` which discards all but each nth input coordinate.

/pgfplots/`restrict x to domain=`⟨*min*⟩:⟨*max*⟩
/pgfplots/`restrict y to domain=`⟨*min*⟩:⟨*max*⟩
/pgfplots/`restrict z to domain=`⟨*min*⟩:⟨*max*⟩

Appends x (or y or z) coordinate filters which set the respective coordinate to `-inf` if it is below ⟨*min*⟩ and to `+inf` if it is above ⟨*max*⟩.

For logarithmic axes, ⟨*min*⟩ and ⟨*max*⟩ are *logs* of the respective values. A variant which uses the non-logarithmic number might be to use `restrict expr to domain={\pgfmathrawx}{`⟨*min*⟩`}{`⟨*max*⟩`}`.

Furthermore, it sets the `unbounded coords=jump` key which leads to interrupted plots.



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}[
    restrict y to domain=-10:10,
    samples=1000,
    %  some fine tuning for the display:
    width=10cm, height=210pt,
    xmin=-4.7124, xmax=4.7124,
    xtick={-4.7124,-1.5708,...,10},
    xticklabels={$-\frac32 \pi$,$-\pi/2$,$\pi/2$,$\frac32 \pi$},
    axis x line=center,
    axis y line=center]

\addplot[blue] gnuplot[id=tangens,domain=-1.5*pi:1.5*pi] {tan(x)};
\legend{$\tan(x)$}
\end{axis}
\end{tikzpicture}
```

/pgfplots/restrict expr to domain={⟨*expression*⟩}{⟨⟨*min*⟩:⟨*max*⟩⟩}

Appends an x coordinate filter which sets the x coordinate to `-inf` if the ⟨*expression*⟩ evaluates to something less than ⟨*min*⟩ and to `inf` if ⟨*expression*⟩ evaluates to something larger than ⟨*max*⟩.

Furthermore, it sets the `unbounded coords=jump` key which leads to interrupted plots.

In contrast to `restrict x to domain`, ⟨*expression*⟩ can depend on anything which is valid during `\addplot`, in particular `\coordindex` or table columns (`\thisrow{`⟨*column name*⟩`}` and friends). The expression doesn't need to depend on x at all.

/pgfplots/filter discard warning=true|false (initially true)

Issues a notification in your logfile whenever coordinate filters discard coordinates.

4.22 Miscellaneous Options

/pgfplots/disablelogfilter=true|false (initally false, default true)

Disables numerical evaluation of $\log(x)$ in TEX. If you specify this option, any plot coordinates and tick positions must be provided as $\log(x)$ instead of x. This may be faster and – possibly – more accurate than the numerical log. The current implementation of $\log(x)$ normalizes x to $m \cdot 10^e$ and computes

$$\log(x) = \log(m) + e \log(10)$$

where $y = \log(m)$ is computed with a newton method applied to $\exp(y) - m$. The normalization involves string parsing without TEX-registers. You can savely evaluate $\log(1 \cdot 10^{-7})$ although TEX-registers would produce an underflow for such small numbers.

/pgfplots/disabledatascaling=true|false (initally false, default true)

Disables internal re-scaling of input data. Normally, every input data like plot coordinates, tick positions or whatever, are parsed without using TEX's limited number precision. Then, a transformation like

$$T(x) = 10^{q-m} \cdot x - a$$

is applied to every input coordinate/position where m is "the order of x" base 10. Example: $x = 1234 = 1.234 \cdot 10^3$ has order $m = 4$ while $x = 0.001234 = 1.234 \cdot 10^{-3}$ has order $m = -2$. The parameter q is the order of the axis' width/height.

The **effect** of the transformation is that your plot coordinates can be of *arbitrary magnitude* like 0.0000001 and 0.0000004. For these two coordinates, PGFPLOTS will use 100pt and 400pt internally. The transformation is quit fast since it relies only on period shifts. This scaling allows precision beyond TEX's capabilities.

The option "`disabledatascaling`" disables this data transformation. This has two consequences: first, coordinate expressions like (⟨`axis cs:`x,y⟩) have the same effect like (⟨x,y⟩), no re-scaling is applied. Second, coordinates are restricted to what TEX can handle[38].

So far, the data scale transformation applies only to normal axis (logarithmic scales do not need it).

[38] Please note that the axis' scaling requires to compute $1/(x_{\max} - x_{\min})$. The option `disabledatascaling` may lead to overflow or underflow in this context, so use it with care! Normally, the data scale transformation avoids this problem.

/pgfplots/execute at begin plot={⟨*commands*⟩}

This axis option allows to invoke {⟨*commands*⟩} at the beginning of each \addplot command. The argument {⟨*commands*⟩} can be any TEX content.

You may use this in conjunction with x filter=... to reset any counters or whatever. An example would be to change every 4th coordinate.

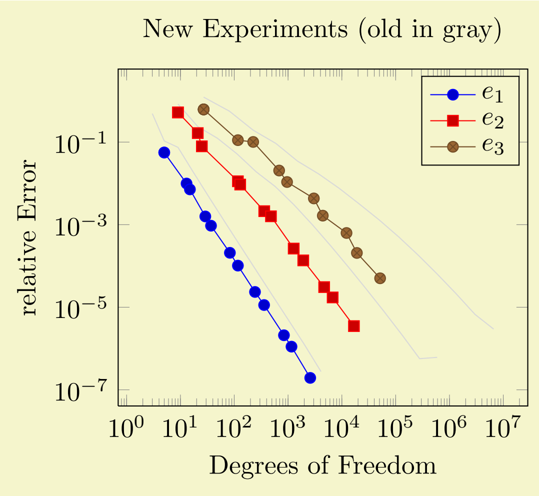
/pgfplots/execute at end plot={⟨*commands*⟩}

This axis option allows to invoke {⟨*commands*⟩} after each \addplot command. The argument {⟨*commands*⟩} can be any TEX content.

/pgfplots/forget plot={⟨*true,false*⟩} (initially false)

Allows to include plots which are not remembered for legend entries, which do not increase the number of plots and which are not considered for cycle lists.

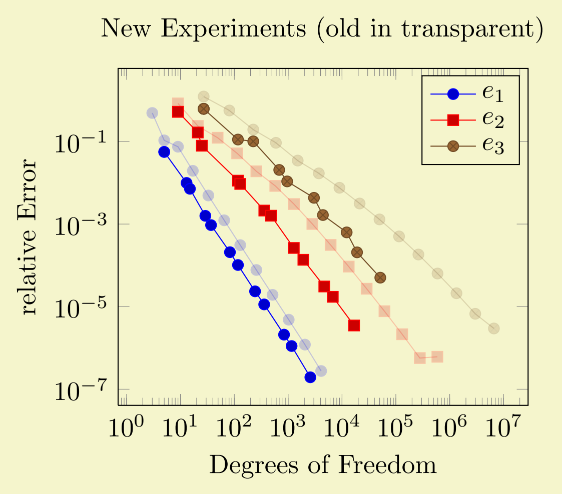
A forgotten plot can be some sort of decoration which has a separate style and does not influence the axis state, although it is processed as any other plot. Provide this option to \addplot as in the following example.



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{loglogaxis}[
        %  some descriptions:
        table/x=Basis,
        table/y={L2/r},
        xlabel=Degrees of Freedom,
        ylabel=relative Error,
        title=New Experiments (old in gray),
        legend entries={$e_1$,$e_2$,$e_3$}
    ]
    \addplot[black!15,forget plot]
        table {plotdata/oldexperiment1.dat};
    \addplot[black!15,forget plot]
        table {plotdata/oldexperiment2.dat};
    \addplot[black!15,forget plot]
        table {plotdata/oldexperiment3.dat};
    \addplot table {plotdata/newexperiment1.dat};
    \addplot table {plotdata/newexperiment2.dat};
    \addplot table {plotdata/newexperiment3.dat};
    \end{loglogaxis}
\end{tikzpicture}
```

Since forgotten plots won't increase the plot index, they will use the same cycle list entry as following plots.

The style every forget plot can be used to configure styles for each such plot:



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{loglogaxis}[
        forget plot style={opacity=0.2},
        %  same as above:
        table/x=Basis,
        table/y={L2/r},
        xlabel=Degrees of Freedom,
        ylabel=relative Error,
        title=New Experiments (old in transparent),
        legend entries={$e_1$,$e_2$,$e_3$},
    ]
    \foreach \exp in {1,2,3} {
      \addplot+[forget plot]
        table {plotdata/oldexperiment\exp.dat};
      \addplot table {plotdata/newexperiment\exp.dat};
    }
    \end{loglogaxis}
\end{tikzpicture}
```

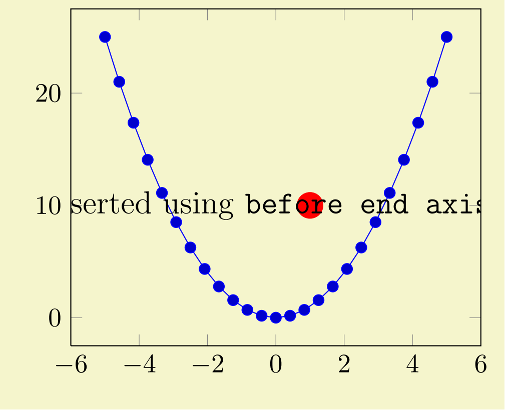
Here, the \addplot+ command means we are using the same cycle list as the following plot and forget plot style modifies every forget style and yields transparency of the "old experiments".

Please note that `every plot no` ⟨*index*⟩ styles are not applicable here.

A forgotten plot will be stacked normally if `stack plots` is enabled!

`/pgfplots/before end axis/.code={⟨...⟩}`

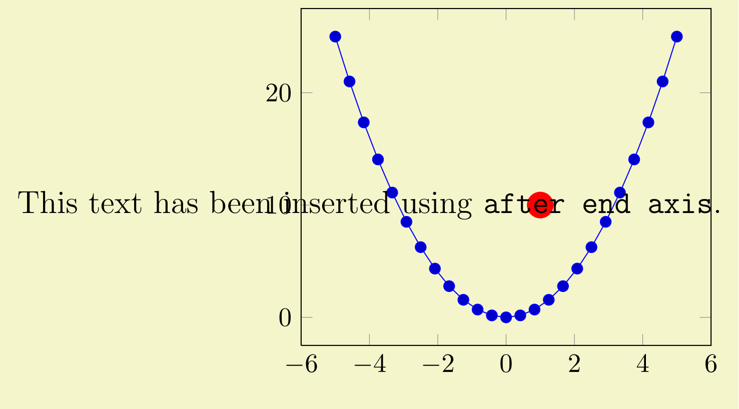
Allows to insert `{`⟨*commands*⟩`}` just before the axis is ended. This option takes effect inside of the clipped area.



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\pgfplotsset{every axis/.append style={
    before end axis/.code={
        \fill[red] (axis cs:1,10) circle(5pt);
        \node at (axis cs:-4,10)
            {\large This text has been inserted
             using \texttt{before end axis}.};
    }}}
\begin{tikzpicture}
    \begin{axis}
    \addplot {x^2};
    \end{axis}
\end{tikzpicture}
```

`/pgfplots/after end axis/.code={⟨...⟩}`

Allows to insert `{`⟨*commands*⟩`}` right after the end of the clipped drawing commands. While `befor end axis` has the same effect as if `{`⟨*commands*⟩`}` had been placed inside of your axis, `after end axis` allows to access axis coordinates without being clipped.



