Graphics for LaTeX users

Agostino De Marco

Abstract

This article presents the most important ways to produce technical illustrations, diagrams and plots, which are relevant to LaTeX users. Graphics is a huge subject *per se*, therefore this is by no means an exhaustive tutorial. And it should not be so since there are usually different ways to obtain an equally satisfying visual result for any given graphic design. The purpose is to stimulate readers' creativity and point them to the right direction. The article emphasizes the role of tikz for programmed graphics and of inkscape as a LaTeX-aware visual tool. A final part on scientific plots presents the package pgfplots.

Sommario

Questo articolo presenta gli strumenti più importanti per produrre illustrazioni tecniche, diagrammi e grafici, che sono rilevanti per gli utenti di LaTeX. La grafica è un argomento di per sé molto vasto, quindi questo tutorial non ha la pretesa di essere esauriente. Né dovrebbe esserlo poiché di solito ci sono più modi per ottenere un risultato visivo soddisfacente per un determinato progetto grafico. Il tentativo è di stimolare la creatività del lettore e di indirizzarlo nella direzione giusta. L'articolo sottolinea il ruolo di tikz per la grafica realizzata con codice LaTeX e di inkscape come strumento visivo capace di interfacciarsi con un sistema TeX. L'ultima parte riguarda i grafici scientifici e presenta il pacchetto pgfplots.

1 Introduction

People writing in technical professions — whether they are primarily technical communicators, engineers, scientists, or others — spend a lot of time describing technology, experiments, how things work, what a project entails, and so forth.

On-the-job technical communications often use graphics, such as the illustrations reported in Figure 1, rather than text to convey key points and information. Graphics are photographs, drawings, flowcharts, fancy tables, and other visual representations. As research shows, they play a critically important role in technical and scientific writing. Visual material convey certain kinds of information more clearly, succinctly, and forcefully than words.

One of the golden rules of traditional typography says that both the text and the accompanying visual material has to be composed to create a readable, coherent, and visually satisfying whole that works invisibly, without the awareness of the reader. Typographers and graphic designers claim that an even distribution of typeset material and graphics, with a minimum of distractions and anomalies, is aimed at producing clarity and transparency. This is even more true for scientific or technical texts, where also precision and consistency are of the utmost importance.

Authors of technical texts are required to be aware and adhere to all the typographical conventions on symbols. The most important rule in all circumstances is *consistency*. This means that a given symbol is supposed to always be presented in the same way, whether it appears in the text body, a title, a figure, a table, or a formula. A number of fairly distinct subjects exist in the matter of typographical conventions where proven typesetting rules have been established. Some examples include: (a) the correct display of units of measurement, (b) mathematical formulae, both inline and in display, (c) chemical elements and formulae, (d) numbers, (e) abbreviations. All the rules have to be applied also in visual material.

When dealing with graphics, a typical anomaly may arise when textual annotations do not match with the general design of the main document. This may happen because differences in font usage are evident or because visual signs are inappropriately crafted. Fortunately, LaTeX can be used natively to produce all sorts of visuals, to typeset the annotations of pictures and drawings, and to produce professional quality graphs. More flexible approaches are also available, with the possibility to combine the typesetting strength of LaTeX with specialized graphical software (external to the TeX system).

In the following sections we will focus on illustrations, how they are designed, how they can be generated and handled, and how their textual annotations are typeset with LaTeX. In the second part of the article we will see how scientific plots can be produced according to the same set of quality criteria.

2 Illustrations: general guidelines

The term *illustration* will refer to all kind of pictorial graphics — photographs, drawings, diagrams, and schematics. As mentioned in previous section, it is important in typography to maintain a consistency between text and graphics. When this is achieved the aesthetic result is of such a good quality that the fame of LaTeX as a tool to produce 'beautiful documents' is readily confirmed.

When it comes to producing graphics in the LaTeX world the reader is referred to the book *The LaTeX Graphics Companion* by Goosens *et al.* (2007), where many techniques can be found that let us generate, manipulate, and integrate graphics with texts. Due to several recent improvements in the TeX typesetting system, that brought the users to harness more efficiently both the features of PDF and the resources of their operating system — such as fonts installed outside TeX —, the Graphics Companion does not address some techniques that nowadays are considered standard. These rely on the program pdflatex — or on the more recent xelatex and lualatex — and on the power of the pgf package with his high-level interface tikz. One of the aims of this article is to cover these aspects.

There are many benefits coming from a careful use of visual material in technical documents. These include the following:

- Readers look for and want graphics. They gain more knowledge from communications with graphics, and remember more from communications with graphics.
- Graphics enhance a communication's visual appeal, thereby increasing the readers' concentration on its message.
- Graphics convey some kinds of information much more efficiently than prose. An example of what a reader perceives when flipping through a technical publication is shown by Figure 2. In the picture, the right-hand page contains a detailed illustration with several annotated indications. Well-crafted graphics really can say more than many lines of text.
- Graphics enable writers to convey information to readers who do not share a common language with the writers — or with each other. Graphics communicate information so effectively that they sometimes convey the entire message. An example is given by Figure 1a where the concept of Reflex in modern cameras is so evident.

Examples of visual material of all kinds are shown in the book by Harris (1996), a comprehensive illustrated reference on information graphics.

Generally speaking, when planning a communication, authors should look for places where graphics provide the best way for them to show how something looks (in drawings or photographs), explain a process (flowcharts), make detailed information readily accessible (tables), or clarify the relationship among groups of data (graphs). When the document is typeset with LaTeX, authors have a number of options to produce and handle graphics. Details of the most popular and up-to-date techniques are going to be discussed in the following sections.

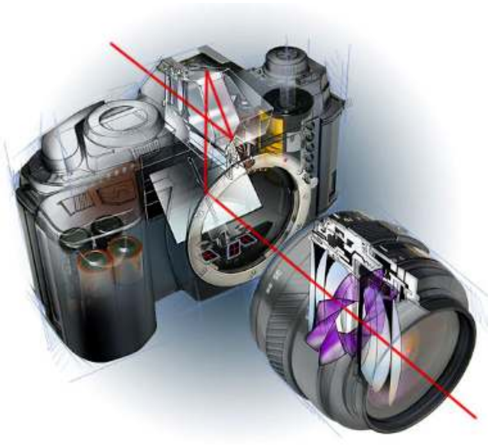
2.1 Guidelines for illustration design

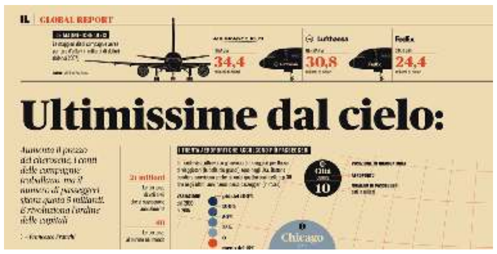
When planning to include an illustration in a document one should keep in mind that, at some point, readers' attention will be going back and forth between the text and the figure, necessarily. Authors should make the effort of having the readers feel at ease during the process. Therefore, having chosen the type of graphic, it must be designed appropriately, with a special focus on usability. Graphics should have the same good qualities of author's prose, easy for readers to understand and use. Here are some general usability rules:

(i) It can be said that graphics have to be designed to support any possible readers' tasks. This is a well-known reader-centered strategy: authors should imagine their readers in the act of using their graphic. Then they should design it to support readers' efforts. In drawings or photographs for step-by step instructions, for example, one should show objects from the same angle that readers will see them when performing the actions described in the illustration and text. This principle is also valid for table design, where one should arrange columns and rows in such an order that will help readers rapidly find the particular pieces of information they are looking for.

(ii) Another important point is about readers' knowledge and expectations; these should be considered carefully by an author/illustrator. Of course, readers will find graphics useful and persuasive only if they can understand them. Some types of graphic are familiar to us all, but other types can be interpreted only by people with specialized knowledge. If one works in a field that employs specialized graphics, then these graphics only can be used when communicating with readers in that particular field, who will understand and expect them. However, when writing for a general audience, authors should consider using alternative types of graphic — or include explanations that general readers need in order to interpret special-use graphics. This kind of simplified visuals are often called 'information graphics' or 'infographics'; they include those images frequently used in presentations at formal meetings or the stylized charts and graphs used in newspapers and magazines (see, for instance, Figure 1b). Many are used for these purposes; however, for every chart, graph, map, diagram, or table used in a presentation or publication, there are thousands that are utilized in other occasions, for what are called operational purposes (Harris, 1996).

(iii) A well-known general rule-of-thumb when designing visual material is that of *seeking simplicity*. As seen in the preceding point, by simplifying their graphics authors can also make their illustrations easy to understand and use. Simplicity is especially important for graphics that will be read on a computer screen or from a projected


(a) An example of technical illustration showing the Reflex principle.



