# Manual for Package PGFPLOTS

Version 1.2.2

http://sourceforge.net/projects/pgfplots

Christian Feuersänger[*]
Institut für Numerische Simulation
Universität Bonn, Germany

February 16, 2009

**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, piecewise constant plots, bar plots and area plots. 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. 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 plots, especially for use in scientific contexts (logplots).

It is build completely on Ti*k*Z and PGF and may be used as Ti*k*Z library.

# 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.

## 2.1 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:

1. Due to a small math bug in PGF 2.00: you *can't* write '`-x^2`' – use '`0-x^2`' instead. The same holds for '`exp(-x^2)`' – use '`exp(0-x^2)`' instead.
2. Starting with PGFPLOTS 1.1, `\tizkstyle` 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 5.10 for more detail.

I apologize for any inconvenience caused by these changes.

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.

## 2.2 The Team

PGFPLOTS has been written by Christian Feuersänger and Pascal Wolkotte as a spare time project. We hope it is useful and provides valueable plots.

## 2.3 Acknowledgements

I thank God for all hours of enjoyed programming. I thank Pascal Wolkotte, author of package `pgfgraph`, who joined development and contributed code for axis lines and tick alignment. 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.

## 2.4 Installation and Prerequisites

### 2.4.1 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`.

### 2.4.2 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.

### 2.4.3 Installation in Windows

Windows users often use MikTEX which downloads the latest stable package versions automatically. As far as I know, 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 use configure the MikTEX Package Manager to use this specific directory with higher priority than its default paths.

See also section 2.4.6 for more information about separate TDS directories.

### 2.4.4 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.

### 2.4.5 Installation Into a Local `texmf`-Directory

Copy PGFPLOTS to a local `texmf` directory like `~/texmf` in your home directory. Then, install PGFPLOTS into the subdirectory `texmf/tex/generic/pgfplots` and run `texhash`.

### 2.4.6 Installation Into a Local TDS Compliant `texmf`-Directory

A TDS conforming installation will use the same base directory as in the last section. Since PGFPLOTS comes in TDS conforming directory structure, you can simply unpack the files into a directory of your choice and configure TeX to use this directory as further include path.

Do not forget to run `texhash`.

### 2.4.7 Installation If Everything Else Fails...

If TeX still doesn't find your files, you can copy all `.sty` and all `.code.tex`-files into your current project's working directory.

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

## 2.5 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").

### 2.5.1 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).

### 2.5.2 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. The `axis equal` key will be confused if $x$ and $y$ have a very different scale.
3. You may have found a bug – please contact the developers.

### 2.5.3 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.

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 explictly.

Please read the corresponding sections in [3, 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.

### 2.5.4 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 7 for how to enlarge them.

### 2.5.5 Problems with Language Settings and Active Characters

Both, PGF and PGFPLOTS use a lot of characters – which may lead to incompatibilites 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).

### 2.5.6 Other Problems

Please contact the authors (preferrably by mail). Visit the webpages shown above to get contact information.

# 3 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}
```

A small LaTeX–example file can be found in

```
doc/latex/pgfplots/pgfplotsexample.tex.
```

**ConTeXt:** `\usemodule[pgfplots]` and

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

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

A small 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
```

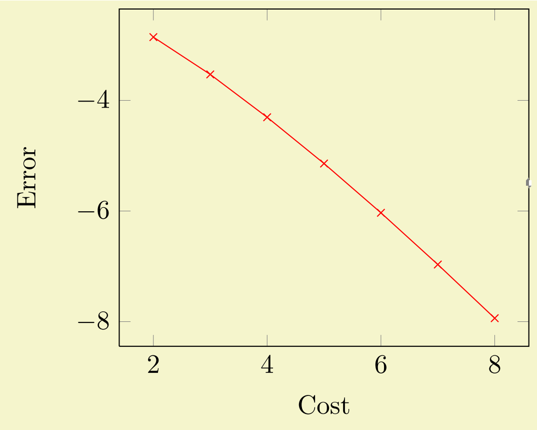
A small plain–TeX–example file can be found in

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

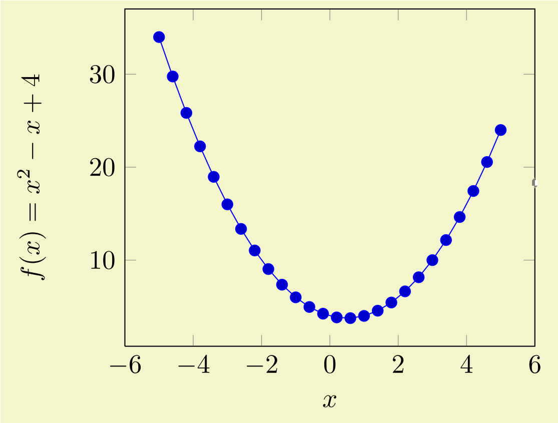
You may need to set low–level drivers if you intend to use `dvipdfm`, see section 2.5.3.

## 3.2 A first plot

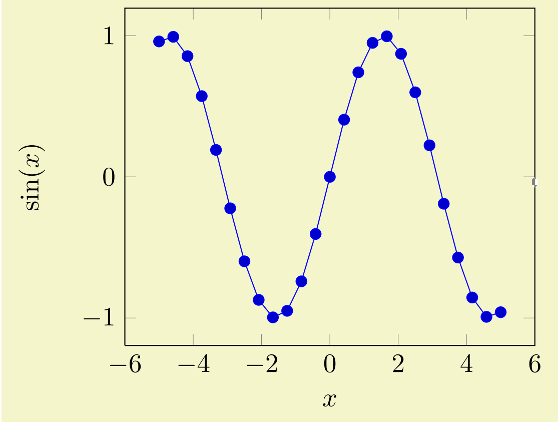
Plotting is done using `\begin{axis} ... \addplot ...; \end{axis}`, where `\addplot` is an interface to the TikZ `plot` commands.



```
\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}
```



```
\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}
```



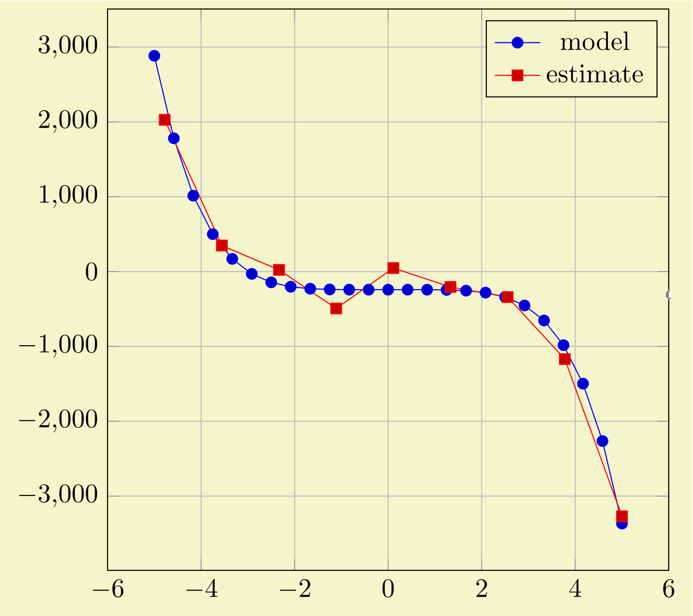
```
\begin{tikzpicture}
    \begin{axis}[
        xlabel=$x$,
        ylabel=$\sin(x)$
    ]
    % invoke external gnuplot as
    % calculator:
    \addplot gnuplot[id=sin]{sin(x)};
    \end{axis}
\end{tikzpicture}
```

The `plot coordinates`, `plot gnuplot` and `plot expression` commands are three of the several supported ways to create plots, see section 4.2 for more details[1] and the remaining ones (`plot file`, `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.

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

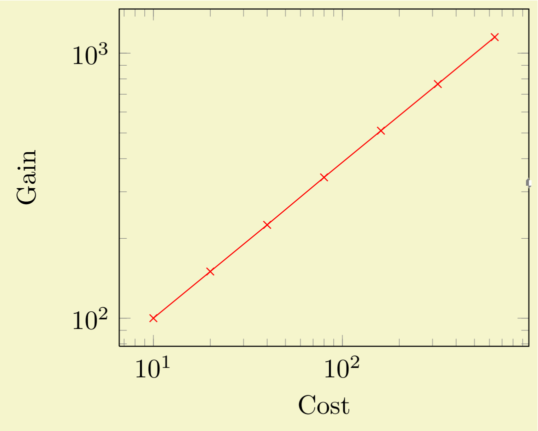


```
\begin{tikzpicture}
    \begin{axis}[
        height=9cm,
        width=9cm,
        grid=major,
    ]

    \addplot plot[id=filesuffix] gnuplot{(-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 always uses the natural logarithm, i.e. basis $e \approx 2.718$. Now, the axis description also contains minor ticks and the labels are placed at $10^i$.

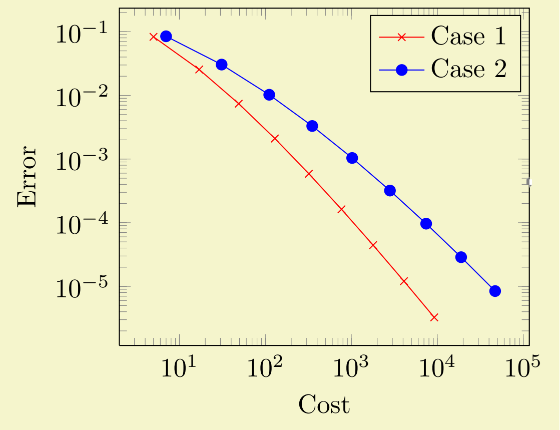


```
\begin{tikzpicture}
\begin{loglogaxis}[xlabel=Cost,ylabel=Gain]
\addplot[color=red,mark=x] coordinates {
    (10,100)
    (20,150)
    (40,225)
    (80,340)
    (160,510)
    (320,765)
    (640,1150)
};
\end{loglogaxis}
\end{tikzpicture}
```

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



```
\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}
```

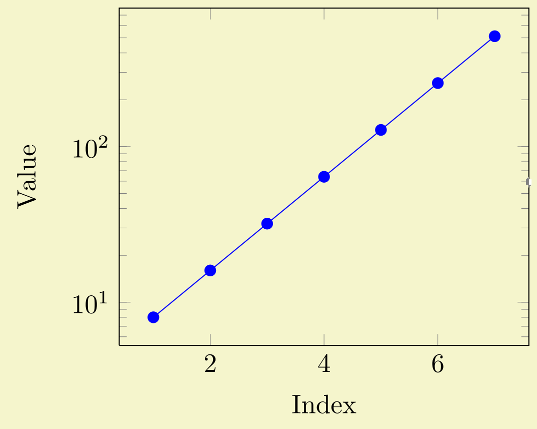
The first plot employs inline coordinates; the second one reads numerical data from file and plots column '`Cost`' versus '`Error`'.
Besided the environment "`loglogaxis`" you can use

- `\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}
```

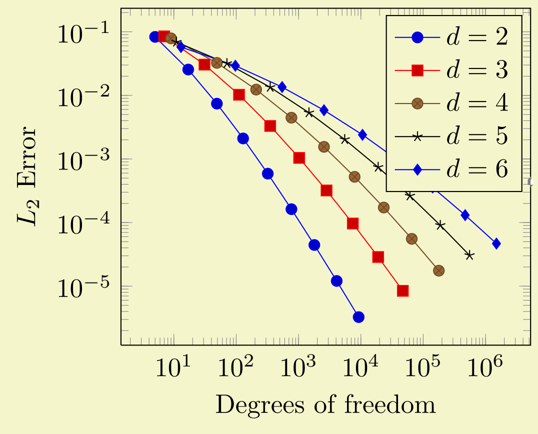
which is the same as `\begin{semilogyaxis}...\end{semilogyaxis}`.



```
\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:

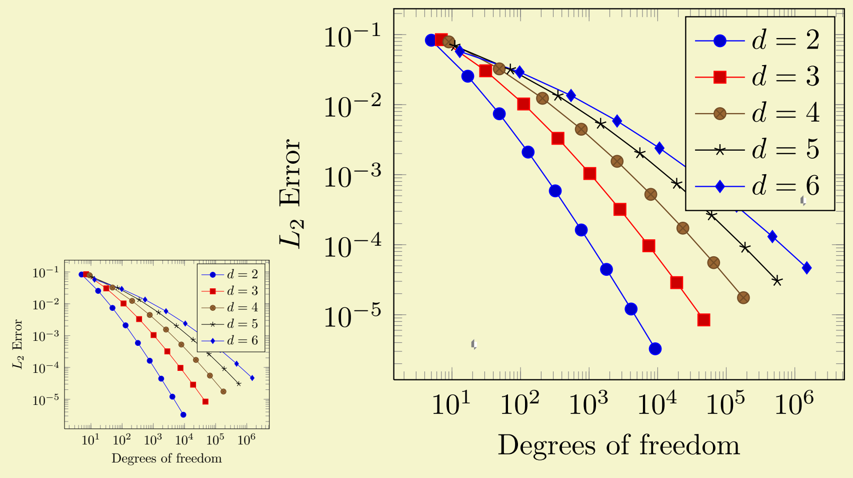


```
\begin{tikzpicture}
\begin{loglogaxis}[
    xlabel={Degrees of freedom},
    ylabel={$L_2$ Error}
]
\addplot coordinates {
    (5,8.312e-02)    (17,2.547e-02)   (49,7.407e-03)
    (129,2.102e-03)  (321,5.874e-04)  (769,1.623e-04)
    (1793,4.442e-05) (4097,1.207e-05) (9217,3.261e-06)
};

\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.

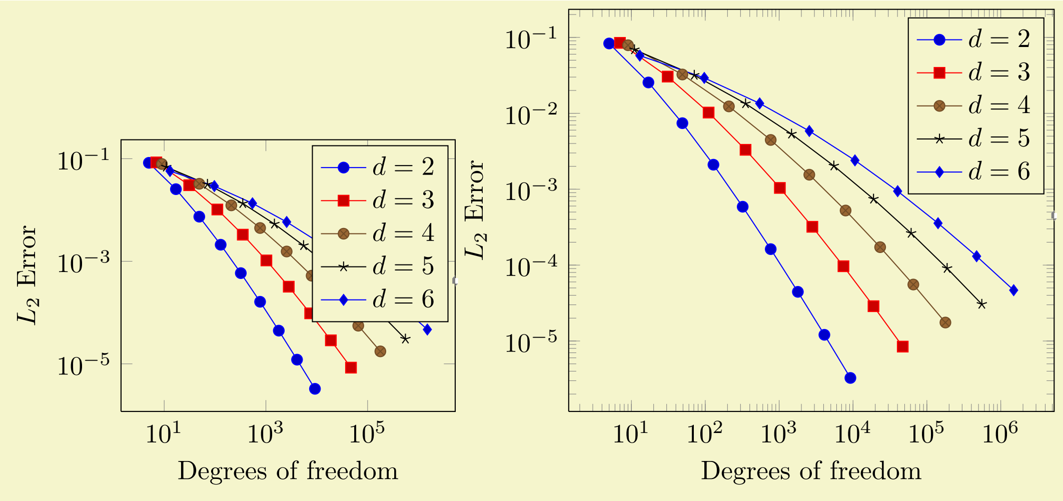
## 3.6 Scaling plots

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.



```
\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:



```
\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}
```

# 4 Command 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{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`.

```
\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.

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

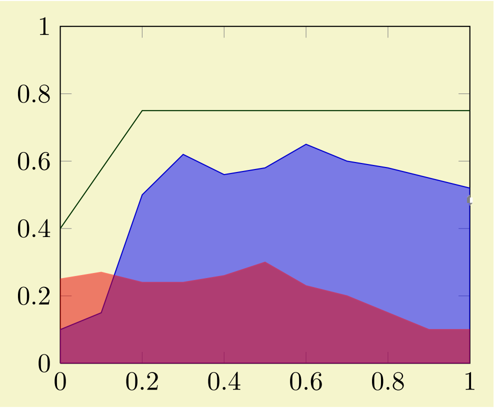
The axis environment for logarithmic scaling of both, $x$ and $y$ axes, As for the other axis possibilities, there is a style '`every loglog axis`' which is installed at the environment's beginning.

They are all equivalent to

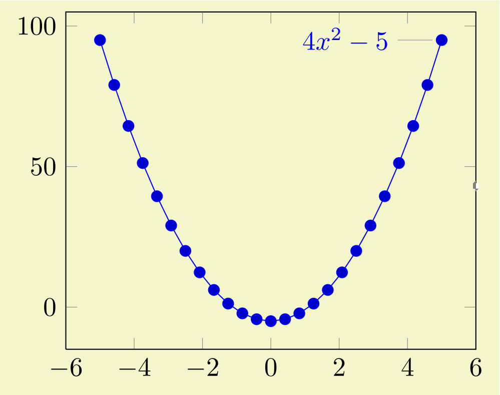
```
\begin{axis}[
    xmode=log|normal,
    ymode=log|normal]
...
\end{axis}
```

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

## 4.2 The Plot Command



```
\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}
```



```
\begin{tikzpicture}
\begin{axis}
\addplot plot
    [id=parable,domain=-5:5]
    gnuplot{4*x**2 - 5}
    node[pin=180:{$4x^2-5$}]{};
\end{axis}
\end{tikzpicture}
```

\addplot[⟨*style options*⟩] plot[⟨*behavior options*⟩] ⟨*input data*⟩ ⟨*trailing path commands*⟩;

This is the main plotting command, available within each axis environment.

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

The ⟨*style options*⟩ can be omitted in which case the next entry from the `cycle list` will be inserted as ⟨*style options*⟩. These keys characterize the plot's type like linear interpolation, smooth plot, constant interpolation or bar plot and define colors, markers and line specifications.

The optional ⟨*behavior options*⟩ can be used to modify plot variants, for example to add error bars. They are described when needed.

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-drawing commands (for example `plot coordinate {...}`). If ⟨*trailing path commands*⟩ is not empty, these arguments are appended to the final drawing command.

Some more details:

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

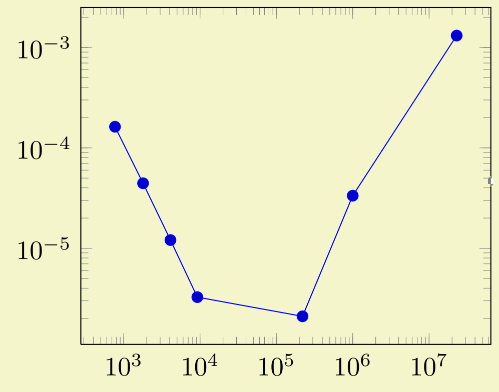
```
\pgfplotsset{every axis plot/.append style={...}}
```

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 has number 0.

- The ⟨*style options*⟩ are remembered for the legend. Furthermore, they are available as '`current plot style`' as long as the path is not yet finished or in associated error bars.
- See subsection 5.3 for a list of available markers and line styles.
- For log plots, PGFPLOTS will compute the natural logarithm log(·) numerically. This works with normal fixed point numbers or in scientific notation. For example, the following numbers are valid input to `\addplot`.



```
\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.

- For normal plots, 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 at least 5 significant decimal digits for the mantisse. As soon as the axes limits are completely known, PGFPLOTS applies a transformation which maps these floating point numbers into TEX-precision using transformations

$$T_x(x) = 10^{s_x} \cdot x - a_x \text{ and } T_y(y) = 10^{s_y} \cdot y - a_y$$

with properly chosen integers $s_x, s_y \in \mathbb{Z}$ and shifts $a_x, a_y \in \mathbb{R}$. Section 5.13 contains a description of `disabledatascaling` and provides more details about the transformation.

- As a consequence of the coordinate parsing routines, you can't use the mathematical expression parsing method of PGF as coordinates (that means: you will need to provide coordinates without suffixes like "cm" or "pt" and you can't invoke mathematical functions).
- If you did not specify axis limits for $x$ and $y$ 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}`.

### 4.2.1 Coordinate Lists

`\addplot` `coordinates` `{`⟨*coordinate list*⟩`};`
`\addplot[`⟨*style options*⟩`]` `plot[`⟨*behavior options*⟩`]` `coordinates` `{`⟨*coordinate list*⟩`}` ⟨*trailing path commands*⟩`;`

The '`plot coordinates`' command is provided by Ti*k*Z and reads its input data from a sequence of point coordinates.

```
\addplot plot coordinates {
    (0,0)
    (0.5,1)
    (1,2)
};
```

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

```
\addplot plot coordinates {
    (0,0)   +- (0.1,0)
    (0.5,1) +- (0.4,0.2)
    (1,2)
    (2,5)   +- (1,0.1)
};
```

or

```
\addplot plot coordinates {
    (1300,1e-6) +- (0.1,0.2)
    (2600,5e-7) +- (0.2,0.5)
    (4000,1e-7) +- (0.1,0.01)
};
```

These error coordinates are only used in case of error bars, see section 5.6. 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, `plot 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 49 for an example). Meta data (if any) must be provided after the coordinates and after error bar bounds (if any) in square brackets:

```
\addplot plot coordinates {
    (1300,1e-6) [1]
    (2600,5e-7) [2]
    (4000,1e-7) [3]
};
```

Please refer to the documentation of `scatter src` on page 47 for more information about per point meta data.

### 4.2.2 Reading Coordinates From Files

`\addplot file {`⟨*name*⟩`};`
`\addplot[`⟨*style options*⟩`] plot[`⟨*behavior options*⟩`] file {`⟨*name*⟩`}` ⟨*trailing path commands*⟩`;`

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

```
\addplot plot 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 [3, 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 `scatter src` 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 47 for information and examples about per point meta data and page 49 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=0,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.

### 4.2.3 Reading Coordinates From Tables

`\addplot table [`⟨*column selection*⟩`]{`⟨*file*⟩`};`

`\addplot[`⟨*style options*⟩`] plot[`⟨*behavior options*⟩`] table [`⟨*column selection*⟩`]{`⟨*file*⟩`}` ⟨*trailing path commands*⟩`;`

PGFPLOTS comes with a new plotting command, the '`plot table`' macro. It's usage 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}\table
...
\addplot table[x=dof,y=L2] from \table;
...
\addplot table[x=dof,y=Lmax] from \table;
...
```

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 5.6 for details about error bars.
- It is possible to read per point meta data (usable in `scatter src`, see page 47) 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|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.
- It *is* possible to create new columns out of existing ones, 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.

`/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/x index={⟨column index⟩}`

`/pgfplots/table/y 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 error`={⟨*column name*⟩}<br>
`/pgfplots/table/y error`={⟨*column name*⟩}<br>
`/pgfplots/table/x error index`={⟨*column index*⟩}<br>
`/pgfplots/table/y error index`={⟨*column index*⟩}

These keys define input sources for error bars with explicit error values. Please see section 5.6 for details.

`/pgfplots/table/meta`={⟨*column name*⟩}<br>
`/pgfplots/table/meta index`={⟨*column index*⟩}

These keys define input sources for per point meta data. Please see page 47 for details about meta data or the documentation for `plot coordinates` and `plot file` for further information.

`/pgfplots/table/col sep`=`space`|`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").

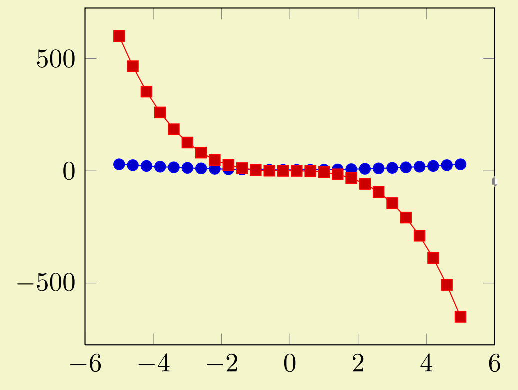
**Attention:** Usually, `\addplot table` only picks required entries, requiring linear runtime complexity. As soon as `read completely` is activated, tables are loaded completely into memory. Due to datastructures issues ("macro append runtime"), the runtime complexity for `read completely` is $O(N^2)$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

### 4.2.4 Computing Coordinates with Mathematical Expressions

`\addplot expression {`⟨*math expr*⟩`} ;`<br>
`\addplot[`⟨*style options*⟩`] plot[`⟨*behavior options*⟩`] expression {`⟨*math expr*⟩`}` ⟨*trailing path commands*⟩`;`

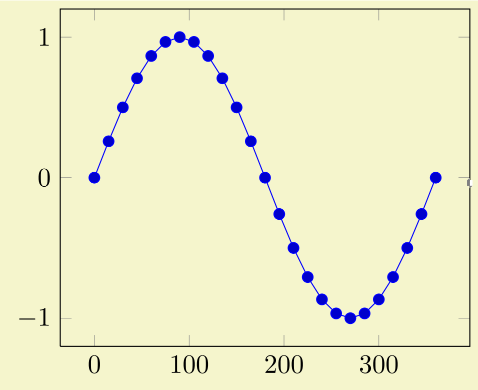
This input method allows to provide mathematical expressions which will be sampled. But unlike `plot gnuplot`, the expressions are evaluated using the math parser of pgf, no external program is required.

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.



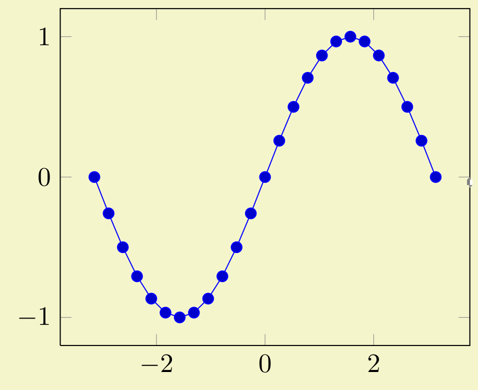
```
\begin{tikzpicture}
\begin{axis}
    \addplot expression {x^2 + 4};
    \addplot expression {-5*x^3 - x^2};
\end{axis}
\end{tikzpicture}
```

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



```
\begin{tikzpicture}
\begin{axis}
    \addplot expression[domain=0:360]
        {sin(x)};
\end{axis}
\end{tikzpicture}
```

If you want to use radians, use



```
\begin{tikzpicture}
\begin{axis}
    \addplot expression[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`, for example. Then, `x` will be the same as `\t` (there won't be a short-hand name for user defined variable names).

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 constanst `pi` and `e`, `^` (power operation), `factorial`[2], `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 [3], the FPU routines have been developed as part of PGFPLOTS. The documentation for both parts can be found in [3].

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

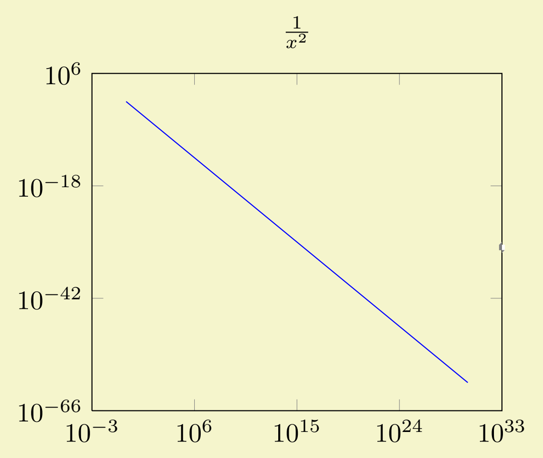
3. If the $x$ axis is logarithmic, samples will be drawn logarithmically.

4. Please note that plot expression does not allow per point meta data.

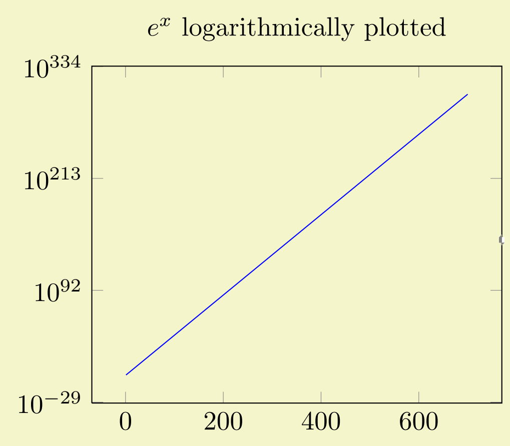
**About the precision and number range:** Starting with version 1.2, `plot expression` uses a floating point unit. The FPU provides the full data range of scientific computing with a relative precision between $10^{-4}$ and $10^{-6}$. The `/pgf/fpu` key provides some more details.