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\pgfplotsset{every axis/.append style={
    after end axis/.code={
        \fill[red] (axis cs:1,10) circle(5pt);
        \node at (axis cs:-4,10)
            {\large This text has been inserted using \texttt{after end axis}.};
    }}}
\begin{tikzpicture}
    \begin{axis}
    \addplot {x^2};
    \end{axis}
\end{tikzpicture}
```

`/pgfplots/clip marker paths=true|false` (initially `false`)

The initial choice `clip marker paths=false` causes markers to be drawn *after* the clipped region. Only their positions will be clipped. As a consequence, markers will be drawn completely, or not at all. The value `clip marker paths=true` is here for backwards compatibility: it does not introduce special marker treatment, so markers may be drawn partially if they are close to the clipping boundary[39].

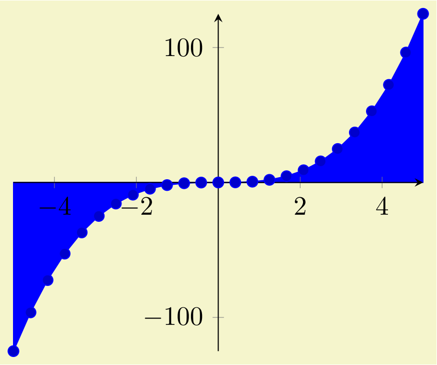
[39] Please note that clipped marker paths may be slightly faster during TEX compilation.

/pgfplots/clip=true|false (initially true)

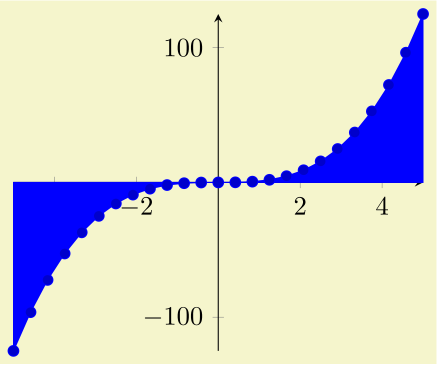
Whether any paths inside an axis shall be clipped.

/pgfplots/axis on top=true|false (initially false)

If set to true, axis lines, ticks, tick labels and grid lines will be drawn on top of plot graphics.



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{axis}[
        axis on top=true,
        axis x line=middle,
        axis y line=middle]
    \addplot+[fill] {x^3} \closedcycle;
    \end{axis}
\end{tikzpicture}
```



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{axis}[
        axis on top=false,
        axis x line=middle,
        axis y line=middle]
    \addplot+[fill] {x^3} \closedcycle;
    \end{axis}
\end{tikzpicture}
```

Please note that this feature does not affect plot marks. I think it looks unfamiliar if plot marks are crossed by axis descriptions.

/pgf/fpu={⟨*true,false*⟩} (initially true)

This key activates or deactivates the floating point unit. If it is disabled (false), the core PGF math engine written by Mark Wibrow and Till Tantau will be used for plot expression. However, this engine has been written to produce graphics and is not suitable for scientific computing. It is limited to fixed point numbers in the range ±16384.00000.

If the fpu is enabled (true, the initial configuration) the high-precision floating point library of PGF written by Christian Feuersänger will be used. It offers the full range of IEEE double precision computing in TEX. This FPU is also part of PgfplotsTable, and it is activated by default for create col/expr and all other predefined mathematical methods.

Use

```
\pgfkeys{/pgf/fpu=false}
```

in order to de-activate the extended precision. If you prefer using the fp (fixed point) package, possibly combined with Mark Wibrows corresponding PGF library, the fpu will be deactivated automatically. Please note, however, that fp has a smaller data range (about $\pm 10^{17}$) and may be slower.

5 Related Libraries

This section describes some libraries which come with PGFPLOTS, but they are more or less special and need to be activated separately.

5.1 Dates as Input Coordinates

```
\usepgfplotslibrary{dateplot} % LaTeX and plain TeX
\usepgfplotslibrary[dateplot] % ConTeXt
\usetikzlibrary{pgfplots.dateplot} % LaTeX and plain TeX
\usetikzlibrary[pgfplots.dateplot] % ConTeXt
```

A library which allows to use dates like `2008-01-01` or dates with time like `2008-01-01 11:35` as input coordinates in plots. The library converts dates to numbers and tick labels will be pretty-printed dates (or times).

This library is documented in section 4.20 on page 193.

5.2 Clickable Plots

```
\usepgfplotslibrary{clickable} % LaTeX and plain TeX
\usepgfplotslibrary[clickable] % ConTeXt
\usetikzlibrary{pgfplots.clickable} % LaTeX and plain TeX
\usetikzlibrary[pgfplots.clickable] % ConTeXt
```

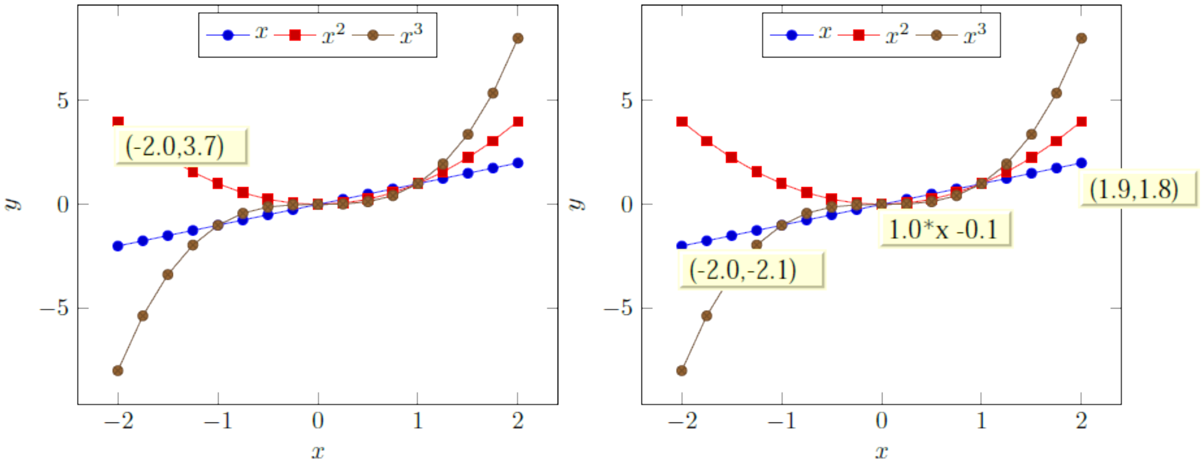
A library which generates small popups whenever one clicks into a plot. The popup displays the coordinate under the mouse pointer. Furthermore, the library allows to display slopes if one holds the mouse pressed and drags it to another point in the plot.

It is completely sufficient to write

```
\usepgfplotslibrary{clickable}
```

in the document preamble. This will automatically prepare every plot.

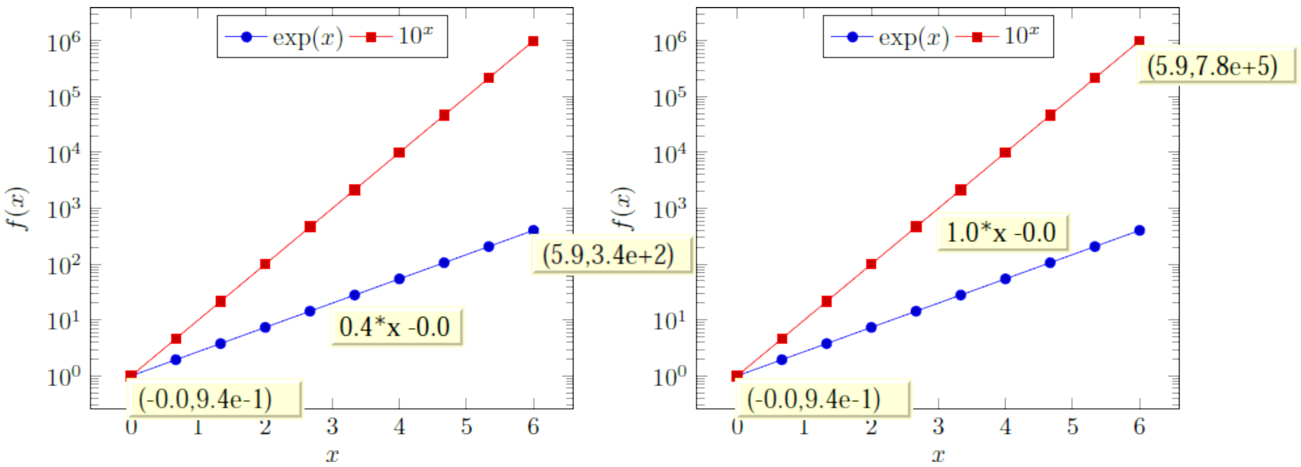
The library works with Acrobat Javascript and PDF forms: every plot becomes a push–button.



These screen shots show the result of clicking into the axis range (left column) and of dragging from one point to another (right column). The second case shows the result of drag- and drop: it displays start- and end points and the equation for the line segment between between the first point of the drag- and drop and the second point where the mouse has been released. The line segment is

$$l(x; x_0, y_0, x_1, y_1) = m \cdot x + n$$

where $m = (y_1 - y_0)/(x_1 - x_0)$ is the slope and n the offset chosen such that $l(x_0; \dots) = y_0$. For logarithmic plots, logarithms will be applied before computing slopes.



These screen shots show the result of drag- and drop for *logarithmic* axes: the end points show, again, the coordinates (without logs) and the form field in the middle shows the slope and offset of the linear equation in log coordinates.

The log basis for any logarithmic axes is usually 10, but it respects the current setting of `log basis x` and `log basis y`. The applied log will always use the same logarithm which is also used for the axis descriptions (this is not necessarily the same as used by PgfplotsTable!).

This document has been produced with the `clickable` library, so it is possible to load it into Acrobat Reader and simply click into a plot.

A click places an annotation at the coordinate under the mouse pointer, a snap–to–nearest feature is not available (yet?).

Requirements:

- The library relies on the LaTeX packages `insdljs` ("Insert document level Javascript") and `eforms` which are both part of the freely available `AcroTeX` education bundle [4][40]. The `insdljs` package creates a temporary file with extension `.djs`.
- At the time of this writing, only Adobe Acrobat Reader interpretes Javascript and Forms properly. The library doesn't have any effect if the resulting document is used in other viewers (as far as I know).

Note that although this library has been written for pgfplots, it can be used independently of an pgfplots environment.

Compatibility issues: There a several restrictions when using this library. Most of them will vanish in future versions – but up to now, I can't do magic.

- The library does not yet support rotated axes. Use `clickable=false` for those axes.
- The library works only with `pdflatex`, `dvips` or `dvipdfm` are not supported[41].
- Up to now, it is *not* possible to use this library together with the `external` library and other image externalization methods of section 7.

  To be more precise, the exported pdf documents will work correctly, but the `\includegraphics` command does not import the dynamic features. Please note that the exported pdf documents will only work if `\usepackage[pdftex]{eforms}` is placed *before* loading pgf, TikZ or pgfplots.

  As long as you are working on a draft version of your document, you might want to use

  ```
  \pgfkeys{/pgf/images/include external/.code={\href{file:#1}{\pgfimage{#1}}}}
  ```

  in your preamble. This will generate hyper links around the graphics files which link to the exported figures. Clicking on the hyper links opens the exported figure which, in turn, has been generated with the `clickable` library and allows dynamic features[42].

[40] These packages rely on LaTeX, so the library is only available for LaTeX, not for plain TeX or ConTeXt.

[41] In fact, they should be. I don't really know why they don't . . . any hint is welcome.

[42] This special treatment needs the external files in the same base directory as the main document, so this approach is most certainly *not* suitable for a final document.

- The library automatically calls `\begin{Form}` at `\begin{document}` and `\end{Form}` at the end of the document. This environment of `hyperref` is necessary for dynamic user interaction and should be kept in mind if the document contains other form elements.

Acknowledgements:

- I have used a javascript `sprintf` implementation of Kevin van Zonneveld [6] (the javascript API has only a limited set of conversions).

It is possible to customize `pgfplots.clickable` with several options.

`/pgfplots/clickable=true|false` (initially `true`)

Allows to disable the library for single plots.

`/pgfplots/annot/js fillColor={`⟨*javascript color*⟩`}` (initially `["RGB",1,1,.855]`)

Sets the background (fill) color of the short popup annotations.

Possible choices are `transparent`, gray, RGB or CMYK color specified as four–element–arrays of the form `["RGB",` ⟨*red*⟩`,`⟨*green*⟩`,`⟨*blue*⟩`]`. Each color component is between 0 and 1.

Again: this option is for Javascript. It is *not* possible to use colors as in PGF.

`/pgfplots/annot/point format={`⟨*sprintf-format*⟩`}` (initially `(%.1f,%.1f)`)

Allows to provide an `sprintf` format string which is used to fill the annotations with text. The first argument to `sprintf` is the x-coordinate and the second argument is the y-coordinate.

The `every semilogx axis`, `every semilogy axis` and `every loglog axis` styles have been updated to