(b) A newspaper illustration. This example shows a particular kind of artwork known as 'infographics.'

Figure 1: Examples of on-the-job technical illustrations.

image; people have more difficulty reading from these media than from paper. Here are some effective strategies for keeping your graphics simple: (*a*) Include only a manageable amount of material. Sometimes, it's better to separate information into two or more graphics than to cram it all into one. (*b*) Eliminate unnecessary details. Like unnecessary words in prose, superfluous details in graphics create extra, unproductive work for readers and obscure the really important information. In many cases the elimination of extraneous detail can simplify and improve the effectiveness of a graph.

(IV) One of the most important points about illustration design is the effectiveness of textual labels. Important content should always be labelled clearly. Labels help readers locate the information in a graphic and understand what it shows. In diagrams, every part that is important to readers has to be labelled. But authors should avoid labeling other features because unnecessary labels clutter a graphic, making it difficult to understand and use. An appropriate wording should be chosen for all labels, which should be placed where they are easily seen. If necessary, a line can be drawn from the label to the item it refers to. Authors have to avoid placing a label on top of an important part in their graphic. It has to be noted that labels placed in a graphic are much easier than a key for readers to use.

(V) A special mention goes to informative titles. Titles help readers to find the graphics they are looking for and also to know what the graphics contain once they locate them. Typically, titles may go in figure captions — for example, "Figure 3. Effects of Temperature on the Strength of M312." Titles can be made brief and informative at the same time. In some cases more words can be used if they are needed in order to give readers precise information about the graphic. For this purpose LATEX provides the `figure` environment where one can use the `\caption` macro. There are extension packages, such as `float` and `caption`, that help users customize the style of captions.

(VI) Readers might seek a specific figure whose location is not obvious from the regular table of contents. To help this process authors should provide a separate list of the figures and the pages where they can be found. Lists of figures are generated in LATEX by the native macro `\listoffigures`.

2.2 Interplay between graphics and text

To enable graphics to achieve their potential for usability and persuasiveness, they should be carefully integrated with a communication's prose. Here are four common strategies authors can use to create a single, unified message in which their graphics and prose work harmoniously together.

(I) First of all, graphics have to be introduced in the text. When people read, they read one sentence and then the next, one paragraph and then the next, and so on. In a manuscript, when one wants to make sure that the next element the reader scans is a table or a chart rather than a sentence or a paragraph, one needs to direct people's attention from the prose to the graphic and tell them how the graphic relates to the statements they just read.

(II) Whatever kind of introduction is made in the text, it should be placed at the exact point where one wants the readers to focus their attention on the graphic. For this purpose, graphics should be placed near the point they are referenced to in the text. When readers come to a statement asking them to look at a graphic, they lift their eyes from the prose and search for the graphic. That search has to be made as short and simple as possible. Ideally, the graphic should be

Figure 2: A technical book in the hands of a reader. The right-hand page contains a full-height annotated illustration.

placed on the same page as the prose references to it. If there is not enough room, the graphic should be put on the facing page or the page that follows. If the figure is placed farther away than that (for instance, after two pages or in an appendix), the text should mention the number of the page on which the figure can be found. These strategies are handled in LaTeX by a careful positioning of floating objects in the source file and by using the cross-referencing mechanism (enanced with packages like varioref, cleveref and hyperref).

(iii) Having introduced the visual material properly, one has to state the conclusions that one wants readers to draw. One way to integrate graphics into the text is to state explicitly those conclusions. Otherwise, readers may draw conclusions that are quite different from the ones that the author has in mind.

(iv) When appropriate, graphics may include explanations, or longer annotations. Sometimes illustrations designers can help readers understand the message by incorporating explanatory statements into the figures—for instance, "Counterclockwise moments are positive by convention."

It has to be mentioned the existence of a well written section entitled "Graphics guidelines" in the pgf package documentation (Tantau, 2016), a LaTeX extension package that will be introduced later on in this paper. This material, that we encourage to read, is not only about pgf, but about general guidelines and principles concerning the creation of graphics for scientific presentations, papers, and books.

3 Drawing and annotating with native LaTeX extensions

In this section we introduce some facilities offered by LaTeX and its extension packages for producing graphic material directly in the source document.

Some of the available drawing facilities are standalone, in the sense that they rely totally on the program latex or pdflatex and do not require functionalities of other programs. Some other drawing tools, instead, rely on other programs distributed with the standard TeX system (or publicly available).

There are several drawing tools that one can pick up and use once a full TeX system is installed. But we have basicly three main facilities, which are readily listed:

(i) The package tikz, which is a high-level interface to the low-level graphics package pgf. This is considered the standard drawing facility. At the time of writing this paper (and probably for years to come) it is the most popular standalone tool for producing line graphics in the LaTeX world.[1] tikz comes with several specialized extension packages (tikz libraries). It is a very powerful package, flexible, easy to use, and stunning.

(ii) The package pstricks and its companion packages. This was the tool of choice before tikz came into the scene. pstricks is designed to embed low-level picture drawing primitives available in the PostScript language. These primitives are

1. To learn more about pgf, see the extensive documentation on CTAN `http://www.ctan.org/pkg/pgf` and the collection of examples on `http://www.texample.net`.

exposed to the user in terms of LATEX macros so that they can work smoothly with the typesetting engine. With an an up-to-date TEX distribution pstricks can be used with pdflatex, provided that the option `-shell-escape` has been enabled.[2]

(iii) The native LATEX environment picture. This environment is part of LATEX kernel and, when enhanced with the standard packages pict2e and curve2e, allows for the creation of line graphics from a number of fairly basic constructs.

The above list mentions the most important ways to produce graphics with LATEX and they will be treated in reverse order in next sections. But they are not the only possible options. The following is a list of other graphics packages and tools included in standard TEX distributions:

- The package Xy-pic. A tool which is best suited to graphs and diagrams, but has capabilities for other formats.[3]
- The package ePiX. A tool to produce mathematical figures. It creates pstricks, TikZ, or eepic macros.[4]
- The program METAPOST. Similar to the program METAFONT used to create early TEX fonts. It outputs special PostScript files that can be imported by both LATEX and PDFLATEX. it is the default drawing program used by Knuth himself. Its source code may be included into a LATEX file and, via the emp package, the METAPOST code is executed and the resulting graph is imported into the typeset document. METAPOST is now integrated in LuaTEX via the mplib library. Using LuaTEX, one can include METAPOST figures directly in the TEX/LATEX file with the luamplib package, without using any external software.[5]
- The program MetaFun. An extension to METAPOST.[6]
- The program asymptote. A descriptive vector graphics software and language for technical drawing. The language is inspired to METAPOST but with an improved syntax taken from C++.[7] LATEX users can use the package asymptote that provides the environment asy to enclose Asymptote language code into LATEX sources. See De Marco (2009) for a presentation of this powerful tool.

This latter list of tools is reported for the sake of completeness. They can be considered of secondary importance for the scope of the present article and the interested readers are referred to the suggested links and references.

2. To learn more about pstricks, see the documentation on CTAN `http://www.ctan.org/pkg/pstricks` and the dedicated section on TUG website `http://tug.org/PSTricks`.
3. `http://www.tug.org/applications/Xy-pic/Xy-pic.html`
4. `http://mathcs.holycross.edu/~ahwang/current/ePiX.html`
5. `http://www.tug.org/metapost.html`
6. `http://wiki.contextgarden.net/MetaFun`
7. `https://ctan.org/pkg/asymptote`

Next three subsections present some selected examples of technical illustrations made, respectively, with the environments: picture (from package picture), pspicture (from package pstricks), and tikzpicture (from package tikz).

4 The standard LATEX picture environment

This section presents briefly the standard picture environment. Examples of usage of this environment are reported in Figure 3, Figure 4 and Figure 5. A picture has two main dimensions, width and height, that users can pass to the environment as arguments. In addition to picture, macros such as `\put`, `\line`, `\vector` and several other commands have arguments that expect numbers that are used as factor for `\unitlength`. The latter is a TEX length that can be set by users with the macro `\setlength` to scale their drawings as appropriate.

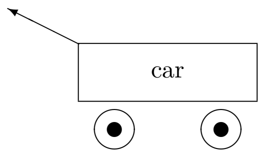
LATEX was born in 1984 with the native picture environment that allowed users to draw lines, vectors, circles and 'ovals' with some limitations. Special fonts were used to draw all graphic objects. This was a severe limitation, because at LATEX's birth all fonts usable with LATEX could contain only 128 glyphs. This was true with text fonts as well as with LATEX special drawing fonts.