[2] Starting with PGF versions newer than 2.00, you can use the postfix operator `!` instead of `factorial`.

In case the `fpu` does not provide the desired mathematical function or is too slow[3], you should consider using the `plot gnuplot` method which invokes the external, freely available program `gnuplot` as desktop calculator.



```
\begin{tikzpicture}
    \begin{loglogaxis}[
        title={$\frac{1}{x^2}$}]
    \addplot[blue]
        expression[domain=1:1e30]
        {x^-2};
    \end{loglogaxis}
\end{tikzpicture}
```



```
\begin{tikzpicture}
    \begin{semilogyaxis}[
        title={$e^x$ logarithmically plotted}]
    \addplot[blue]
        expression[domain=1:700]
        {exp(x)};
    \end{semilogyaxis}
\end{tikzpicture}
```

`\addplot` **{⟨*math expression*⟩}** `;`
`\addplot[`⟨*style options*⟩`] plot[`⟨*behavior options*⟩`]` **{⟨*math expression*⟩}** ⟨*trailing path commands*⟩`;`

Use

`\addplot {`⟨*math expression*⟩`};`

as short-hand equivalent for

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

`\addplot` **(⟨*x expression*⟩,⟨*y expression*⟩)** `;`
`\addplot[`⟨*style options*⟩`] plot[`⟨*behavior options*⟩`]` **(⟨*x expression*⟩,⟨*y expression*⟩)** ⟨*trailing path commands*⟩`;`

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, you can't use round braces for either ⟨*x expression*⟩ or ⟨*y expression*⟩. You will need to introduce curly braces additionally to round braces.

`/pgfplots/domain=`⟨*start*⟩`:`⟨*end*⟩ (initially `[-5:5]`)

Determines the plotted range. This is not necessarily the same as the axis limits (which are configured with the `xmin`/`xmax` options).

[3] Or in case you find a bug. . .

This option is used for `plot expression` and for `plot gnuplot`.

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

**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`)

Sets the number of sample points for `plot expression` and `plot gnuplot`.

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[4]

```
\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.

**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`.

### 4.2.5 Computing Coordinates with Mathematical Expressions (gnuplot)

`\addplot gnuplot {`⟨*gnuplot code*⟩`};`
`\addplot[`⟨*style options*⟩`] plot[`⟨*behavior options*⟩`] gnuplot {`⟨*gnuplot code*⟩`}` ⟨*trailing path commands*⟩`;`

In contrast to `plot expression`, the `plot gnuplot` command 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[5]).

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.
- `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`.

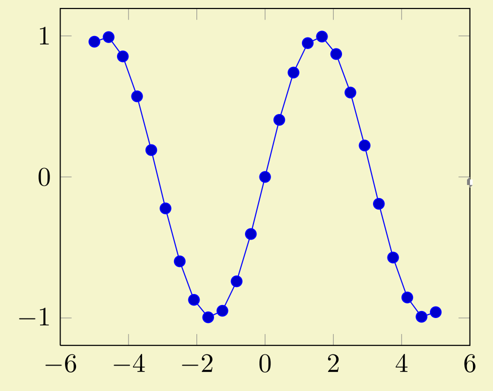
Since system calls are a potential danger, they need to be enabled explicitly using command line options, for example

[4] 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).

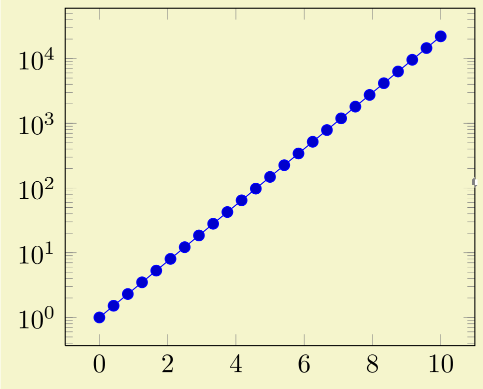
[5] Please note that PGFPLOTS produces slightly different files than TikZ when used with `plot gnuplot` (it configures high precision output). You should use different ids to avoid conflicts in such a case.

```
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.



```
\begin{tikzpicture}
\begin{axis}
\addplot plot[id=sin]
    gnuplot{sin(x)};
\end{axis}
\end{tikzpicture}
```



```
\begin{tikzpicture}
\begin{semilogyaxis}
\addplot plot[id=exp,domain=0:10]
    gnuplot{exp(x)};
\end{semilogyaxis}
\end{tikzpicture}
```

The ⟨*style options*⟩ determine the appearance of the plotted function; these parameters also affect the legend. The ⟨*behavior options*⟩ are specific to the gnuplot interface. These options are described in all detail in [3, section 18.6]. A short summary is shown below.

Please note that `plot gnuplot` does not allow per point meta data.

Please refer to [3, section 18.6] for more details about `plot function` and the `gnuplot` interaction.

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

Use

`\addplot function {`⟨*gnuplot code*⟩`};`

as alias for

`\addplot gnuplot {`⟨*gnuplot code*⟩`};`

`/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 [3, section 18.6] for details).

`/tikz/raw gnuplot` (no value)

Disables the use of `samples` and `domain`.

### 4.2.6 Using External Graphics as Plot Sources

`\addplot graphics {`⟨*file name*⟩`};`

`\addplot[⟨style options⟩] plot[⟨behavior options⟩] graphics {⟨file name⟩} ⟨trailing path commands⟩;`

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 shaded surface 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.

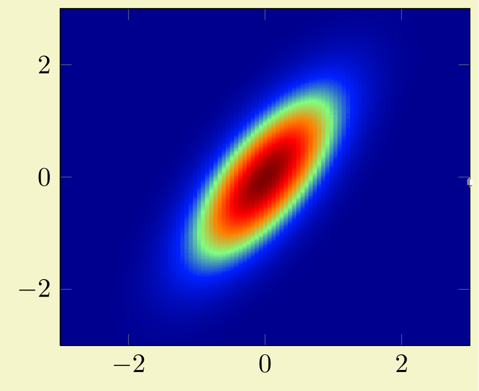
Let's start with an example: Suppose we use, for example, matlab to generate a surface plot like

```
[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
```

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



```
\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[6] 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).

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 rectanglar path which is as large as the bounding box (Matlab (tm) computes such `.eps` images). In this case, use the "Ungroup" method 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.

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

**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.

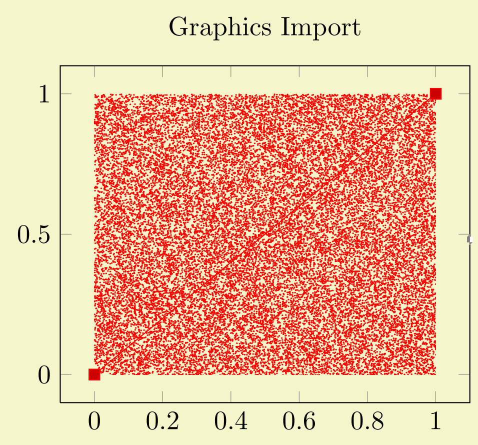
Let's follow this approach in a further example.

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:



```
\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.

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

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

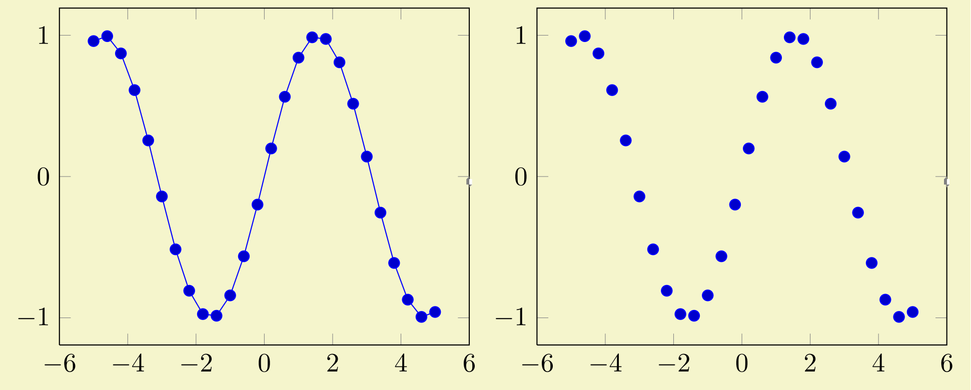
/pgfplots/plot graphics (no value)

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 independent of an axis.

\addplot+[⟨*style options*⟩] . . . ;

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



```
\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}
```

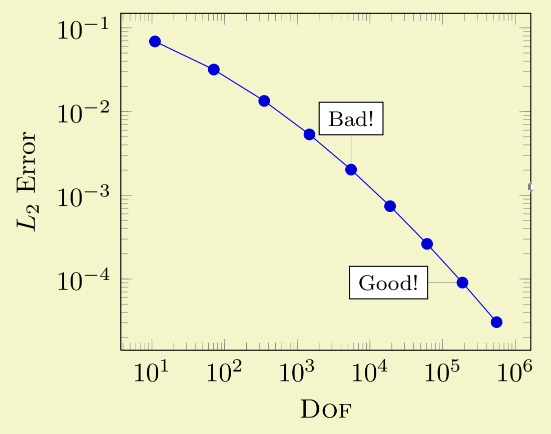
## 4.3 Accessing Axis Coordinates for Annotations

Coordinate system axis cs

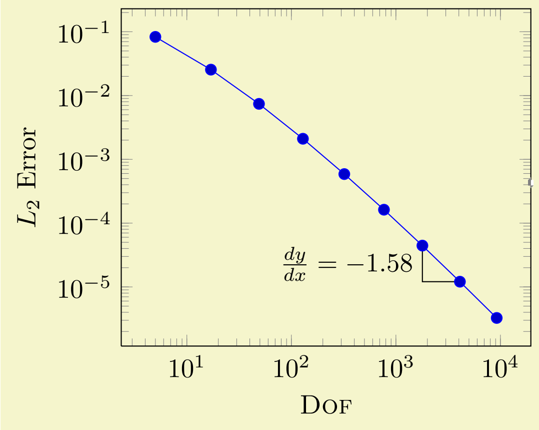
PGFPLOTS provides a new coordinate system for use inside of an axis, the "axis coordinate system", `axis cs`.

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);
```



```
\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}
```



```
\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!

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.

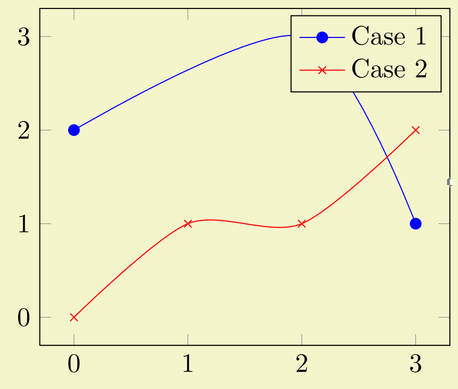
Predefined node current plot end

This coordinate will be defined for every plot. It is the last coordinate of the current plot.

## 4.4 Legend Commands

\addlegendentry{⟨*name*⟩}

Adds a single legend entry to the legend list. This will also enable legend drawing.



```
\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.

Optional argument are accepted with`\addlegendentry[`⟨*key-value-list*⟩`]{...}`. This does mainly affect some keys affecting the legend layout, support is very limited.

Using `\addlegendentry` disables the key `legend entries`.

\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 a comma–separated list of entries, one for each plot. It is processed using the PGF-foreach command[7]. 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.

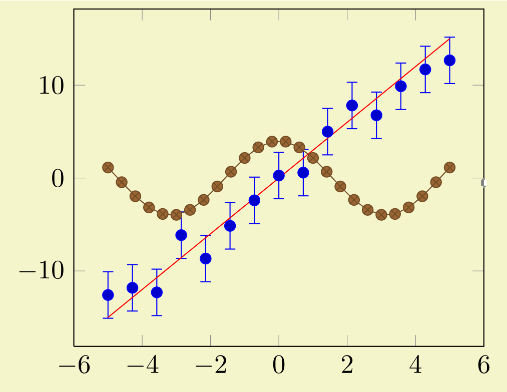
### 4.4.1 Legend Appearance

The legend appearance can be configured with the help of several styles and options. These options are described in section 5.4.2, under Axis Descriptions.

### 4.4.2 `\label` and `\ref` for Legend Creation

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.

[7] Older versions of PGFPLOTS used `\legend{first\\second\\third\\}` instead of comma–separated lists. This syntax is still accepted.



```
\begin{tikzpicture}[baseline]
\begin{axis}
    \addplot+[only marks]
        expression[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*⟩}

When used after `\addplot`, this command creates a LaTeX label named `{`⟨*label name*⟩`}`[8]. 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 independant label[9].

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.

`\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`.

`/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}}
```

[8] This feature is *only* available in LaTeX, sorry.

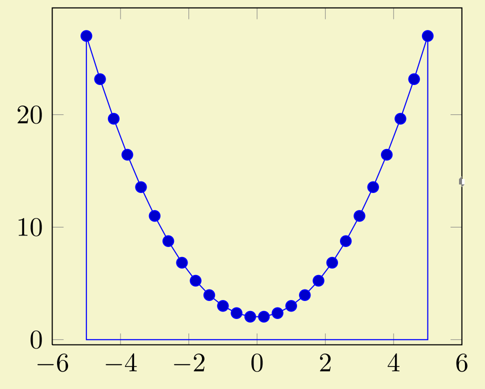
[9] Please note that you can't use the label/ref mechanism in conjunction with image externalization as this will (naturally) lead to undefined references.

## 4.5 Closing Plots

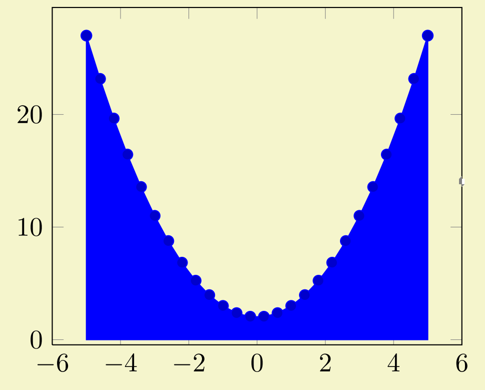
`\closedcycle`

Provide `\closedcycle` as ⟨*trailing path commands*⟩ after `\addplot` to draw a closed line from the last plot coordinate to the first one.

Use `\closedcycle` whenevery you intend to fill the area under a plot.

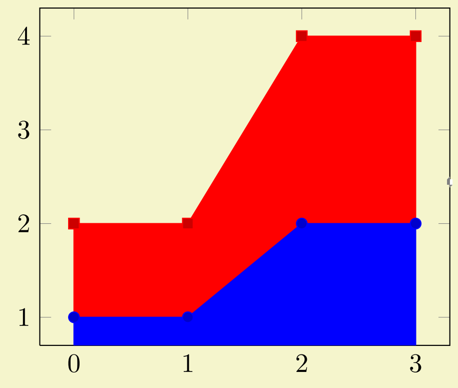


```
\begin{tikzpicture}
    \begin{axis}
    \addplot {x^2+2} \closedcycle;
    \end{axis}
\end{tikzpicture}
```



```
\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[10].



```
\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}
```

## 4.6 Other Commands

`\autoplotspeclist`

This command should no longer be used, although it will be kept as technical implementation detail. Please use the '`cycle list`' option, section 5.3.6.

[10] 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 unsymmetric, `\closedcycle` will use `const plot mark right` as reversed sequence for `const plot mark left`.

`\pgfmathlogtologten`⟨*number*⟩

Assigns the result of ⟨*number*⟩/ log(10) to `\pgfmathresult`.

`\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[11] for `{`⟨*number*⟩`}`. This method is used for every tick label.

The number is printed using the current number printing options, see section 5.7 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.

`\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.

# 5 Option Reference

There are several required and even more optional arguments to modify axes. They are used like

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

The overall appeareance can be changed with

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

for example. There are several other styles predefined to modify the appearance, see section 5.10.

`\pgfplotsset{`⟨*key-value-list*⟩`}`

Defines or sets all options in `{`⟨*key-value-list*⟩`}`.

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,

[11] This method was previously `\prettyprintnumber`. It's functionality has been included into pgf and the old command is now deprecated.

```
\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 5.10 in more detail.

## 5.1 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 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`'.

## 5.2 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 [3, section 18.8].

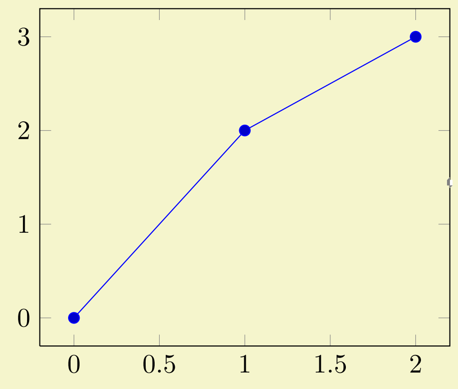
Plot types are part of the plot style, so they are set with options. The following list contains a short summary of the PGF plot library, [3, section 18.8].

### 5.2.1 Linear Plots

`/tikz/sharp plot` (no value)

\addplot+[sharp plot]

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



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

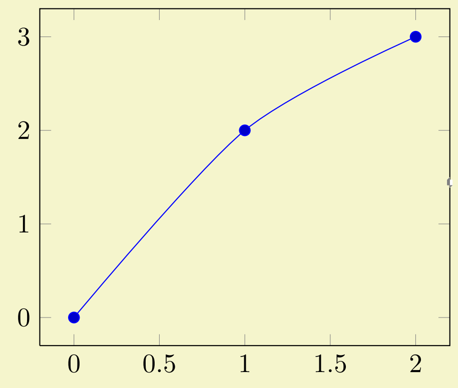
The '+' here means to use the normal plot cycle list and append 'sharp plot' to its option list.

### 5.2.2 Smooth Plots

/tikz/smooth (no value)

\addplot+[smooth]

Smooth plots interpolate smoothly between successive points.



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

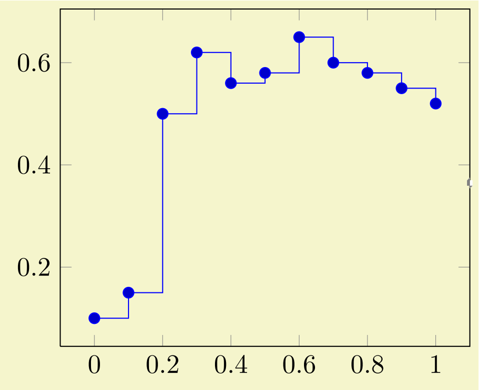
### 5.2.3 Constant Plots

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

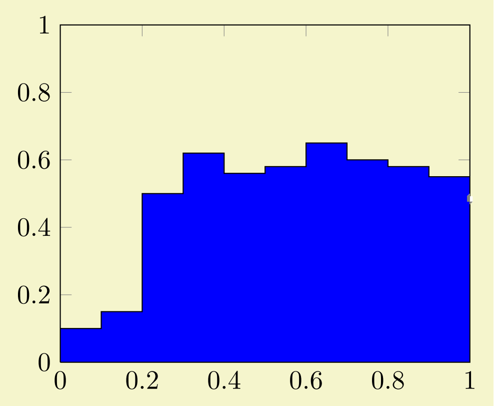
/tikz/const plot (no value)

\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.



```
\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}
```



```
\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)

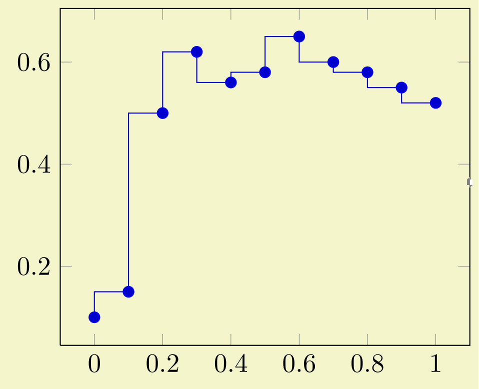
`\addplot+[const plot mark left]`

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.

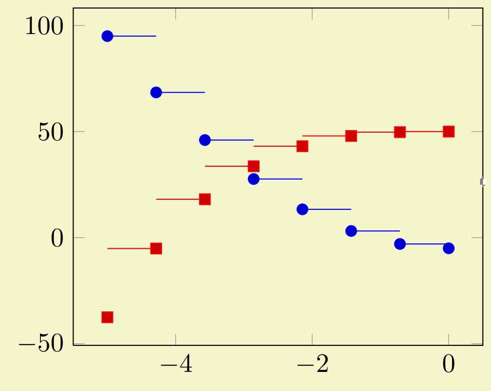


```
\begin{tikzpicture}
\begin{axis}
\addplot+[const plot mark right]
coordinates
{(0,0.1)    (0.1,0.15)  (0.2,0.5)   (0.3,0.62)
 (0.4,0.56) (0.5,0.58)  (0.6,0.65)  (0.7,0.6)
 (0.8,0.58) (0.9,0.55)  (1,0.52)};
\end{axis}
\end{tikzpicture}
```

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

`\addplot+[jump mark left]`

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



```
\begin{tikzpicture}
\begin{axis}[samples=8]
\addplot+[jump mark left]
    expression[domain=-5:0]
    {4*x^2 - 5};

\addplot+[jump mark right]
    expression[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.

### 5.2.4 Bar Plots

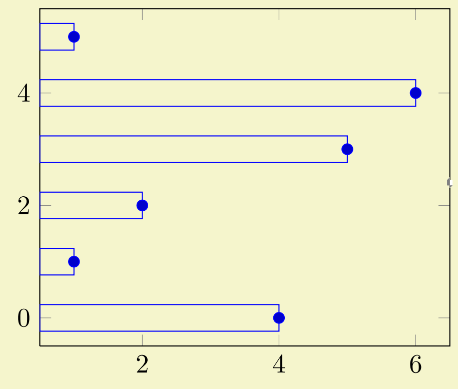
Bar plots place horizontal or vertical bars at coordinates. Multiple bar plots in one axis can be stacked on top of each other or aligned next to each other.

/tikz/xbar (no value)

\addplot+[xbar]

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

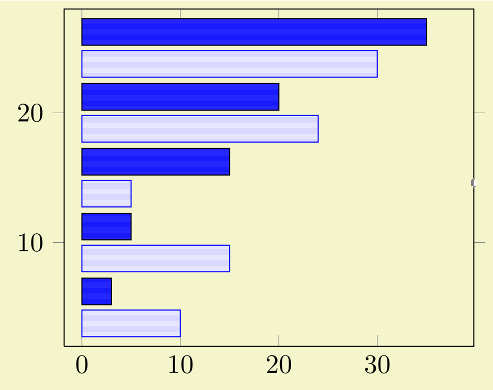
This option is used on a per-plot basis and configures only the visualization of coordinates. The figure-wide style `/pgfplots/xbar` also sets reasonable options for ticks, legends and multiple plots.



```
\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.



```
\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.

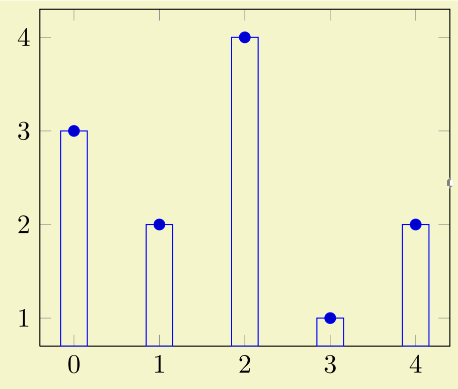
```
/pgfplots/xbar/.style={
    bar cycle list,
    tick align=outside,
    /pgfplots/legend image code/.code=
        {\draw[##1,bar width=3pt,yshift=-0.2em,bar shift=0pt]
            plot coordinates {(0cm,0.8em) (2*\pgfplotbarwidth,0.6em)};},
    /pgf/bar shift={%
            % total width = n*w + (n-1)*skip
            % -> subtract half for centering
            -0.5*(\numplots*\pgfplotbarwidth + (\numplots-1)*#1)  +
            % the '0.5*w' is for centering
            (.5+\plotnum)*\pgfplotbarwidth + \plotnum*#1},
    /tikz/xbar},
```

The formular for `bar shift` assigns shifts dependend 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]`

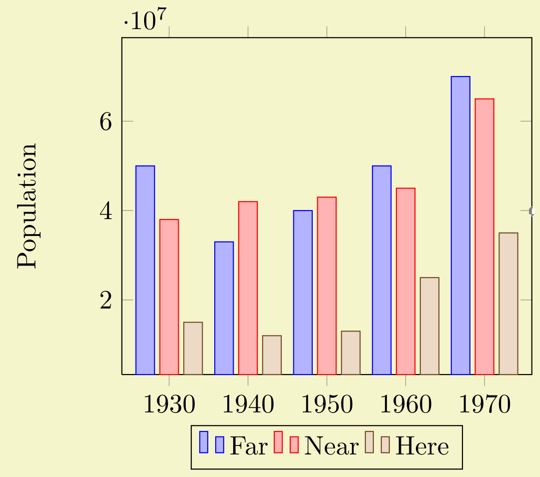
Like `xbar`, this option generates bar plots. It draws vertical bars between the $(x = 0)$ line and each input coordinate.



```
\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:



```
\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`.

`/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*⟩`}`. Furthermore, it installs the style `bar cycle list`. It is defined similarly to `/pgfplots/xbar`.

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

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

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

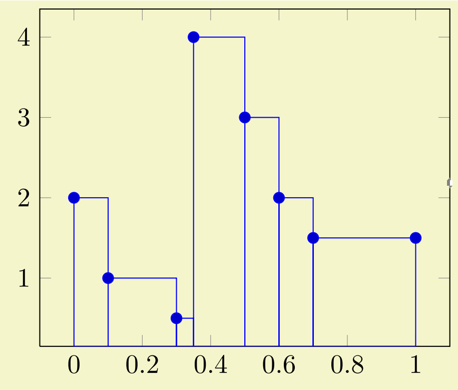
Configures a shift for `xbar` and `ybar`. Use `bar shift` together with `bar width` to draw multiple bar plots into the same axis. It is accepted to provide mathematical expressions.

`/tikz/ybar interval` (no value)

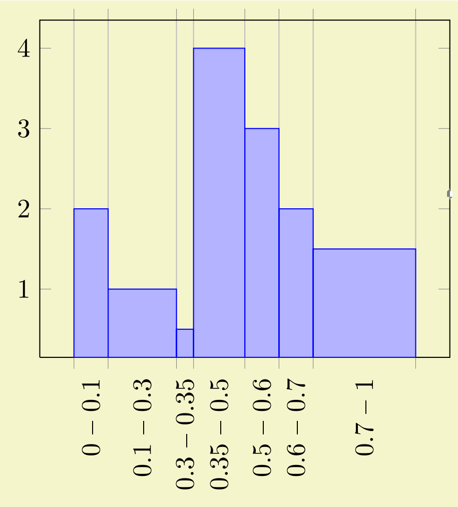
`\addplot+[ybar interval]`

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

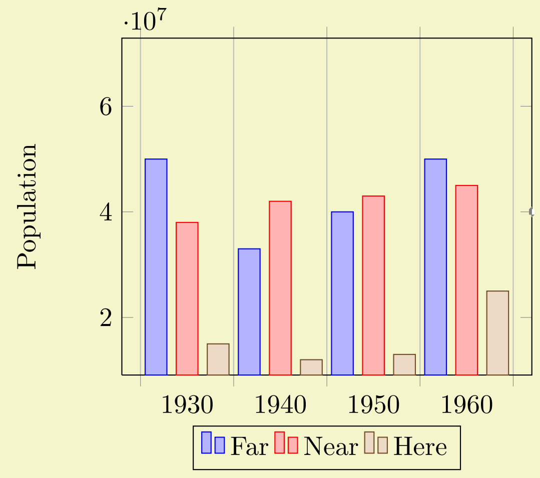
It is installed on a per-plot basis and configures *only* the visualization of coordinates. See the style `/pgfplots/ybar interval` which configures the appearance of the complete figure.