```
\pgfplotsset{
    every semilogy axis/.append style={/pgfplots/annot/point format={(\% .1f,\%.1e)}},
    every semilogx axis/.append style={/pgfplots/annot/point format={(\% .1e,\%.1f)}},
    every loglog axis/.append style={/pgfplots/annot/point format={(\% .1e,\%.1e)}}
}
```

such that every logarithmic coordinate is displayed in scientific format.

`/pgfplots/annot/slope format={`⟨*sprintf-format*⟩`}` (initially `%.1f*x %+.1f`)

Allows to provide an `sprintf` format string which is used to fill the slope–annotation with text. The first argument is the slope and the second the line offset.

`/pgfplots/annot/printable=true|false` (initially `false`)

Allows to configure whether the small annotations will be printed. Otherwise, they are only available on screen.

`/pgfplots/annot/font={`⟨*javascript font name*⟩`}` (initially `font.Times`)

Allows to choose a javascript font for the annotations. Possible choices are limited to what javascript accepts (which is *not* the same as LaTeX). The default fonts and its names are shown below.

Font Name	Name in Javascript
Times-Roman	font.Times
Times-Bold	font.TimesB
Times-Italic	font.TimesI
Times-BoldItalic	font.TimesBI
Helvetica	font.Helv
Helvetica-Bold	font.HelvB
Helvetica-Oblique	font.HelvI
Helvetica-BoldOblique	font.HelvBI
Courier	font.Cour
Courier-Bold	font.CourB
Courier-Oblique	font.CourI
Courier-BoldOblique	font.CourBI
Symbol	font.Symbol
ZapfDingbats	font.ZapfD

/pgfplots/annot/textSize={⟨*Size in Point*⟩} (initially 11)

Sets the text size of annotations in points.

5.2.1 Using the Clickable Library in Other Contexts

This library provides essentially one command, \pgfplotsclickablecreate which creates a clickable area of predefined size, combined with javascript interaction code. It can be used independently of PGFPLOTS.

\pgfplotsclickablecreate[⟨*required key-value-options*⟩]

Creates an area which is clickable. A click produces a popup which contains information about the point under the cursor.

The complete (!) context needs to be provided using key-value-pairs, either set before calling this method of inside of [⟨*required key-value-options*⟩].

This command actually creates an AcroForm which invokes javascript whenever it is clicked. A javascript Object is created which represents the context (axis limits and options). This javascript object is available at runtime.

This method is public and it is *not* restricted to PGFPLOTS. The PGFPLOTS hook simply initialises the required key-value-pairs.

This method does not draw anything. It initialises only a clickable area and javascript code.

The required key-value-pairs are documented below.

Attention: Complete key-value validation is *not* performed here. It can happen that invalid options will produce javascript bugs when opened with Acrobat Reader. Use the javascript console to find them.

All options described in the following are only interesting for users who intend to use this library without PGFPLOTS.

/pgfplots/annot/width={⟨*dimension*⟩} (initially -)

This required key communicates the area's width to \pgfplotsclickablecreate. It must be a TeX dimension like 5cm.

/pgfplots/annot/height={⟨*dimension*⟩} (initially -)

This required key communicates the area's height to \pgfplotsclickablecreate. It must be a TeX dimension like 5cm.

/pgfplots/annot/jsname={⟨*string*⟩} (initially -)

This required key communicates a unique identifier to \pgfplotsclickablecreate. This identifier is used to identify the object in javascript, so there can't be more than one of them. If it is empty, a default identifier will be created.

/pgfplots/annot/xmin={⟨*number*⟩}
/pgfplots/annot/xmax={⟨*number*⟩}
/pgfplots/annot/ymin={⟨*number*⟩}
/pgfplots/annot/ymax={⟨*number*⟩} (initially empty)

These required keys communicate the axis limits to \pgfplotsclickablecreate. They should be set to numbers which can be assigned to a javascript floating point number (standard IEEE double precision).

5.3 Units in Labels

by Nick Papior Andersen

```
\usepgfplotslibrary{units} % LaTeX and plain TeX
\usepgfplotslibrary[units] % ConTeXt
\usetikzlibrary{pgfplots.units} % LaTeX and plain TeX
\usetikzlibrary[pgfplots.units] % ConTeXt
```

A library which allows to use automatic typesetting of units in labels. The library utilizes different keys to typeset the final output in a consistent way. Calling one of the commands automatically sets the key 'use units=true' so one has not to worry of this.

PGFPLOTS has the capability of supporting units. This provides quick customization of the plot as well as the addition of units in labels.

Loading the library automatically enables the typesetting of units in labels. Currently it only supports predefined SI units but a per-user customization is also implemented such that it can be used in any way you like.

First the key which enables you to switch on/off the unit system.

/pgfplots/use units={⟨*boolean*⟩} (initially true)

This key simply enables PGFPLOTS to use what is described next. This key will be set to true if you load the library. You can use this to temporarily determine whether the unit library should be used in plots.

/pgfplots/x unit={⟨*unit*⟩} (initially empty)
/pgfplots/y unit={⟨*unit*⟩} (initially empty)
/pgfplots/z unit={⟨*unit*⟩} (initially empty)

These keys sets the unit in their respective axis. In SI units you could for instance set the x unit in Newton as x unit=N.

/pgfplots/x unit prefix={⟨*prefix*⟩} (initially empty)
/pgfplots/y unit prefix={⟨*prefix*⟩} (initially empty)
/pgfplots/z unit prefix={⟨*prefix*⟩} (initially empty)

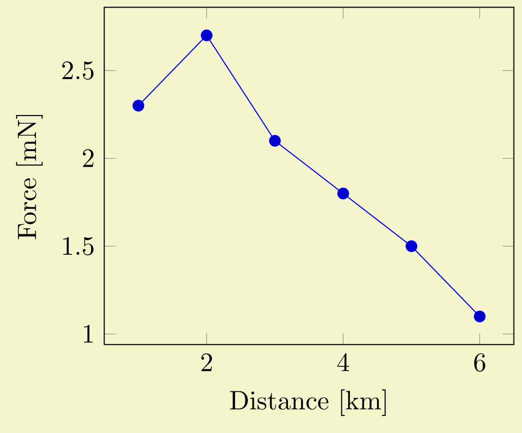
These keys sets the prefix of the unit. If a value on the y axis is in kilo you would set the y unit prefix=k. Prefix will be typeset in front of the unit.

This command will not intervene with the basis of the axis system. I.e. a prefix as just mentioned will not divide every y axis number by 1000. In order to do this see key ⟨*axis*⟩ SI prefix, see Section 5.3.1.

Notice that if the ⟨*axis*⟩ unit isn't set the entire unit will not be typeset.

Remarks: Remember that all typesetting of labels occur within a $$ environment. Therefore one can use \frac and other mathematics commands.

Often one just have to utilize the above mentioned keys. It is the basis of the unit typesetting system provided by PGFPLOTS.



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
  \begin{axis}[use units,
    x unit=m,x unit prefix=k,
    y unit=N,y unit prefix=m,
    xlabel=Distance,ylabel=Force]
    \addplot coordinates {
        (1,2.3)
        (2,2.7)
        (3,2.1)
        (4,1.8)
        (5,1.5)
        (6,1.1)
    };
  \end{axis}
\end{tikzpicture}
```

Below is an example of what would be yielded according to the styles

```
%  x label becomes ‘‘Temperature [T]’’, y label becomes ‘‘Nothing’’
\pgfplotsset{use units,x unit=T,xlabel=Temperature,ylabel=Nothing}
%  x label becomes ‘‘Temperature’’, y label becomes ‘‘Nothing’’
\pgfplotsset{use units,x unit prefix=m,xlabel=Temperature,ylabel=Nothing}
```

Notice the second example. Only setting the prefix will not activate the unit typesetting. Therefore one should ensure to use the x unit key if the typesetting of the labels should be done.

For typesetting the units one can also change the appearance. For instance one might not like the square brackets which surround the unit. These can luckily be changed using the below keys.

/pgfplots/unit marking pre={⟨*pre*⟩} (initially \left[)
/pgfplots/unit marking post={⟨*post*⟩} (initially \right])
/pgfplots/unit markings=parenthesis|square brackets|slash space (initially square brackets)

These keys sets the surroundings of the unit. The initial yields $\left[\frac{1}{2}\right]$ such that you can typeset fractions in units. Be aware that you can only obtain large fractions if you use \dfrac. These can easily be set using the option key unit markings where the options typesets as the following

```
\pgfplotsset{x unit=T,unit markings=parenthesis} %  x unit becomes `` \left(T\right)''
\pgfplotsset{x unit=T,unit markings=square brackets} %  x unit becomes `` \left[T\right]''
\pgfplotsset{x unit=T,unit markings=slash space} %  x unit becomes `` / T''
```

Notice that all typesetting of units first inserts a space and then the unit marking pre code.

Of course you can just manually set each of them with the unit marking pre and unit marking post keys. Just remember that they are typeset within a $$.

One will typically typeset the unit with a specific font. To do so an option of changing the typesetting command is supplied.

/pgfplots/unit code/.code 2 args={⟨...⟩}

This can be utilized to great extend. As a default the typesetting of the units is as \mathrm{⟨*unit prefix*⟩⟨*unit*⟩}. But if one for instance wishes to utilize the package siunitx, which has great capabilities in typesetting both units, numbers and angles, one can just set the key as

```
\pgfplotsset{unit code/.code 2 args={\si{#1#2}}}
```

which would yield the unit as \si{⟨*unit prefix*⟩⟨*unit*⟩}.

The first argument is typeset as ⟨*unit prefix*⟩ and the second argument is ⟨*unit*⟩.

The most important thing is that the command needs exactly two arguments. So if you would like a command that typesets the prefix in bold face and the unit in normal roman font you should call

```
\pgfplotsset{unit code/.code 2 args={\mathbf{#1}\mathrm{#2}}
```

5.3.1 Preset SI prefixes

To support the SI system a number of preset keys are defined. This should yield a more intuitive way of supplying the prefix as well as add some more functionality. For instance it provides an easy scaling mechanism.

/pgfplots/x SI prefix=yocto|...|milli|centi|deci|deca|hecto|kilo|...|yotta (initially none)
/pgfplots/y SI prefix=yocto|...|milli|centi|deci|deca|hecto|kilo|...|yotta (initially none)
/pgfplots/z SI prefix=yocto|...|milli|centi|deci|deca|hecto|kilo|...|yotta (initially none)
/pgfplots/change x base=true|false (initially false)
/pgfplots/change y base=true|false (initially false)
/pgfplots/change z base=true|false (initially false)

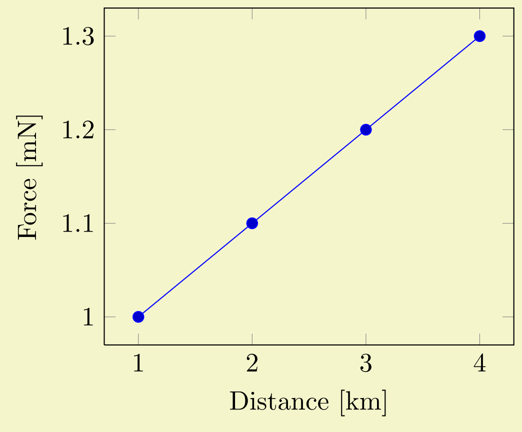
These keys sets the prefix of the unit. The allowed prefixes are:

Prefix	Power
yocto	−24
zepto	−21
atto	−18
femto	−15
pico	−12
nano	−9
micro	−6
milli	−3
centi	−2
deci	−1

Prefix	Power
deca	1
hecto	2
kilo	3
mega	6
giga	9
tera	12
peta	15
exa	18
zetta	21
yotta	24

As well as resetting the base of the axis if the key `change` ⟨*axis*⟩ `base=true`. Just **remember** to set the `change` ⟨*axis*⟩ `base` before using the ⟨*axis*⟩ `SI prefix` key.

See the utilization as in the example below.



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
  \begin{axis}[change x base,
    x SI prefix=kilo,x unit=m,
    y SI prefix=milli,y unit=N,
    xlabel=Distance,ylabel=Force]
    \addplot coordinates {
        (1000,1)
        (2000,1.1)
        (3000,1.2)
        (4000,1.3)
    };
  \end{axis}
\end{tikzpicture}
```

Notice that the `x axis` has changed base without displaying the $\cdot 10^3$. This is done by using the key `change x base`. Even though you have used the key `y SI prefix=milli` the base isn't changed on the `y axis`. Try adding `change y base` just after `change x base` and see the result!

The above keys are the easy implementation of the base change. Below is a further customization of the base change. It makes it easy to implement a prefix with a custom base change.

`/pgfplots/axis base prefix`=axis `{`⟨*axis*⟩`}` base `{`⟨*base*⟩`}` prefix `{`⟨*prefix*⟩`}` (initially `empty`)

One can utilize this key to customize further of the base and setting the prefix.

```
\pgfplotsset{change x base,axis base prefix={axis x base -3 prefix k}}
\pgfplotsset{change x base,x SI prefix=kilo}
```

The above two commands are thus equivalent. Remember that the base should operate in opposite of prefix!

5.4 Grouping plots

by Nick Papior Andersen

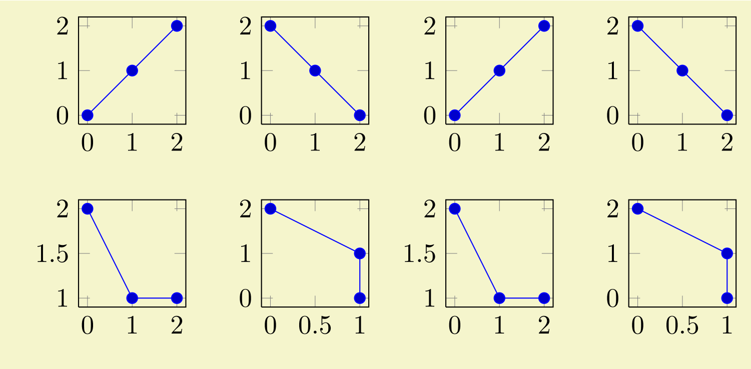
```
\usepgfplotslibrary{groupplots} % LaTeX and plain TeX
\usepgfplotslibrary[groupplots] % ConTeXt
\usetikzlibrary{pgfplots.groupplots} % LaTeX and plain TeX
\usetikzlibrary[pgfplots.groupplots] % ConTeXt
```

A library which allows the user to typeset several plots in a matrix like structure. Often one has to compare two plots to one another, or you simply need to display two plots in conjunction with each other. Either way the following section describes this library which makes this typesetting easy.