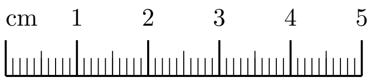
This implied that there were only two thicknesses available for all graphic objects, and, worst of all, the line and vector slopes and the circle diameters were available in only a limited number. Line slope parameters could only be integer relatively prime numbers in the range $[-6, +6]$, while for vectors they could range only within $[-4, +4]$. This meant that only a limited set of lines and vectors could be drawn.

Filled circles (disks) were limited to diameters from 1 pt to 15 pt, while unfilled circles were limited to diameters from 1 pt to 40 pt; circles of diameter larger than 15 pt were drawn as four quarter circles, and each of these were available also for the rounded corners of 'ovals.'

This environment allowed typesetting of text by means of extensions of the `\makebox` macros, and framed text by means of `\framebox`; fine tuning of the position of the text allowed placement of any text in the precise required position and, most important of all, the fonts used were the same ones used in the text of the document as a whole. Typographically, this feature was and remains the most valuable of this built-in environment.

Leslie Lamport, in his second edition of the LATEX handbook (Lamport, 1994) fixed the syntax of a new extended picture environment, where most if not all limitations of the standard implementation could be overcome: unlimited slopes of lines and vectors, any circle radius, arbitrary line thickness also for curved lines, real third order Bézier curves, *et cetera* (see for example Figure 3).





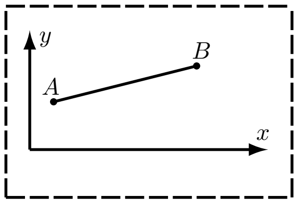
```
\setlength{\unitlength}{1mm}
% in preamble
\newcounter{cms}
% ...
\begin{picture}(50,39)
  \put(0,7){\makebox(0,0)[bl]{cm}}
  \multiput(10,7)(10,0){5}{%
    \addtocounter{cms}{1}\makebox(0,0)[
    b]{
      \arabic{cms}}}
  \put(15,20){\circle{6}}
  \put(30,20){\circle{6}}
  \put(15,20){\circle*{2}}
  \put(30,20){\circle*{2}}
  \put(10,24){\framebox(25,8){car}}
  \put(10,32){\vector(-2,1){10}}
  \multiput(1,0)(1,0){49}{\line(0,1)
    {2.5}}
  \multiput(5,0)(10,0){5}{\line(0,1)
    {3.5}}
  \thicklines
  \put(0,0){\line(1,0){50}}
  \multiput(0,0)(10,0){6}{\line(0,1)
    {5}}
\end{picture}
```

Figure 3: Lamport used this drawing in his handbook to describe his picture environment.

These extensions are nowadays incorporated in the package pict2e.

In addition to the standard pict2e package, users may rely on the curve2e package developed by Claudio Beccari.[8] This extension to pict2e enhances the syntax of the `\line` command and introduces two new commands: `\Line`, which allows the user to specify the relative x and y displacements from the current point, and `\LINE`, which has two absolute coordinates as its arguments. Similarly, `\Vector` and `\VECTOR` are defined and extend the `\vector` command. The package also defines a `\polyline` command for drawing polylines between two (minimum) or more vertices that are specified as arguments, as well as a `\Curve` command for drawing third-order Bézier curves. This macro needs a series of nodes on the curve together with the tangent at each node. Finally, curve2e introduces an `\Arc` command and some variants for drawing circular arcs of any radius and any angular aperture.

8. See the documentation on CTAN: `https://ctan.org/pkg/curve2e`.



```
% in preamble
\usepackage{pict2e}
% ...
\begin{picture}(120 ,80)
  \put(30,30){\circle*{3}}
  \put(30,33){\makebox(0,0)[br]{$A$}}
  \put(90,43){\circle*{3}}
  \put(88,47){\makebox(0,0)[bl]{$B$}}
  \linethickness{1.2pt}
  \Line(30,30)(90,43)
  \put(10,10){\vector(1,0){100}}
  \put(110,14){\makebox(0,0)[b]{$x$}}
  \put(10,10){\vector(0,1){60}}
  \put(14,70){\makebox(0,0)[l]{$y$}}
  % dashed box
  \put(0,0){\dashbox{5}(120,80){}}
\end{picture}
```

Figure 4: An example showing some basic commands offered by the standard picture environment enhanced by the pict2e package.

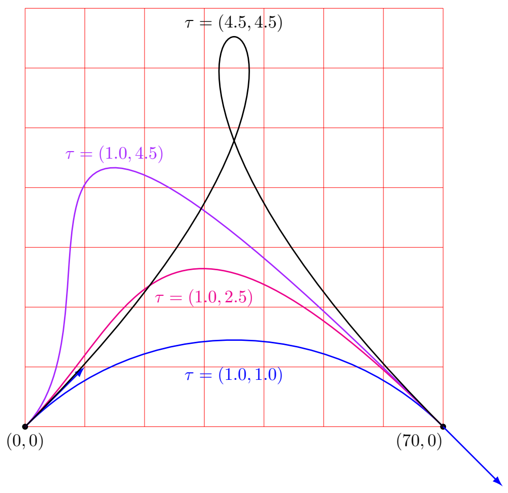
Recently the pict2e package has incorporated some features proposed by curve2e and some drawings are made possible by simply including pict2e. Figure 4 shows an example of a very simple illustration and the related LaTeX code. The example in Figure 5 demonstrates the use of tension parameters that modify the shape of a Bezier curve connecting two points.

For more details on the use of the picture environment the reader might want to consult reference Beccari (2011) for a review of the available commands and for examples of graphics with this native drawing tool.

5 Using pstricks

The pstricks package is presented very briefly in this section. Explaining all the features of the package is beyond the scope of this article. For an extended treatment of the subject we encourage to consult the *LaTeX Graphics Companion* (Goosens *et al.*, 2007), which contains an entire chapter on pstricks, and the book by Herbert Voß on graphics and PostScript for TeX and LaTeX (Voß, 2011).

The extension package pstricks provides the environment pspicture that will contain all commands



```
% in preamble
\usepackage{pict2e}
\usepackage{curve2e}
% ...
\begin{picture}(80,70)
  \put(0,0){\GraphGrid(70,70)}
  \put( 0,0){\circle*{1}}
  \put(70,0){\circle*{1}}
  \put( 0,0){\makebox(0,-1)[ct]{$(0,0)$}}%
  \put(70,0){\makebox(0,-1)[rt]{$(70,0)$}}%

  \thicklines

  % Default tension
  \put(0,0){\color{blue}
    \Curve(0,0)<1,1>(70,0)<1,-1>
    \put( 0,0){\Vector(10,10)}
    \put(70,0){\Vector(10,-10)}
    \put(35,10){\makebox(0,0)[ct]{$\tau
     =(1.0,1.0)$}}%
  }

  \put(0,0){\color{magenta}
    \Curve(0,0)<1,1>(70,0)<1,-1;1.0,2.5>
    \put(30,23){\makebox(0,0)[ct]{$\tau
     =(1.0,2.5)$}}%
  }

  \put(0,0){\color[rgb]{0.65,0.15,1.0}
    \Curve(0,0)<1,1>(70,0)<1,-1;1.0,4.5>
    \put(15,47){\makebox(0,0)[ct]{$\tau
     =(1.0,4.5)$}}%
  }

  \Curve(0,0)<1,1>(70,0)<1,-1;4.5,4.5>
  \put(35,69){\makebox(0,0)[ct]{$\tau
     =(4.5,4.5)$}}%
\end{picture}
```

Figure 5: An example showing some basic commands offered by the standard picture environment enhanced by the curve2e package.

```
% arara: latex
% arara: dvips
% arara: ps2pdf
\documentclass[%
  border={0.6cm 0.6cm 0.6cm 0.6cm}% lbrt
  ]{standalone}

\usepackage[pdf]{pstricks}

\begin{document}
\begin{pspicture}(4,5)
  \psgrid
\end{pspicture}
\end{document}
```

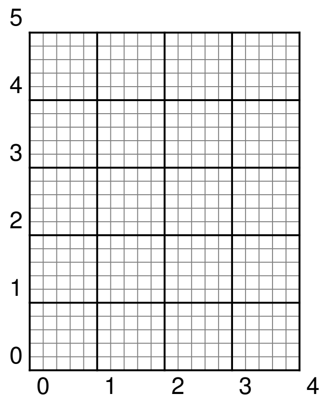


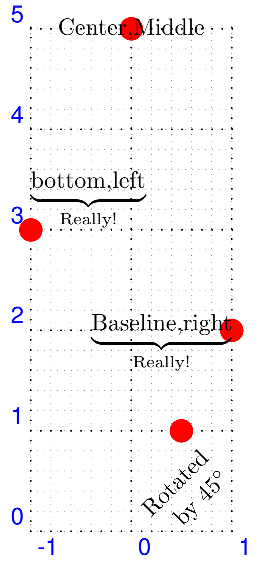
Figure 6: The basic command \psgrid offered by the pspicture environment.

necessary to produce a drawing. One important tool for drawing pictures with pstricks is the grid. It can be activated with the \psgrid macro. If no further arguments are given in the command it produces a grid with width and height as determined by the size of the enclosing pspicture. Figure 6 shows the basic commands offered by the pspicture environment. Thanks to the class standalone the output document is a PDF containing the cropped diagram. The work flow that produces the final PDF is handled by the editor texworks coupled with arara.[9]

The way pstricks works is an example of a drawing package where some of the functionalities are provided by an external program, in this case dvips. The set of macros that are collectively known as pstricks exploit the PostScript language to a great degree by writing to the output file — a .dvi file in this case — the raw PostScript code necessary to draw all of the required objects.