```
\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}
```



```
\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}
```



```
\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}
```

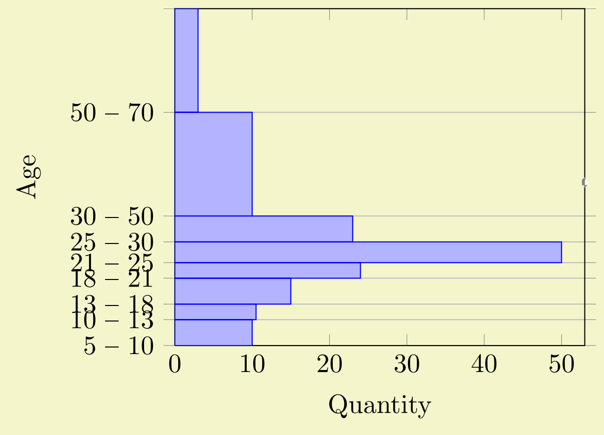
`/pgfplots/ybar interval={`*⟨relative width⟩*`}` (style, default `1`)

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.



```
\begin{tikzpicture}
\begin{axis}[
    xmin=0,xmax=53,
    ylabel=Age,
    xlabel=Quantity,
    y label style={yshift=0.7cm},
    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)

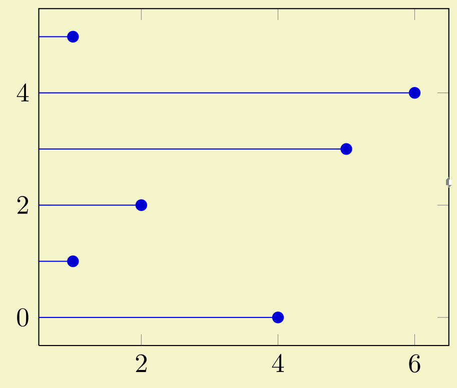
These are style keys which set `x tick label as interval` and configure the tick appearance to be {⟨*start*⟩} – {⟨*end*⟩} for each tick interval.

### 5.2.5 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)

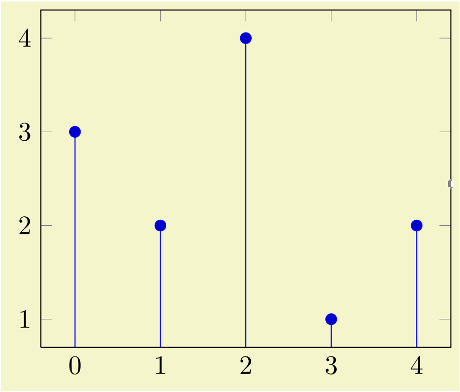
\addplot+[xcomb]



```
\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]

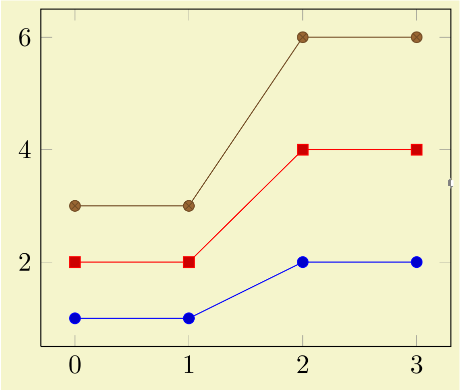


```
\begin{tikzpicture}
\begin{axis}
\addplot+[ycomb] plot coordinates
    {(0,3) (1,2) (2,4) (3,1) (4,2)};
\end{axis}
\end{tikzpicture}
```

### 5.2.6 Stacked Plots

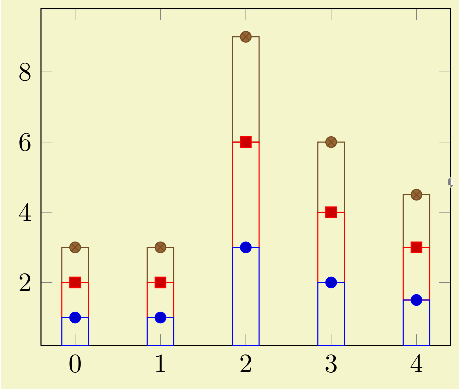
`/pgfplots/stack plots=x|y|false` (initially `false`)

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.

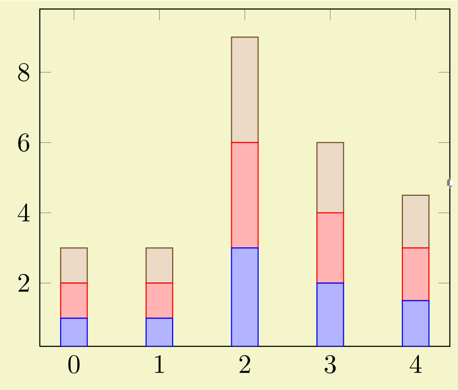


```
\begin{tikzpicture}
    \begin{axis}[stack plots=y]
    \addplot coordinates
        {(0,1) (1,1) (2,2) (3,2)};
    \addplot coordinates
        {(0,1) (1,1) (2,2) (3,2)};
    \addplot coordinates
        {(0,1) (1,1) (2,2) (3,2)};
    \end{axis}
\end{tikzpicture}
```

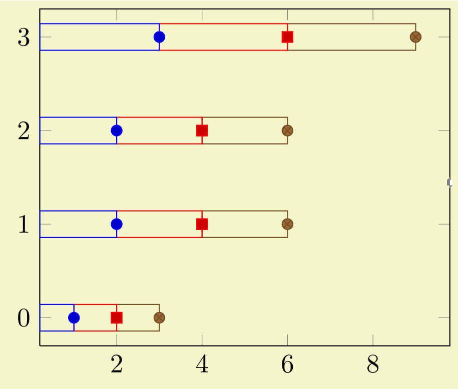
`stack plots` is particularly useful for bar plots. The following examples demonstrate its functionality. Normally, it is advisable to use the styles `ybar stacked` and `xbar stacked` which also set some other options.



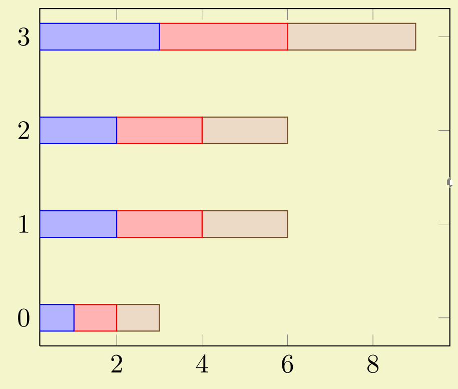
```
\begin{tikzpicture}
    \begin{axis}[stack plots=y,/tikz/ybar]
    \addplot coordinates
        {(0,1) (1,1) (2,3) (3,2) (4,1.5)};
    \addplot coordinates
        {(0,1) (1,1) (2,3) (3,2) (4,1.5)};
    \addplot coordinates
        {(0,1) (1,1) (2,3) (3,2) (4,1.5)};
    \end{axis}
\end{tikzpicture}
```



```
\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}
```



```
\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}
```



```
\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`)

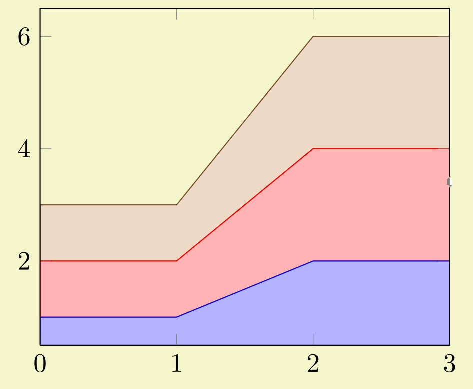
A style similar to `/pgfplots/xbar stacked` for the interval based bar plot variant.

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

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

### 5.2.7 Area Plots

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



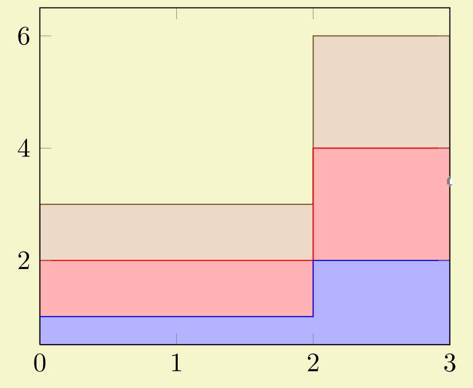
```
\begin{tikzpicture}
    \begin{axis}[
        stack plots=y,
        area style,
        enlarge x limits=false]
    \addplot coordinates
        {(0,1) (1,1) (2,2) (3,2)}
        \closedcycle;
    \addplot coordinates
        {(0,1) (1,1) (2,2) (3,2)}
        \closedcycle;
    \addplot coordinates
        {(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.

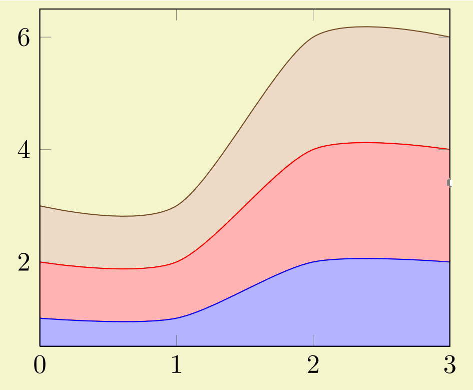
`/pgfplots/area style` (style, no value)

A style which sets

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



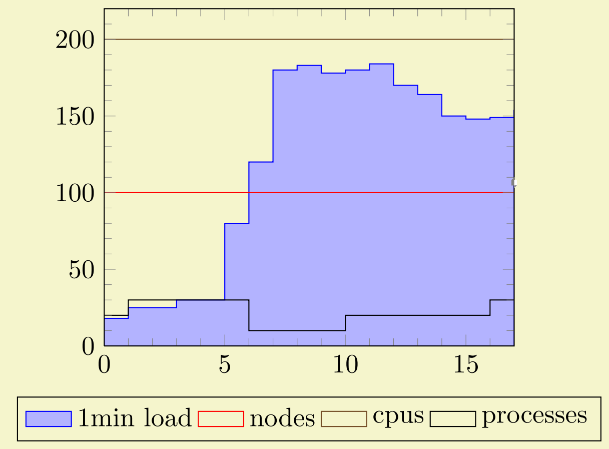
```
\begin{tikzpicture}
    \begin{axis}[
        const plot,
        stack plots=y,
        area style,
        enlarge x limits=false]
    \addplot coordinates
        {(0,1) (1,1) (2,2) (3,2)}
        \closedcycle;
    \addplot coordinates
        {(0,1) (1,1) (2,2) (3,2)}
        \closedcycle;
    \addplot coordinates
        {(0,1) (1,1) (2,2) (3,2)}
        \closedcycle;
    \end{axis}
\end{tikzpicture}
```



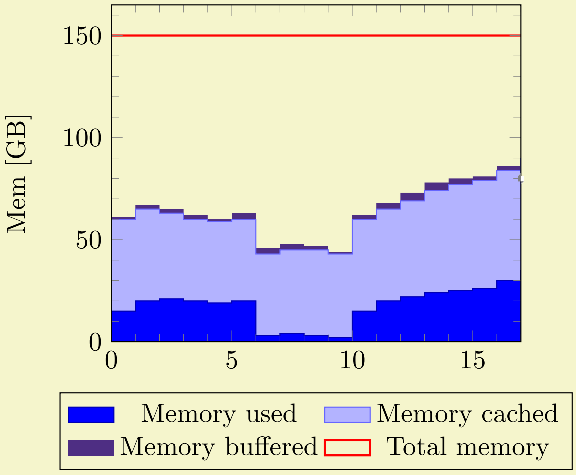
```
\begin{tikzpicture}
    \begin{axis}[
        smooth,
        stack plots=y,
        area style,
        enlarge x limits=false]
    \addplot coordinates
        {(0,1) (1,1) (2,2) (3,2)}
        \closedcycle;
    \addplot coordinates
        {(0,1) (1,1) (2,2) (3,2)}
        \closedcycle;
    \addplot 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}\table
\pgfplotstabletypeset\table
```



```
\pgfplotstableread
    {pgfplots.timeseries.dat}
    {\table}

\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 \table
        \closedcycle;
    \addplot table[x=time,y=nodes] from \table;
    \addplot table[x=time,y=cpus] from \table;
    \addplot table[x=time,y=processes]
        from \table;
    \legend{1min load,nodes,cpus,processes}
    \end{axis}
\end{tikzpicture}
```



```
\pgfplotstableread{pgfplots.timeseries.dat}\table

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

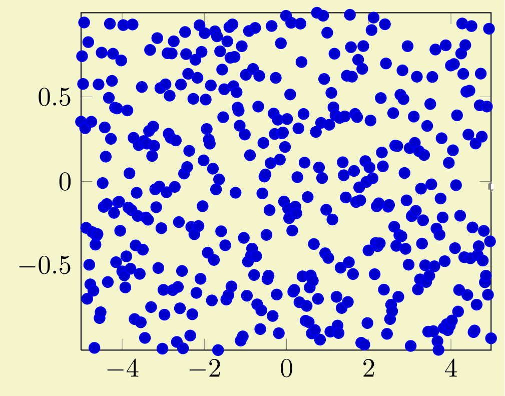
### 5.2.8 Scatter Plots

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.



```
\begin{tikzpicture}
    \begin{axis}[enlargelimits=false]
    \addplot+[only marks]
        expression[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 5.3 on page 52.

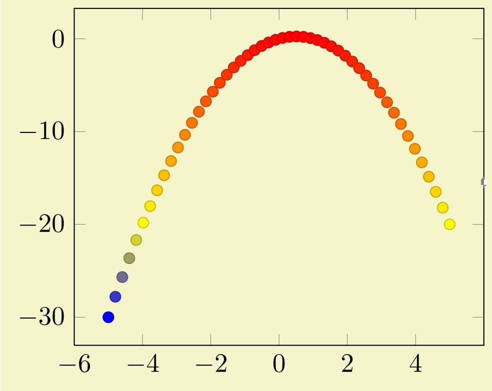
More sophisticated scatter plots change the marker appearance for each data point. An example is that marker colors depend on the magnitude of function values $f(x)$ or other provided coordinates. The term "scatter plot" will be used for this type of plots in the following sections.

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

`/tikz/scatter` (no value)

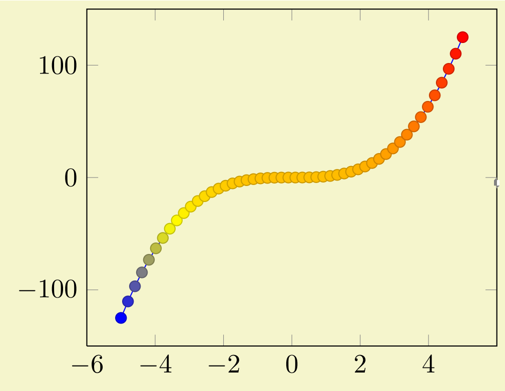
`\addplot+[scatter]`

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.



```
\begin{tikzpicture}
    \begin{axis}
    \addplot+[scatter,only marks]
        expression[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.



```
\begin{tikzpicture}
    \begin{axis}
    \addplot+[scatter]
        expression[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|x|y|z|explicit|explicit symbolic` (initially `none`)

This key is necessary for any scatter plot. It needs to be provided as `{`⟨*behavior option*⟩`}` for `\addplot` to configure the value used to determine marker appearances.

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 choices `x`, `y` and `z` will use either the $x$, $y$ or $z$ coordinates to determine marker options[12]. 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. Finally, `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 somethting with it. See, for example, the `scatter/classes` style below.

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

[12] The coordinates are used after any coordinate filters, logarithms or stacked-plot computations have been applied.

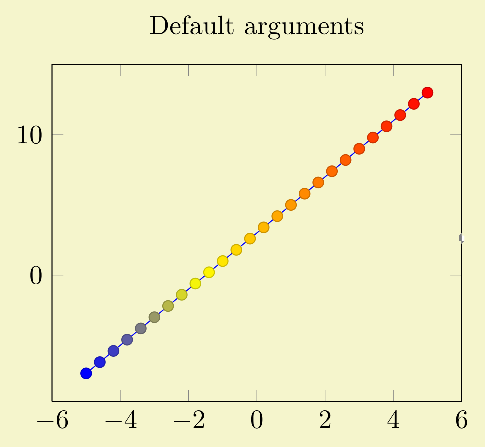
```
\begin{tikzpicture}
    \begin{axis}
        % provide color data explicitly using [<data>]
        % behind coordinates:
        \addplot+[scatter]
            [scatter src=explicit]
            coordinates {
                (0,0) [1.0e10]
                (1,2) [1.1e10]
                (2,3) [1.2e10]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

        % 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`.

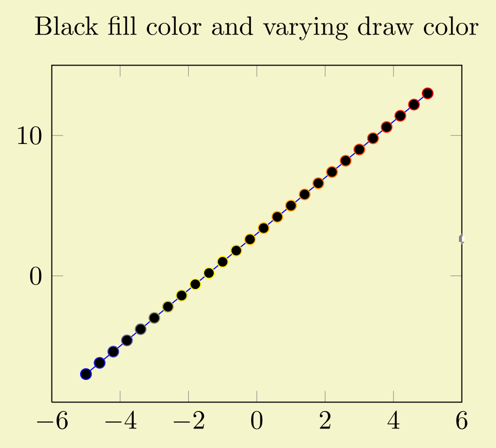
`/pgfplots/scatter/use mapped color={`⟨*options for each marker*⟩`}` (style, initially `draw=mapped color!80!black,fill=mapped color`)

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 colormap linearly. Then, it evaluates the options provided as `{`⟨*options for each marker*⟩`}` which are expected to depend on `mapped color`.

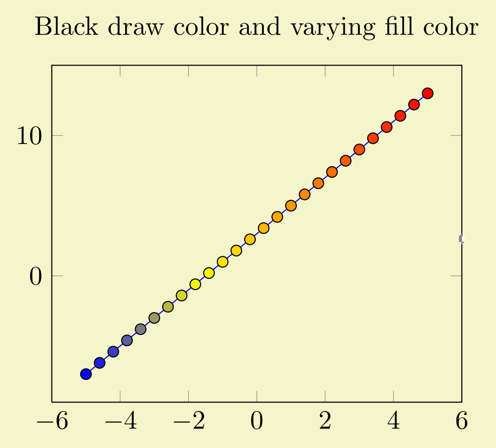
The user interface for colormaps is described in section 5.3.5.



```
\begin{tikzpicture}
\begin{axis}[title=Default arguments]
\addplot+[scatter]
    expression[scatter src=y]
    {2*x+3};
\end{axis}
\end{tikzpicture}
```



```
\begin{tikzpicture}
\begin{axis}[
    title=Black fill color and varying draw color,
    scatter/use mapped color=
        {draw=mapped color,fill=black}]
\addplot+[scatter]
    expression[scatter src=y]
    {2*x+3};
\end{axis}
\end{tikzpicture}
```



```
\begin{tikzpicture}
\begin{axis}[
    title=Black draw color and varying fill color,
    scatter/use mapped color=
        {draw=black,fill=mapped color}]
\addplot+[scatter]
    expression[scatter src=y]
    {2*x+3};
\end{axis}
\end{tikzpicture}
```

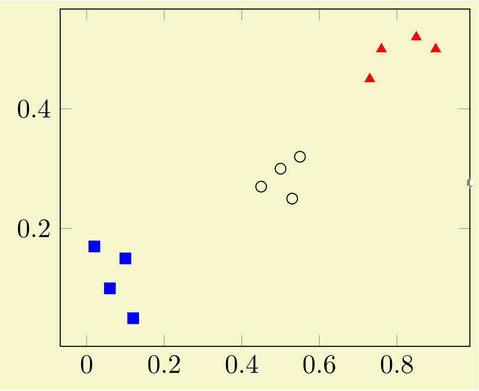
This key is actually a style which redefines `@pre marker code` and `@post marker code` (see below).

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

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 classname*⟩} contains a comma-separated list which associates appearance options to each class label.

If you need different `scatter/classes` arguments per plot, they must be given as {⟨*behavior option*⟩}, not as style option.

[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.



```
\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]
        plot[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.

```
\begin{tikzpicture}
\begin{axis}[scatter/classes={
    0={mark=square*,blue},%
    1={mark=triangle*,red},%
    2={mark=o,draw=black,fill=black}}]

    % Assumes datafile.dat looks like
    % x             y               label
    % 5.000000e-01  7.500000e-01    1
    % 1.000000e+00  6.718750e-01    2
    % 1.000000e+00  5.597630e-01    2
    % 5.000000e-01  1.250000e-01    2
    % 1.000000e+00  6.603350e-01    0
    % 5.000000e-01  0.000000e+00    0
    % 0.000000e+00  5.000000e-01    0
    % 1.000000e+00  0.000000e+00    2
    % 5.000000e-01  1.250000e-01    1
    % 1.000000e+00  6.213180e-01    1
    % ...
    \addplot[scatter,only marks]
            table[scatter src=explicit symbolic,x index=x,y index=y,meta=label]
            {datafile.dat}
        ;
\end{axis}
\end{tikzpicture}
```

In general, the format of `{`⟨*styles for each classname*⟩`}` 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 classname*⟩`}`! So, instead of assigning `mark options`, you can simply provide the options directly. They apply only to markers anyway.

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

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

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

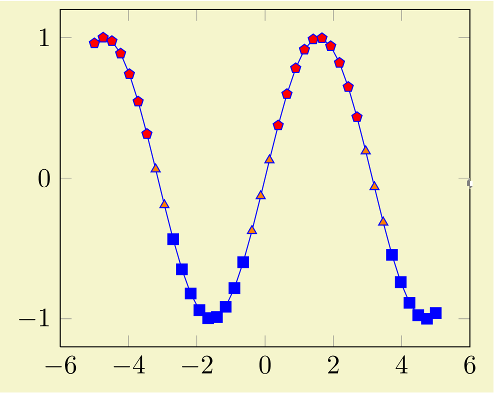
The scatter plot algorithm works as follows:

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[14] 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 code of `scatter/@pre marker code` is evaluated (without arguments).
5. The standard code which draws markers is evaluated.
6. 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[15].



```
\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]
   expression[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.

### 5.2.9 Interrupted Plots

Sometimes it is desireable to draw parts of a single plot separately, without connection between the parts (discontinuities). There is limited support for such an application.

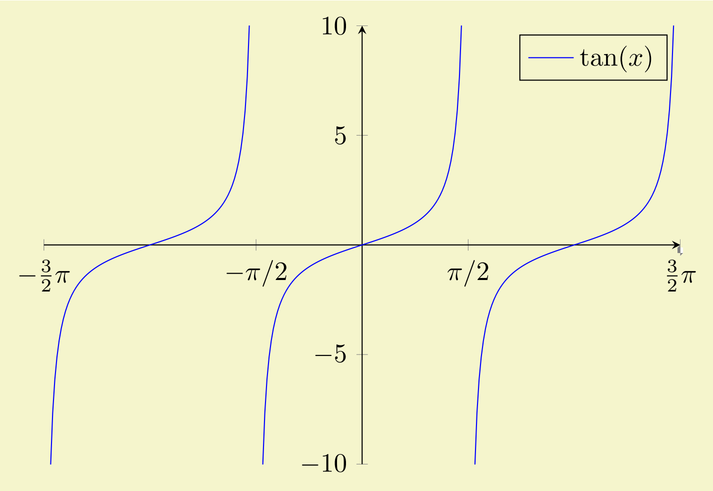
[14] During the evaluation, the public macros `\pgfplotspointmeta` and `\pgfplotspointmetarange` indicate the source coordinate and the source coordinate range in the format $a : b$ (for log–axis, they are given in fixed point representation and for linear axes in floating point).

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

`/pgfplots//pgfplots/forget plot`={⟨*true,false*⟩} (initially `false`)

This key tells PGFPLOTS to add a plot without changing cycle list position and legends. This key is described in all detail on page 113.

However, it can be used to get the interesting effect of “interrupted plot”, so it is also discussed here:



```
\begin{tikzpicture}
\begin{axis}[
    width=10cm, height=210pt,
    xmin=-4.7124, xmax=4.7124,
    ymin=-10, ymax=10,
    xtick={-4.7124,-1.5708,...,10},
    xticklabels={$-\frac32 \pi$,$-\pi/2$,$\pi/2$,$\frac32 \pi$},
    axis x line=center,axis y line=center,
    no markers,
    samples=100]

% Use gnuplot as calculator here. The first two plots won't be counted:
\addplot gnuplot[id=tan0,forget plot,domain=-1.5*pi+0.003:-0.5*pi-0.003] {tan(x)};
\addplot gnuplot[id=tan1,forget plot,domain=-0.5*pi+0.003: 0.5*pi-0.003] {tan(x)};
\addplot gnuplot[id=tan2,            domain= 0.5*pi+0.003: 1.5*pi-0.003] {tan(x)};
\legend{$\tan(x)$}
\end{axis}
\end{tikzpicture}
```

The interesting part is in the `\addplot` commands. The `id` is specific to the gnuplot interface (and can be omitted). The `domain` option defines separate domains for every plot part. Due to the `forget plot` key, the `cycle list` position is not updated so all three plots use the same line specification. Furthermore, only the last command affects the legend (and advances the cycle list).

**Remark:** The `forget plot` feature is *not very sophisticated*. In particular, it has the following **restrictions**:

1. Besides the `cycle list` side–effect, no styles are communicated between successive plots.
2. It won’t work together with `stack plots`.

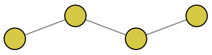
## 5.3 Markers and Linestyles

The following options of Ti*k*Z are available to plots.

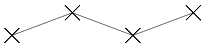
### 5.3.1 Markers

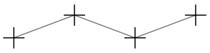
This list is copied from [3, section 29]:

`mark=*`

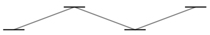


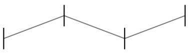
`mark=x`

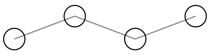


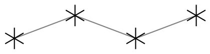
`mark=+` 

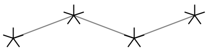
And with `\usetikzlibrary{plotmarks}`:

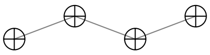
`mark=−` 

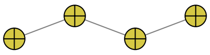
`mark=|` 

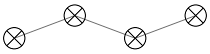
`mark=o` 

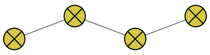
`mark=asterisk` 

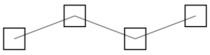
`mark=star` 

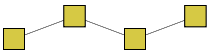
`mark=oplus` 

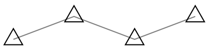
`mark=oplus*` 

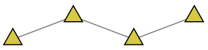
`mark=otimes` 

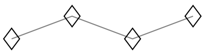
`mark=otimes*` 

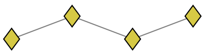
`mark=square` 

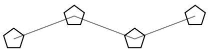
`mark=square*` 

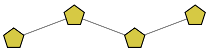
`mark=triangle` 

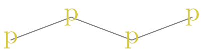
`mark=triangle*` 

`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.

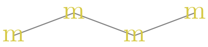
All these options have been drawn with the additional options

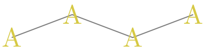
```
\draw[
    gray,
    thin,
    mark options={%
        scale=2,fill=yellow!80!black,draw=black
    }
]
```

Please see section 5.3.4 for how to change draw- and fill colors.

/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.

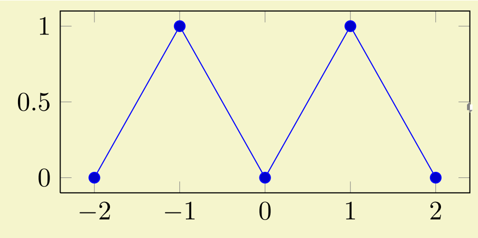
The first choice is highly flexible and possibly slow, the second is very fast and usually enough.