```
\begin{groupplot}[⟨options⟩]
  ⟨environment contents⟩
\end{groupplot}
```

Once you have loaded the `groupplots` library you will gain access to this environment. This environment is limited to the same restrictions as the `axis` environment. It actually utilizes this environment so consider it as an extension of this. What is important to note is that `[`⟨*options*⟩`]` gets applied to all plots in the entire environment. This can be really handy when you need the same `xmin`, `xmax`, `ymin` and `ymax`.

With such an environment one can typeset plots in matrix like styles

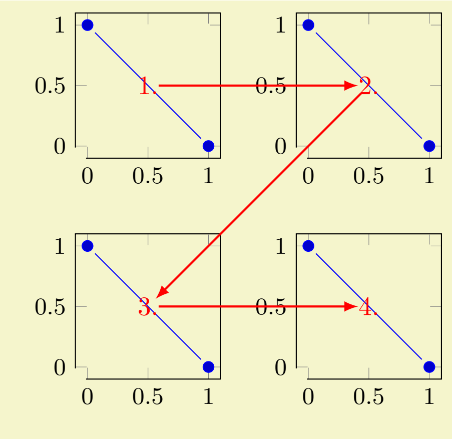


```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
%  Example using groupplots library
\begin{tikzpicture}
  \begin{groupplot}[group style={group size=2 by 2},height=3cm,width=3cm]
    \nextgroupplot
    \addplot coordinates {(0,0) (1,1) (2,2)};
    \nextgroupplot
    \addplot coordinates {(0,2) (1,1) (2,0)};
    \nextgroupplot
    \addplot coordinates {(0,2) (1,1) (2,1)};
    \nextgroupplot
    \addplot coordinates {(0,2) (1,1) (1,0)};
  \end{groupplot}
\end{tikzpicture}
%  Same example created as done without the library
\begin{tikzpicture}
  \begin{axis}[name=plot1,height=3cm,width=3cm]
    \addplot coordinates {(0,0) (1,1) (2,2)};
  \end{axis}
  \begin{axis}[name=plot2,at={($(plot1.east)+(1cm,0)$)},anchor=west,height=3cm,width=3cm]
    \addplot coordinates {(0,2) (1,1) (2,0)};
  \end{axis}
  \begin{axis}[name=plot3,at={($(plot1.south)-(0,1cm)$)},anchor=north,height=3cm,width=3cm]
    \addplot coordinates {(0,2) (1,1) (2,1)};
  \end{axis}
  \begin{axis}[name=plot4,at={($(plot2.south)-(0,1cm)$)},anchor=north,height=3cm,width=3cm]
    \addplot coordinates {(0,2) (1,1) (1,0)};
  \end{axis}
\end{tikzpicture}
```

The equivalent code is seen as the second example and it is clear that you have to type a lot less. So how do you use it? First of all you need to utilize the new environment `groupplot`. Within this environment the following command works.

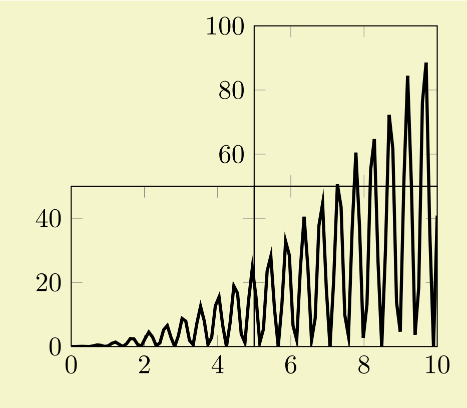
`\nextgroupplot[`⟨*options*⟩`]` ⟨*normal plot commands*⟩

This command shifts the placement of the plot. Therefore one should always start the environment `groupplot` with the command `\nextgroupplot` in order to create the first plot. The `[`⟨*options*⟩`]` are the options that are supplied to the following plot commands until the next `\nextgroupplot` command is seen by TEX. The order in which figures are typeset are as seen in the next example.



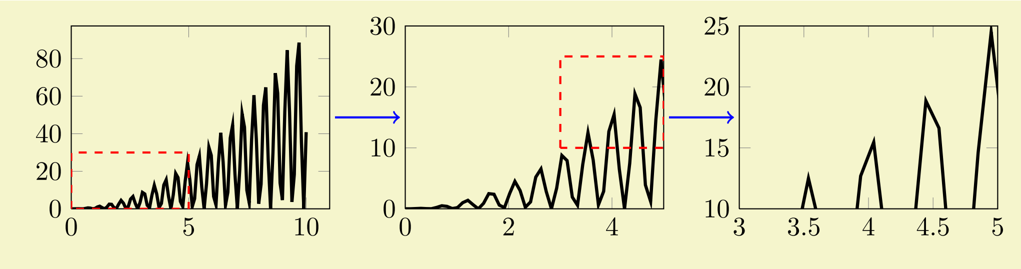
```
\begin{tikzpicture}[shorten >=4pt,shorten <=4pt]
  \begin{groupplot}[group style={group size=2 by 2},
    height=3.5cm,width=3.5cm,/tikz/font=\small]
    \nextgroupplot% 1
    \addplot coordinates {(0,1) (1,0)};
    \nextgroupplot% 2
    \addplot coordinates {(0,1) (1,0)};
    \nextgroupplot% 3
    \addplot coordinates {(0,1) (1,0)};
    \nextgroupplot% 4
    \addplot coordinates {(0,1) (1,0)};
  \end{groupplot}
  \draw[thick,>=latex,->,red]
    (group c1r1.center) node {1.}  --
    (group c2r1.center) node {2.};
  \draw[thick,>=latex,->,red]
    (group c2r1.center)  --
    (group c1r2.center) node {3.};
  \draw[thick,>=latex,->,red]
    (group c1r2.center)  --
    (group c2r2.center) node {4.};
\end{tikzpicture}
```

The plot first fills the first row, then the next row and so on. Just like a table, thus the names `group c`⟨*column*⟩`r`⟨*row*⟩. The power of the `groupplot` is to quickly create an aligned structure of plots. But you can also utilize it to structure data more creatively. Consider the next example.



```
\begin{tikzpicture}
 \begin{groupplot}[group style={group size=2 by 2,
     horizontal sep=0pt,vertical sep=0pt,
     xticklabels at=edge bottom},
     xmin=0,ymin=0,
     height=3.7cm,width=4cm,no markers]
  \nextgroupplot[group/empty plot]
  \nextgroupplot[xmin=5,xmax=10,ymin=50,ymax=100]
  \addplot[very thick] file {plotdata/group-1.dat};
  \nextgroupplot[xmax=5,ymax=50]
  \addplot[very thick] file {plotdata/group-1.dat};
  \nextgroupplot[xmin=5,xmax=10,ymax=50,yticklabels={}]
  \addplot[very thick] file {plotdata/group-1.dat};
 \end{groupplot}
\end{tikzpicture}
```

Or for instance zooming in on data as in the next example.



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
  \begin{groupplot}[group style={group size=3 by 1},xmin=0,ymin=0,height=4cm,width=5cm,no markers]
    \nextgroupplot
    \addplot[very thick] file {plotdata/group-1.dat};
    \draw[red,dashed,thick] (axis cs:0,0) rectangle (axis cs:5,30);
    \nextgroupplot[xmax=5,ymax=30]
    \addplot[very thick] file {plotdata/group-1.dat};
    \draw[red,dashed,thick] (axis cs:3,10) rectangle (axis cs:5,25);
    \nextgroupplot[xmin=3,xmax=5,ymin=10,ymax=25]
    \addplot[very thick] file {plotdata/group-1.dat};
  \end{groupplot}
  \draw[thick,blue,->,shorten >=2pt,shorten <=2pt]
          (group c1r1.east) -- (group c2r1.west);
  \draw[thick,blue,->,shorten >=2pt,shorten <=2pt]
          (group c2r1.east) -- (group c3r1.west);
\end{tikzpicture}
```

5.4.1 Grouping options

Here the various available options of the library resides.

/pgfplots/**group style**=group/.cd,{⟨*options*⟩}

This key allows one to faster reach the group options. Some of the options are very similar to the normal commands so they are supplied in another subdirectory `group`. So the following are equivalent.

```
\pgfplotsset{group style={a=2,b=3}}
\pgfplotsset{group/a=2,group/b=3}
\pgfplotsset{group/.cd,a=2,b=3}
```

It can thus decrease the size of your options with out destroying the clearness of where options belong.

All the following keys are in the subdirectory `group`.

/pgfplots/**group/group size**=⟨*columns*⟩ by ⟨*rows*⟩ (initially `1 by 1`)
/pgfplots/**group/columns**=⟨*columns*⟩ (initially `1`)
/pgfplots/**group/rows**=⟨*rows*⟩ (initially `1`)

These keys determines the total number of plots that can be in one environment `groupplot`. It is thus important not to add more `\nextgroupplot` in the environment than ⟨*columns*⟩×⟨*rows*⟩. This is critical to set if one uses more than 1 more plot. As the key `group size` uses `columns` and `rows` you should stick to either `group size` or both `columns` and `rows`.

/pgfplots/**group/horizontal sep**=⟨*dimension*⟩ (initially `1cm`)
/pgfplots/**group/vertical sep**=⟨*dimension*⟩ (initially `1cm`)

The spacing between the plots in the horizontal and vertical direction, respectively. If you thus want them to be *glued* together you should set them both to a length of `0pt`.

/pgfplots/**group/every plot/.style**={⟨*style*⟩} (initially `empty`)

This style is used on every plot as the first style. It is thus equivalent as ⟨*options*⟩ in the `groupplot` environment.

/pgfplots/**group/xticklabels at**=`all|edge top|edge bottom` (initially `all`)
/pgfplots/**group/yticklabels at**=`all|edge left|edge right` (initially `all`)

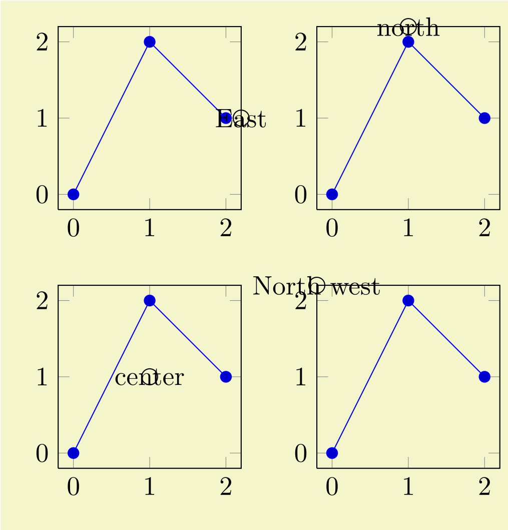
In order to determine which plots get tick labels typeset one can use these keys. By default all axis gets typeset normally and thus have both x and y axis tick labels. If one sets

```
\pgfplotsset{group/xticklabels at=edge bottom,group/yticklabels at=edge right}
```

only the bottom row gets tick labels on the x axis and only the last column gets tick labels on the y axis on their right side. These keys are specially handy when using *glued* plots.

/pgfplots/**group/group name**={⟨*name*⟩} (initially `group`)

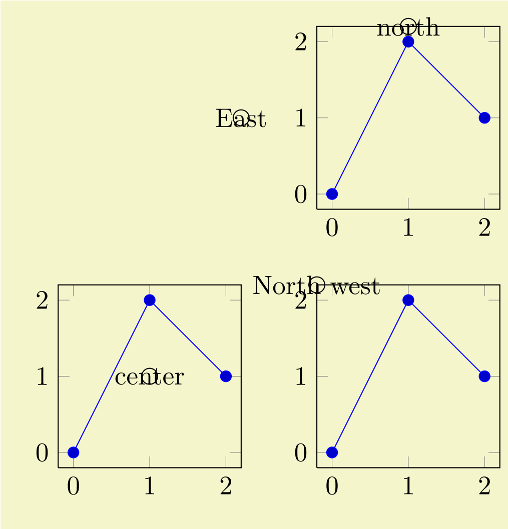
This sets what you can refer the plots to after typesetting. Thus you can use their anchors later. See the following example



```
\begin{tikzpicture}
  \begin{groupplot}[group style={
        group name=my plots,group size=2 by 2},
    width=4cm,height=4cm]
    \nextgroupplot
    \addplot coordinates{(0,0) (1,2) (2,1)};
    \nextgroupplot
    \addplot coordinates{(0,0) (1,2) (2,1)};
    \nextgroupplot
    \addplot coordinates{(0,0) (1,2) (2,1)};
    \nextgroupplot
    \addplot coordinates{(0,0) (1,2) (2,1)};
  \end{groupplot}
  \draw (my plots c1r1.east)
      circle (3pt) node {East};
  \draw (my plots c2r1.north)
      circle (3pt) node {north};
  \draw (my plots c1r2.center)
      circle (3pt) node {center};
  \draw (my plots c2r2.north west)
      circle (3pt) node {North west};
\end{tikzpicture}
```

/pgfplots/group/empty plot/.style={⟨*style*⟩} (initially /pgfplots/hide axis)

This key can be used as an option to the command \nextgroupplot. This makes the next plot invisible (only the axes) but maintains it anchors and name. If you want it to behave in another style then you can redefine it. Consider the same example as before.



```
\begin{tikzpicture}
  \begin{groupplot}[group style={
        group name=my plots,group size=2 by 2},
    width=4cm,height=4cm]
    \nextgroupplot[group/empty plot]
    \nextgroupplot
    \addplot coordinates{(0,0) (1,2) (2,1)};
    \nextgroupplot
    \addplot coordinates{(0,0) (1,2) (2,1)};
    \nextgroupplot
    \addplot coordinates{(0,0) (1,2) (2,1)};
  \end{groupplot}
  \draw (my plots c1r1.east)
      circle (3pt) node {East};
  \draw (my plots c2r1.north)
      circle (3pt) node {north};
  \draw (my plots c1r2.center)
      circle (3pt) node {center};
  \draw (my plots c2r2.north west)
      circle (3pt) node {North west};
\end{tikzpicture}
```

Notice that you need to call a \nextgroupplot again after to jump to the next plot.