The most important basic geometric objects are produced by the macros \psline, \psdots, \pspolygon, \pscircle, \psellipse, \psarc,

9. http://texdoc.net/texmf-dist/doc/support/arara/arara-usermanual.pdf



```
% arara: latex
% arara: dvips
% arara: ps2pdf
\documentclass[%
  border={0.6cm 0.6cm 0.6cm 0.6cm}% l b r t
  ]{standalone}
\usepackage[pdf]{pstricks}
\usepackage{pst-all}
\usepackage{pstricks-add}

\begin{document}
\begin{pspicture}(-1,0)(1,5)
  \psgrid[griddots=10,subgriddots=3,
    gridlabelcolor=blue](-1,0)(1,5)
  \psdots[linecolor=red,dotsize=10pt]
    (0,5)(-1,3)(1,2)(0.5,1)
  \rput(0,5){Center,Middle}
  \rput[bl](-1,3){%
    $\underbrace{\text{bottom,left}}_{\text{Really!}}$}
  \rput[Br](1,2){%
    $\underbrace{\text{Baseline,right}}_{\text{Really!}}$}
  \rput[tr]{45}(0.5,1){
    \parbox{5cm}{\flushright Rotated\\ by $45^{\circ}$}
  }
\end{pspicture}
\end{document}
```

Figure 7: Placing whatever,wherever in pspicture environment.

`\pscurve`, `\psbezier`, whose names are self-explanatory. Figure 7 shows a possible customization of the grid made by passing a number of arguments to `\psgrid`. The same diagram contains a few examples of how a text box can be placed on the canvas. Figure 8 demonstrates the use of `\psline` and `\pscurve` with their arguments to obtain simple lines with various line endings.

A showcase of graphics produced with pstricks and other companion packages is given by Figure 9.[10] Figure 9a represents a power balance in a form known as *Sankey diagram* and is made by coupling to pstricks the package pst-node. Figure 9b shows a three-dimensional scene produced with the package pst-solides3d. In Figure 9c the power of the package pst-plot is demonstrated by plotting the diagram of a mathematical function using 4000 connected points. In Figure 9d pst-plot is used to illustrate the construction of a hypocycloid curve. Finally, Figures 9e and 9f demonstrate the potential of package pst-3dplot.[11]

6 Using pgf/tikz

This is an introduction to drawing diagrams/pictures using the tikz package, which is built on top of pgf, a platform- and format-independent macro package for creating graphics. The pgf package is smoothly integrated with TEX and LATEX and, as a result, tikz also lets users incorporate text and mathematics in their diagrams. The tikz package also supports the beamer class, which is used for creating incremental computer presentations.

The main purpose here is to emphasize the potential of tikz and inspire a creative use of this powerful extension. The interested reader is referred to the excellent package documentation itself (Tantau, 2016) for more detailed information.

The pgf package, where 'pgf' means 'portable graphics format,' is a package for creating *inline* graphics, that is, it defines a number of TEX commands that can draw graphics within the typesetting process. Graphics objects are put into boxes and treated as normal items to be taken care of by the LATEX output routine.

As occurs with the environments picture and pspicture, when one uses pgf the graphics are *programed*, just as documents are programed when TEX is used. The package users get all the advantages of the TEX-approach to typesetting for their graphics: quick creation of simple graphics, precise positioning, the use of macros, often superior typography. But also all the disadvantages are inherited: steep learning curve, no wysiwyg, small changes require a recompilation, and the code does not really *show* how things will look like.

10. For an extensive gallery of examples visit this link http://tug.org/PSTricks/main.cgi?file=examples .
11. http://texdoc.net/texmf-dist/doc/generic/pst-3dplot/pst-3dplot-doc.pdf .

```
\begin{pspicture}(5,5)
  \psgrid[griddots=10,subgriddots=3,
    gridlabelcolor=blue]
  \psline{-*}(1,4)(2,4)
  \psline{-,linewidth=3pt}(3,4)(4,4)
  \psline{->,linecolor=magenta,
    linewidth=2pt}(2.5,4)(2.5,2.5)
  \pscurve{|-|}(1,2)(2.5,1)(4,2)
\end{pspicture}
```

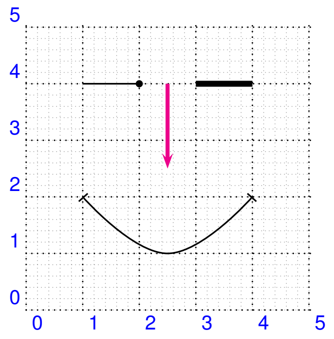


Figure 8: Lines and line endings with pstricks.

The pgf system is designed as a combination of three software layers sitting one on top of each other. The lower layer, called by the author the "system layer," is in charge of low-level tasks related to the production of the final output. In practice, the generic user will never be using the TeX macros provided by the system layer. Next, we have the "basic layer," providing a set of basic commands that allow to produce complex graphics in a much easier manner than by using the system layer directly. However, also this layer of drawing macros is not conceived to be used directly by the generic user. Finally, the pgf exposes a "frontend layer," i. e. a set of commands or a special syntax that makes using the functionalities implemented by basic layer easier. This frontend is what is called "Ti*k*Z"—hence the double possibility to name the package. The name tikz is an acronym of 'tikz ist kein Zeichenprogramm' (German for 'tikz is not a drawing program') and provides commands and environments for specifying and "drawing" graphical objects in a document.

6.1 Command \tikz and environment tikzpicture

Once \usepackage{tikz} is used in the preamble, LaTeX users have two options to produce a diagram with the tikz frontend:

(*a*) The command \tikz as the following example

```
  \tikz \draw (0pt,0pt) -- (20pt,6pt);
```

that yields the line , or

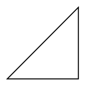
```
\tikz\fill[orange] (1ex,1ex) circle(1ex);
```

that yields the orange circle ●. Observe that the argument passed to \tikz is a string terminated by a semicolon.

(*b*) The environment tikzpicture to embed more elaborated graphic commands, as the following example

```
\begin{tikzpicture}
  \draw (0,0) -- (1,0) -- (1,1) -- cycle;
\end{tikzpicture}
```

that gives the triangle



Both the command \tikz and the environment tikzpicture can be used in running text for simple drawings. For instance, the following draws a 0.4×0.2 crossed rectangle: ⊠.

```
The following draws a $0.4 \times 0.2$
crossed rectangle:
\begin{tikzpicture}
  \draw (0.0,0.0) rectangle (0.4,0.2);
  \draw (0.0,0.0) -- (0.4,0.2);
  \draw (0.0,0.2) -- (0.4,0.0);
\end{tikzpicture}\,.
```

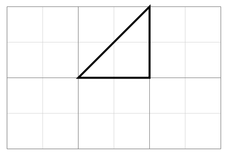
Although there is the chance to use or implement other types of frontend layers to the pgf system, the tikz frontend is by far the most popular.