/tikz/mark size={⟨*dimension*⟩}

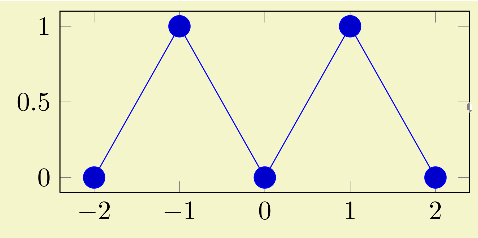
This TikZ option allows to set marker sizes to {⟨*dimension*⟩}. For circular markers, {⟨*dimension*⟩} is the radius, for other plot marks it is about half the width and height.

/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.



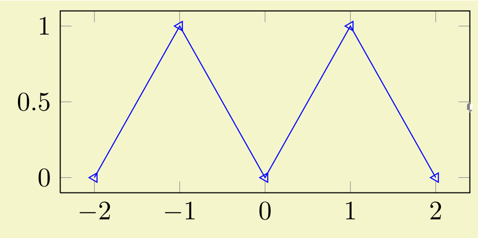
```
\begin{tikzpicture}
\begin{axis}[y=2cm]
    \addplot coordinates
        {(-2,0) (-1,1) (0,0) (1,1) (2,0)};
\end{axis}
\end{tikzpicture}
```



```
\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/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.



```
% 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}
```

`/pgfplots/no markers` (style, no value)

A style which appends `mark=none` to `every axis plot post`, which disables markers for every plot (even if the cycle list contains markers).

`/tikz/mark options={⟨options⟩}`

Resets `every mark` to `{⟨options⟩}`.

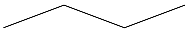
Markers paths are not subjected to clipping as other parts of the figure. Markers are either drawn completely or not at all.

Ti*k*Z offers more options for marker fine tuning, please refer to [3] for details.

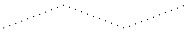
### 5.3.2 Line Styles

The following line styles are predefined in Ti*k*Z.

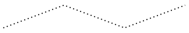
`/tikz/solid` (style, no value)



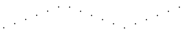
`/tikz/dotted` (style, no value)



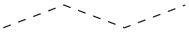
`/tikz/densely dotted` (style, no value)



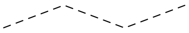
`/tikz/loosely dotted` (style, no value)



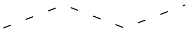
`/tikz/dashed` (style, no value)



`/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 [3] for details.

### 5.3.3 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`[16].

It may be useful to change fonts only for specific axis descriptions, for example using

```
\pgfplotsset{
    tick label style={font=\small},
    label style={font=\small},
    legend style={font=\footnotesize}
}
```

[16] ConTEXt and plain TEX users need to provide other statements, of course.

/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

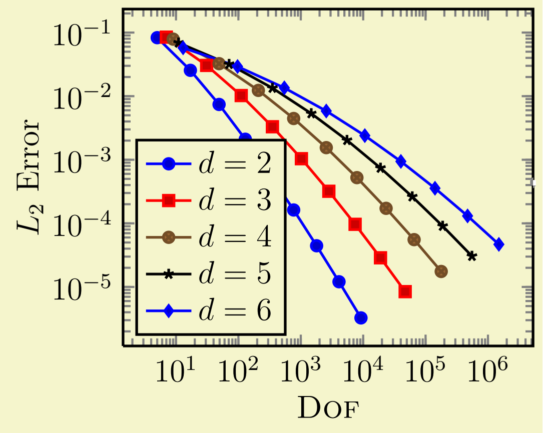
```
\pgfplotsset{every axis/.append style={
    thick,
    tick style={semithick}}}
```

The '`every axis plot`' style can be used to change line widths for plots only.

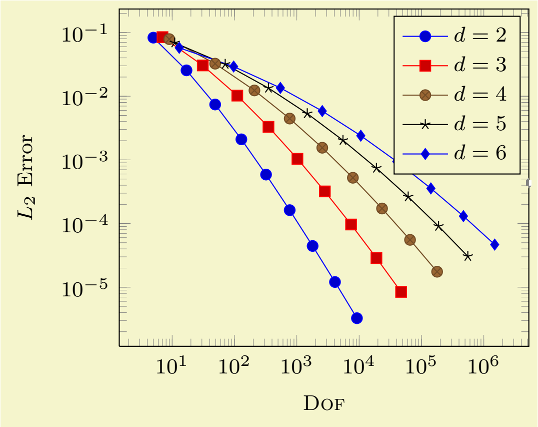
/tikz/ultra thin (no value)
/tikz/very thin (no value)
/tikz/semithick (no value)
/tikz/thick (no value)
/tikz/very thick (no value)
/tikz/ultra thick (no value)

These Ti*k*Z styles provide different predefined line widths.

This example shows the same plots as on page 11 (using `\plotcoords` as place holder for the commands on page 11), with different line width and font size.



```
\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}
```

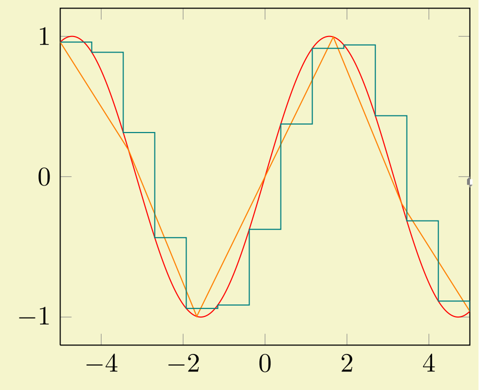


```
\pgfplotsset{every axis/.append style={
    font=\footnotesize,
    thin,
    tick style={ultra thin}}}
\begin{tikzpicture}
    \begin{loglogaxis}[
        xlabel=\textsc{Dof},
        ylabel=$L_2$ Error
    ]
    % see above for this macro:
    \plotcoords
    \legend{$d=2$,$d=3$,$d=4$,$d=5$,$d=6$}
    \end{loglogaxis}
\end{tikzpicture}
```

### 5.3.4 Colors

PGF uses the color support of `xcolor`. Therefore, the main reference for how to specify colors is the `xcolor` manual [1]. The PGF manual [3] 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 [1] (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.



```
\begin{tikzpicture}
    \begin{axis}[enlarge x limits=false]
    \addplot[red]
        expression[samples=500] {sin(deg(x))};

    \addplot[orange]
        expression[samples=7] {sin(deg(x))};

    \addplot[teal,const plot]
        expression[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, ■ red!50!green, ■ blue!50!yellow.

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 [1] for more details and color models.

`/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.

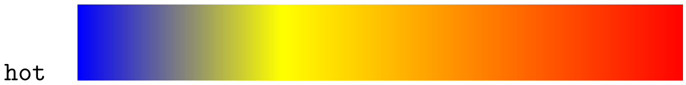
Use `draw=none` to disable drawing and `fill=none` to disable filling[17].

Since these keys belong to Ti*k*Z, the complete documentation can be found in the Ti*k*Z manual [3, Section "Specifying a Color"].

### 5.3.5 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 colormaps are described below.

Colormaps can be used, for example, in scatter plots (see section 5.2.8).

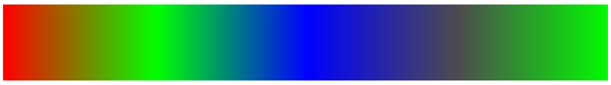
You can use `colormap` to create new color maps (see below).

`/pgfplots/colormap`={⟨*name*⟩}{⟨*color specification*⟩}

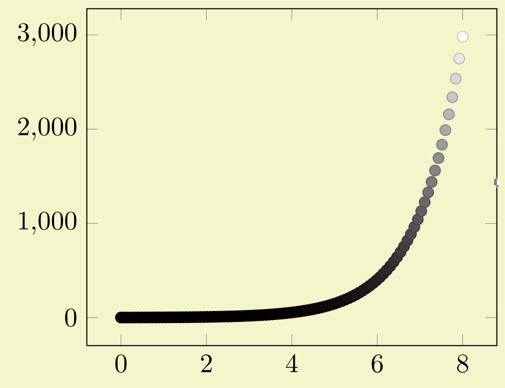
Defines a new colormap named `{`⟨*name*⟩`}` according to `{`⟨*color specification*⟩`}` and activates it using `colormap name={`⟨*name*⟩`}`.

The syntax of `{`⟨*color specification*⟩`}` is the same as those for PGF shadings described in [3, VIII – Shadings]: it consists of a series of colors along with a length.

```
rgb(0cm)=(1,0,0); rgb(1cm)=(0,1,0); rgb255(2cm)=(0,0,255); gray(3cm)=(0.3);  color(4cm)=(green)
```



The single colors are separated by semicolons '`;`'. The length describes how much of the bar is occupied by the interval. Each entry has the form ⟨*color model*⟩`(`⟨*length*⟩`)=(`⟨*arguments*⟩`)`. The line above means that the left end of the colormap 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.



```
\begin{tikzpicture}
    \begin{axis}[
        colormap={bw}{gray(0cm)=(0); gray(1cm)=(1)}]
    \addplot+[scatter,only marks]
        plot[scatter src=y,domain=0:8,samples=100]
        {exp(x)};
    \end{axis}
\end{tikzpicture}
```

[17] Up to now, plot marks always have a stroke color (some also have a fill color). This restriction may be lifted in upcoming versions.

The complete length of a colormap 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`.

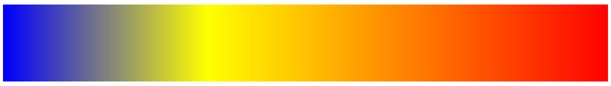
Available colormaps are shows below.

**Remark:** Currently, only equidistant `{`⟨*color specification*⟩`}`s are supported (each interval must have the same length).

`/pgfplots/colormap/hot` (style, no value)

A style which installs the colormap

`{color(0cm)=(blue); color(1cm)=(yellow); color(2cm)=(orange); color(3cm)=(red)}`

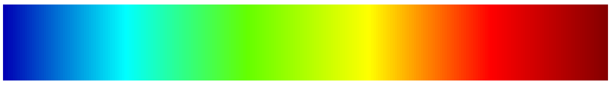


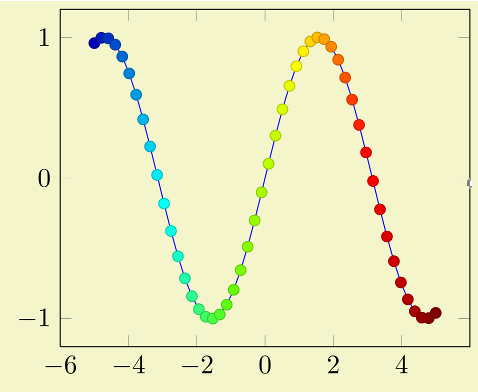
This is the preconfigured colormap.

`/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)}`,





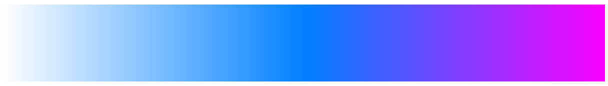
```
\begin{tikzpicture}
    \begin{axis}[colormap/bluered]
    \addplot+[scatter]
        expression[scatter src=x,samples=50]
        {sin(deg(x))};
    \end{axis}
\end{tikzpicture}
```

**Remark:** The style `bluered` (re-)defines the colormap and activates it. TEX will be slightly faster if you call `\pgfplotsset{colormap/bluered}` in the preamble (to create the colormap once) and use `colormap name=bluered` whenever you need it. This remark holds for every colormap style which follows. But you can simply ignore this remark.

`/pgfplots/colormap/cool` (style, no value)

A style which installs the colormap

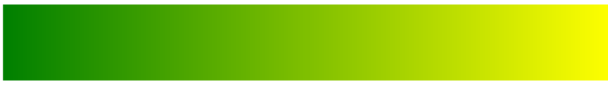
`{rgb255(0cm)=(255,255,255); rgb255(1cm)=(0,128,255); rgb255(2cm)=(255,0,255)}`



`/pgfplots/colormap/greenyellow` (style, no value)

A style which installs the colormap

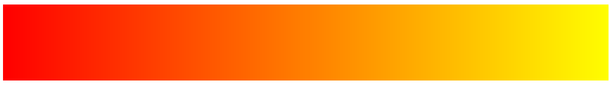
`{rgb255(0cm)=(0,128,0); rgb255(1cm)=(255,255,0)}`



`/pgfplots/colormap/redyellow` (style, no value)

A style which installs the colormap

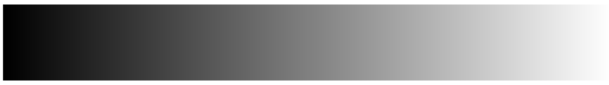
`{rgb255(0cm)=(255,0,0); rgb255(1cm)=(255,255,0)}`



`/pgfplots/colormap/blackwhite` (style, no value)

A style which installs the colormap

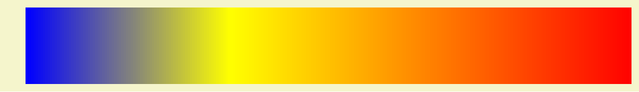
`{gray(0cm)=(0); gray(1cm)=(1)}`



`\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 [3] 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.

### 5.3.6 Options Controlling Linestyles

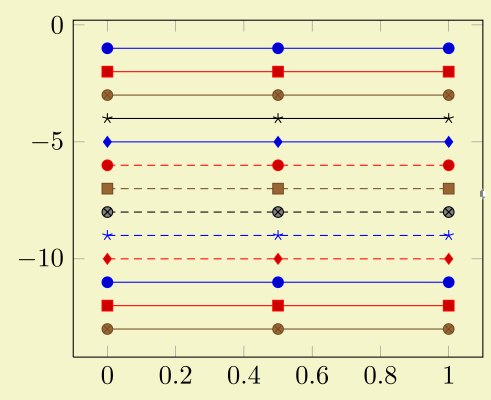
`/pgfplots/cycle list={`⟨*list*⟩`}`
`/pgfplots/cycle list name={`⟨*\macro*⟩`}`

Allows to specify a list of plot specifications which will be used for each `\addplot`-command without explicit plot specification.

There are several possiblities to change it:

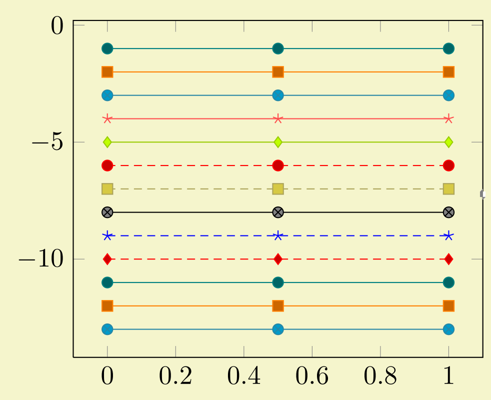
1. Use one of the predefined lists[18],
   - `color` (from top to bottom)

[18] These lists were named `\coloredplotspeclist` and `\blackwhiteplotspeclist` which appeared to be unnecessarily long, so they have been renamed. The old names are still accepted, however.



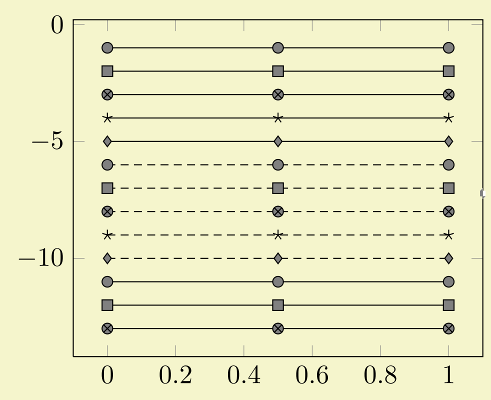
```
\begin{tikzpicture}
\begin{axis}[
    stack plots=y,stack dir=minus,
    cycle list name=color]
\addplot coordinates {(0,1) (0.5,1) (1,1)};
\addplot coordinates {(0,1) (0.5,1) (1,1)};
\addplot coordinates {(0,1) (0.5,1) (1,1)};
\addplot coordinates {(0,1) (0.5,1) (1,1)};
\addplot coordinates {(0,1) (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)



```
\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)

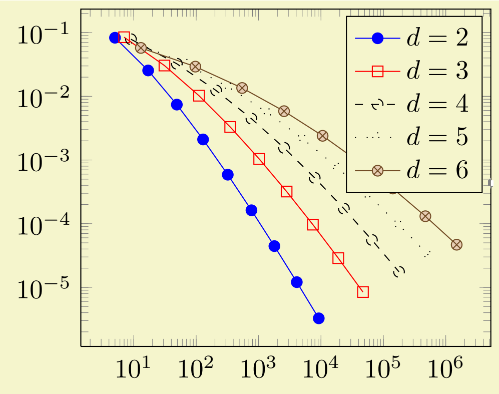


```
\begin{tikzpicture}
\begin{axis}[
    stack plots=y,stack dir=minus,
    cycle list name=black white]
\addplot coordinates {(0,1) (0.5,1) (1,1)};
\addplot coordinates {(0,1) (0.5,1) (1,1)};
\addplot coordinates {(0,1) (0.5,1) (1,1)};
\addplot coordinates {(0,1) (0.5,1) (1,1)};
\addplot coordinates {(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}
```

These predefined cycle lists habe been created with

```
\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*\\%
}
```

2. Provide the list explicitly,



```
\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}`).

3. Define macro names and use them with ‘`cycle list name`’:

```
\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}
```

**Remark:** You can also terminate single entries with '\\' as in

```
\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.

## 5.4 Axis Descriptions

Axis descriptions are labels for $x$ and $y$ axis and titles. Axis descriptions are drawn after the plot is finished and they are not subjected to clipping. Their placement is always *relative to the axis rectangle*, where $(0,0)$ refers to the lower left corner and $(1,1)$ refers to the upper right one.

Furthermore, axis descriptions can be placed using the predefined node `current axis`. 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 references using `current axis.`⟨*anchor name*⟩. Please see section 5.11, Alignment, for further details.

### 5.4.1 Labels

`/pgfplots/xlabel={`⟨*text*⟩`}`
`/pgfplots/ylabel={`⟨*text*⟩`}`

The options `xlabel` and `ylabel` change axis labels to `{`⟨*text*⟩`}` which is any TEX text. Use "`xlabel={, = characters}`" if characters like '`=`' or '`,`' need to be included literally.

Labels are TikZ-Nodes which are placed with

```
\node
    [style=every axis label,
    style=every axis x label]
\node
    [style=every axis label,
    style=every axis y label]
```

so their position and appearance can be customized. The coordinate `(0,0)` denotes the lower left axis corner and `(1,1)` the upper right.

The default styles are

```
\pgfplotsset{every axis label/.style={}}
\pgfplotsset{every axis x label/.style={
    at={(0.5,0)},
    below,
    yshift=-15pt}}
\pgfplotsset{every axis y label/.style={
    at={(0,0.5)},
    xshift=-35pt,
    rotate=90}}
```

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={...}}
```

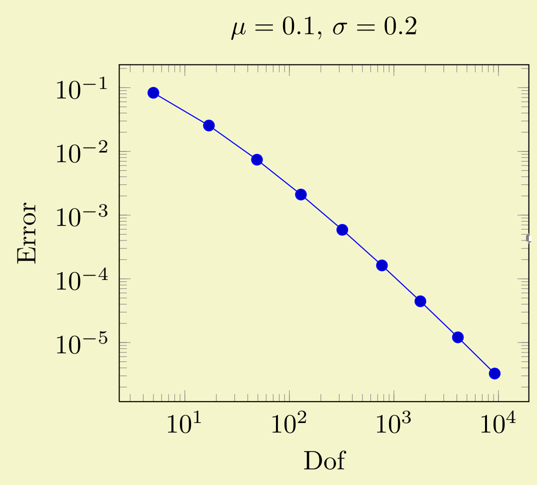
Use `xlabel/.add={`⟨*prefix*⟩`}{`⟨*suffix*⟩`}` to modify an already assigned label.

/pgfplots/title={⟨*text*⟩}

Adds a caption to the plot. This will place a Ti*k*Z-Node with

```
\node[style=every axis title] {text};
```

to the current axis.



```
\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{every axis title/.append style={at={(0.75,1)}}}
```

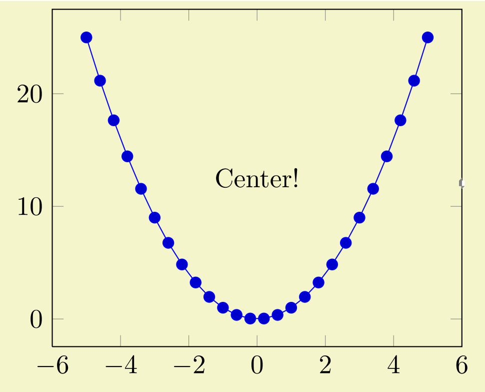
This will place the title at 75% of the $x$-axis. The coordinate $(0, 0)$ is the lower left corner and $(1, 1)$ the upper right one.

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.



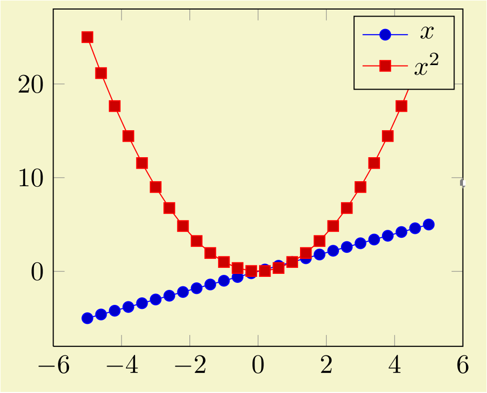
```
\pgfplotsset{every axis/.append style={
    extra description/.code={
        \node at (0.5,0.5) {Center!};
    }}}
\begin{tikzpicture}
    \begin{axis}
    \addplot {x^2};
    \end{axis}
\end{tikzpicture}
```

### 5.4.2 Legend

Legends can be generated in two ways: the first is to use `\addlegendentry` or `\legend` inside of an axis. This method has been presented in section 4.4, Legend Commands. The other method is to use a key.

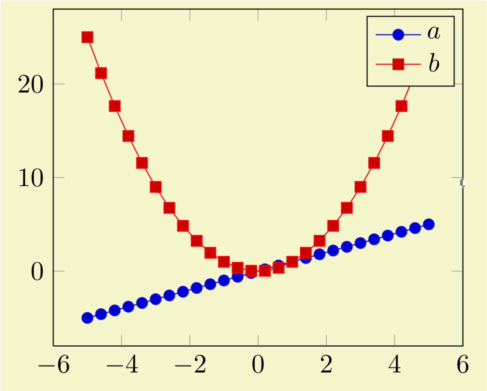
/pgfplots/legend entries={⟨*comma separated list*⟩}

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).



```
\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 is only considered if there is no legend command in the current axis. Please refer to section 4.4, Legend Commands, for details about the commands.



```
\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).

`/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}}
```

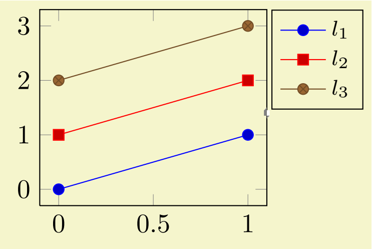
will draw it at the lower left corner of the axis while

```
\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 $(1,1)$.

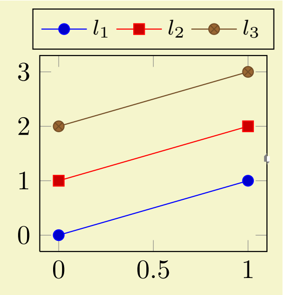
The legend is a Ti*k*Z-matrix, so one can use any Ti*k*Z option which affects nodes and matrizes (see [3, section 13 and 14]). The matrix is created by something like

```
\matrix[style=every axis legend] {
    draw plot specification 1 & \node{legend 1}\\
    draw plot specification 2 & \node{legend 2}\\
    ...
};
```



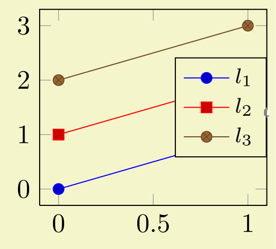
```
\pgfplotsset{every axis legend/.append style={
        at={(1.02,1)},
        anchor=north west}}
\begin{tikzpicture}
\begin{axis}
\addplot coordinates {(0,0) (1,1)};
\addplot coordinates {(0,1) (1,2)};
\addplot coordinates {(0,2) (1,3)};
\legend{$l_1$,$l_2$,$l_3$}
\end{axis}
\end{tikzpicture}
```

Use `legend columns={⟨number⟩}` to configure the number of horizontal legend entries.