5.5 Image Externalization

```
\usepgfplotslibrary{external} % LaTeX and plain TeX
\usepgfplotslibrary[external] % ConTeXt
\usetikzlibrary{pgfplots.external} % LaTeX and plain TeX
\usetikzlibrary[pgfplots.external] % ConTeXt
```

The external library offers a convenient method to export every single tikzpicture into a separate .pdf (or .eps). Later runs of LaTeX will simply include these graphics, thereby reducing typesetting time considerably.

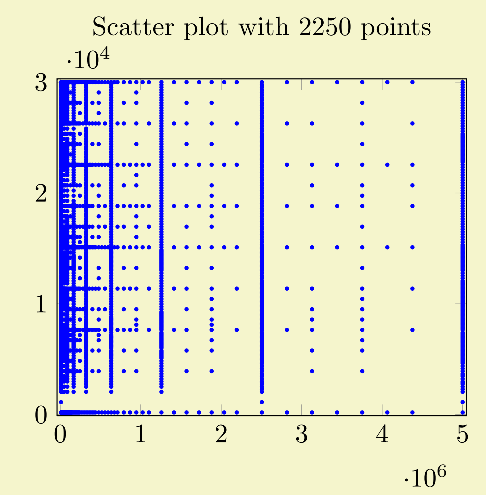
This library is documented in more detail in section 7.1 "Export to PDF/EPS".

The external library has been written by Christian Feuersänger (author of PGFPLOTS). It has been contributed to Ti*k*Z as general purpose library, so the reference documentation along with all tweaks can be found in [5, Section "Externalization Library"]. The command \usepgfplotslibrary{external} is actually just a wrapper which loads \usetikzlibrary{external} or, if this library does not yet exist because the installed PGF has at most version 2.00, it will load a copy which is shipped with PGFPLOTS.

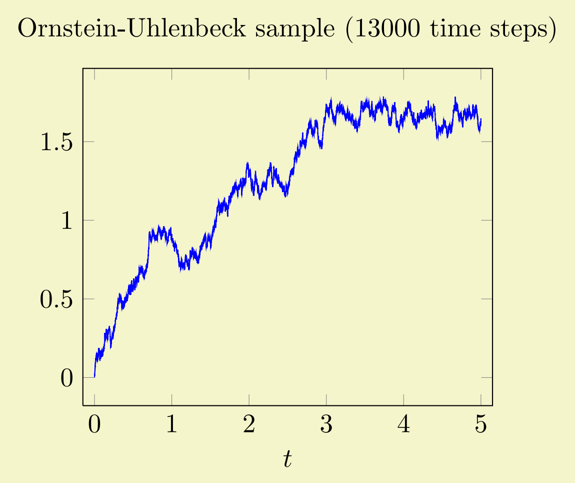
6 Memory and Speed considerations

6.1 Memory Limits of TEX

PGFPLOTS can typeset plots with several thousand points if memory limits of TEX are configured properly. Its runtime is roughly proportional to the number of input points[43].



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}[
    enlargelimits=0.01,
    title style={yshift=5pt},
    title=Scatter plot with $2250$ points]

\addplot[blue,
    mark=*,only marks,mark options={scale=0.3}]
    file[skip first]
    {plotdata/pgfplots_scatterdata3.dat};

\end{axis}
\end{tikzpicture}
```



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
\begin{axis}[
    enlarge x limits=0.03,
    title=Ornstein-Uhlenbeck sample
        ($13000$ time steps),
    xlabel=$t$]

\addplot[blue] file {plotdata/ou.dat};
\end{axis}
\end{tikzpicture}
```

PGFPLOTS relies completely on TEX to do all typesetting. It uses the front-end-layer and basic layer of PGF to perform all drawing operations. For complicated plots, this may take some time, and you may want to read section 7 for how to write single figures to external graphics files. Externalization is the best way to reduce typesetting time.

However, for large scale plots with a lot of points, limitations of TEX's capacities are reached easily.

6.2 Memory Limitations

The default settings of most TEX-distributions are quite restrictive, so it may be necessary to adjust them.

Usually, the log–file or the final error message contains a summary about the used resources, giving a hint which parameter needs to be increased.

6.2.1 MikTEX

For MikTEX, memory limits can be increased in two ways. The first is to use command line switches:

```
pdflatex
    --stack-size=n --save-size=n
    --main-memory=n --extra-mem-top=n --extra-mem-bot=n
    --pool-size=n --max-strings=n
```

[43] In fact, the runtime is pseudo–linear: starting with about 100,000 points, it will become quadratic. This limitation applies to the path length of PGF paths as well. Furthermore, the linear runtime is not possible yet for stacked plots.

Experiment with these settings if MikTEX runs out of memory. Usually, one doesn't invoke `pdflatex` manually: there is a development aid which does all the invocations, so this one needs to be adjusted.

Sometimes it might be better to adjust the MikTEX configuration file permanently, for example to avoid reconfiguring the TEX development program. This can be realized using the command

```
initexmf --edit-config-file=pdflatex
```

which can be typed either on a command prompt in Windows or using Start ≫ Execute. As a result, an editor will be opened with the correct config file. A sample config file could be

```
main_memory=90000000
save_size=80000
```

or any of the config file entries which are listed below can be entered. Thanks to "LeSpocky" for his documentation in
http://blog.antiblau.de/2009/04/21/speicherlimits-von-miktex-erhoehen.

6.2.2 TEXLive or similar installations

For Unix installations, one needs to adjust config files. This can be done as follows:

1. Locate `texmf.cnf` on your system. On my Ubuntu installation, it is in
   `/usr/share/texmf/web2c/texmf.cnf`.

2. Either change `texmf.cnf` directly, or copy it to some convenient place. If you copy it, here is how to proceed:

   - keep only the changed entries in your local copy to reduce conflicts. TEX will always read *all* config files found in its search path.
   - Adjust the search path to find your local copy. This can be done using the environment variable `TEXMFCNF`. Assuming your local copy is in `~/texmf/mytexcnf/texmf.cnf`, you can write

     ```
     export TEXMFCNF=~/texmf/mytexcnf:
     ```

     to search first in your directory, then in all other system directories.

3. You should change the entries

   ```
   main_memory = n
   extra_mem_top = n
   extra_mem_bot = n
   max_strings = n
   param_size = n
   save_size = n
   stack_size = n
   ```

   The log–file usually contains information about the parameter which needs to be enlarged.

An example of this config file thing is shown below. It changes memory limits.

1. Create the file `~/texmf/mytexcnf/texmf.cnf` (and possibly the paths as well).

   ```
   %  newly created file ~/texmf/mytexcnf/texmf.cnf:
   %  If you want to change some of these sizes only for a certain TeX
   %  variant, the usual dot notation works, e.g.,
   %  main_memory.hugetex = 20000000
   main_memory = 230000000 %  words of inimemory available; also applies to inimf&mp
   extra_mem_top = 10000000     %  extra high memory for chars, tokens, etc.
   extra_mem_bot = 10000000     %  extra low memory for boxes, glue, breakpoints, etc.
   save_size = 150000    %  for saving values outside current group
   stack_size = 150000    %  simultaneous input sources

   %  Max number of characters in all strings, including all error messages,
   %  help texts, font names, control sequences.  These values apply to TeX and MP.
   % pool_size = 1250000
   %  Minimum pool space after TeX/MP's own strings; must be at least
   %  25000 less than pool_size, but doesn't need to be nearly that large.
   % string_vacancies = 90000
   %  Maximum number of strings.
   % max_strings = 100000
   %  min pool space left after loading .fmt
   % pool_free = 47500
   ```

2. Run `texhash` such that TeX updates its `~/texmf/ls-R` database.

3. Create the environment variable `TEXMFCNF` and assign the value '`~/texmf/mytexcnf:`' (including the trailing '`:`'!). For my linux system, this can be done using by adding

   ```
   export TEXMFCNF=~/texmf/mytexcnf:
   ```

   to `~/.bashrc`.

Unfortunately, TeX does not allow arbitrary memory limits, there is an upper bound hard coded in the executables.

6.3 Reducing Typesetting Time

pgfplots does a lot of computations ranging from abstract coordinate computations to low level `.pdf` drawing commands (realized by pgf). For complex plots, this may take a considerable time – especially for 3D plots.

One possibility to reduce typesetting time is to tell pgf to generate single, temporary `.pdf` (or `.eps`) documents for a subset (or all) graphics in one run and re-use these temporary images in successive runs. For pgfplots, this is the most effective way to reduce typesetting time. It can be accomplished using the `external` library described in section 7.1.

7 Import/Export From Other Formats

This section contains information of how to single pictures into separate PDF graphics files (or EPS graphics files). Furthermore, it explains a matlab (tm) script which allows to convert from matlab to PGFPLOTS.

7.1 Export to PDF/EPS

It is possible to export images to single PDF-documents using routines of PGF and/or Ti*k*Z.

7.1.1 Using the Automatic Externalization Framework of Ti*k*Z

```
\usepgfplotslibrary{external} % LaTeX and plain TeX
\usepgfplotslibrary[external] % ConTeXt
\usetikzlibrary{pgfplots.external} % LaTeX and plain TeX
\usetikzlibrary[pgfplots.external] % ConTeXt
```

The `external` library offers a convenient method to export every single `tikzpicture` into a separate `.pdf` (or `.eps`). Later runs of LaTeX will simply include these graphics, thereby reducing typesetting time considerably.

The library can also be used to submit documents to authors who do not even have PGFPLOTS or Ti*k*Z installed.

Technical foreword: The `external` library has been written by Christian Feuersänger (author of PGFPLOTS). It has been contributed to Ti*k*Z as general purpose library, so the reference documentation along with all tweaks can be found in [5, Section “Externalization Library”]. The command `\usepgfplotslibrary{external}` is actually just a wrapper which loads `\usetikzlibrary{external}` or, if this library does not yet exist because the installed PGF has at most version 2.00, it will load a copy which is shipped with PGFPLOTS.

The `external` library has been designed such that *no changes* to the document as such are necessary. The idea is as follows:

1. Every `\begin{tikzpicture}` ... `\end{tikzpicture}` gets a file name. The file name can be assigned manually with `\tikzsetnextfilename{`⟨*output file name*⟩`}` or automatically, in which case ⟨*tex file name*⟩`-figure`⟨*number*⟩ is used with an increasing ⟨*number*⟩.
2. The library writes the resulting images using system calls of the form `pdflatex --jobname {`⟨*output file name*⟩`}` automatically, using the write18 system call of TeX. It is the same framework which can be used to call `gnuplot`.

The only steps which are necessary is to use

```
\usepgfplotslibrary{external}
\tikzexternalize
```

somewhere in your document’s preamble. No further modification to the document is necessary. Suppose we have a file called `test.tex`:

```
\documentclass{article}

\usepackage{pgfplots}

\usepgfplotslibrary{external}
\tikzexternalize%  activate externalization!

\begin{document}
    \begin{figure}
        \begin{tikzpicture}
        \begin{axis}
            \addplot {x^2};
        \end{axis}
        \end{tikzpicture}
    \caption{Our first external graphics example}
    \end{figure}

    \begin{figure}
        \begin{tikzpicture}
        \begin{axis}
            \addplot {x^3};
        \end{axis}
        \end{tikzpicture}
    \caption{A second graphics}
    \end{figure}
\end{document}
```

To enable the system calls, we type

```
pdflatex -shell-escape test
```

and LaTeX will now generate the required graphics files `test-figure0.pdf` and `test-figure1.pdf` automatically. Any further call to `pdflatex` will simply use `\includegraphics` and the `tikzpicture`s as such are no longer considered (you need a different command line switch for MikTeX, see the `shell escape` option).

If a figure shall be remade, one can simply delete all or selected graphics files and re-generate them. Alternatively, one can use the command `\tikzset{external/force remake}` somewhere in the document to remake every following picture automatically.