6.2 Grids

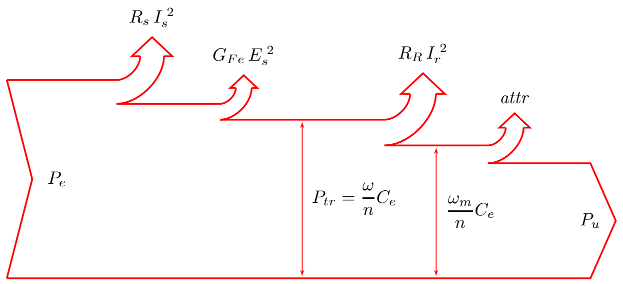
Grids, as in all programmed drawing environments, are the most important support of the trial-and-error process occurring when users develop their pictures. In tikz the simple code

```
\begin{tikzpicture}
\draw[line width=0.1pt,gray!30,step=5mm]
  (0,0) grid (3,2);
\draw[help lines]
  (0,0) grid (3,2);
\draw[thick] (1,1) -- (2,2) -- (2,1)
  -- cycle;
\end{tikzpicture}
```

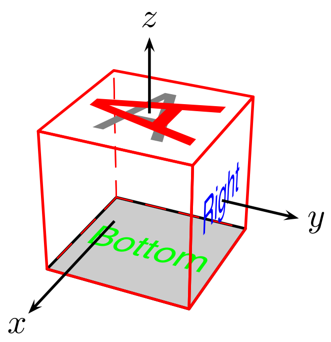
draws a grid and a triangle:



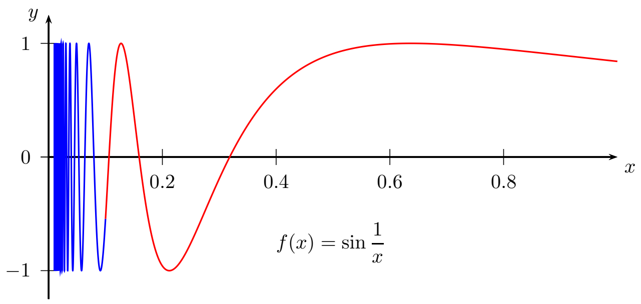
This example demonstrates how to draw a basic 3×2 grid, relative to the origin. The grid consists of two superimposed grids, the coarser of which (*help lines*) is drawn on top of the other. The option gray!30 in the style of the fine grid defines the colour for the grid: one gets it by mixing 30% grey and 70% white. Of course, a grid becomes part of a picture in all cases where a scientific plot has to be represented.



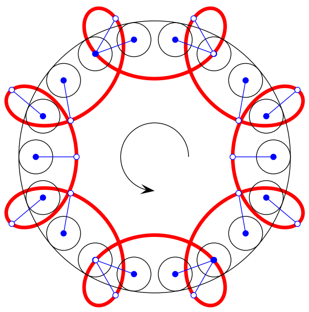
(a) Power balance represented as a Sankey diagram.



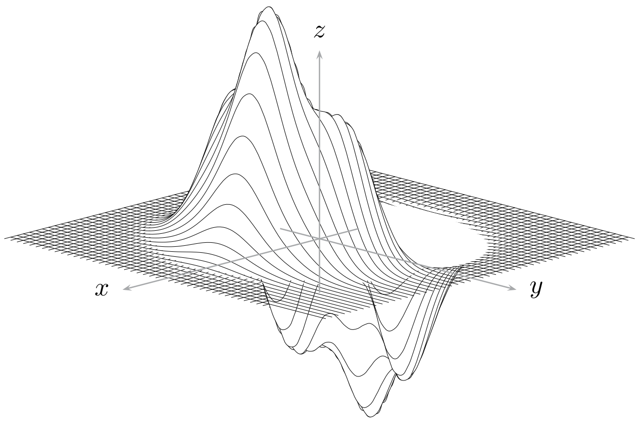
(b) A 3D scene drawn with PSTricks extended with the package pst-solides3d.



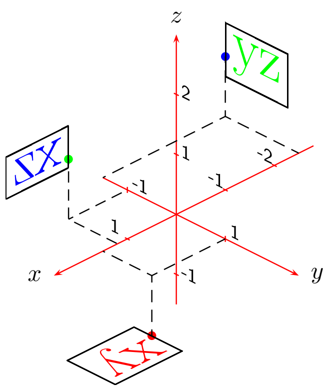
(c) Mathematical function plot obtained with the extension package pst-plot connecting 4000 points.



(d) Construction of a hypocycloid obtained with the extension package pst-plot.



(e) A function $z = f(x, y)$ plotted with pst-plot3d. Reproduced from Goosens *et al.* (2007).



(f) A 3D plot obtained with the extension package pst-plot3d.

Figure 9: Examples of advanced illustrations made with PSTricks.

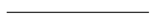
6.3 Paths

Inside a tikzpicture environment everything is drawn by starting a *path* and by *extending* the path. Paths are constructed using the `\path` command. In its basic form, a path is started with a coordinate that becomes the *current* coordinate of the path. Next the path is extended with other coordinates, line segments, *nodes* or other shapes. Line segments may be straight line segments or *cubic spline* segments, which are also known as *cubic splines*. Each line segment extension operation adds a line segment starting at the current coordinate and ending at another coordinate. Path extension operations may update the current coordinate.

The optional argument of the `\path` command is used to control if, and how the path should be drawn. Adding the option `draw` forces the drawing of the path. By default the path is not drawn. A semicolon indicates the end of the path. This code

```
\begin{tikzpicture}
  \path[draw] (0,0) -- (2,0);
  \path (2.5,0) -- (3,0);
\end{tikzpicture}
```

renders as follows:



The first `\path` command in the above tikzpicture draws a line segment from $(0,0)$ to $(2,0)$. The second `\path` command draws an invisible line segment. Both line segments are considered part of the picture, so the picture has a width of 3 cm. The following variant

```
\begin{tikzpicture}
  \path[draw] (0,0) -- (2,0);
  \path[draw, red, thick] (2.5,0) --
    (3,0);
\end{tikzpicture}
```

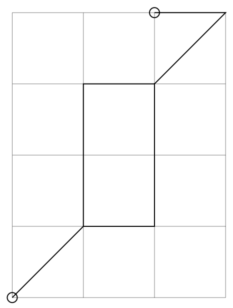
renders both lines:

and modifies their thickness and color.

The command `\draw` is a shorthand for `\path [draw]`. The tikz package has many shorthand notations like this. The following example draws a path that starts at position $(0,0)$.

```
\begin{tikzpicture}
  \draw[help lines] (0,0) grid (3,4);
  \draw (0,0) circle (2pt)
    -- (1,1) rectangle (2,3)
    -- (3,4)
    -- (2,4) circle (2pt);
\end{tikzpicture}
```

First the path is extended by adding a circle. Next the path is extended with a line segment leading to $(1,1)$. Next it is extended with a rectangle, and so on. Except for the circle extension operation, each operation changes the current position of the path. The result is:



It has to be observed that the examples given so far violate every *rule of the maintainability*. For example, what if the rectangle's size were to change, what if its position were to change, what if its colour were to change? Fortunately, tikz provides users a range of commands and techniques for maintaining their diagrams. One of the cornerstones is the ability to label nodes and coordinates and use the labels to construct other nodes and shapes. In addition the packages upports hierarchies. Parent settings may be inherited by descendants in the hierarchy.

6.4 Coordinate labels

Maintaining complex diagrams defined entirely in terms of absolute coordinates is virtually impossible. Fortunately, tikz provides many techniques that help maintain a diagram. In one of these techniques relies on the possibility to you define *coordinate labels* associated to coordinates. The resulting labels can be effectively used instead of the coordinates to build up even the most a complicated diagrams.

User defines a coordinate label by the chosen label name after the `coordinate` keyword. Defining coordinates this way is possible at (almost) any point in a path. Once the label of a coordinate is defined, it can be used as a coordinate. The following, which draws a crossed rectangle (⌧), demonstrates the mechanism.

```
The following, which draws a
crossed rectangle
(\begin{tikzpicture}
  \draw (0.0,0.0) coordinate(lower left)
    -- (0.4,0.2) coordinate(upper right);
  \draw (0.0,0.2) -- (0.4,0.0);
  \draw (lower left) rectangle (upper
    right);
\end{tikzpicture}), demonstrates
the mechanism.
```

A label name may contain spaces.

6.5 Types of path extensions

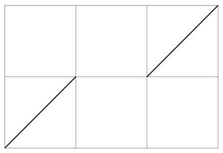
Paths are constructed by extending them. There are several different kinds of path extension operations. The majority of these extension operations modify the current coordinate, but some don't. In the remainder of this section it is therefore assumed that an extension operation modifies the current coordinate unless this is indicated otherwise. For the moment it is assumed that none of the coordinates are relative or incremental coordinates.