```
\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.



```
\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 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/legend image code/.code={⟨...⟩}

Allows to replace the default images which are drawn inside of legends. The first argument to this option is the plot specification, a key-value list which has been determined by \addplot.

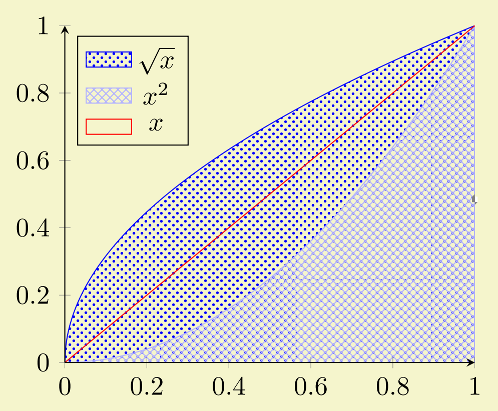
The default is

```
/pgfplots/legend image code/.code={%
    \draw[#1,mark repeat=2,mark phase=2]
        plot coordinates {
            (0cm,0cm)
            (0.3cm,0cm)
            (0.6cm,0cm)%
        };%
}
```

/pgfplots/area legend (style, no value)

A style which sets legend image code to

```
\pgfplotsset{
    /pgfplots/legend image code/.code={%
        \draw[#1] (0cm,-0.1cm) rectangle (0.6cm,0.1cm);
    }}
```



```
% \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]
expression[samples=500]
    {sqrt(x)}    \closedcycle;

\addplot[pattern=crosshatch,
    pattern color=blue!30!white,
    draw=blue!30!white]
expression {x^2} \closedcycle;

\addplot[red] coordinates {(0,0) (1,1)};
\legend{$\sqrt x$,$x^2$,$x$}
\end{axis}
\end{tikzpicture}
```

### 5.4.3 Axis Lines

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)

Allows to choose the location of the axis line(s). Ticks and tick labels are placed accordingly. 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.

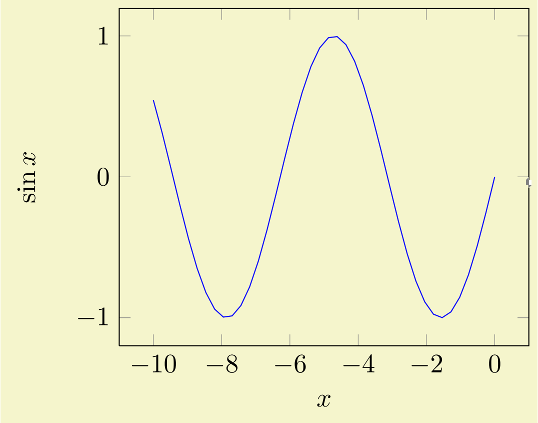
The starred versions ...line* *only* affect the axis lines, without correcting the positions of axis labels, tick lines or other keys which are (possibly) affected by a changed axis line. The non-starred versions are actually styles which set the starred key *and* some other keys which also affect the 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.

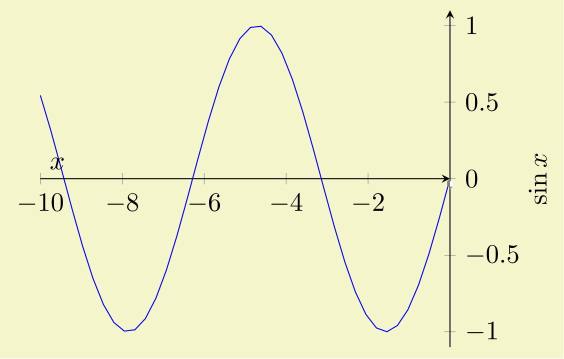
The same holds true for the `y`-variants. The default styles are defined as

```
\pgfplotsset{
    /pgfplots/every non boxed x axis/.style={
        xtick align=center,
        enlarge x limits=false,
        x axis line style={-stealth}
    },
    /pgfplots/every boxed x axis/.style={}
}
```

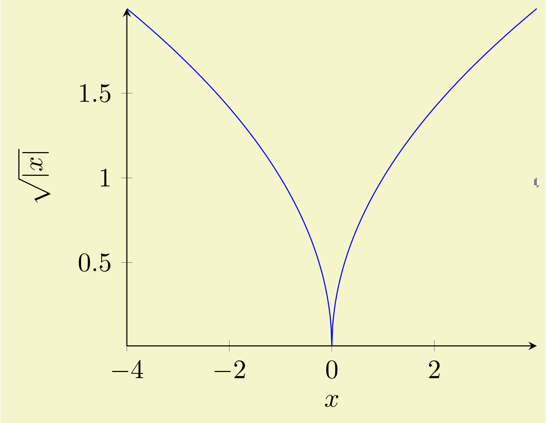
Feel free to overwrite these styles if the default doesn't fit your needs or taste.



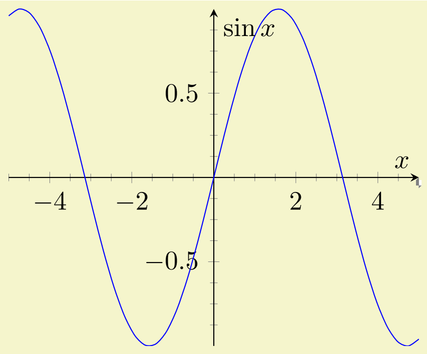
```
\begin{tikzpicture}
\begin{axis}[
    xlabel=$x$,ylabel=$\sin x$]

    \addplot[blue,mark=none]
        expression[domain=-10:0,samples=40]
        {sin(deg(x))};
\end{axis}
\end{tikzpicture}
```



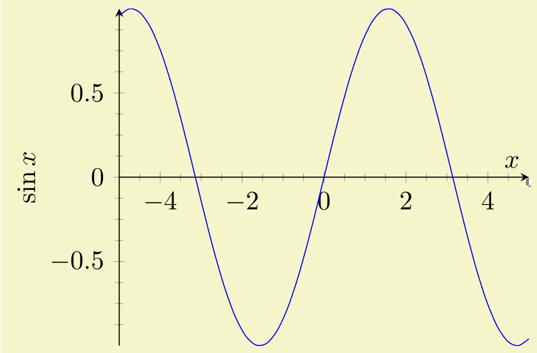
```
\begin{tikzpicture}
\begin{axis}[
    axis x line=middle,
    axis y line=right,
    ymax=1.1, ymin=-1.1,
    xlabel=$x$,ylabel=$\sin x$
]
    \addplot[blue,mark=none]
        expression[domain=-10:0,samples=40]
        {sin(deg(x))};
\end{axis}
\end{tikzpicture}
```



```
\begin{tikzpicture}
\begin{axis}[
    axis x line=bottom,
    axis y line=left,
    xlabel=$x$,ylabel=$\sqrt{|x|}$
]
\addplot[blue,mark=none]
    expression[domain=-4:4,samples=501]
    {sqrt(abs(x))};
\end{axis}
\end{tikzpicture}
```



```
\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}
```



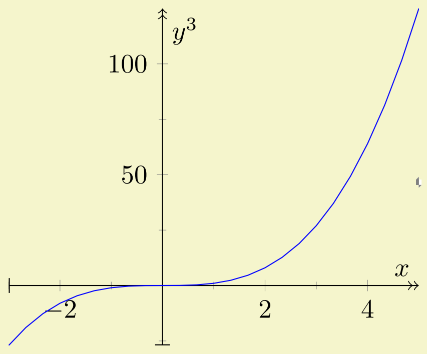
```
\begin{tikzpicture}
\begin{axis}[
    minor tick num=3,
    axis y line=left,
    axis x line=middle,
    xlabel=$x$,ylabel=$\sin x$
    ]
    \addplot[smooth,blue,mark=none]
        [domain=-5:5,samples=40]
        {sin(deg(x))};
\end{axis}
\end{tikzpicture}
```

In case `middle`, the style `every inner axis x line` allows to adjust the appearenace.

`/pgfplots/every inner x axis line` (no value)
`/pgfplots/every inner y 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.

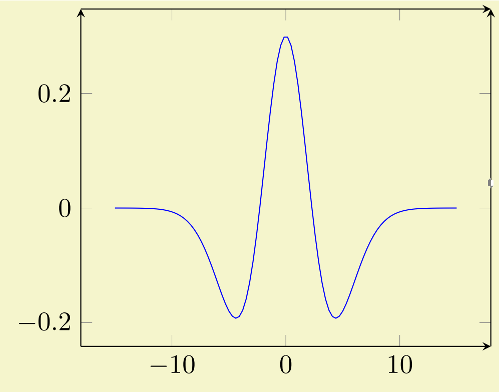
This style affects *only* the line as such.



```
\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] expression[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)

Similar to `every inner x axis line`, this style configures the appearance of all axis lines which are part of the outer box.



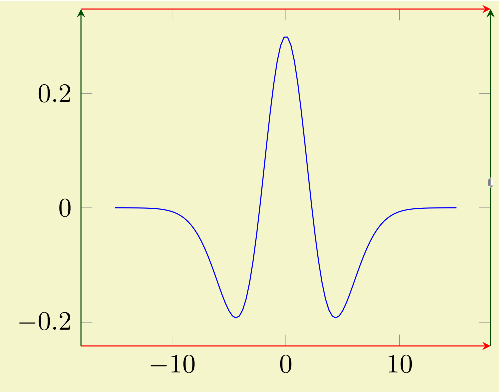
```
\begin{tikzpicture}
\begin{axis}[
    separate axis lines, % important !
    every outer x axis line/.append style=
        {-stealth},
    every outer y axis line/.append style=
        {-stealth},
]
\addplot[blue] plot[id=DoG,
        samples=100,
        domain=-15:15]
  gnuplot{1.3*exp(-x**2/10) - 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*.

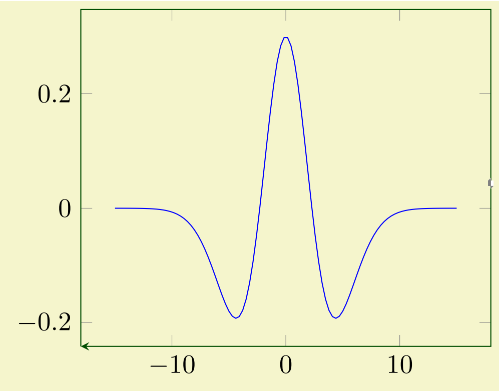
Both cases have their advantages and disadvantages, I fear there is no reasonable default (suggestions are welcome).

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.



```
\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]
    expression[
        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).



```
\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] plot[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*⟩}

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)

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)

A style which will be installed as soon as axis x line (y) will be set to something different than box.

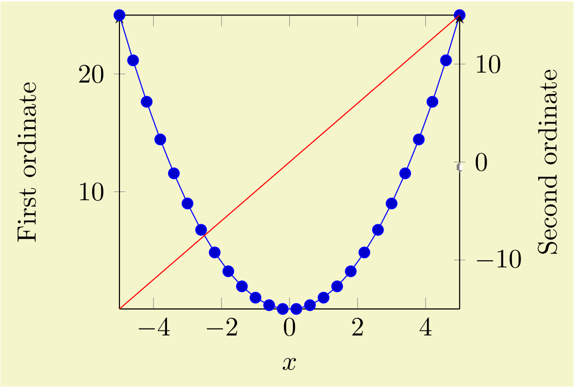
The default is

```
\pgfplotsset{
    /pgfplots/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.

### 5.4.4 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.



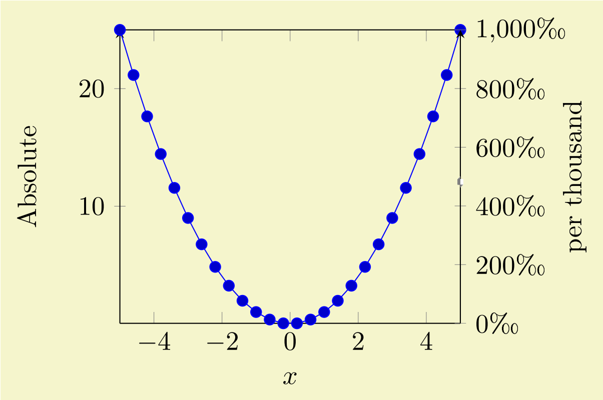
```
\begin{tikzpicture}
  \begin{axis}[
    scale only axis,
    xmin=-5,xmax=5,
    axis y line=left,
    xlabel=$x$,
    ylabel=First ordinate]
  \addplot {x^2};
  \end{axis}

  \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:



```
% \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}
```

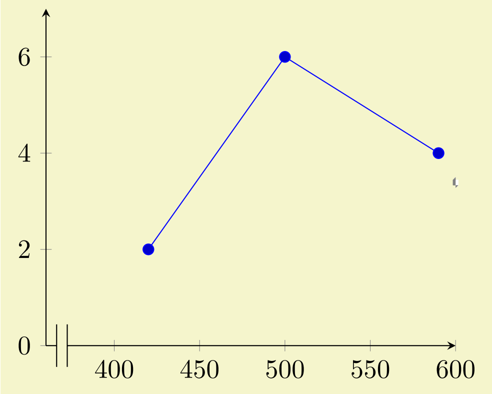
### 5.4.5 Axis Discontinuities

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.

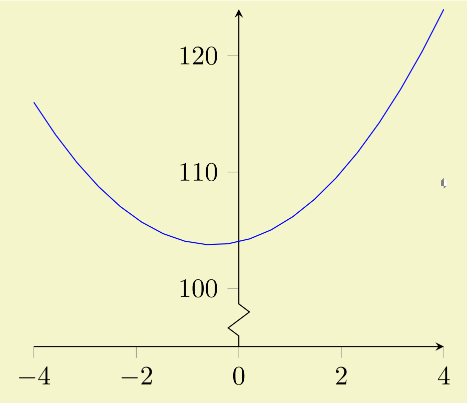
`/pgfplots/axis x discontinuity=crunch|parallel|none` (initially `none`)
`/pgfplots/axis y discontinuity=crunch|parallel|none` (initially `none`)

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 $y$.



```
\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}
```

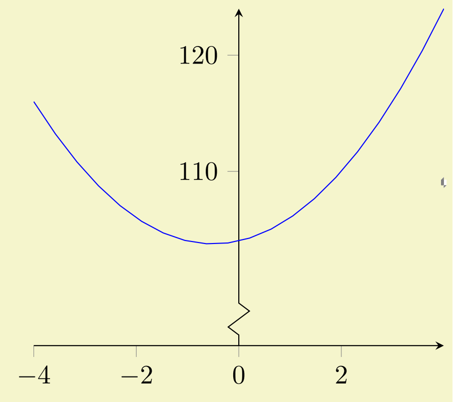


```
\begin{tikzpicture}
\begin{axis}[
    axis x line=bottom,
    axis y line=center,
    tick align=outside,
    axis y discontinuity=crunch,
    ymin=95, enlargelimits=false
]
    \addplot[blue,mark=none]
        expression[domain=-4:4,samples=20]
        {x*x+x+104};
\end{axis}
\end{tikzpicture}
```

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/xtickmax={⟨coord⟩}` (default `axis limits`)
`/pgfplots/ytickmax={⟨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.

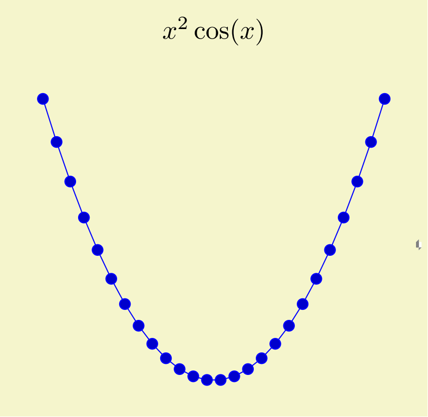


```
\begin{tikzpicture}
\begin{axis}[
    axis x line=bottom,
    axis y line=center,
    tick align=outside,
    axis y discontinuity=crunch,
    xtickmax=3,
    ytickmin=110,
    ymin=95, enlargelimits=false
]
    \addplot[blue,mark=none]
        plot[domain=-4:4,samples=20]
        expression{x*x+x+104};
\end{axis}
\end{tikzpicture}
```

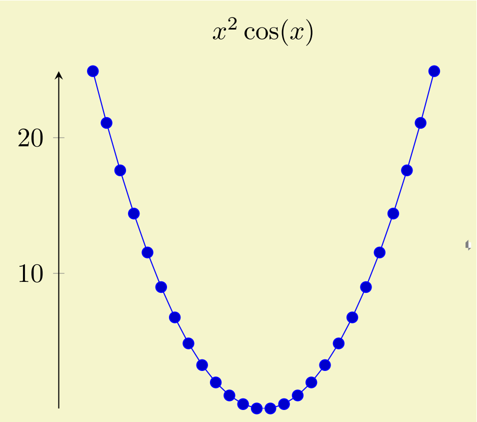
`/pgfplots/hide x axis=true|false` (initially `false`)
`/pgfplots/hide y axis=true|false` (initially `false`)

Allows to hide either the $x$ or the $y$ axis. 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 x axis`.



```
\begin{tikzpicture}
    \begin{axis}[
        hide x axis,
        hide y axis,
        title={$x^2\cos(x)$}]
    \addplot {cos(x)*x^2};
    \end{axis}
\end{tikzpicture}
```



```
\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}
```

/pgfplots/hide axis=true|false (style, default true)

A style which sets both, hide x axis and hide y axis.

### 5.4.6 Adjusting Descriptions for Different Scales

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*⟩}
/tikz/mark size={⟨*integer*⟩}

These keys should be adjusted to the figure's dimensions. Use

```
\pgfplotset{tick label style={font=\footnotesize},
    label style={font=\small},
    legend style={font=\small}
}
```

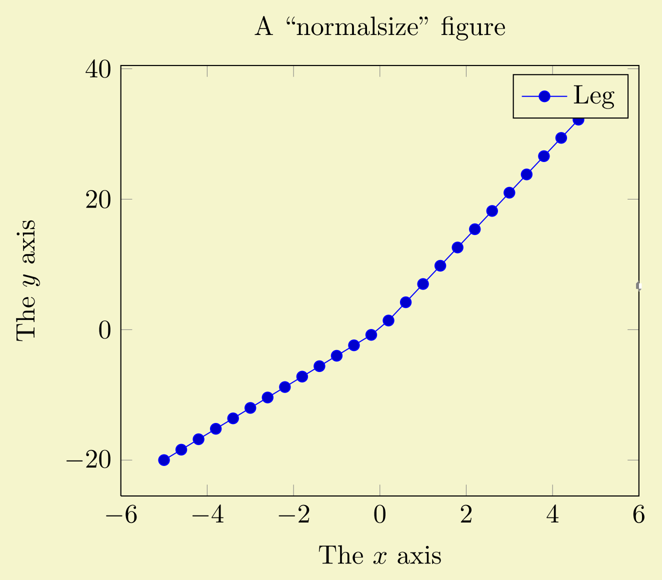
to provide different fonts for different descriptions.

The max space between ticks is described on page 88 and configures the approximate distance between 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.



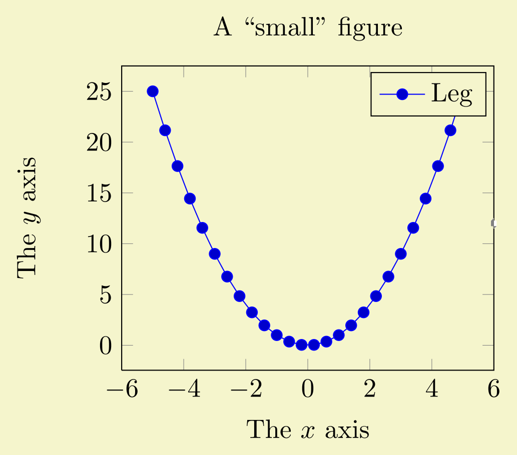
```
\begin{tikzpicture}
    \begin{axis}[normalsize,
        title=A ‘‘normalsize’’ figure,
        xlabel=The $x$ axis,
        ylabel=The $y$ axis,
        legend entries={Leg}]
        \addplot {max(4*x,7*x)};
    \end{axis}
\end{tikzpicture}
```

The initial setting is

```
/pgfplots/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”.



```
\begin{tikzpicture}
    \begin{axis}[small,
        title=A ``small'' figure,
        xlabel=The $x$ axis,
        ylabel=The $y$ axis,
        legend entries={Leg}]
        \addplot {x^2};
    \end{axis}
\end{tikzpicture}
```

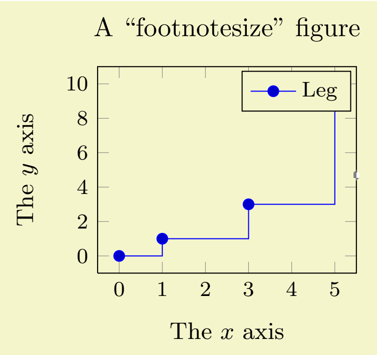
The initial setting is

```
/pgfplots/small/.style={
    /pgfplots/width=6.5cm,
    /pgfplots/height=,
    /pgfplots/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`.



```
\begin{tikzpicture}
    \begin{axis}[footnotesize,
        title=A ``footnotesize'' figure,
        xlabel=The $x$ axis,
        ylabel=The $y$ axis,
        legend entries={Leg}]
        \addplot+[const plot]
            coordinates {
            (0,0) (1,1) (3,3) (5,10)
        };
    \end{axis}
\end{tikzpicture}
```

The initial setting is

```
/pgfplots/footnotesize/.style={
    /pgfplots/width=5cm,
    /pgfplots/height=,
    legend style={font=\footnotesize},
    tick label style={font=\footnotesize},
    label style={font=\small},
    /pgfplots/max space between ticks=20,
    every mark/.append style={mark size=8},
    ylabel style={yshift=-0.3cm}
},
```

As for `small`, it can be convenient to set `footnotesize` and set `width` afterwards.

## 5.5 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[19]. 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

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/x=`{(⟨*x*⟩,⟨*y*⟩)}
`/pgfplots/y=`{(⟨*x*⟩,⟨*y*⟩)}

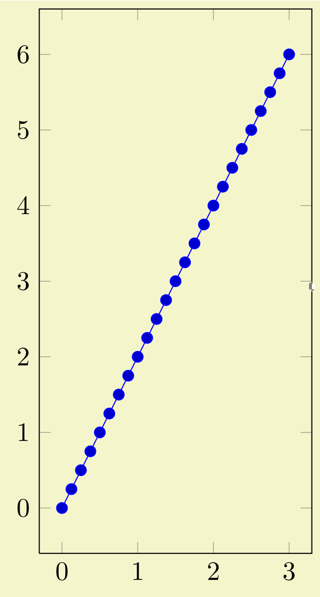
Sets the unit vectors for $x$ (or $y$). Every logical plot coordinate $(x, y)$ is drawn at the position

$$x \cdot \begin{bmatrix} e_{xx} \\ e_{xy} \end{bmatrix} + y \cdot \begin{bmatrix} e_{yx} \\ e_{yy} \end{bmatrix}.$$

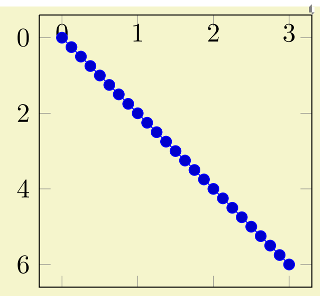
The unit vectors $e_x$ and $e_y$ determine the paper position in the current (always two dimensional) image.

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*⟩}.

[19] I.e. the '`width`' option will not work as expected, but the bounding box is still ok.

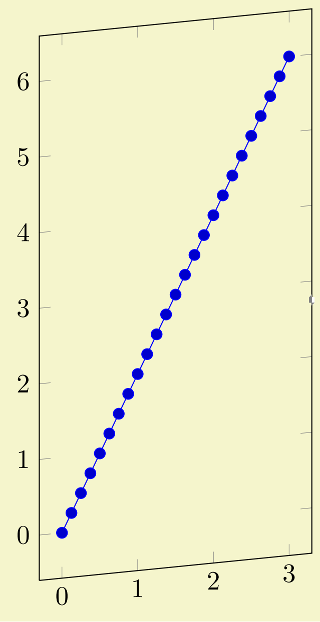


```
\begin{tikzpicture}
\begin{axis}[x=1cm,y=1cm]
\addplot expression[domain=0:3] {2*x};
\end{axis}
\end{tikzpicture}
```



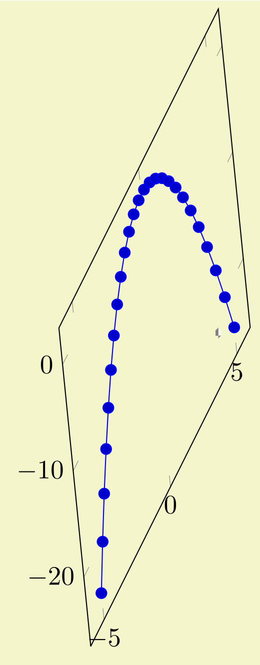
```
\begin{tikzpicture}
\begin{axis}[x=1cm,y=-0.5cm]
\addplot expression[domain=0:3] {2*x};
\end{axis}
\end{tikzpicture}
```

The second syntax, `x={(`$\langle x\rangle$`,`$\langle y\rangle$`)}` sets $e_x = (\langle x\rangle, \langle y\rangle)^T$ explicitly[20]; the corresponding `y` key works similiarly. This allows to define skewed or rotated axes.



```
\begin{tikzpicture}
\begin{axis}[x={(1cm,0.1cm)},y=1cm]
\addplot expression[domain=0:3] {2*x};
\end{axis}
\end{tikzpicture}
```

[20] 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.



```
\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}
```

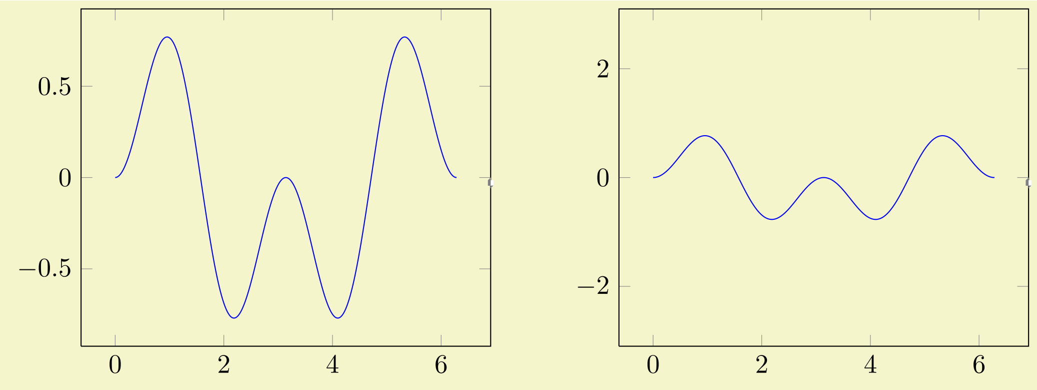
won't have any effect because an axis rescales its coordinates (see the `width` option).

**Limitations:** Unfortunately, skewed axes are **not available for bar plots**.

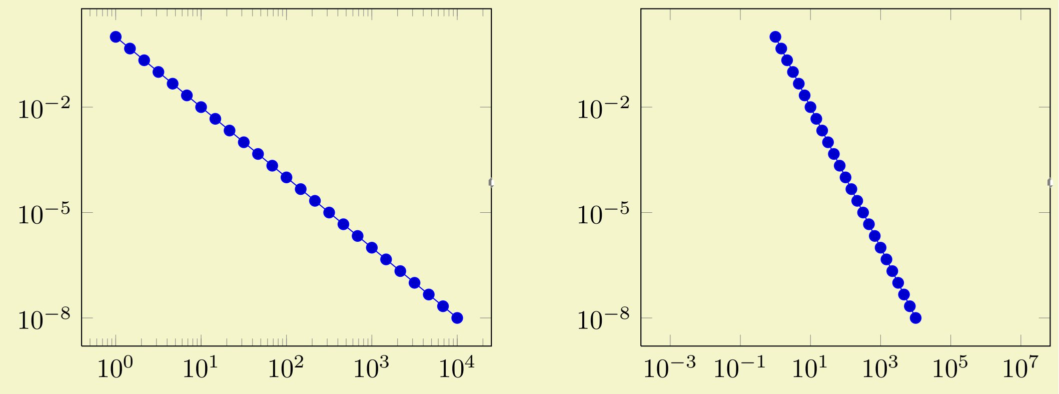
`/pgfplots/axis equal`=`{`⟨*true,false*⟩`}` (initially `false`)

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.



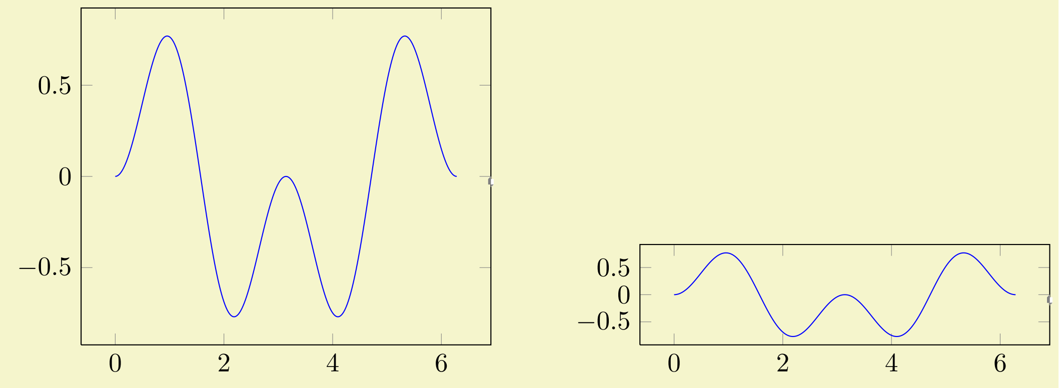
```
\begin{tikzpicture}
    \begin{axis}[axis equal=false]
        \addplot[blue] expression[domain=0:2*pi,samples=300] {sin(deg(x))*sin(2*deg(x))};
    \end{axis}
\end{tikzpicture}
\hspace{1cm}
\begin{tikzpicture}
    \begin{axis}[axis equal=true]
        \addplot[blue] expression[domain=0:2*pi,samples=300] {sin(deg(x))*sin(2*deg(x))};
    \end{axis}
\end{tikzpicture}
```



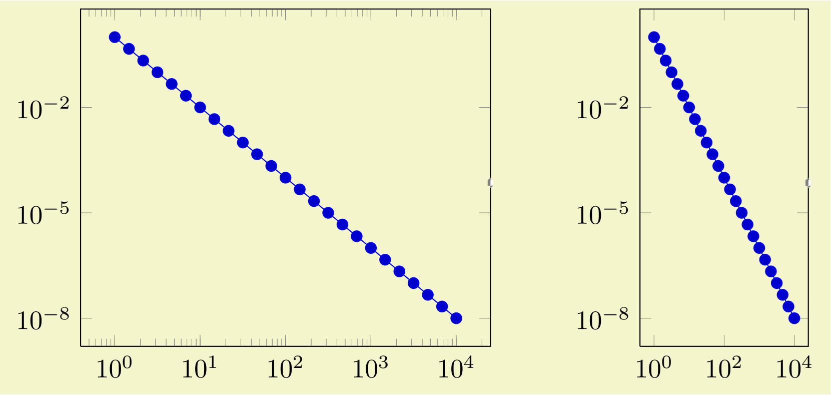
```
\begin{tikzpicture}
    \begin{loglogaxis}[axis equal=false]
        \addplot expression[domain=1:10000] {x^-2};
    \end{loglogaxis}
\end{tikzpicture}
\hspace{1cm}
\begin{tikzpicture}
    \begin{loglogaxis}[axis equal=true]
        \addplot expression[domain=1:10000] {x^-2};
    \end{loglogaxis}
\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).



```
\begin{tikzpicture}
    \begin{axis}[axis equal image=false]
        \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]
        \addplot[blue] expression[domain=0:2*pi,samples=300] {sin(deg(x))*sin(2*deg(x))};
    \end{axis}
\end{tikzpicture}
```



```
\begin{tikzpicture}
    \begin{loglogaxis}[axis equal image=false]
        \addplot expression[domain=1:10000] {x^-2};
    \end{loglogaxis}
\end{tikzpicture}
\hspace{1cm}
\begin{tikzpicture}
    \begin{loglogaxis}[axis equal image=true]
        \addplot expression[domain=1:10000] {x^-2};
    \end{loglogaxis}
\end{tikzpicture}
```

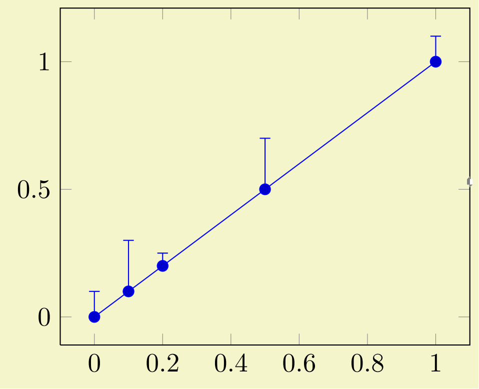
## 5.6 Error Bars

PGFPLOTS supports error bars for normal and logarithmic plots.