There are three ways to modify the file names of externalized figures:

- Changing the overall file name using a `prefix`,
- Changing the file name for a single figure using `\tikzsetnextfilename`,
- Changing the file name for a restricted set of figures using `figure name`.

`/tikz/external/prefix={`⟨*file name prefix*⟩`}` (initially `empty`)

A shortcut for `\tikzsetexternalprefix{`⟨*file name prefix*⟩`}`, see below.

`\tikzsetexternalprefix{`⟨*file name prefix*⟩`}`

Assigns a common prefix used by all file names. For example,

```
\tikzsetexternalprefix{figures/}
```

will prepend `figures/` to every external graphics file name.

`\tikzsetnextfilename{`⟨*file name*⟩`}`

Sets the file name for the *next* Ti*k*Z picture or `\tikz` short command. It will *only* be used for the next picture.

Pictures for which no explicit file name has been set will get automatically generated file names.

Please note that `prefix` will still be prepended to `{`⟨*file name*⟩`}`.

```
\documentclass{article}
%  main document, called main.tex
\usepackage{tikz}

\usepgfplotslibrary{external}
\tikzexternalize[prefix=figures/]%  activate with a name prefix

\begin{document}

\tikzsetnextfilename{firstplot}
\begin{tikzpicture} %  will be written to ’figures/firstplot.pdf’
\begin{axis}
    \addplot {x};
\end{axis}
\end{tikzpicture}

\begin{tikzpicture} %  will be written to ’figures/main-figure0.pdf’
   \draw[help lines] (0,0) grid (5,5);
\end{tikzpicture}
\end{document}
```

```
pdflatex -shell-escape main
```

/tikz/external/figure name={⟨*name*⟩}

Same as \tikzsetfigurename{⟨*name*⟩}.

\tikzsetfigurename{⟨*name*⟩}

Changes the names of *all* following figures. It is possible to change figure name during the document using \tikzset{external/figure name={⟨*name*⟩}}. A unique counter[44] will be used for each different {⟨*name*⟩}, and each counter will start at 0.

The value of prefix will be applied after figure name has been evaluated.

```
\documentclass{article}
%  main document, called main.tex
\usepackage{tikz}

\usepgfplotslibrary{external}
\tikzexternalize%  activate externalization!

\begin{document}

%  will be written to ’main-figure0.pdf’
\begin{tikzpicture}
\begin{semilogyaxis}
    \addplot {exp(x)};
\end{semilogyaxis}
\end{tikzpicture}

{
  \tikzset{external/figure name={subset_}}
  A simple image is \tikz \fill (0,0) circle(5pt);. %  will be written to ’subset_0.pdf’

  \begin{tikzpicture} %  will be written to ’subset_1.pdf’
     \begin{axis}
         \addplot {x^2};
     \end{axis}
  \end{tikzpicture}
}%  here, the old file name will be restored:

\begin{tikzpicture} %  will be written to ’main-figure1.pdf’
   \begin{axis}
           \addplot[domain=1e-3:100] {1/x};
    \end{axis}
\end{tikzpicture}
\end{document}
```

The scope of figure name ends with the next closing brace (as all values set by \tikzset do).

[44] These counters are stored into into different *macros*. In other words: no TEX register will be needed.

Remark: Use `\tikzset{external/figure name/.add={⟨prefix⟩}{⟨suffix⟩}}` to prepend a ⟨*prefix*⟩ and append a ⟨*suffix*⟩ to the actual value of `figure name`. Might be useful for something like

```
\tikzset{external/figure name=main}

%  uses main_0.pdf, main_1.pdf, ...

\section{The first section}
{\tikzset{external/figure name/.add={}{_firstsection}}
    ...
    %  uses main_firstsection_0.pdf, main_firstsection_1.pdf, ...
}

\section{The second section}
{\tikzset{external/figure name/.add={}{secondsection_}}
    ...
    %  uses main_secondsection_0.pdf, main_secondsection_1.pdf, ...
    \subsection{Second subsection}
    {\tikzset{external/figure name/.add={}{sub_}}
        ...
        %  uses main_secondsection_sub_0.pdf, main_secondsection_sub_1.pdf, ...
    }
    %  uses main_secondsection_2.pdf, main_secondsection_3.pdf, ...
}
```

`/tikz/external/system call={`⟨*template*⟩`}`

A template string used to generate system calls. Inside of `{`⟨*template*⟩`}`, the macro `\image` can be used as placeholder for the image which is about to be generated while `\texsource` contains the main file name (in truth, it contains `\input{`⟨*main file name*⟩`}`, but that doesn't matter).

The default is

```
\tikzset{external/system call={pdflatex \tikzexternalcheckshellescape -halt-on-error
    -interaction=batchmode -jobname "\image" "\texsource"}
```

where `\tikzexternalcheckshellescape` inserts the value of the configuration key `shell escape` if and only if the current document has been typeset with `-shell-escape`[45].

For `eps` output, you can (and need to) use

```
\tikzset{external/system call={latex \tikzexternalcheckshellescape -halt-on-error
    -interaction=batchmode -jobname "\image" "\texsource";
    dvips -o "\image".ps "\image".dvi}}
```

The argument `{`⟨*template*⟩`}` will be expanded using `\edef`, so any control sequences will be expanded. During this evaluation, '`\\`' will result in a normal backslash, '`\`'. Furthermore, double quotes '`"`', single quotes '`'`', semicolons and dashes '`-`' will be made to normal characters if any package uses them as macros. This ensures compatibility with the `german` package, for example.

`/tikz/external/shell escape={`⟨*command-line arg*⟩`}` (initially `-shell-escape`)

Contains the command line option for `latex` which enables the `\write18` feature. For TEX-Live, this is `-shell-escape`. For MikTEX, you should use `\tikzexternalize[shell escape=-enable-write18]`.

`/tikz/external/mode=convert with system call|list and make|...` (initially `convert with system call`)

This allows to change the default operation mode. There are a handful of choices possible, all of them are described in detail in [5, section "Externalization Library"]. The most useful ones are probably the initial configuration `convert with system call` and the specialized choice `list and make`.

The choice `list and make` configures the library to check if there are already external graphics and uses them. If there are no graphics, the library will *skip* the figure. However, it will also generate a `makefile` to generate the graphics, and a list of all required graphics files.

[45] Note that this is always true for the default configuration. This security consideration applies mainly for `mode=list and make` which will also work *without* shell escapes.

It is not required to use `make`: the library expects you to generate the images somehow and it doesn't care about the "how". Using `make -f` ⟨*name-of-tex-file*⟩`.makefile -j 2` allows parallel execution which might, indeed, be an option. Furthermore, the makefile also supports file dependencies: if one of your data tables has been updated, the external graphics will be remade automatically. pgfplots tells the external library about any file dependencies (input files and tables).

The two modes have the following characteristics:

1. `convert with system call` is automatic and does everything on–the–fly. However, it *can't* work with `\ref` and/or `\label` information in external pictures.
2. `list and make` requires either manual (by calling the system calls manually) or semi–automatic conversion (using the generated ⟨*main*⟩`.makefile`), and multiple runs of `pdflatex`. The generated Makefile can be processed in parallel. Furthermore, `list and make` provides *full support* for `\ref` and `\label`: any `\label` defined inside of an externalized graphics is still available for the main document.

   If you have legends with `legend to name` or `\label`/`\ref`, you need to generate the graphics defining the `\label` (or `legend to name`), then run `pdflatex` twice on the main document. Afterwards, you can externalize the legend graphics.

The complete reference documentation and remaining options are documented in [5, "Externalization Library"]. This reference also contains information about

- how to use `\tikzset{external/force remake}` and `\tikzset{external/remake next}` to remake selected figures,
- how to disable the externalization partially with `\tikzset{external/export=false}` or completely with `\tikzexternaldisable`,
- how to optimize the speed of the conversion process using `\tikzset{external/optimize command away=\myExpensiveMacro}`,
- how to add further remake-dependencies with `\tikzpicturedependsonfile{`⟨*name*⟩`}` and/or `\tikzexternalfiledependsonfile{`⟨*external file*⟩`}{`⟨*name*⟩`}`,
- how to typeset such a document without pgf installed or
- how to provide work-arounds with `.pdf` images and bounding box restrictions.

Using the Library Without pgf or pgfplots Installed There is a small replacement package `tikzexternal.sty` which can be used once every figure has been exported. The idea is to uncomment `\usepackage{tikz}` and `\usepackage{pgfplots}` and write `\usepackage{tikzexternal}` instead:

```
%  \usepackage{tikz}
%  \usepackage{pgfplots}
\usepackage{tikzexternal}
\tikzexternalize%  activate externalization

\begin{document}
\begin{tikzpicture}
    ...
\end{tikzpicture}
...
\end{document}
```

You do not need pgf, Ti*k*Z or pgfplots installed. What you need is `tikzexternal.sty` and all generated figures (consisting of the image files, '`.pdf`' and the '`.dpth`' files containing information of the `baseline` option). The file `tikzexternal.sty` is shipped with pgf in the directory

```
latex/pgf/utilities/tikzexternal.sty
```

and a copy is shipped with pgfplots in

```
tex/generic/pgfplots/oldpgfcompatib/pgfplotsoldpgfsupp_tikzexternal.sty
```

Just copy the file into your directory and rename it to `tikzexternal.sty`.

Attention: The small replacement package doesn't support key–value interfaces. Thus, it is necessary to use `\tikzsetexternalprefix` instead of the `prefix` option and `\tikzsetfigurename` instead of the `figure name` option since `\tikzset` is not available in such a context. Also, you may want to define a dummy–macro `\pgfplotsset` if you have used `\pgfplotsset`.

7.1.2 Using the Externalization Framework of PGF "By Hand"

Another way to export TEX-pictures to single graphics files is to use the externalization framework of PGF, which requires more work but works more generally than the `external` library. The basic idea is to encapsulate the desired parts with

`\beginpgfgraphicnamed{`⟨*output file name*⟩`}`
⟨*picture contents*⟩
`\endpgfgraphicnamed`.

Furthermore, one needs to tell PGF the name of the main document using

`\pgfrealjobname{`⟨*the real job's name*⟩`}`

in the preamble. This enables two different modes:

1. The first is the normal typesetting mode. LATEX checks whether a file named `{`⟨*output file name*⟩`}` with one of the accepted file extensions exists – if that is the case, the graphics file is included with `\pgfimage` and the ⟨*picture contents*⟩ is skipped. If no such file exists, the ⟨*picture contents*⟩ is typeset normally. This mode is applied if `\jobname` equals `{`⟨*the real job's name*⟩`}`.

2. The second mode applies if `\jobname` equals `{`⟨*output file name*⟩`}`, it initiates the "conversion mode" which is used to write the graphics file `{`⟨*output file name*⟩`}`. In this case, *only* ⟨*picture contents*⟩ is written to `\jobname`, the complete rest of the LATEX is processed as normal, but it is silently discarded.

   This mode needs to be started manually with `pdflatex --jobname` ⟨*output file name*⟩ for every externalized graphics file.

A complete example may look as follows.

```
\documentclass{article}

\usepackage{pgfplots}

\pgfrealjobname{test}

\begin{document}
    \begin{figure}
        \beginpgfgraphicnamed{testfigure}
        \begin{tikzpicture}
        \begin{axis}
            \addplot {x^2};
        \end{axis}
        \end{tikzpicture}
        \endpgfgraphicnamed
    \caption{Our first external graphics example}
    \end{figure}

    \begin{figure}
        \beginpgfgraphicnamed{testfigure2}
        \begin{tikzpicture}
        \begin{axis}
            \addplot {x^3};
        \end{axis}
        \end{tikzpicture}
        \endpgfgraphicnamed
    \caption{A second graphics}
    \end{figure}
\end{document}
```

The file is named `test.tex`, and it is processed (for example) with

```
pdflatex test
```

Now, we type

```
pdflatex --jobname testfigure test
pdflatex --jobname testfigure2 test
```

to enter conversion mode. These last calls will *only* write the contents of our named graphics environments, one for {⟨*testfigure*⟩} and one for {⟨*testfigure2*⟩} into the respective output files `testfigure.pdf` and `testfigure2.pdf`.

In summary, one needs `\pgfrealjobname` and calls `pdflatex --jobname {`⟨*graphics file*⟩`}` for every externalized graphics environment. Please note that it is absolutely necessary to use the syntax above, *not* `\begin{pgfgraphicnamed}`.

These steps are explained in much more detail in Section“Externalizing Graphics” of [5].

Attention: Do not forget a correct `\pgfrealjobname` statement! If it is missing, externalization simply won't work. If it is wrong, any call to LaTeX will produce empty output files.

It should be noted that this approach of image externalization is not limited to Ti*k*Z picture environments. In fact, it collects everything between the begin and end statements into the external file. It is implicitly assumed that the encapsulated stuff is one box, but you can also encapsulate complete paragraphs using something like the LaTeX minipage (or a `\vbox` which is not as powerful but does not affect the remaining document that much).