6.5.1 The move-to operation

The *move-to* operation is the most intuitive and adds a coordinate to the path, making it the current coordinate. The following example uses three move-to operations. The first move-to operation defines the lower left corner of the grid. The remaining move-to operations define the starts of two line segments. The code

```
\draw[help lines] (0,0) grid (3,2);
\draw (0,0) -- (1,1) % then move-to
(2,1) -- (3,2);
```

yields:

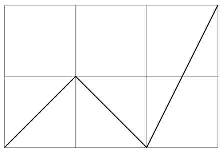


6.5.2 The line-to operation

The *line-to* operation is represented by the `--` directive and adds a straight line segment to the path. The line segment is from the current coordinate and ends in the given coordinate. The example code

```
\draw[help lines] (0,0) grid (3,2);
\draw (0,0) -- (1,1) --
  (2,0) -- (3,2);
```

yields:

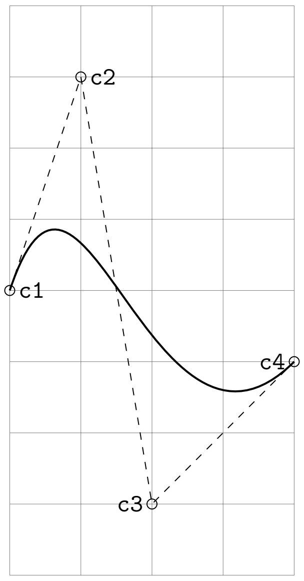


6.5.3 The curve-to operation

The *curve-to* operation is represented by the `..` directive and adds a cubic Bézier spline segment to the path. The start point of the curve is the current point of the path. The end point is last given coordinate, and the control points are all the intermediate coordinates passed to the operation. The following example code

```
% grid
\draw[help lines] (-2,-4) grid (2,4);

% define labels (nodes)
\path (-2, 0) coordinate(c1)
  (-1, 3) coordinate(c2)
  ( 0,-3) coordinate(c3)
  ( 2,-1) coordinate(c4);

% segments connecting nodes
\draw[dashed] (c1) -- (c2) -- (c3) -- (c
    4);

% control points
\draw (c1) circle (2pt)
  (c2) circle (2pt)
  (c3) circle (2pt)
  (c4) circle (2pt)
  (c1);

% the curve
\draw[thick] (c1) .. controls (c2)
  and (c3) .. (c4);

% text labels
\path
(c1) node[anchor=west] {\texttt{c1}}
(c2) node[anchor=west] {\texttt{c2}}
(c3) node[anchor=east] {\texttt{c3}}
(c4) node[anchor=east] {\texttt{c4}};
```

yields:



The above drawing demonstrates the operation. The curve starts at `c1` and ends at `c4`. The control points are given by `c2` and `c3`. The tangent of the spline segment at `c1` is equal to the tangent of the line segment `c1 -- c2`. Likewise, the tangent at `c4` is given by the tangent of the line segment `c3 -- c4`.

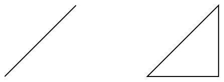
As an alternative, in the this curve-to operation the `and` can be replaced by `..` directives.

6.5.4 The cycle operation

The *cycle* operation closes the current path by adding a straight line segment from the current point to the most recent destination point of a move-to operation. The cycle operation has three applications. First it closes the path, which is required if one wishes to fill the path with a colour. Second, it connects the start and end line segments in the path. Third, it avoids the need to reference the start point of the path. The following example code

```
\draw (0,0) -- (1,1)
  (2,0) -- (3,0) --
  (3,1) -- cycle;
```

yields:

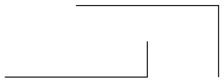


6.5.5 Connecting points with horizontal/vertical lines

This operation is equivalent to two line-to operations connecting the current coordinate and the given coordinate. It may follow the `-|` directive, that is, the first operation adds a horizontal and the second a vertical line segment. The following example code

```
\draw (0.0,0.0) -| (2.0,0.5)
  (1.0,1.0) -| (3.0,0.0);
```

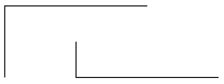
yields:



As an alternative, the operation may follow the `|-` directive. This time, however, the first operation adds a vertical and the second a horizontal line segment, as in the following example

```
\draw (0.0,0.0) |- (2.0,1.0)
  (1.0,0.5) |- (3.0,0.0);
```

resulting in:

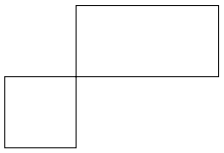


6.5.6 The rectangle operation

The rectangle operation adds a rectangle to the path. The rectangle is constructed by making the current coordinate and the given coordinate, respectively, the lower left and upper right corners of the rectangle. The following example

```
\draw (0,0) rectangle (1,1)
  rectangle (3,2);
```

yields:



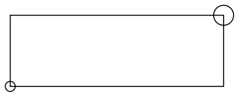
The given coordinate in the first `rectangle` operation here becomes the current coordinate in the next one.

6.5.7 The circle operation

The circle operation adds a circle to the path. The centre of the circle is given by the current coordinate of the path and its radius is the dimension passed as argument. This operation does not change the current coordinate of the path. The following example

```
\draw (0,0) circle (2pt)
  rectangle (3,1)
  circle (4pt);
```

yields:

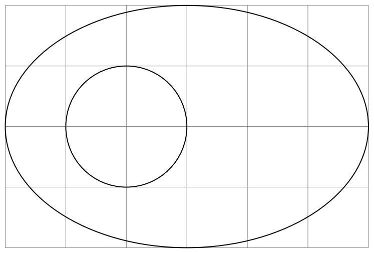


6.5.8 The ellipse operation

The ellipse operation adds an ellipse to the path. The centre of the ellipse is given by the current coordinate of the path and its semi-width and semi-height are passed as arguments. This operation does not change the current coordinate of the path. The following example

```
\begin{tikzpicture}[scale=0.85]
  \draw[help lines] (0,0) grid (6,4);
  \draw (2,2) ellipse (1cm and 1cm)
    (3,2) ellipse (3cm and 2cm);
\end{tikzpicture}
```

yields:

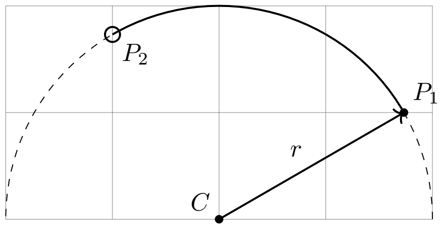


6.5.9 The arc operation

The arc operation adds an arc to the path. The arc starts at the current point, P. The user supplies two angles, α and β, and a radius r. The arc is determined by a circle of radius r. The centre of the circle, C, is determined by the equation $P = C + R \times (\cos\alpha, \sin\alpha)$. The end point of the arc is given by $P = C + R \times (\cos\beta, \sin\beta)$. The arc is drawn in counterclockwise direction from the start point to the end point, which becomes the new current coordinate of the path. The following example illustrates the construction. Only the upper half of the circle is drawn. The resulting arc is drawn with a continuous line. The code

```
\begin{tikzpicture}[scale=1.5]
\draw[help lines] (0,0) grid (4,2);
\draw[dashed] (4,0) coordinate(p0)
  arc (0:180:2cm);
\draw[fill=black] (2,0) coordinate(c)
  circle(1pt)
  node[anchor=south east] {$C$};
\path (p0) arc (0:30:2cm)
  coordinate(p30);
\draw[fill=black] (p30) circle(1pt)
  node[anchor=south west] {$P_1$};
\draw[thick] (p30) arc (30:120:2cm)
  coordinate(p120)
  circle (2pt)
  node[anchor=north west] {$P_2$};
\draw[->,thick] (c) --
  node[anchor=south east] {$r$} (p30);
\end{tikzpicture}
```

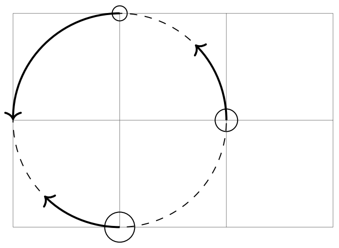
yields:



Further examples of arcs are drawn with the following code

```
\begin{tikzpicture}[scale=1.5]
  \draw[help lines] (0,0) grid (3,2);
  \draw[dashed] (1,1) circle (1cm);
  \draw (1,2) coordinate(a) circle (2pt)
    (2,1) coordinate(b) circle (3pt)
    (1,0) coordinate(c) circle (4pt);
  \draw[->,thick] (a) arc (90:180:1cm);
  \draw[->,thick] (b) arc (0:45:1cm);
  \draw[->,thick] (c) arc (270:225:1cm);
\end{tikzpicture}
```

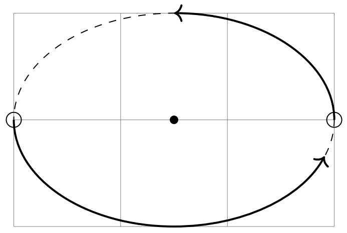
resulting in:



The arc operation can be performed along an ellipse as well. It adds an ellipse segment to the path. The construction of the ellipse segment is similar to the construction of the arc segment. In this case, instead of passing the radius, the user must provide the half width and the half height of the ellipse. The following code

```
\begin{tikzpicture}[scale=1.5]
  \draw[help lines] (0,-1) grid (3,1);
  \draw[dashed] (1.5,0) circle (1.5cm and
     1cm);
  \draw[fill=black] (1.5,0) coordinate(c)
     circle(1pt);
  \draw (3,0) coordinate(a) circle (2pt);
  \draw (0,0) coordinate(b) circle (2pt);
  \draw[->,thick] (a) arc (0:90:1.5cm and
     1cm);
  \draw[->,thick] (b) arc (180:340:1.5cm
    and 1cm);
\end{tikzpicture}
```

demonstrates the elliptical arcs:



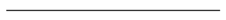
6.6 Actions on paths

Most of the examples shown so far are conceived to emphasize the default path style. This may not always be what users want. For example, one may want to draw a line in a certain colour, change the default line width, fill a shape with a colour, and so on. In tikz terminology this is achieved with *path actions*, which are operations acting on an existing path. The user first constructs the path and then apply the action. At the basic level the command `\draw` is defined in terms of an action on a path: the action results in the path being drawn. As pointed out before `\draw` is a shorthand for `\path[draw]`.