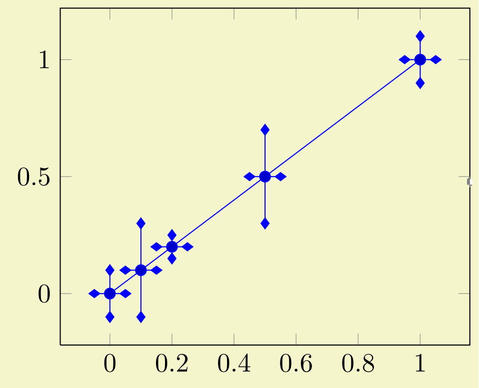
Error bars are enabled for each plot separately, using ⟨*behavior options*⟩ after `\addplot`:

```
\addplot plot[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.

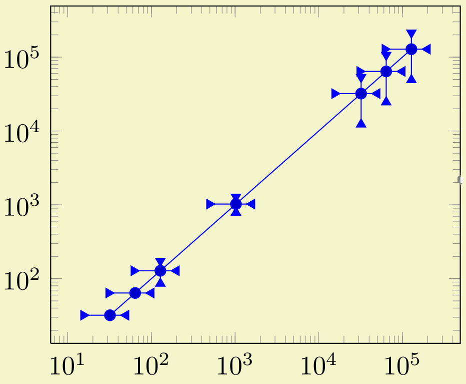


```
\begin{tikzpicture}
\begin{axis}
\addplot plot[error bars/.cd,
    y dir=plus,y explicit]
coordinates {
    (0,0)      +- (0.5,0.1)
    (0.1,0.1) +- (0.05,0.2)
    (0.2,0.2) +- (0,0.05)
    (0.5,0.5) +- (0.1,0.2)
    (1,1)      +- (0.3,0.1)};
\end{axis}
\end{tikzpicture}
```

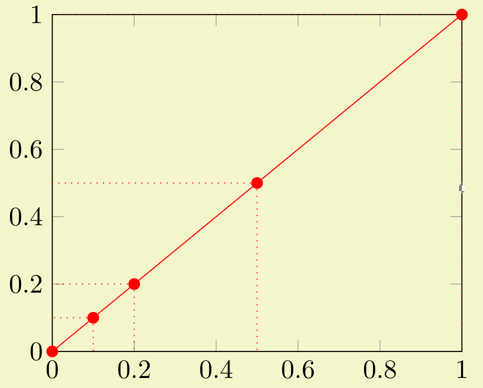


```
\begin{tikzpicture}
\begin{axis}
\addplot plot[error bars/.cd,
    y dir=both,y explicit,
    x dir=both,x fixed=0.05,
    error mark=diamond*]
coordinates {
    (0,0)      +- (0.5,0.1)
    (0.1,0.1) +- (0.05,0.2)
    (0.2,0.2) +- (0,0.05)
    (0.5,0.5) +- (0.1,0.2)
    (1,1)      +- (0.3,0.1)};
\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 |



```
\pgfplotstabletypeset{pgfplots.testtable2.dat}

\begin{tikzpicture}
\begin{loglogaxis}
\addplot plot[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}
```



```
\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`)

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 $\epsilon_x$ 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`)

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`)

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)

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 5.6.1.

`/pgfplots/error bars/x explicit relative` (no value)
`/pgfplots/error bars/y 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 5.3.

`/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

```
/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}}%
    };
}
```

### 5.6.1 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.

## 5.7 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*⟩`]{`⟨$x$⟩`}`.

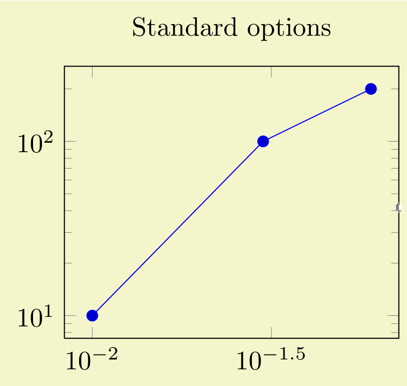
Please refer to the manual of PGFPLOTSTABLE (shipped with this package) for details about the number options.

`/pgfplots/log identify minor tick positions=true|false` (initially `false`)

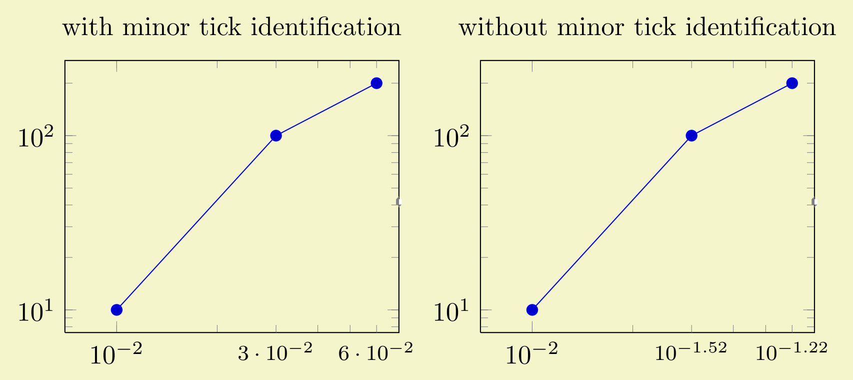
Set this to `true` if you want to identify log–plot tick labels at positions

$$i \cdot 10^j$$

with $i \in \{2, 3, 4, 5, 6, 7, 8, 9\}$, $j \in \mathbb{Z}$. This may be valueable in conjunction with the ‘`extra x ticks`’ and ‘`extra y ticks`’ options.



```
\begin{tikzpicture}%
\begin{loglogaxis}
    [title=Standard options,
    width=6cm]
\addplot coordinates {
    (1e-2,10)
    (3e-2,100)
    (6e-2,200)
};
\end{loglogaxis}
\end{tikzpicture}%
```



```
\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.

`/pgfplots/log base 10 number format code/.code={⟨...⟩}`

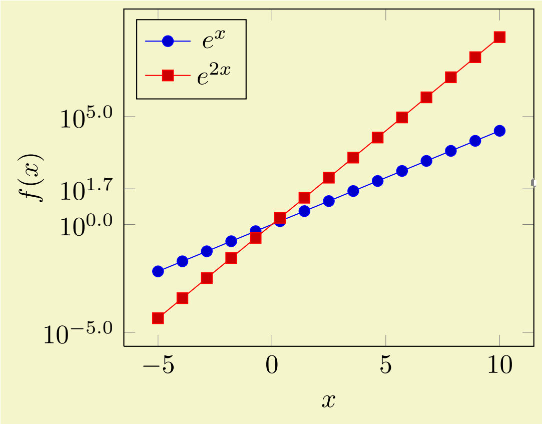
Allows to change the overall appearance of base 10 log plot tick labels. The default is

```
log base 10 number format code/.code={%
    $10^{\pgfmathprintnumber{#1}}$}
```

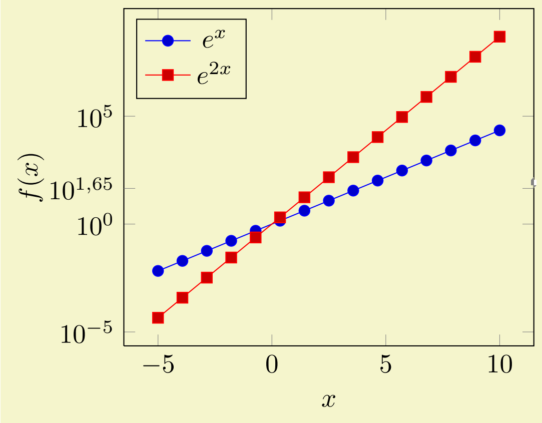
where the '`log plot exponent style`' allows to change number formatting options.

`/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`'.



```
\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}
```



```
\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}
```

## 5.8 Specifying the Plotted Range

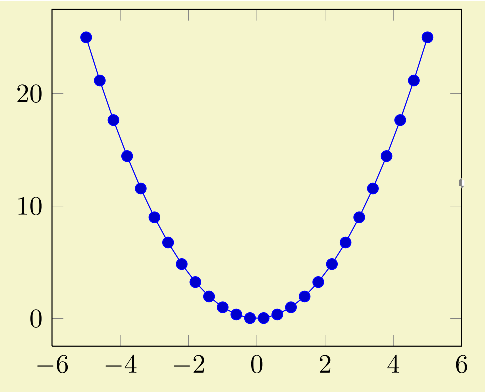
`/pgfplots/xmin={⟨coord⟩}`
`/pgfplots/ymin={⟨coord⟩}`
`/pgfplots/xmax={⟨coord⟩}`
`/pgfplots/ymax={⟨coord⟩}`

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.

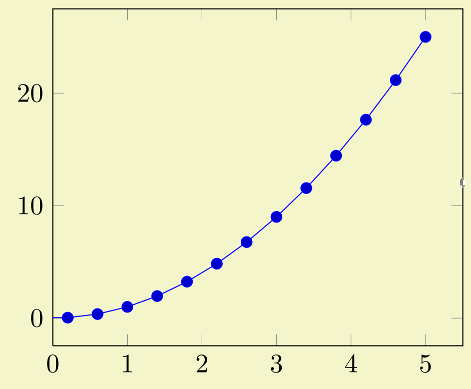
Each missing limit will be determined automatically.

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`.

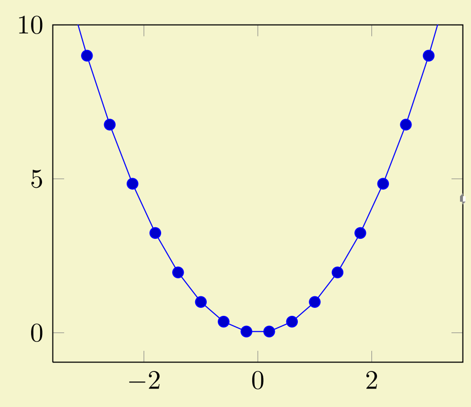
Axis limits can be increased automatically using the `enlargelimits` option.



```
\begin{tikzpicture}
    \begin{axis}
    \addplot {x^2};
    \end{axis}
\end{tikzpicture}
```



```
\begin{tikzpicture}
    \begin{axis}[xmin=0]
    \addplot {x^2};
    \end{axis}
\end{tikzpicture}
```



```
\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)

Allows to choose between linear (=normal) or logarithmic axis scaling or logplots for each $x, y$-combination.

/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=true|false|auto|{⟨*val*⟩} (initially auto)
/pgfplots/enlarge y limits=true|false|auto|{⟨*val*⟩} (initially auto)

Enlarges the axis size for one axis 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 all axes.
- 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.
- All other values like '`enlarge x limits=0.1`' will enlarge all axis limits relatively (in this example, 10% of the axis limits will be added at all sides).

A small value of `enlarge x limits` may avoid problems with large markers near the boundary.

`/pgfplots/enlargelimits=true|false|auto|{⟨val⟩}` (style, default `true`)

A style which sets `enlarge x limits` and `enlarge y limits` to the specified value.

`\begin{pgfplotsinterruptdatabb}`
⟨*environment contents*⟩
`\end{pgfplotsinterruptdatabb}`

Everything in `{`⟨*environment contents*⟩`}` will not contribute to the data bounding box.

## 5.9 Tick and Grid Options

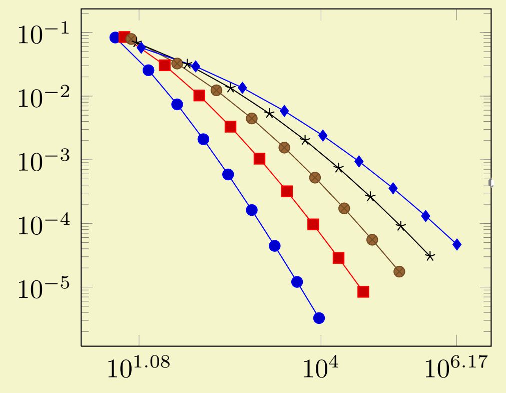
`/pgfplots/xtick=\empty|data|{`⟨*coordinate list*⟩`}` (initially `{`⟨⟩`}`)
`/pgfplots/ytick=\empty|data|{`⟨*coordinate list*⟩`}` (initially `{`⟨⟩`}`)

The options `xtick` and `ytick` assigns a list of *Positions* where ticks shall be placed. The argument is either the command `\empty`, `data` or a list of coordinates. 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*⟩`}`. If this list is empty, PGFPLOTS will compute a default list.

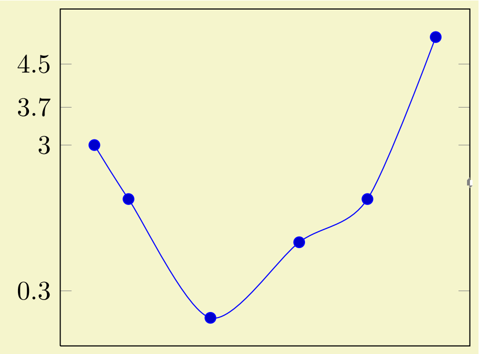
`{`⟨*coordinate list*⟩`}` will be used inside of a `\foreach \x in {`⟨*coordinate list*⟩`}` statement. The format is as follows:

- `{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 [3, Section 34] for a more detailed definition of the options.
- Please be careful with whitespaces inside of `{`⟨*coordinate list*⟩`}` (at least around the dots).

For logplots, PGFPLOTS will apply log(·) to each element in '`{`⟨*coordinate list*⟩`}`'.



```
\begin{tikzpicture}
    \begin{loglogaxis}[xtick={12,9897,1468864}]
    % see above for this macro:
    \plotcoords
    \end{loglogaxis}
\end{tikzpicture}
```



```
\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 "feasable" 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. Which positions depends 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 "feasable" step sizes $h$ as in the case of linear axis.

The default tick positions can be reconfigured with

- '`max space between ticks={`⟨*number*⟩`}`' where the integer argument denotes 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.
- '`try min ticks={`⟨*number*⟩`}`' 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.
- '`try min ticks log={`⟨*number*⟩`}`' The same for logarithmic axis.

The total number of ticks may still vary because not all fractional numbers in the axis' range are valid tick positions.

The tick *appearance* can be (re-)configured with

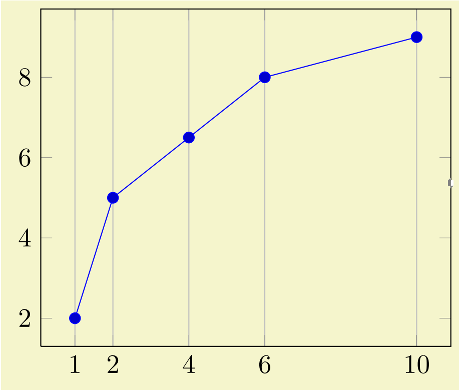
```
\pgfplotsset{every tick/.style={very thin,gray}}
\pgfplotsset{every minor tick/.style={}}
```

or

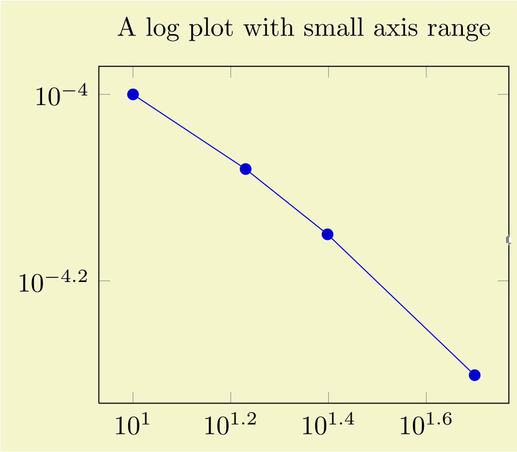
```
\pgfplotsset{every tick/.append style={very thin,gray}}
\pgfplotsset{every minor tick/.append style={black}}
```

Please prefer the '`.append style`' versions whenever possible to ensure compatibility with future versions.

These style commands can be used at any time. The tick line width can be configured with '`major tick length`' and '`minor tick length`'.



```
\begin{tikzpicture}
\begin{axis}[xtick=data,xmajorgrids]
    \addplot coordinates {
        (1,2)
        (2,5)
        (4,6.5)
        (6,8)
        (10,9)
    };
\end{axis}
\end{tikzpicture}
```

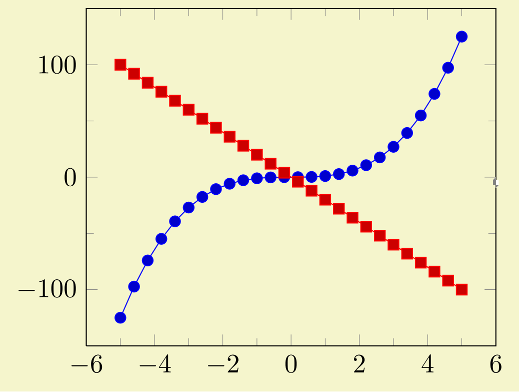


```
\begin{tikzpicture}
\begin{loglogaxis}[
    title=A log plot with small axis range]

    \addplot coordinates {
        (10,1e-4)
        (17,8.3176e-05)
        (25,7.0794e-05)
        (50,5e-5)
    };
\end{loglogaxis}
\end{tikzpicture}
```

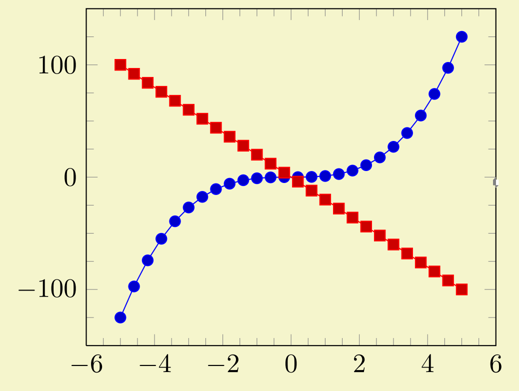
/pgfplots/minor tick num={⟨*number*⟩}

Sets both, `minor x tick num` and `minor y tick num` to {⟨*number*⟩}.

Minor ticks will be disabled if the major ticks don't have the same distance.



```
\begin{tikzpicture}
    \begin{axis}[minor tick num=1]
    \addplot {x^3};
    \addplot {-20*x};
    \end{axis}
\end{tikzpicture}
```

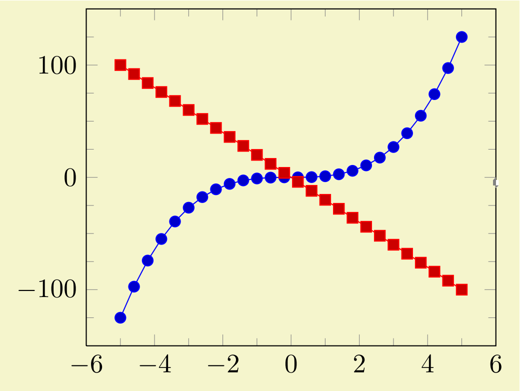


```
\begin{tikzpicture}
    \begin{axis}[minor tick num=3]
    \addplot {x^3};
    \addplot {-20*x};
    \end{axis}
\end{tikzpicture}
```

/pgfplots/minor x tick num={⟨*number*⟩} (initially 0)
/pgfplots/minor y tick num={⟨*number*⟩} (initially 0)

Sets the number of minor tick lines used for linear $x$ or $y$ axis separately.

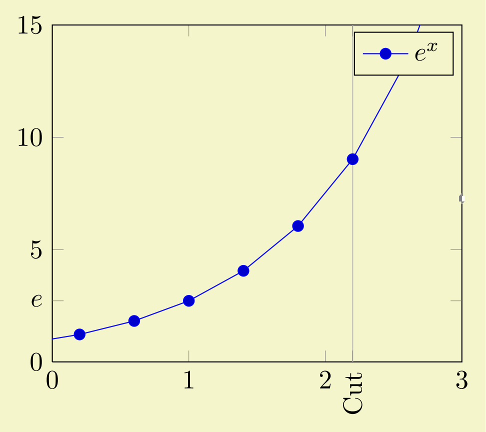
Minor ticks will be disabled if the major ticks don't have the same distance.



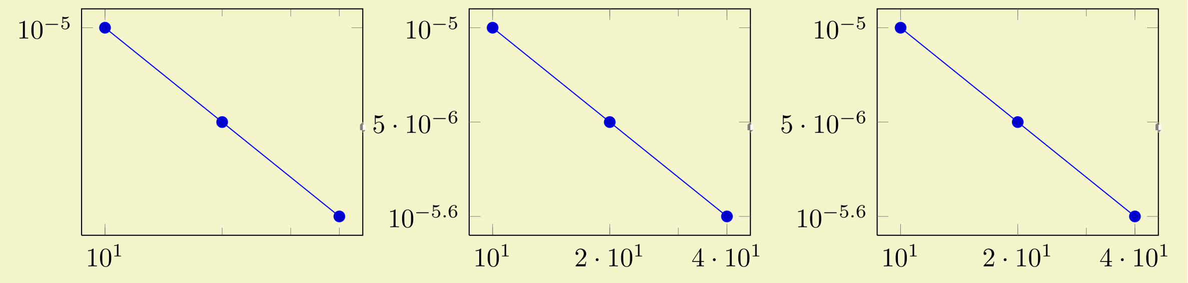
```
\begin{tikzpicture}
    \begin{axis}[minor x tick num=1,
                 minor y tick num=3]
    \addplot {x^3};
    \addplot {-20*x};
    \end{axis}
\end{tikzpicture}
```

/pgfplots/extra x ticks={⟨*coordinate list*⟩}
/pgfplots/extra y ticks={⟨*coordinate list*⟩}

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.



```
\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}
```



```
\pgfplotsset{every axis/.append style={width=5.3cm}}
\begin{tikzpicture}
\begin{loglogaxis}[
    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}[
    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}[
    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/max space between ticks={`⟨*number*⟩`}` (initially 35)
`/pgfplots/try min ticks={`⟨*number*⟩`}` (initially 4)
`/pgfplots/try min ticks log={`⟨*number*⟩`}` (initially 3)

see Options `xtick` and `ytick` for a description.

`/pgfplots/tickwidth={`⟨*dimension*⟩`}` (initially `0.15cm`)
`/pgfplots/major tick length={`⟨*dimension*⟩`}` (initially `0.15cm`)

Sets the width of major tick lines.

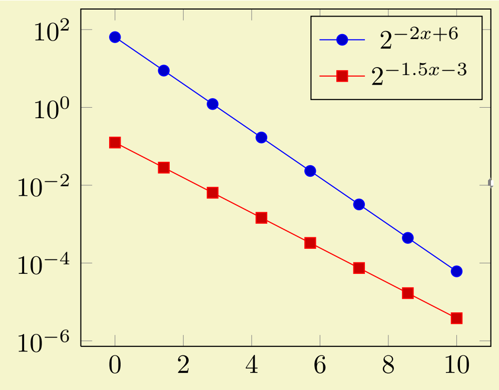
`/pgfplots/subtickwidth={`⟨*dimension*⟩`}` (initially `0.1cm`)
`/pgfplots/minor tick length={`⟨*dimension*⟩`}` (initially `0.1cm`)

Sets the width of minor tick lines.

`/pgfplots/xtickten={`⟨*exponent base 10 list*⟩`}`
`/pgfplots/ytickten={`⟨*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}`'.



```
\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 plot[id=pow2]
    gnuplot {2**(-1.5*x -3)};
\addlegendentry{$2^{-1.5x -3}$}
\end{semilogyaxis}
\end{tikzpicture}
```

/pgfplots/xticklabels={⟨*label list*⟩}
/pgfplots/yticklabels={⟨*label list*⟩}

Assigns a *list* of tick *labels* to each tick position. Tick *positions* are assigned using the `xtick` and `ytick`-options.

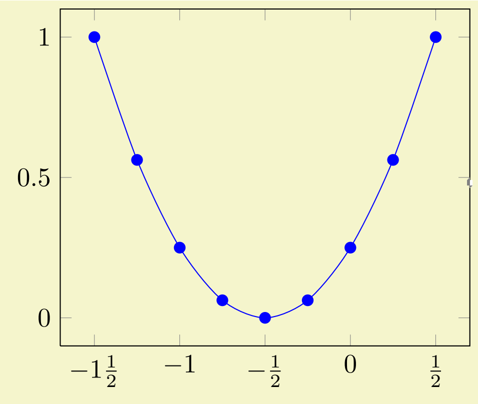
This is one of two options to assign tick labels directly. The other option is `xticklabel={`⟨*command*⟩`}` (or `yticklabel={`⟨*command*⟩`}`). Option '`xticklabel`' offers higher flexibility while '`xticklabels`' is easier to use.

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}, $e$}.
```



```
\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}
```

/pgfplots/xticklabel={⟨*command*⟩}
/pgfplots/yticklabel={⟨*command*⟩}

Use `xticklabel` or `yticklabel` to change the TEX-command which creates the tick *labels* assigned to each tick position (see options `xtick` and `ytick`).

This is one of two options to assign tick labels directly. The other option is '`xticklabels={`⟨*label list*⟩`}`' (or `yticklabels={`⟨*label list*⟩`}`). 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 (a counter).