`/pgf/images/aux in dpth=true|false` (initially `false`)

If this boolean is set to `true`, any `\label` information generated inside of the external image is stored into the already mentioned `.dpth` file. The main document can thus reference label information of externalized parts of the document (although you may need to run `latex` several times).

Label support is provided for `\ref`, and probably `\cite`. The `\pageref` command is only partially supported.

Using the Library Without PGF Installed Simply uncomment the packages `\usepackage{tikz}` and `\usepackage{pgfplots}` and use

```
\long\def\beginpgfgraphicnamed#1#2\endpgfgraphicnamed{%
    \begingroup
    \setbox1=\hbox{\includegraphics{#1}}%
    \openin1=#1.dpth
    \ifeof1 \box1
    \else
        \read1 to\pgfincludeexternalgraphicsdp \closein1
        \dimen0=\pgfincludeexternalgraphicsdp\relax
        \hbox{\lower\dimen0 \box1 }%
    \fi
    \endgroup
}
```

instead. This will include the generated graphics files (and it will respect the `baseline` information stored in `.dpth` files). Consequently, you won't need PGF or PGFPLOTS installed. See Section“Externalizing Graphics” of [5] for details.

7.2 Exporting Mesh Data From Matlab To pgfplots

While it is easy to write Matlab vectors to files (using `save P.dat data -ASCII`), it is more involved to export mesh data.

The main problem is to communicate the mesh structure to PGFPLOTS.

Here is an example how to realize this task: in Matlab, we have mesh data `X`, `Y` and `Z` which are matrizes of the same size. For example, suppose we have

```
[X,Y] = meshgrid( linspace(-1,1,5), linspace(4,5,10) );
Z = X + Y;
surf(X,Y,Z)
```

as data. Then, we can generate an $N \times 3$ table containing all single elements in column–wise ordering with

```
data = [ X(:) Y(:) Z(:) ]
save P.dat data -ASCII
```

where the second command stores the $N \times 3$ table into `P.dat`. Finally, we can use

`\addplot3[surf,mesh/rows=10,mesh/ordering=colwise,shader=interp] file {P.dat};`

in PGFPLOTS to read this data. We need to provide either the number of rows (10 here) or the number of columns – and the ordering (which is `colwise` for Matlab matrizes).

An alternative which is faster in pgfplots would be to transpose the matrizes in Matlab and tell pgfplots they are in `rowwise` ordering. So, the last step becomes

```
XX=X'; YY=Y'; ZZ=Z';
data = [ XX(:) YY(:) ZZ(:) ]
save P.dat data -ASCII
```

with pgfplots command

`\addplot3[surf,mesh/cols=10,mesh/ordering=rowwise,shader=interp] file {P.dat};`.

7.3 matlab2pgfplots.m

This is a Matlab (tm) script which attempts to convert a matlab figure to pgfplots. It requires Matlab version 7.4 (or higher).

Attention: The author of pgfplots does not have enough time to maintain this script as much as he wants to. In other words, it supports only a small subset of pgfplots. You may also want to look at `matlab2tikz`, a conversion script of Nico Schlömer available at

`http://www.mathworks.com/matlabcentral/fileexchange/22022-matlab2tikz`

which also uses pgfplots for the LaTeX conversion.

The idea of `matlab2pgfplots.m` is to

- use a complete matlab figure as input,
- acquire axis labels, axis scaling (log or normal) and legend entries,
- acquire all plot coordinates

and write an equivalent `.pgf` file which typesets the plot with pgfplots.

The intention is *not* to simulate matlab. It is a first step for a conversion. Type

```
> help matlab2pgfplots
```

on your matlab prompt for more information about its features and its limitations.

This script is experimental.

7.4 matlab2pgfplots.sh

A `bash`-script which simply starts matlab and runs

```
    f=hgload( 'somefigure.fig' );
    matlab2pgfplots( 'outputfile.pgf', 'fig', f );
```

See matlab2pgfplots.m above.

7.5 SVG Output

It is possible to write every single Ti*k*Z picture into a scalable vector graphics (`.svg`) file. This has nothing to do with pgfplots, it is a separate driver of pgf. Please refer to [5, Section "Producing HTML / SVG Output"].

7.6 Generate pgfplots Graphics Within Python

Mario Orne DÍAZ ANADÓN contributed a small python script `pgfplots.py` which provides a simple interface to generate pgfplots figures from within python. It can be found in the pgfplots installation directory, in `pgfplots/scripts/pgfplots/pgfplots.py`; documentation can be found in the file.

8 Utilities and Basic Level Commands

This section documents commands which provide access to more basic elements of PGFPLOTS. Most of them are closely related to the basic level of PGF, especially various point commands which are specific to an axis. Some of them are general purpose utilities like loops.

However, most elements in this section are only interesting for advanced users – and perhaps only for special cases.

8.1 Utility Commands

`\foreach`⟨*variables*⟩ `in` ⟨*list*⟩ `{`⟨*commands*⟩`}`

A powerful loop command provided by Ti*k*Z, see [5, Section Utilities].

Iterating 1. Iterating 2. Iterating 3. Iterating 4.

```
\foreach \x in {1,2,...,4} {Iterating \x. }%
```

A PGFPLOTS related example could be

```
\foreach \i in {1,2,...,10} {\addplot table {datafile\i}; }%
```

`\pgfplotsforeachungrouped`⟨*variable*⟩ `in` ⟨*list*⟩ `{`⟨*command*⟩`}`

A specialised variant of `\foreach` which can do two things: it does not introduce extra groups while executing ⟨*command*⟩ and it allows to invoke the math parser for (simple!) ⟨x_0⟩,⟨x_1⟩,...,⟨x_n⟩ expressions.

Iterating 1. Iterating 2. Iterating 3. Iterating 4. All collected = , 1, 2, 3, 4.

```
\def\allcollected{}
\pgfplotsforeachungrouped \x in {1,2,...,4} {Iterating \x. \edef\allcollected{\allcollected, \x}}%
All collected = \allcollected.
```

A more useful example might be to work with tables. The following example is taken from PGFPLOTSTABLE:

```
\pgfplotsforeachungrouped \i in {1,2,...,10} {%
    \pgfplotstablevertcat{\output}{datafile\i} %  appends ‘datafile\i’ -> ‘\output’
}%
%  since it was ungrouped, \output is still defined (would not work
%  with \foreach)
```

Remark: The special syntax ⟨*list*⟩=⟨x_0⟩,⟨x_1⟩,...,⟨x_n⟩, i.e. with two leading elements, followed by dots and a final element, invokes the math parser for the loop. Thus, it allows larger number ranges than any other syntax if `/pgf/fpu` is active. In all other cases, `\pgfplotsforeachungrouped` invokes `\foreach` and provides the results without TEX groups.

`\pgfplotsinvokeforeach{`⟨*list*⟩`} {`⟨*command*⟩`}`

A variant of `\pgfplotsforeachungrouped` (and such also of `\foreach`) which replaces any occurrence of `#1` inside of ⟨*command*⟩ once for every element in ⟨*list*⟩. Thus, it actually assumes that `{`⟨*command*⟩`}` is like a `\newcommand` body.

In other words, `{`⟨*command*⟩`}` is invoked for every element of `{`⟨*list*⟩`}`. The actual element of `{`⟨*list*⟩`}` is available as `#1`.

As `\pgfplotsforeachungrouped`, this command does *not* introduce extra scopes (i.e. it is ungrouped as well).

The difference to `\foreach \x in` ⟨*list*⟩`{`⟨*command*⟩`}` is subtle: the `\x` would *not* be expanded whereas `#1` is.

Invoke them:[a] [b][c] [d]

```
\pgfkeys{
  otherstyle a/.code={[a]},
  otherstyle b/.code={[b]},
  otherstyle c/.code={[c]},
  otherstyle d/.code={[d]}}
\pgfplotsinvokeforeach{a,b,c,d}
    {\pgfkeys{key #1/.style={otherstyle #1}}}
Invoke them:
\pgfkeys{key a} \pgfkeys{key b}
\pgfkeys{key c} \pgfkeys{key d}
```

The counter example would use a macro (here `\x`) as loop argument:

Invoke them:[d] [d][d] [d]

```
\pgfkeys{
  otherstyle a/.code={[a]},
  otherstyle b/.code={[b]},
  otherstyle c/.code={[c]},
  otherstyle d/.code={[d]}}
\pgfplotsforeachungrouped \x in {a,b,c,d}
    {\pgfkeys{key \x/.style={otherstyle \x}}}
Invoke them:
\pgfkeys{key a} \pgfkeys{key b}
\pgfkeys{key c} \pgfkeys{key d}
```

Restrictions: you can't nest this command yet (since it does not introduce protection by scopes).

`\pgfmathparse{`⟨*expression*⟩`}`

Invokes the PGF math parser for ⟨*expression*⟩ and defines `\pgfmathresult` to be the result.

The result is '42.0'.

```
\pgfmathparse{1+41}

The result is '\pgfmathresult'.
```

Please refer to [5] for more details.

`\pgfplotstableread{`⟨*file*⟩`}`

Please refer to the manual of PgfplotsTable, `pgfplotstable.pdf`, which is part of the PGFPLOTS-bundle.

`\pgfplotstabletypeset{`⟨\macro⟩`}`

Please refer to the manual of PgfplotsTable, `pgfplotstable.pdf`, which is part of the PGFPLOTS-bundle.

`\pgfplotsiffileexists{`⟨*filename*⟩`}{`⟨*true code*⟩`}{`⟨*false code*⟩`}`

Invokes `{`⟨*true code*⟩`}` if `{`⟨*filename*⟩`}` exists and `{`⟨*false code*⟩`}` if not. Can be used in looping macros, for example to plot every data file until there are no more of them.

`\pgfplotsutilifstringequal{`⟨*first*⟩`}{`⟨*second*⟩`}{`⟨*true code*⟩`}{`⟨*false code*⟩`}`

A simple "strcmp" tool which invokes `{`⟨*true code*⟩`}` if `{`⟨*first*⟩`}` =`{`⟨*second*⟩`}` and `{`⟨*false code*⟩`}` otherwise. This does not expand macros.

`\pgfkeys`
`\pgfeov`
`\pgfkeysvalueof`
`\pgfkeysgetvalue`

These commands are part of the Ti*k*Z way of specifying options, its sub-package `pgfkeys`. The `\pgfplotsset` command is actually nothing but a wrapper around `\pgfkeys`.

A short introduction into `\pgfkeys` can be found in [7] whereas the complete reference is, of course, the Ti*k*Z manual [5].

The key `\pgfkeysvalueof{`⟨*key name*⟩`}` expands to the value of a key; `\pgfkeysgetvalue{`⟨*key name*⟩`}{`⟨\macro⟩`}` stores the value of ⟨*key name*⟩ into ⟨\macro⟩. The `\pgfeov` macro is used to delimit arguments for code keys in `\pgfkeys`, please refer to the references mentioned above.

8.2 Commands Inside Of PGFPLOTS Axes

`\autoplotspeclist`

This command should no longer be used, although it will be kept as technical implementation detail. Please use the '`cycle list`' option, section 4.6.6.

`\logten`

Expands to the constant log(10). Useful for logplots because $\log(10^i) = i \log(10)$. This command is only available inside of an Ti*k*Z-picture.

`\pgfmathprintnumber{`⟨*number*⟩`}`

Generates pretty–printed output[46] for `{`⟨*number*⟩`}`. This method is used for every tick label.

The number is printed using the current number printing options, see the manual of PgfplotsTable which comes with this package for the different number styles, rounding precision and rounding methods.

`\plotnum`

Inside of `\addplot` or any associated style, option or command, `\plotnum` expands to the current plot's number, starting with 0.

`\numplots`

Inside of any of the axis environments, associated style, option or command, `\numplots` expands to the total number of plots.

`\coordindex`

Inside of an `\addplot` command, this macro expands to the number of the actual coordinate (starting with 0).

It is useful together with `x filter` or `y filter` to (de-)select coordinates.

8.3 Path Operations

`\path`
`\draw`
`\fill`
`\node`
`\matrix`

These commands are Ti*k*Z drawing commands all of which are documented in [5]. They are used to draw or fill paths, generate text nodes or aligned text matrices. They are equivalent to `\path[draw]`, `\path[fill]`, `\path[node]`, `\path[matrix]`, respectively.

`\path ... --`⟨*coordinate*⟩ `...;`

A Ti*k*Z path operation which connects the current point (the last one before `--`) and ⟨*coordinate*⟩ with a straight line.

`\path ... |-`⟨*coordinate*⟩ `...;`

A Ti*k*Z path operation which connects the current point and ⟨*coordinate*⟩ with *two* straight lines: first vertical, then horizontal.

`\path ... -|`⟨*coordinate*⟩ `...;`

A Ti*k*Z path operation which connects the current point and ⟨*coordinate*⟩ with *two* straight lines: first horizontal, then vertical.