The following are some other shorthand commands that are defined in terms of path actions inside the tikzpicture environment.

`\draw` Shorthand for `\path[draw]`.
Example:

```
\draw (0,0) -- (3,0);
```



`\fill` Shorthand for `\path[fill]`.
Example:

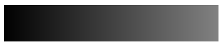
```
\fill[gray!30] (0,0) rectangle (3,0.5);
```

`\filldraw` Shorthand for `\path[filldraw]`.
Example:

```
\filldraw[fill=gray!30,draw=black,thick]
  (0,0) rectangle (3,0.5);
```

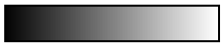
`\shade` Shorthand for `\path[shade]`.
Example:

```
\shade[left color=black,right color=gray]
  (0,0) rectangle (3,0.5);
```



`\shadedraw` Shorthand for `\path[shadedraw]`.
Example:

```
\shadedraw[left color=black,right color=
    white,thick]
  (0,0) rectangle (3,0.5);
```

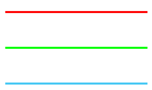


6.7 Colour

The tikz package knows several colours. Some colours are inherited from the xcolor package.[12] There are several techniques to define a new name for a colour. To learn more, the reader is referred to the documentation of xcolor.

Some path actions also let users define a colour. For example, one may draw a path with the given colour. There are different ways to control the colour. The option `color` determines the colour for drawing and filling, and the colour of text in nodes. (Nodes are explained later on in this section.) One may set the colour of the whole tikzpicture or set the colour of a given path action. Setting the colour of the whole picture is done by passing a `color` option to the environment. Setting the colour of a path action is done by passing the option to the `\path` command (or derived shorthand commands). The following is an example that draws three lines: one in red, one in green, and one in 60% cyan and 40% white.

```
\begin{tikzpicture}[thick,color=red]
  \draw (0,2) -- (2,2);
  \draw[color=green] (0,1.5) -- (2,1.5);
  \draw[color=cyan!60]
    (0,1) -- (2,1);
\end{tikzpicture}
```



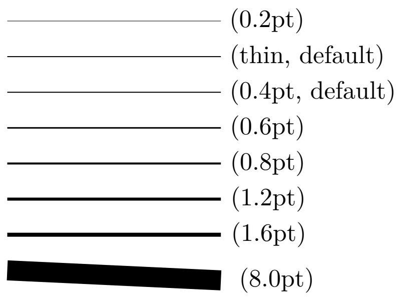
It is usually possible to omit the `color=` part when one specifies colour options.

6.8 Line width

In tikz there are several path actions affecting the line style, including the style that determines the line width, the line cap, and the line join. The following code provide some examples.

12. `http://texdoc.net/texmf-dist/doc/latex/xcolor/xcolor.pdf`

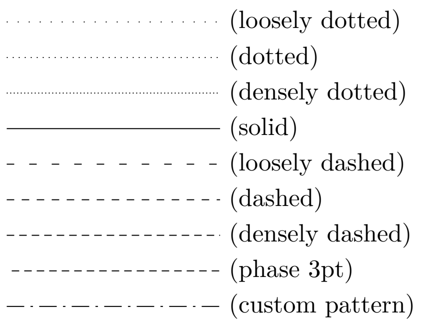
```
\draw[very thin] (0,3.5) -- (3,3.5)
  node[anchor=west] {(0.2pt)};
\draw[thin] (0,3) -- (3,3)
  node[anchor=west] {(thin, default)};
\draw[line width=0.4pt] (0,2.5) --
    (3,2.5)
  node[anchor=west] {(0.4pt, default)};
\draw[semithick] (0,2) -- (3,2)
  node[anchor=west] {(0.6pt)};
\draw[thick] (0,1.5) -- (3,1.5)
  node[anchor=west] {(0.8pt)};
\draw[very thick] (0,1.0) -- (3,1.0)
  node[anchor=west] {(1.2pt)};
\draw[ultra thick] (0,0.5) -- (3,0.5)
  node[anchor=west] {(1.6pt)};
\draw[line width=8pt] (0,0) -- (3,-4pt)
  node[anchor=west] {(8.0pt)};
```



6.9 Dash patterns

The drawing of lines inb tikz also depends on the *dash pattern* and *dash phase* settings. The dash pattern determines a basic pattern for the line that is repeated cyclicly. The dash phase shifts the dash pattern. By default the dash pattern is `solid`. The following shows the relevant path actions that affect dash patterns.

```
\draw[loosely dotted] (0,3.5) -- (3,3.5)
  node[anchor=west] {(loosely dotted)};
\draw[dotted] (0,3) -- (3,3)
  node[anchor=west] {(dotted)};
\draw[densely dotted] (0,2.5) -- (3,2.5)
  node[anchor=west] {(densely dotted)};
\draw[solid] (0,2.0) -- (3,2.0)
  node[anchor=west] {(solid)};
\draw[loosely dashed] (0,1.5) -- (3,1.5)
  node[anchor=west] {(loosely dashed)};
\draw[dashed] (0,1.0) -- (3,1.0)
  node[anchor=west] {(dashed)};
\draw[densely dashed] (0,0.5) -- (3,0.5)
  node[anchor=west] {(densely dashed)};
\draw[densely dashed,
  dash phase=3pt] (0,0.0) -- (3,0.0)
  node[anchor=west] {(phase 3pt)};
\draw[dash pattern=on 7pt off 2.5pt
  on 1pt off 2.5pt] (0,-0.5) -- (3,-0.5)
  node[anchor=west] {(custom pattern)};
```



6.10 Predefined styles

Hard-coding a line width or a dash pattern command is not always a good idea. It is usually better to define a style for a certain line width, for a dash style, or a combination of the two. The advantages of doing this are that you only have to define the style once and can use it several times. Using styles gives you a consistent appearance for the resulting lines, and if you want to make a global change to the style then you only have to make one change in your LaTeX file. Later on in this section it will be explained how users can define their own styles.

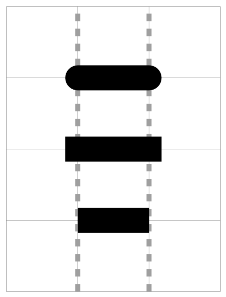
Several previous examples given so far make use of some predefined line width and dash pattern styles. The default line width is `thin`. The default dash pattern is `solid`.

6.11 Line caps and joins

The drawing of a path depends on several parameters. The *line cap* determines how lines start and end. The *line join* determines how line segments are joined.

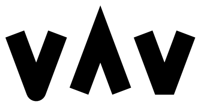
The following examples demonstrates different line cap types.

```
\begin{tikzpicture}[line width=8pt]
  \draw[help lines] (0,0) grid (3,4);
  \draw[line width=2pt,dashed,gray!75]
    (1,0) -- (1,4) (2,0) -- (2,4);
  \draw[line cap=round] (1,3) -- (2,3);
  \draw[line cap=rect] (1,2) -- (2,2);
  \draw[line cap=butt] (1,1) -- (2,1);
\end{tikzpicture}
```



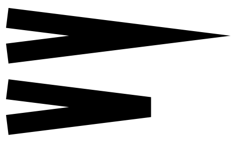
The following examples demonstrates different line join types.

```
\begin{tikzpicture}[line width=8pt]
  \draw[line join=round]
    (0.0,.8)--(0.3,.0)--(0.6,.8);
  \draw[line join=miter]
    (0.9,.0)--(1.2,.8)--(1.5,.0);
  \draw[line join=bevel]
    (1.8,.8)--(2.1,.0)--(2.4,.8);
\end{tikzpicture}
```



To avoid sharp-angled miter joins that protrude too far beyond the joining point, `tikz` provides the control option `miter limit`. It poses a limit on how far the miter join may protrude the joining point. If the join protrudes beyond the limit then the join style is changed to `bevel`. The limit is equal to a fraction of the line width. An example of use is the following.

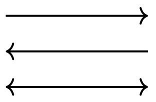
```
\begin{tikzpicture}
  [line width=8pt,line join=miter]
  \draw (0,0) -- (0.25,2) -- (0.5,0);
  \draw[miter limit=8]
    (1,0) -- (1.25,2) -- (1.5,0);
\end{tikzpicture}
```



6.12 Arrows

Arrows are also drawn using path actions. The following example demonstrates some common ways to how to draw them.