**\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 and
- `\axisdefaultticklabellog` for logplots, see below.

(the same holds for `yticklabel`). The defaults are set to

```
\def\axisdefaultticklabel{%
    $\pgfmathprintnumber{\tick}$%
}

\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 5.7 and logarithmic axis with `log number format code`, see below.

You can change the appearance of tick labels with

```
\pgfplotsset{every tick label/.append style={
    font=\tiny,
    /pgf/number format/sci}}
```

and/or

```
\pgfplotsset{every x tick label/.append style={
    above,
    /pgf/number format/fixed zerofill}}
```

and

```
\pgfplotsset{every y tick label/.append style={font=\bfseries}}
```

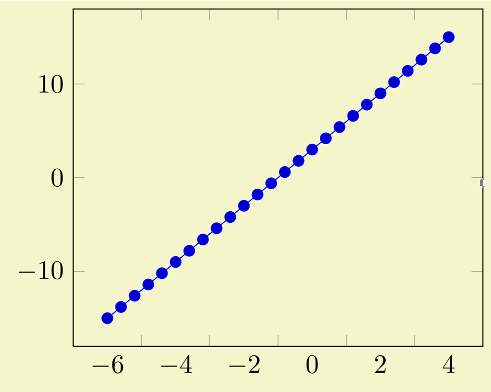
Another possibility is to use

```
\begin{axis}[y tick label style={above,
    /pgf/number format/fixed zerofill}
]
...
\end{axis}
```

which has the same effect as the '`every x tick label`' statement above. This is possible for all PGFPLOTS-`every`-styles, see section 5.10.

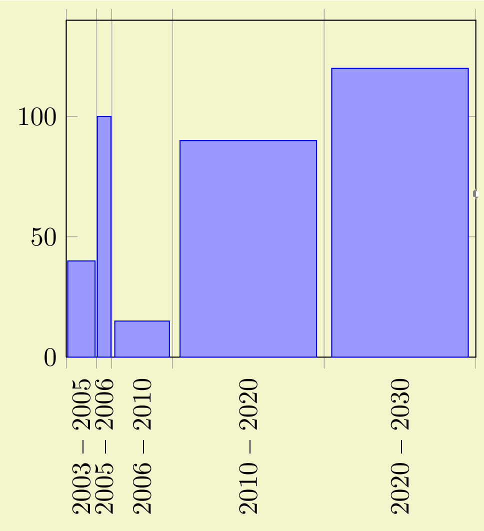
`/pgfplots/x tick label as interval=true|false` (initially `false`)
`/pgfplots/y tick label as interval=true|false` (initially `false`)

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.



```
\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.



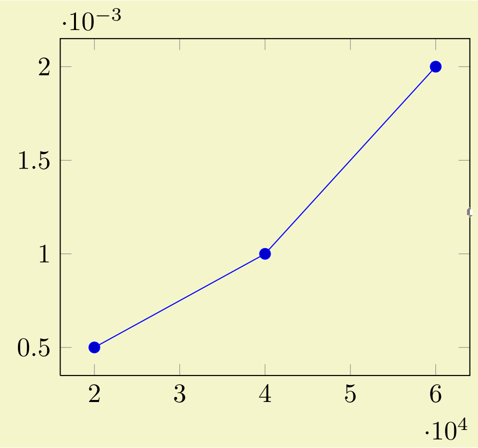
```
\begin{tikzpicture}
\begin{axis}[
    ybar interval=0.9,
    x tick label as interval,
    xmin=2003,xmax=2030,
    ymin=0,ymax=140,
    xticklabel={
       $\pgfmathprintnumber{\tick}$
    -- $\pgfmathprintnumber{\nexttick}$},
    xtick=data,
    x tick label style={
        rotate=90,anchor=east,
        /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/scaled x ticks=true|false|base 10:`⟨*e*⟩`|real:`⟨*number*⟩`|manual:{`⟨*text*⟩`}{`⟨*tick scale code*⟩`}` (initially `true`)

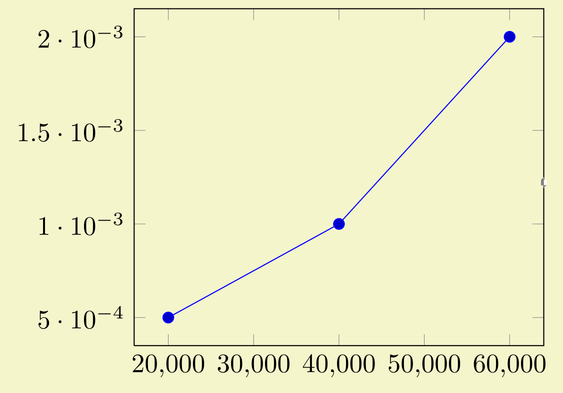
`/pgfplots/scaled y ticks=true|false|base 10:`⟨*e*⟩`|real:`⟨*number*⟩`|manual:{`⟨*text*⟩`}{`⟨*tick scale code*⟩`}` (initially `true`)

`/pgfplots/scaled ticks=true|false|base 10:`⟨*e*⟩`|real:`⟨*number*⟩`|manual:{`⟨*text*⟩`}{`⟨*tick scale code*⟩`}` (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).



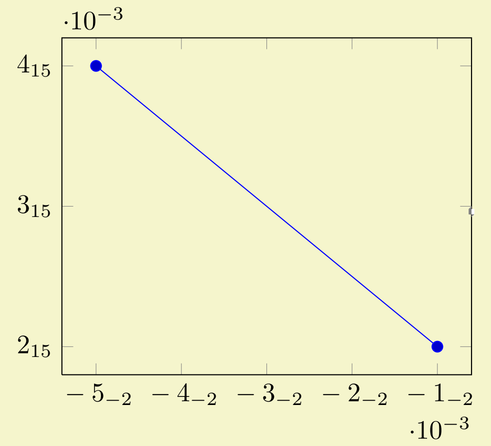
```
\begin{tikzpicture}
\begin{axis}[scaled ticks=true]
    \addplot coordinates {
        (20000,0.0005)
        (40000,0.0010)
        (60000,0.0020)
    };
\end{axis}
\end{tikzpicture}%
```



```
\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 $y$.

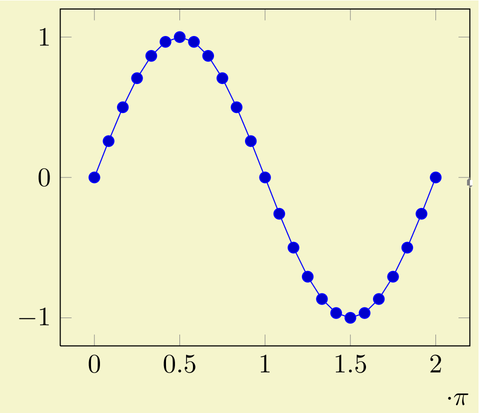
The value `base 10:`$\langle e \rangle$ allows to adjust the algorithm manually. For example, `base 10:3` will divide every tick label by $10^3$:



```
\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:`$\langle number \rangle$ allows to divide every tick by a fixed $\langle number \rangle$. For example, the following plot is physically ranged from 0 to $2\pi$, but the tick scaling algorithm is configured to divide every tick label by $\pi$.

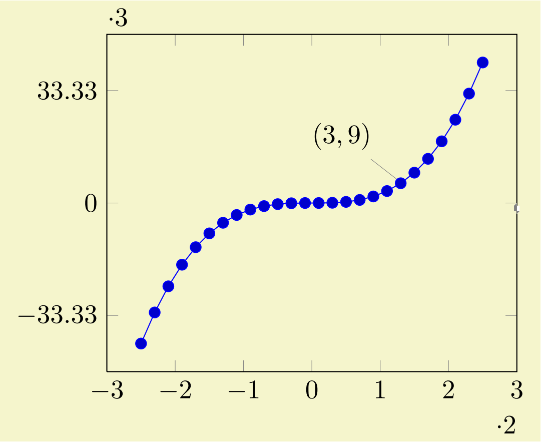


```
\begin{tikzpicture}
    \begin{axis}[
        xtick={0,1.5708,...,10},
        domain=0:2*pi,
        scaled x ticks={real:3.1415},
        xtick scale label code/.code={$\cdot \pi$}]
    \addplot {sin(deg(x))};
    \end{axis}
\end{tikzpicture}
```

Setting `scaled ticks=real:`⟨*number*⟩ also changes the `tick scale label code` to

```
\pgfkeys{/pgfplots/xtick scale label code/.code={$\cdot \pgfmathprintnumber{#1}$}}.
```

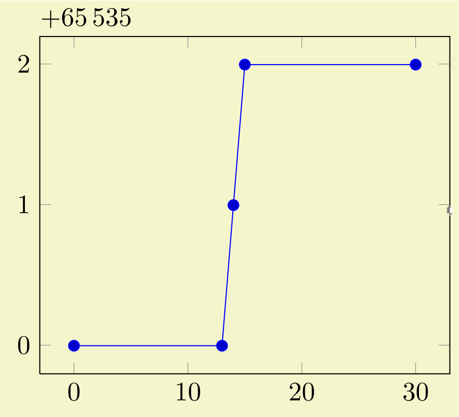
A further – not very useful – example is shown below. Every $x$ tick label has been divided by 2, every $y$ tick label by 3.



```
\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, ⟨*number*⟩ 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:{`⟨*text*⟩`}{`⟨*tick scale code*⟩`}` allows even more customization. It allows *full control* over the displayed scaling label *and* the scaling code: `{`⟨*text*⟩`}` is used as-is inside of the tick scaling label while `{`⟨*tick scale code*⟩`}` is supposed to be a one-argument-macro which scales each tick. Example:



```
\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 plot 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, `{`⟨*tick scale 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 [3] contains detailed documentation about its math engine (including floating point[21]).

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={`⟨...⟩`}`

Allows to change the default code for scaled tick labels. The default is

```
xtick scale label code/.code={$\cdot 10^{#1}$}.
```

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 both, `xtick scale label code` and the corresponding variant for $y$.

`/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.

`/pgfplots/xtick pos=left|right|both` (initially `both`)
`/pgfplots/ytick pos=left|right|both` (initially `both`)

Allows to choose where to place the small tick lines. In the default configuration, this does also affect tick *labels*, see below.

For $x$, the additional choices `bottom` and `top` can be used which are equivalent to `left` and `right`, respectively. Both are accepted for $y$.

`/pgfplots/tickpos=left|right|both`

A style which sets both, `xtick pos` and `ytick pos`.

`/pgfplots/xticklabel pos=left|right|default` (initially `default`)
`/pgfplots/yticklabel 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 axis – keep it to `default` for non-boxed figures.

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/ticklabelpos=left|right|default`

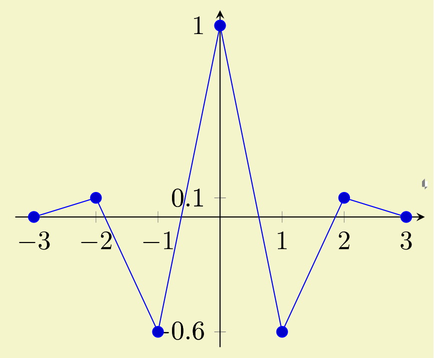
A style which sets both, `xticklabel pos` and `yticklabel pos`.

`/pgfplots/xtick align=inside|center|outside` (initially `inside`)

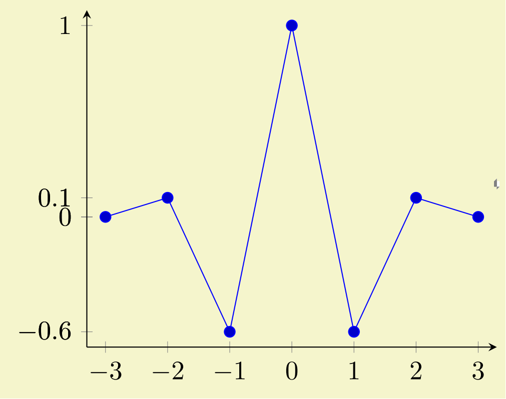
[21] However, that particular stuff is newer than PGF 2.00. At the time of this writing, it is only available as (public) CVS version.

/pgfplots/ytick align=inside|center|outside (initially inside)

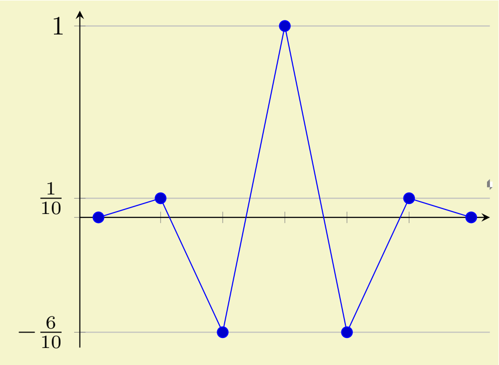
Allows to change the location of the ticks relative to the axis lines. Default is “inside”.



```
\begin{tikzpicture}
\begin{axis}[
    xtick=data,ytick=data,
    xtick align=center,
    axis x line=center,
    axis y line=center,
    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}
```



```
\begin{tikzpicture}
\begin{axis}[
    xtick=data,ytick=data,
    axis x line=bottom,
    ytick align=outside,
    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}
```



```
\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/tick align=inside|center|outside (style, initially inside)

A style which sets both, xtick align and ytick align to the specified value.

/pgfplots/xminorticks=true|false (initially true)
/pgfplots/yminorticks=true|false (initially true)
/pgfplots/xmajorticks=true|false (initially true)
/pgfplots/ymajorticks=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[22].

[22] A uniform list means the difference between all elements is the same for linear axis or, for logarithmic axes, log(10).

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:

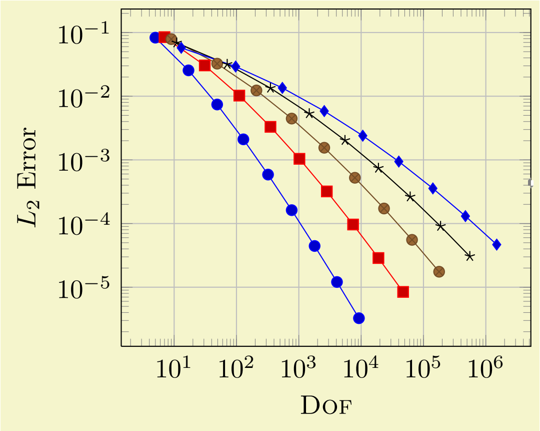
```
\pgfplotsset{every tick/.append style={color=black}} % applies to major and minor ticks,
\pgfplotsset{every minor tick/.append style={thin}}  % applies only to minor ticks,
\pgfplotsset{every major tick/.append style={thick}} % applies only to major ticks.
```

There is also the style "`every tick`" which applies to both, major and minor ticks.

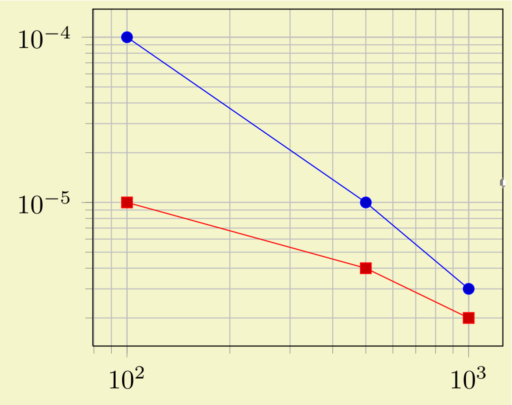
`/pgfplots/xminorgrids=true|false` (initially `true`)
`/pgfplots/yminorgrids=true|false` (initially `true`)
`/pgfplots/xmajorgrids=true|false` (initially `true`)
`/pgfplots/ymajorgrids=true|false` (initially `true`)
`/pgfplots/grids=minor|major|both|none` (initially `both`)

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 11.



```
\begin{tikzpicture}
\begin{loglogaxis}[
    xlabel={\textsc{Dof}},
    ylabel={$L_2$ Error},
    grid=major
]
% see above for this macro:
\plotcoords
\end{loglogaxis}
\end{tikzpicture}
```



```
\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{every axis grid/.style={style=help lines}}
\pgfplotsset{every minor grid/.append style={color=blue}}
\pgfplotsset{every major grid/.append style={thick}}
```

`/pgfplots/xtickmin={⟨coord⟩}`
`/pgfplots/ytickmin={⟨coord⟩}`
`/pgfplots/xtickmax={⟨coord⟩}`
`/pgfplots/ytickmax={⟨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 73 under section 5.4.5, "Axis Discontinuities"'.

## 5.10 Style Options

### 5.10.1 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.

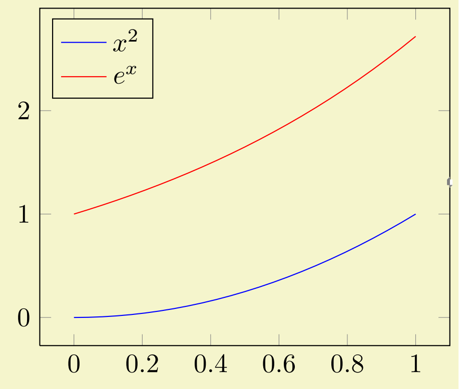
#### Styles installed for single plots

/pgfplots/every axis plot (style, initially empty)

Installed for each plot. This style may contain ⟨*behavior options*⟩ like samples, gnuplot parameters, error bars and it may contain ⟨*style 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.



```
\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 $\# = 0, 1, 2, 3, 4, \ldots$. This option may also contain ⟨*behavior options*⟩.

`/pgfplots/every forget plot` (style, initially `empty`)

Used for every plot which has `forget plot` activated. This option may also contain ⟨*behavior options*⟩.

### Styles for axis descriptions

`/pgfplots/every axis label` (style, initially `empty`)

Used for $x$ and $y$ axis label. You can use '`at=(`⟨*x*,*y*⟩`)` to set its position where $(0, 0)$ refers to the lower left corner and $(1, 1)$ to the upper right one.

`/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)

Used only $x$ or only for $y$ labels, installed after '`every axis label`'.

The initial settings are

```
\pgfplotsset{
    every axis x label/.style={at={(0.5,0)},below,yshift=-15pt},
    every axis y label/.style={at={(0,0.5)},xshift=-35pt,rotate=90}}
```

The predefined node `current axis` can be used to refer to anchors of the unfinished picture. For example '`at={(current axis.origin)}`' will position a label at the *data* coordinate $(0, 0)$. More useful is probably '`at={(current axis.right of origin)}`', see section 5.11 for more details. This remark holds for any axis description, but it is mostly useful for axis labels.

**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/xlabel style={`⟨*key-value-list*⟩`}`
`/pgfplots/ylabel 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`).

`/pgfplots/every axis title` (style, no value)

Used for any axis title. The `at=(`⟨*x*,*y*⟩`)` command works as for '`every axis label`'.

The initial setting is

```
\pgfplotsset{every axis title/.style={at={(0.5,1)},above,yshift=6pt}}
```

`/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 for `every axis label`, the legend's position can be placed using coordinates between 0 and 1, see above.

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*⟩}.

### Styles for axis lines

/pgfplots/every outer x axis line (style, initially empty)
/pgfplots/every outer y 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)

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*⟩}

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 5.4.3 on page 70 for details about styles for axis lines.

### 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/every x tick label (style, initially empty)
/pgfplots/every y 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/xticklabel style={⟨*key-value-list*⟩}
/pgfplots/yticklabel 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).

`/pgfplots/every x tick scale label` (style, no value)
`/pgfplots/every y tick scale label` (style, no value)

Configures placement and display of the nodes containing the order of magnitude of tick labels, see section 5.9 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*⟩`}`

An abbreviation for `every x tick scale label/.append style={`⟨*key-value-list*⟩`}` (or the respective style for $y$, `every y tick scale label`).

`/pgfplots/every x tick` (style, initially `empty`)
`/pgfplots/every y tick` (style, initially `empty`)

Installed for tick *lines* on either $x$ or $y$ axis.

`/pgfplots/x tick style={`⟨*key-value-list*⟩`}`
`/pgfplots/y tick style={`⟨*key-value-list*⟩`}`

An abbreviation for `every x tick/.append style={`⟨*key-value-list*⟩`}` (or the respective style for $y$, `every y tick`).

`/pgfplots/every minor x tick` (style, initially `empty`)
`/pgfplots/every minor y 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*⟩`}`

An abbreviation for `every minor x tick/.append style={`⟨*key-value-list*⟩`}` (or the respective style for $y$, `every minor y tick`).

`/pgfplots/every major x tick` (style, initially `empty`)
`/pgfplots/every major y 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*⟩`}`

An abbreviation for `every major x tick/.append style={`⟨*key-value-list*⟩`}` (or the respective style for $y$, `every major y tick`).

`/pgfplots/every extra x tick` (style, no value)
`/pgfplots/every extra y 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*⟩`}`

An abbreviation for `every extra x tick/.append style={`⟨*key-value-list*⟩`}` (or the respective style for $y$, `every extra y 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`)

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*⟩`}`

An abbreviation for `every axis x grid/.append style={`⟨*key-value-list*⟩`}` (or the respective style for $y$, `every axis y grid`).

`/pgfplots/every minor x grid` (style, initially `empty`)
`/pgfplots/every minor y 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*⟩`}`

An abbreviation for `every minor x grid/.append style={`⟨*key-value-list*⟩`}` (or the respective style for $y$, `every minor y grid`).

`/pgfplots/every major x grid` (style, initially `empty`)
`/pgfplots/every major y 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*⟩`}`

An abbreviation for `every major x grid/.append style={`⟨*key-value-list*⟩`}` (or the respective style for $y$, `every major y grid`).

### Styles for error bars

`/pgfplots/every error bar` (style, initially `thin`)

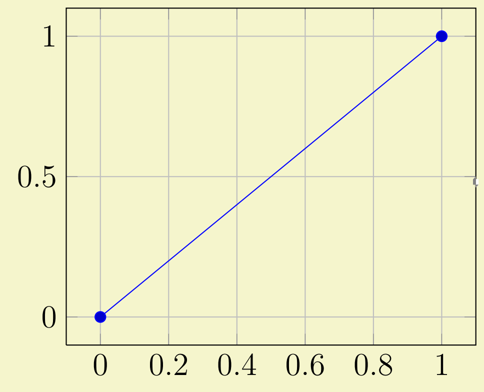
Installed for every error bar.

`/pgfplots/error bars/error bar style={`⟨*key-value-list*⟩`}`

An abbreviation for `every error bar/.append style={`⟨*key-value-list*⟩`}`.

### 5.10.2 (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[23].



```
\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}
```

## 5.11 Alignment Options and Bounding Box Control

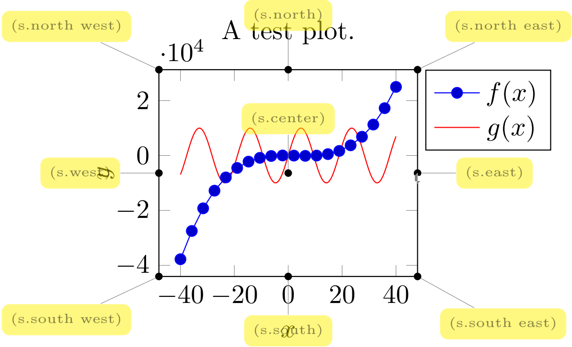
/pgfplots/anchor={⟨*name*⟩} (initially `south west`)

This option shifts the axis horizontally and vertically such that the axis anchor (a point on the axis) is placed at coordinate $(0,0)$.

Anchors are useful in conjunction with horizontal or vertical alignment of plots, see the examples below.

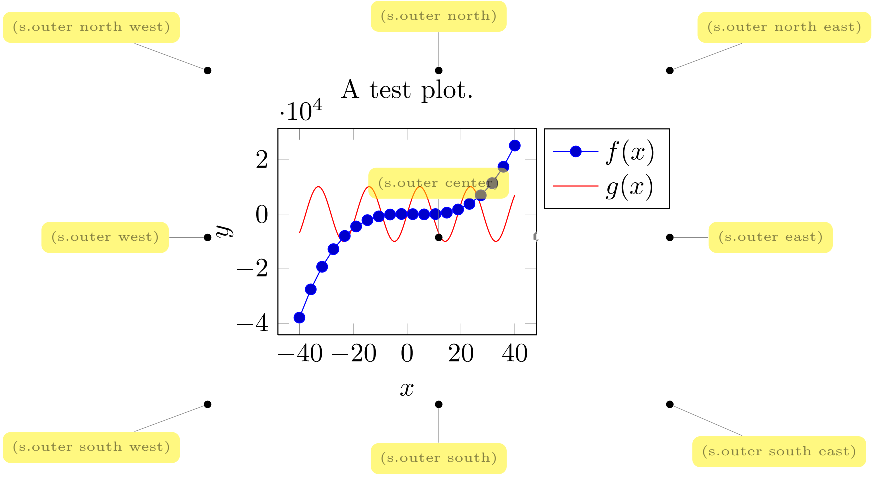
There are four sets of anchors available: anchors positioned on the axis rectangle, anchors on the outer bounding box and anchors which have one coordinate on the outer bounding box and the other one at a position of the axis rectangle. Finally, one can place anchors near the origin.

In more detail, we have Anchors on the axis rectangle,

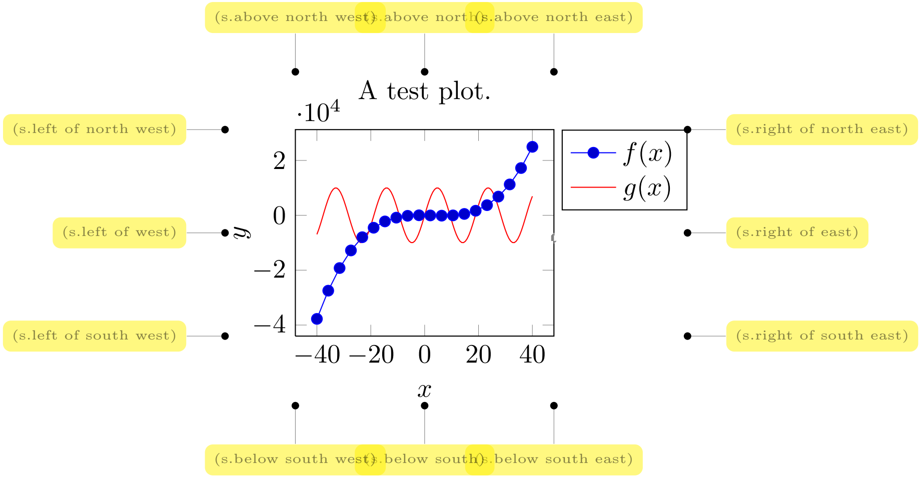


Anchors on the outer bounding box,

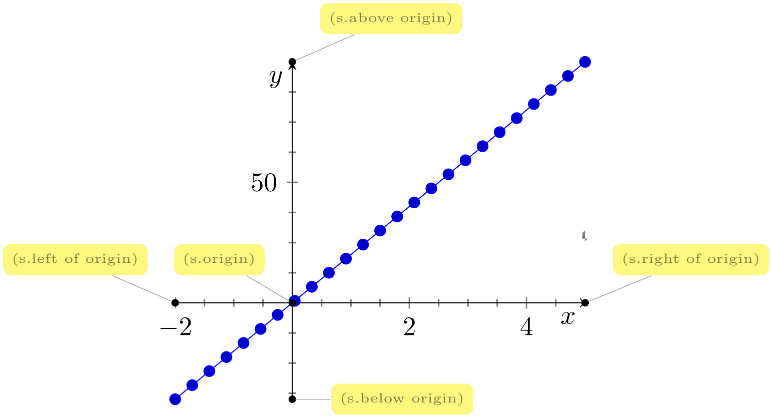
[23] This was possible in a previous version and is still supported for backwards compatibility. But in some cases, it may not work as expected.



There are anchors which have one coordinate on the outer bounding box, and one on the axis rectangle,

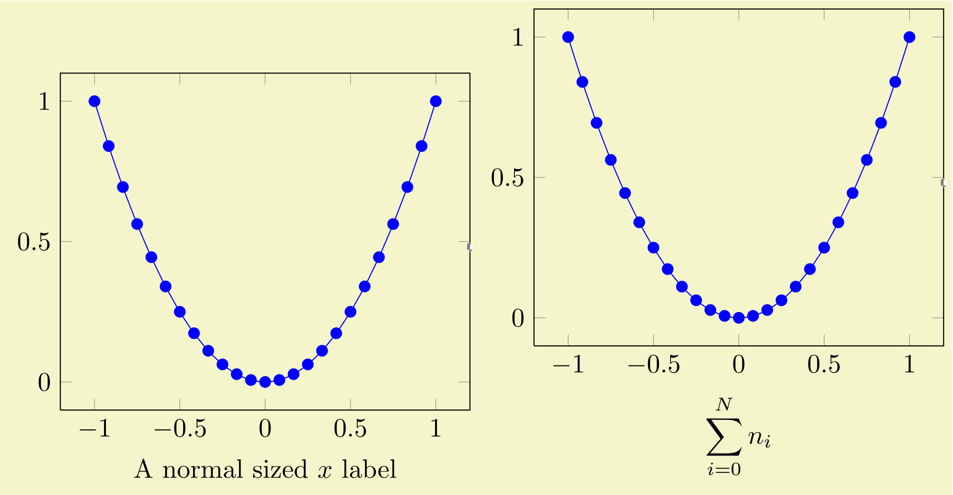


And finally, we have origin anchors which are especially useful when axis lines pass through the origin,

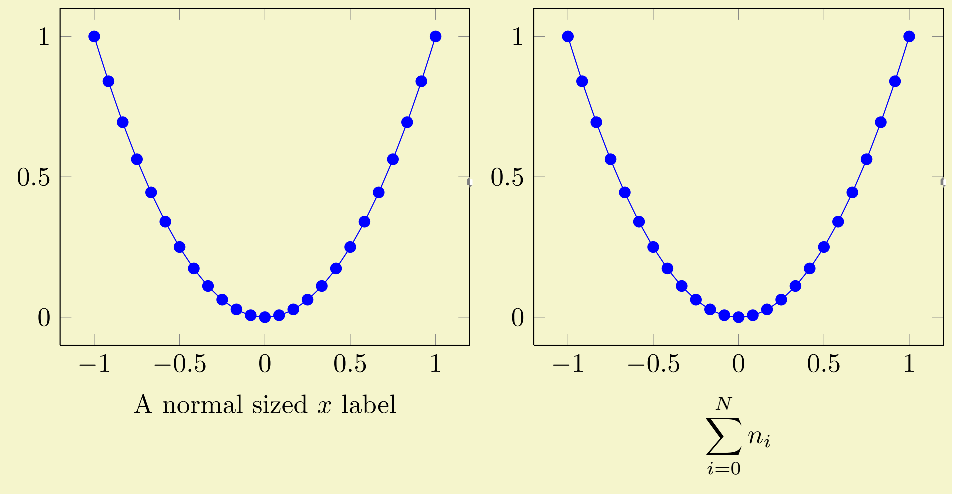


The default value is `anchor=south west`. You can use anchors in conjunction with the Ti*k*Z `baseline` option and/or `\begin{pgfinterruptboundingbox}` to perform alignment.

**Vertical alignment with `baseline`** 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:



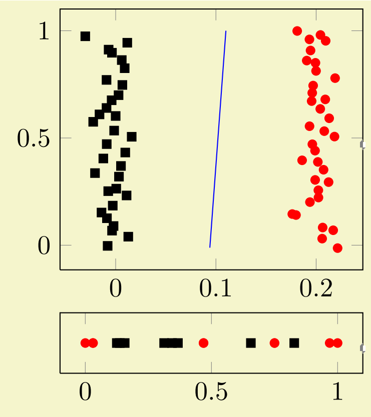
```
\pgfplotsset{domain=-1:1}
\begin{tikzpicture}
    \begin{axis}[xlabel=A normal sized $x$ label]
    \addplot[smooth,blue,mark=*] {x^2};
    \end{axis}
\end{tikzpicture}%
\hspace{0.15cm}
\begin{tikzpicture}
    \begin{axis}[xlabel={$\displaystyle \sum_{i=0}^N n_i $ }]
    \addplot[smooth,blue,mark=*] {x^2};
    \end{axis}
\end{tikzpicture}
```



```
\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` option configures Ti*k*Z to shift position $y = 0$ to the text's baseline and the `south west` anchor shifts the axis such the $y = 0$ is at the lower left axis corner.