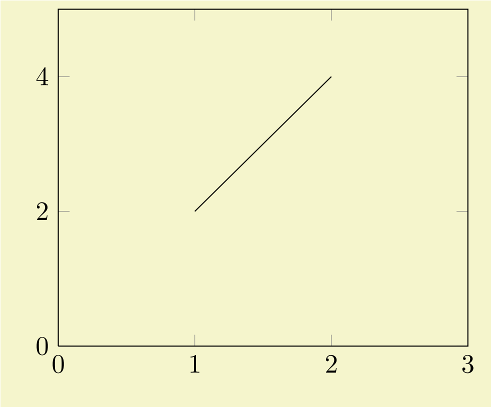
`/tikz/xshift={`⟨*dimension*⟩`}`
`/tikz/yshift={`⟨*dimension*⟩`}`

These Ti*k*Z keys allow to shift something by `{`⟨*dimension*⟩`}` which is any TeX size (or expression).

[46] This method was previously `\prettyprintnumber`. It's functionality has been included into PGF and the old command is now deprecated.

\pgfplotsextra{⟨*low-level path commands*⟩}

A command to execute {⟨*low-level path commands*⟩} in a PGFPLOTS axis. Since any drawing commands inside of an axis need to be postponed until the axis is complete and the scaling has been initialised, it is not possible to simply draw any paths. Instead, it is necessary to draw them as soon as the axis is finished. This is done automatically for every Ti*k*Z path – and it is also done manually if you write \pgfplotsextra{⟨*commands*⟩}.



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{axis}[xmin=0,xmax=3,ymin=0,ymax=5]
    \pgfplotsextra{%
        \pgfpathmoveto{\pgfplotspointaxisxy{1}{2}}%
        \pgfpathlineto{\pgfplotspointaxisxy{2}{4}}%
        \pgfusepath{stroke}%
    }
    \end{axis}
\end{tikzpicture}
```

The example above initialises an axis and executes the basic level path commands as soon as the axis is ready. The execution of multiple \path, \addplot and \pgfplotsextra commands is in the same sequence as they occur in the environment[47].

8.4 Specifying Basic Coordinates

\pgfplotspointaxisxy{⟨*x coordinate*⟩}{⟨*y coordinate*⟩}
\pgfplotspointaxisxyz{⟨*x coordinate*⟩}{⟨*y coordinate*⟩}{⟨*z coordinate*⟩}

Point commands like \pgfpointxy which take logical, absolute coordinates and return a low–level point. Every transformation from user transformations to logarithms are applied.

Since the transformations are initialised after the axis is complete, this command needs to be postponed (see \pgfplotsextra).

\pgfplotspointrelaxisxy{⟨*rel x coordinate*⟩}{⟨*rel y coordinate*⟩}
\pgfplotspointrelaxisxyz{⟨*rel x coordinate*⟩}{⟨*rel y coordinate*⟩}{⟨*rel z coordinate*⟩}

Point commands which take *relative* coordinates such that $x = 0$ is the *lower* x axis limit and $x = 1$ the *upper* x axis limit.

These commands are used for rel axis cs.

Please note that the transformations are only initialised if the axis is complete! This means you need to provide \pgfplotsextra.

\pgfplotspointdescriptionxy{⟨*x fraction*⟩}{⟨*y fraction*⟩}
\pgfplotsqpointdescriptionxy{⟨*x fraction*⟩}{⟨*y fraction*⟩}

Point commands such that {0}{0} is the lower left corner of the axis' bounding box and {1}{1} the upper right one; everything else is in-between. The 'q' variant is quicker as it doesn't invoke the math parser on its arguments.

They are used for axis description cs, see section 4.8.1.

\pgfplotspointunitx
\pgfplotspointunity
\pgfplotspointunitz

Low–level point commands which return the x, y or z unit vectors.

The point \pgfplotspointxyz{1}{0}{0} is the same as \pgfplotspointunitx, the {0}{1}{0} coordinate the unit y vector and the {0}{0}{1} coordinate the unit z vector.

The unit z vector is only defined for three dimensional axes.

[47] Except for stacked plots where the sequence may be reverse, see the key reverse stack plots.

`\pgfplotsunitxlength`
`\pgfplotsunitylength`
`\pgfplotsunitzlength`
`\pgfplotsunitxinvlength`
`\pgfplotsunityinvlength`
`\pgfplotsunitzinvlength`

Macros which expand to the vector length $\|x_i\|$ of the respective unit vector x_i or the inverse vector length, $1/\|x_i\|$. These macros can be used inside of `\pgfmathparse`, for example.

The x_i are the `\pgfplotspointunitx` variants.

`\pgfplotspointaxisorigin`

A point coordinate at the origin, $(0, 0, 0)$. If the origin is not part of the axis limits, the nearest point on the boundary is returned instead.

This is the same coordinate as returned by the `origin` anchor.

`\pgfplotsqpointoutsideofaxis{`⟨*three-char-string*⟩`}{`⟨*coordinate*⟩`}{`⟨*normal distance*⟩`}`

Provides a point coordinate on one of the available four axes in case of a two dimensional figure or on one of the available twelve axes in case of a three dimensional figure.

The desired axis is uniquely identified by a three character string, provided as first argument to the command. The first of the three characters is '`0`' if the x coordinate of the specified axis passes through the lower axis limit. It is '`1`', if the x coordinate of the specified axis passes through the upper axis limit. Furthermore, it is '`2`' if it passes through the origin. The second character is also either `0`, `1` or `2` and it characterizes the position on the y axis. The third character is for the third dimension, the z axis. It should be left at '`0`' for two dimensional plots. However, *one* of the three characters should be '`v`', meaning the axis varies. For example, `v01` denotes $\{(x, y_{\min}, z_{\max})|x \in \mathbb{R}\}$.

The second argument, ⟨*coordinate*⟩ is the logical coordinate on that axis. Since two coordinate of the axis are fixed, ⟨*coordinate*⟩ refers to the varying component of the axis. It must be a number without unit; no math expressions are supported here.

The third argument ⟨*normal distance*⟩ is a dimension like `10pt`. It shifts the coordinate away from the designated axis in direction of the outer normal vector. The outer normal vector always points away from the axis. It is computed using `\pgfplotspointouternormalvectorofaxis`.

There are several variants of this command which are documented in the source code. One of them is particularly useful:

`\pgfplotsqpointoutsideofaxisrel{`⟨*three-char-string*⟩`}{`⟨*axis fraction*⟩`}{`⟨*normal distance*⟩`}`

This point coordinate is a variant of `\pgfplotsqpointoutsideofaxis` which allows to provide an ⟨*axis fraction*⟩ instead of an absolute coordinate. The fraction is a number between 0 (lower axis limit) and 1 (upper axis limit), i.e. it is given in percent of the total axis. It is possible to provide negative values or values larger than one.

The `\pgfplotsqpointoutsideofaxisrel` command is similar in spirit to `rel axis cs`.

There is one speciality in conjunction with reversed axes: if the axis has been reversed by `x dir=reverse` and, in addition, `allow reversal of rel axis cs` is true, the value 0 denotes the *upper* limit while 1 denotes the *lower* limit. The effect is that coordinates won't change just because of axis reversal.

`\pgfplotspointouternormalvectorofaxis{`⟨*three-char-string*⟩`}`

A point command which yields the outer normal vector of the respective axis. The normal vector has length 1 (computed with `\pgfpointnormalised`). It is the same normal vector used inside of `\pgfplotsqpointoutsideofaxis` and its variants.

The output of this command will be cached and re-used during the lifetime of an axis.

`\pgfplotsticklabelaxisspec{`⟨*x, y or z*⟩`}`

Expands to the three-character-identification for the axis containing tick labels for the chosen axis, either ⟨x⟩, ⟨y⟩ or ⟨z⟩.

`\pgfplotsvalueoflargesttickdimen{`⟨*x, y or z*⟩`}`

Expands to the largest distance of a tick position to its tick label bounding box in direction of the outer unit normal vector. It does also include the value of the `ticklabel shift` key.

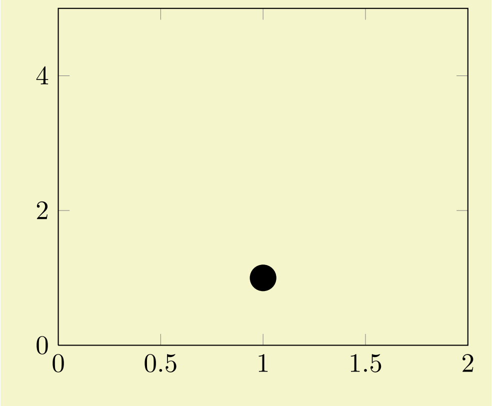
This value is used for `ticklabel cs`.

`\pgfplotstransformcoordinatex{`⟨*x coordinate of an axis*⟩`}`
`\pgfplotstransformcoordinatey{`⟨*y coordinate of an axis*⟩`}`
`\pgfplotstransformcoordinatey{`⟨*z coordinate of an axis*⟩`}`

Defines `\pgfmathresult` to be the low-level PGF coordinate corresponding to the input argument.

The command applies any `[xyz] coord trafo` keys, data scalings and/or logarithms or whatever PGFPLOTS does to map input coordinates to internal coordinates.

The result can be used inside of a `\pgfpointxy` statement (i.e. it still needs to be scaled with the respective PGF unit vector).



```
% Preamble: \pgfplotsset{width=7cm,compat=1.3}
\begin{tikzpicture}
    \begin{axis}[xmin=0,xmax=2,ymin=0,ymax=5]
    \pgfplotsextra{%
        \pgfplotstransformcoordinatex{1}%
        \let\xcoord=\pgfmathresult
        \pgfplotstransformcoordinatey{1}%
        \let\ycoord=\pgfmathresult
        \pgfpathcircle
            {\pgfqpointxy{\xcoord}{\ycoord}}
            {5pt}%
        \pgfusepath{fill}%
    }%
    \end{axis}
\end{tikzpicture}
```

Please note that the transformations are only initialised if the axis is complete! This means you need to provide `\pgfplotsextra` as is shown in the example above.

`\pgfplotsconvertunittocoordinate{`⟨*x, y or z*⟩`}{`⟨*dimension*⟩`}`

Converts a dimension (with unit!) to a corresponding x, y or z coordinate. The result will be written to `\pgfmathresult` (without units).

It is possible to use the result as arguments for the `\pgfpointxyz` commands.

The effect is to multiply `{`⟨*dimension*⟩`}` with the inverse length of the unit vector for the specified axis. These lengths are precomputed in PGFPLOTS so the operation is fast.

```
\pgfplotsconvertunittocoordinate{x}{5pt}
%  now, the command uses exactly 5pt in x direction:
\pgfqpointxyz{\pgfmathresult}{4}{3}
```

`\pgfplotsmathfloatviewdepthxyz{`⟨*x*⟩`}{`⟨*y*⟩`}{`⟨*z*⟩`}`
`\pgfplotsmathviewdepthxyz{`⟨*x*⟩`}{`⟨*y*⟩`}{`⟨*z*⟩`}`

Both macros define `\pgfmathresult` to be the “depth” of a three dimensional point $\bar{x} = (x, y, z)$. The depth is defined to be the scalar product of $\bar{x}$ with $\vec{d}$, the view direction of the current axis.

For `\pgfplotsmathfloatviewdepthxyz`, the arguments are parsed as floating point numbers and the result is encoded in floating point. A fixed point representation can be generated with `\pgfmathfloattofixed{\pgfmathresult}`.

For `\pgfplotsmathviewdepthxyz`, TEX arithmetics is employed for the inner product and the result is assigned in fixed point. This is slightly faster, but has considerably smaller data range.

Both commands can only be used *inside* of a three dimensional PGFPLOTS axis (as soon as the axis is initialised, see `\pgfplotsextra`).

`\ifpgfplotsthreedim`⟨*true code*⟩`\else`⟨*else code*⟩`\fi`

A TEX `\if` which evaluates the ⟨*true code*⟩ if the axis is three dimensional and the ⟨*else code*⟩ if not.

Index

References

[1] C. Feuersänger. PgfplotsTable package – Loading, rounding and formatting tables in LaTeX. Available as separate package \usepackage{pgfplotstable}, as part of pgfplots.

[2] C. Feuersänger. Programming in TeX and Library Functions from pgf and pgfplots. Available as part of pgfplots, TeX-programming-notes.pdf, March 31, 2010.

[3] U. Kern. Extending LaTeX's color facilities: the `xcolor` package.

[4] D. P. Story. The AcroTeX eDucation Bundle. `http://www.ctan.org/tex-archive/macros/latex/contrib/acrotex`. Sub packages `insdljs` and `eforms` are required for the clickable library.

[5] T. Tantau. Ti*k*Z and pgf manual. `http://sourceforge.net/projects/pgf`. *v.* $\geq$ 2.00.

[6] K. van Zonneveld. PhP to javascript conversion project (GPL). `http://kevin.vanzonneveld.net/techblog/article/phpjs_licensing`.

[7] J. Wright and C. Feuersänger. Implementing keyval input: an introduction. `http://pgfplots.sourceforge.net` as `.pdf`, 2008.