**Horizontal Alignment** If you place multiple `axes` into a single `tikzpicture` and use the '`anchor`'-option, you can control horizontal alignment:



```
\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}
```

**Bounding box restrictions** Bounding box restrictions can be realized with several methods of PGF:

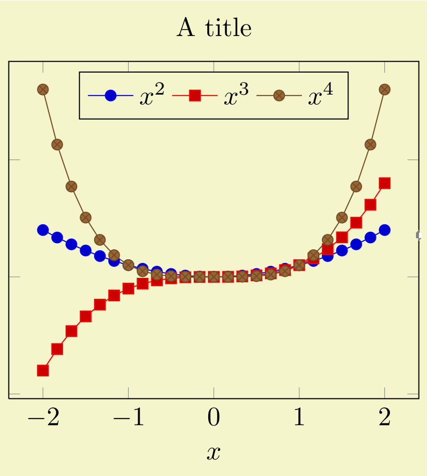
1. The `overlay` option,
2. The `pgfinterruptboundingbox` environment,
3. The `\useasboundingbox` path.

`/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).

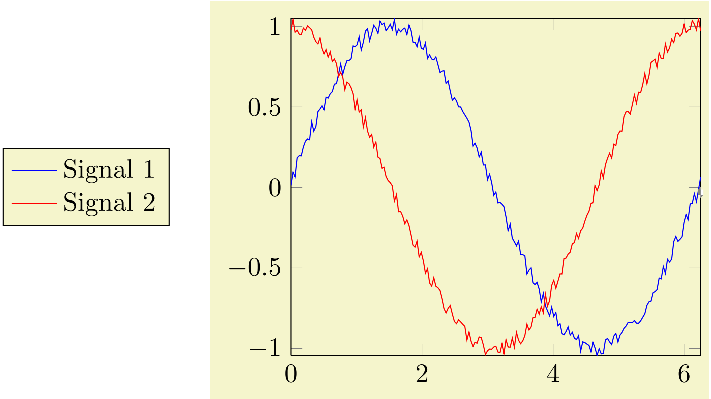
For example, one may want to include only certain parts of the axis into the final bounding box. This would allow horizontal alignment (centering):



```
\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.

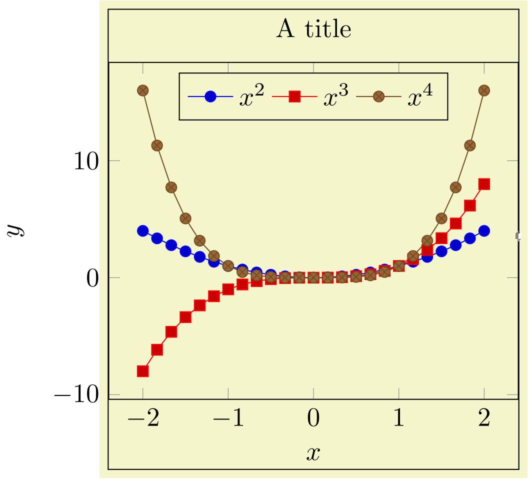
The following example places a legend somewhere without affecting the bounding box.



```
\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 [3].

An alternative to `overlay` is shown below: the figure has a truncated bounding box with is shown using `\fbox`.



```
\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 [3]).

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.

`/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 [3]. The common syntax is `at={(`⟨$x$,$y$⟩`)}`.

## 5.12 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.

`/pgfplots/x coord trafo/.code={⟨...⟩}`
`/pgfplots/y coord trafo/.code={⟨...⟩}`
`/pgfplots/x coord inv trafo/.code={⟨...⟩}`
`/pgfplots/y 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[24].

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 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.

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).

### 5.12.1 Dates as Input Coordinates

The already mentioned application of using dates as input coordinates has been predefined. It relies on the PGF calendar library which converts dates to numbers in the julian calendar. Then, one coordinate unit is one day.

`\usetikzlibrary{dateplot} % LaTeX and plain TeX`
`\usetikzlibrary[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.

The result of the transformation are numbers where one unit is one day.

The transformation is realized using the PGF-calendar module, see [3, Calendar Library]. This reference also contains more information about extended syntax options.

The inverse transformation provides the following three macros which are available during tick label evaluation:

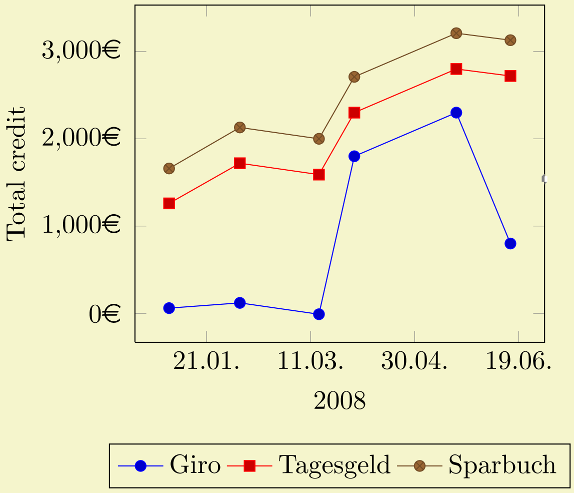
[24] Of course, if coordinates have been generated by gnuplot or PGF, this does no longer hold.

- `\year` expands to the year component,
- `\month` expands to the month component,
- `\day` expands to the day component.

This allows to use `\day.\month.\year` 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 |



```
% requires \usetikzlibrary{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}
```

`/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.

## 5.13 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[25].

So far, the data scale transformation applies only to normal axis (logarithmic scales do not need it).

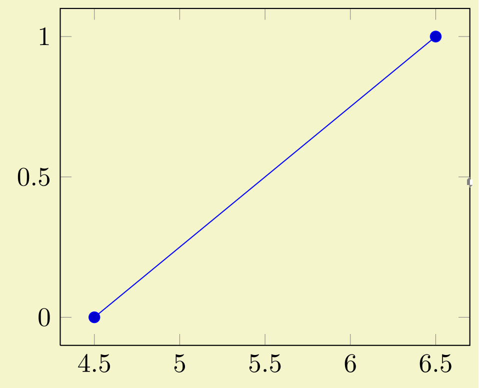
`/pgfplots/x filter/.code={⟨...⟩}`
`/pgfplots/y filter/.code={⟨...⟩}`

The code keys `x filter` and `y filter` allow coordinate filtering. 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.

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.

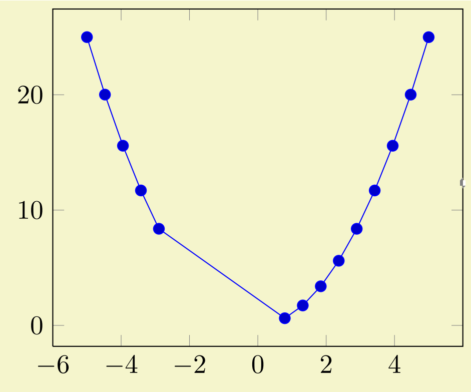


```
\begin{tikzpicture}
\begin{axis}[x filter/.code=
    {\pgfmathadd{#1}{0.5}}]
\addplot coordinates {
    (4,0)
    (6,1)
};
\end{axis}
\end{tikzpicture}
```

Please refer to [3, 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.

[25] 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.



```
\begin{tikzpicture}
\begin{axis}[
    samples=20,
    x filter/.code={
        \ifnum\coordindex>4\relax
            \ifnum\coordindex<11\relax
                \def\pgfmathresult{}
            \fi
        \fi
    }]
\addplot {x^2};
\end{axis}
\end{tikzpicture}
```

There is also a style key which simplifies selection by index, see below.

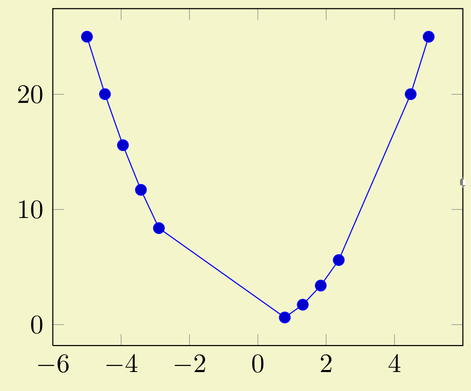
PGFPLOTS invokes the filter with argument `#1` set to the input coordinate. For $x$-filters, this is the $x$-coordinate as it is specified to `\addplot`, for $y$-filters it is the $y$-coordinate.

If the corresponding axis is logarithmic, `#1` is the *logarithm* of the coordinate as a real number, for example `#1=4.2341`.

The arguments to coordinate filters are not transformed. You may need to call coordinate parsing routines.

`/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 $\langle begin\rangle \leq i < \langle end\rangle$ will be skipped.



```
\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/filter discard warning`=`true`|`false` (initially `true`)

Issues a notification in your logfile whenever coordinate filters discard coordinates.

`/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.

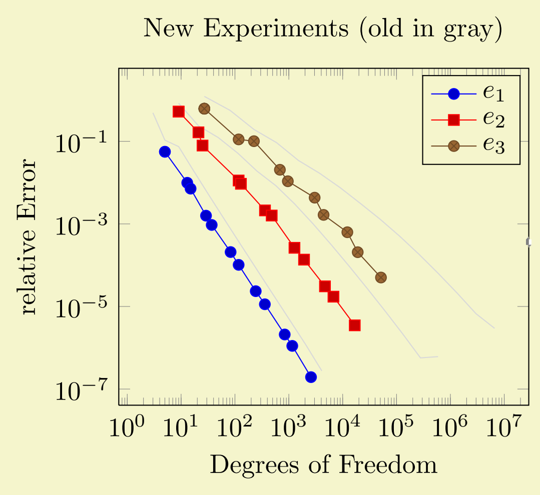
`/pgfplots/execute at end plot`={⟨*commands*⟩}

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. Please provide this option as ⟨*behavior option*⟩ to `\addplot` as in the following example.



```
\begin{tikzpicture}
    \begin{loglogaxis}[
        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] plot[forget plot]
        table {plotdata/oldexperiment1.dat};
    \addplot[black!15] plot[forget plot]
        table {plotdata/oldexperiment2.dat};
    \addplot[black!15] plot[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}
```

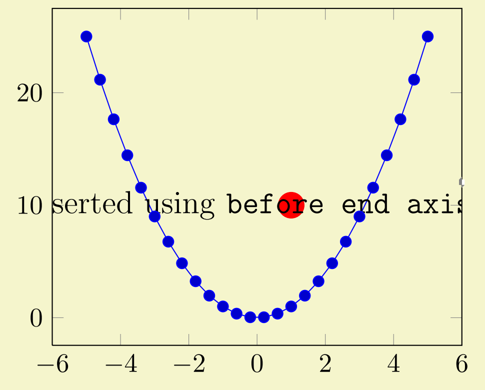
Since forgotten plots won't increase the plot index, they will use the same `cycle list` entry as following plots. This can be used to "interrupt" plots as is described in section 5.2.9.

The style `every forget plot` can be used to configure styles for each such plot. 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={`⟨...⟩`}`

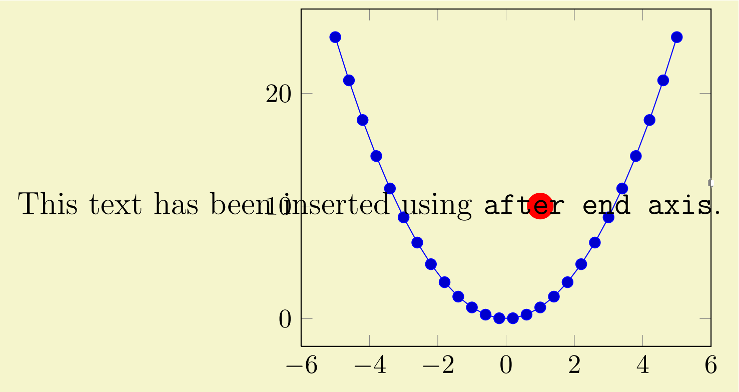
Allows to insert `{`⟨*commands*⟩`}` just before the axis is ended. This option takes effect inside of the clipped area.



```
\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.



```
\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`)

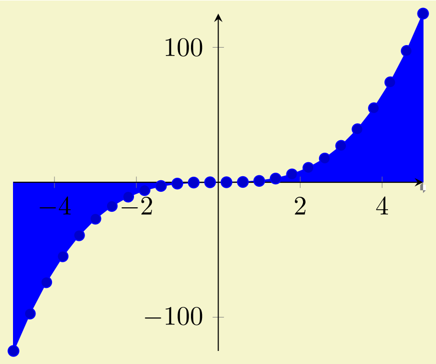
The initial choice `clip marker paths=false` causes markers to be drawn *after* the clipped region. Only their positions will be clipped. As a consequence, markers will be drawn completely, or not at all. The value `clip marker paths=true` is here for backwards compatibility: it does not introduce special marker treatment, so markers may be drawn partially if they are close to the clipping boundary[26].

`/pgfplots/clip=true|false` (initially `true`)

Whether any paths inside an axis shall be clipped.

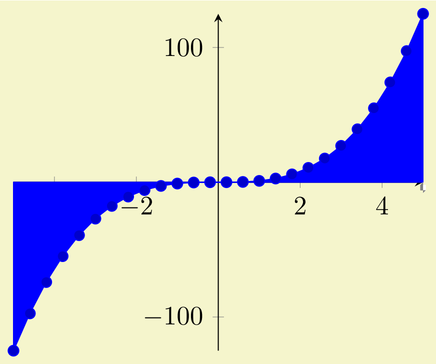
`/pgfplots/axis on top=true|false` (initially `false`)

If set to `true`, axis lines, ticks, tick labels and grid lines will be drawn on top of plot graphics.



```
\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}
```

[26] Please note that clipped marker paths may be slightly faster during TEX compilation.



```
\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 FeuersÃ¤nger 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.

# 6 Related Libraries

This section describes some libraries which come with PGFPLOTS, but they are more or less special and need to be activated separately.

## 6.1 Dates as Input Coordinates

```
\usetikzlibrary{dateplot} % LaTeX and plain TeX
\usetikzlibrary[dateplot] % ConTeXt
```

A library which allows to use dates like 2008-01-01 as input coordinates in plots. The library converts dates to numbers and tick labels will be pretty-printed dates.

This library is documented in section 5.12 on page 110.

## 6.2 Clickable Plots

```
\usetikzlibrary{pgfplotsclickable} % LaTeX and plain TeX
\usetikzlibrary[pgfplotsclickable] % 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.

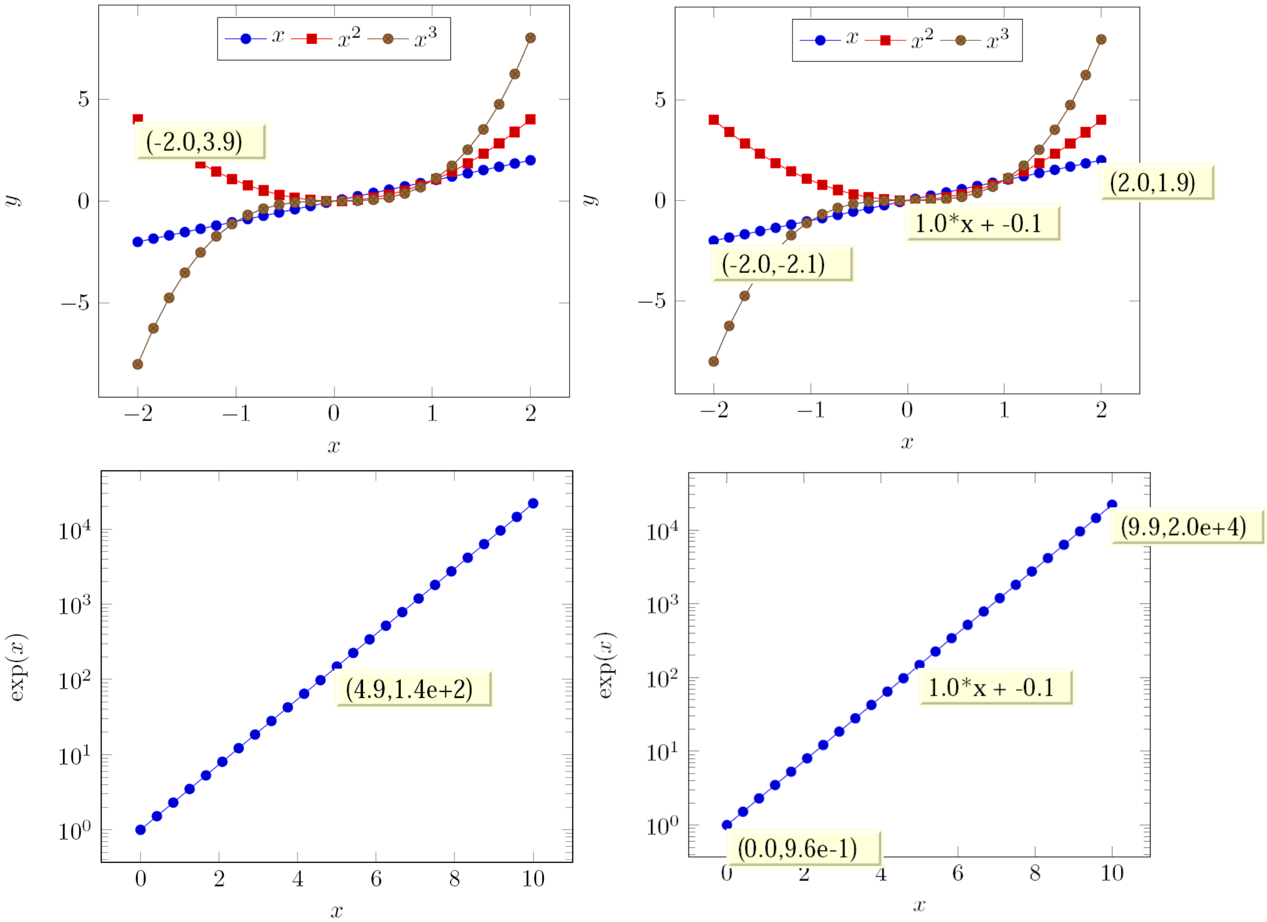
Although this library has been written for PGFPLOTS, it can be used independently of an PGFPLOTS environment.

It is completely sufficient to write

```
\usetikzlibrary{pgfplotsclickable}
```

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 screenshots 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 start- and end points and the slope of the line segment in–between.

This document has been produces with `pgfplotsclickable`, 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 [2][27]. 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).

**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[28].
- Up to now, it is *not* possible to use this library in conjunction with the image externalization methods of section 8.

  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.
- 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 [4] (the javascript API has only a limited set of conversions).

It is possible to customize `pgfplotsclickable` 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

[27] These packages rely on LaTeX, so the library is only available for LaTeX, not for plain TeX or ConTeXt.

[28] In fact, they should be. I don't really know why they don't . . . any hint is welcome.

```
\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 javascripts 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.

### 6.2.1 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 `-`)

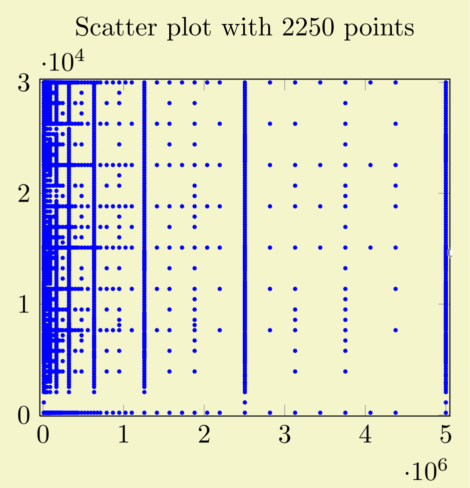
This required key communicates a unique identifier to `\pgfplotsclickablecreate`. This identifier is used to identify the object in javascript, so there can't be more than one of them. If it is empty, a default identifier will be created.

`/pgfplots/annot/xmin={⟨number⟩}`
`/pgfplots/annot/xmax={⟨number⟩}`
`/pgfplots/annot/ymin={⟨number⟩}`
`/pgfplots/annot/ymax={⟨number⟩}` (initially `empty`)

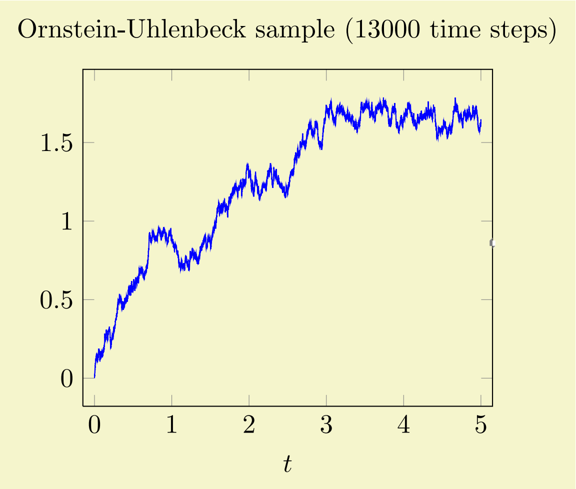
These required keys communicate the axis limits to `\pgfplotsclickablecreate`. They should be set to numbers which can be assigned to a javascript floating point number (standard IEEE double precision).

# 7 Memory and Speed considerations

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[29].



```
\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}
```



```
\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 8 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.

## 7.1 Memory limitations

The default settings of most TEX-distributions are quite restrictive, so it may be necessary to adjust them. For MikTEX, this can be done using simple 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
```

Experiment with these settings if MikTEX runs out of memory.

Usually, the log–file contains a summary about the used ressources, giving a hint which parameter needs to be increased. 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`.

[29] 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.

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.

Unfortunately, TEX does not allow arbitrary memory limits, there is an upper bound hard coded in the executables.

# 8 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.

## 8.1 Export to PDF/EPS

It is possible to export images to single PDF-documents using routines of PGF and/or Ti*k*Z.

### 8.1.1 Using the Externalization framework of PGF "by hand"

The first way to export TEX-pictures to single graphics files is to use the externalization framework of PGF. 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 [3]. This reference also contains information about how to typeset such a document without PGF installed.

I once attempted to write a unix `bash`–script `pgf2pdf.sh` which simplifies these steps in case that every externalized graphics environment is placed into a separate file `.pgf`. Interested readers find it in the installation tree.

**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.

### 8.1.2 Using the automatic Externalization framework of Ti*k*Z

It is also possible to externalize graphics with the high-level library
`\usetikzlibrary{external}`
which comes with (very recent versions of) Ti*k*Z. At the time of this writing, it is **only available in the CVS** 2.0 **version of** PGF, sorry. It is a front-end for `\beginpgfgraphicnamed` which automatically encapsulates every picture in your document with the required externalization commands and performs commands to generate all required graphics files.

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 issues the required calls to `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
`\tikzexternalize`{⟨*the job's real file name*⟩}
as above. No further modification to the document is necessary. Now, the example file `test.tex` of the last subsection reads as follows:

```
\documentclass{article}

\usepackage{pgfplots}

\usetikzlibrary{external}
\tikzexternalize{test}

\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.

The command `\tikzset{external/force remake}` somewhere in the document can be used to remake every following picture automatically. Of course, it is also possible to simply delete every graphics file.

The library can also be configured to produce a list of figures in case system calls are undesired (or unavailable). In that case, `pdflatex --jobname {`⟨*output file name*⟩`}` needs to be invoked for every file name listed in ⟨*real file name*⟩`.figlist`. This step can be done within a script.

The command `\tikzsetexternalprefix{`⟨*file prefix*⟩`}` can be used to prepend a directory name to every figure, for example with

```
\tikzsetexternalprefix{figures/}
```

to produce `figures/test-figure0.pdf` and `figures/test-figure1.pdf` in our example.

The complete reference documentation and remaining options are documented in the documentation for the "PDF externalization library" of [3]. This reference also contains information about how to typeset such a document without PGF installed or how to provide work-arounds with `.pdf` images and bounding box restrictions.

## 8.2 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).

The idea 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 indention 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.

## 8.3 matlab2pgfplots.sh

A `bash`-script which simply starts matlab and runs

```
    f=hgload( 'somefigure.fig' );
    matlab2pgfplots( 'outputfile.pgf', 'fig', f );
```

See matlab2pgfplots.m above.

## 8.4 SVG Output

It is possible to write every single TikZ 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 [3, Section "Producing HTML / SVG Output"].

# Index

# References

[1] U. Kern. Extending LaTeX's color facilities: the `xcolor` package.

[2] 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.

[3] T. Tantau. Ti*k*Z and PGF manual. `http://sourceforge.net/projects/pgf`. $v. \geq 2.00$.

[4] K. van Zonneveld. PhP to javascript conversion project (GPL). `http://kevin.vanzonneveld.net/techblog/article/phpjs_licensing`.