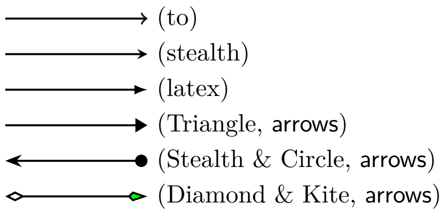
```
\begin{tikzpicture}[thick]
  \draw[->] (0,1.0) -- (2,1.0);
  \draw[<-] (0,0.5) -- (2,0.5);
  \draw[<->] (0,0.0) -- (2,0.0);
\end{tikzpicture}
```



Several arrow head styles are available besides the default one shown above. Some of the styles are provided by the tikz extension library arrows.meta. The following code demonstrates a small selection of arrow types.

```
% in preamble
\usepackage{tikz}
\usetikzlibrary{arrows.meta}

\begin{tikzpicture}[thick]
  \draw[>=to,->] (0,1.0) -- (2,1.0)
```

```
    node[anchor=west] {(to)};
  \draw[>=stealth,->] (0,0.5) -- (2,0.5)
    node[anchor=west] {(stealth)};
  \draw[>=latex,->] (0,0.0) -- (2,0.0)
    node[anchor=west] {(latex)};
 \draw[>=Triangle,->] (0,-0.5) --
    (2,-0.5)
    node[anchor=west] {(Triangle, arrows)
    };
 \draw[Stealth-Circle] (0,-1.0) --
    (2,-1.0)
    node[anchor=west] {(Stealth \& Circle
    , arrows)};
 \draw[{Diamond[open]}-{Kite[fill=green
    ]}] (0,-1.5) -- (2,-1.5)
    node[anchor=west] {(Diamond \& Kite,
    arrows)};
```



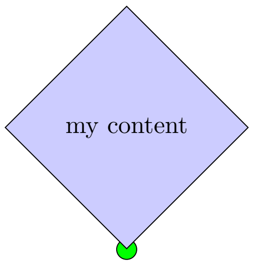
6.13 Nodes and node labels

Diagrams with lines only are rare. Usually, they also contain text, math, or both. Fortunately, tikz has a mechanism for adding text, math, and other material to paths. This is done with the *node path extension operation.*

The node path extension operation allows to place a given content (delimited by curly brackets) at the current position in the path using some given options, and associates a label to the node. Each node added to a path has an *outer shape*. The outer shape is only drawn if draw is part of the options. The default node shape is a rectangle but other shapes are also defined.

The following example draws a circle at position $(1, 0)$ and at the same time places a diamond-shaped node at the current point on the path. The node contains the words 'my content' and shape boundaries have a distance of 10 pt from the text. The fill operation is applied both to the circle and to the diamond, giving them, respectively, a green and a light blue background.

```
% in preamble
\usepackage{tikz}
\usetikzlibrary{shapes.geometric}

\draw (0,1) % current position.
  [fill=green] % options for circle
  circle (4pt) % draw shape circle
  node[anchor=south, % node options
    diamond,
    fill=blue!20,
    inner sep=10pt,draw]
    (c) % node label
    {my content}; % content
```



When adding a node to a path, it is not mandatory to have a label and a set of options. In fact, the simple code

```
\draw (0,1) circle (8pt) node {Circle};
```

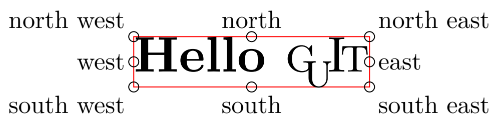
just draws a circle and puts the word 'Circle' on top of it:

Observe that the default behaviour puts the word's center at the current coordinate.

When a node receives a label, ⟨ *label* ⟩, then usually the additional labels ⟨ *label* ⟩.center, ⟨ *label* ⟩.north, ⟨ *label* ⟩.north east, . . . , and ⟨ *label* ⟩.north west are also defined. The positions of these labels correspond to their names, so ⟨ *label* ⟩.north is to the north of the node having label ⟨ *label* ⟩. This holds for the most common node shapes. Next example involves all these auxiliary labels, except for ⟨ *label* ⟩.center. The option anchor in the example is a way to override the node's default insertion point.

```
\begin{tikzpicture}
  \draw (0,0)
    node (hello)
      [scale=2.0,
      inner sep=0pt,outer sep=0pt,
      draw=red]
      {\fbox{\textbf{Hello \GuIT}}};
  \draw (hello.north east) circle (2pt)
    node[anchor=south west] {north east};
  \draw (hello.north) circle (2pt)
    node[anchor=south] {north};
  \draw (hello.north west) circle (2pt)
    node[anchor=south east] {north west};
  \draw (hello.west) circle (2pt)
    node[anchor=east] {west};
  \draw (hello.south west) circle (2pt)
    node[anchor=north east] {south west};
  \draw (hello.south) circle (2pt)
    node[anchor=north] {south};
  \draw (hello.south east) circle (2pt)
    node[anchor=north west] {south east};
  \draw (hello.east) circle (2pt)
```

```
    node[anchor=west] {east};
\end{tikzpicture}
```



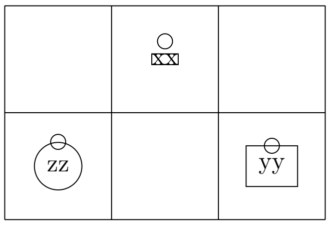
Observe in the above example that the options `inner sep` and `outer sep` are both set to 0 pt. This means that the default rectangular shape containing the words 'Hello GuIT' is the actual bounding box of the text and that the anchor points are precisely located at the boundary of the box. A nonzero `inner sep` (*inner separation*) makes the shape larger than the actual bounding box. A nonzero `outer sep` (*outer separation*) offsets the anchor points outwards of the given dimension.

6.14 Predefined node shapes

Nodes have a shape/style and content. The default node shape is rectangular but `tikz` also predefines the shapes `coordinate`, `rectangle`, `circle`, and `ellipse`. The option `shape=`⟨*shape*⟩ determines the node shape.

The following example shows some of the different node shape options and low-level control.

```
\begin{tikzpicture}[scale=1.5]
  \draw (0,0) grid (3,2);
  \draw (1.5,1.5)
    node (a)
      [draw,inner sep=0pt,outer sep=5pt]
      {xx};
  \draw (2.5,0.5)
    node (b)
      [draw,inner sep=5pt,outer sep=0pt]
      {yy};
  \draw (0.5,0.5)
    node(c)
    [draw,shape=circle] {zz};
  \draw (a.north) circle (2pt);
  \draw (b.north) circle (2pt);
  \draw (c.north) circle (2pt);
\end{tikzpicture}
```

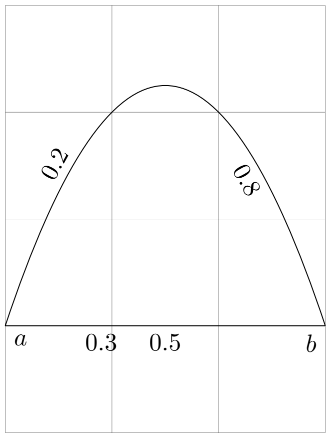


The difference in the inner separations of the rectangular nodes manifests itself in different sizes for the rectangular shapes. Differences in the outer separations result in different distances of labels such as `north`. The higher the outer separation of a node, the further its north label is away from its rectangular shape.

6.15 Node placement

Several node options exist that permit a low-level control of all graphical aspects. Here we show some of these node options with an example.

```
\begin{tikzpicture}[scale=1.5]
  \draw[help lines] (0,0) grid (3,4);
  \draw (0,1) coordinate(a)
    node[anchor=north west] {$a$}
    -- (3,1) coordinate(b)
    node[anchor=north east] {$b$}
    node[pos=0.3,anchor=north] {$0.3$}
    node[pos=0.5,anchor=north] {$0.5$}
    (a) .. controls (1,4) and (2,4) .. (b
    )
    node[pos=0.2,sloped,anchor=south] {$
    0.2$}
    node[pos=0.8,sloped,anchor=north] {$
    0.8$};
\end{tikzpicture}
```

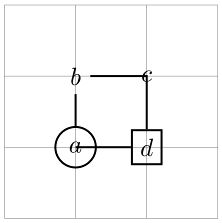


Notice that several nodes can be placed with `pos` options for the same path segment.

6.16 Connecting nodes

The `tikz` package is well-behaved. It will not cross lines unless user says so. This includes the crossing of borderlines of node shapes. For example, let us assume the user created two nodes. One of them is a circle, which is labelled `c`, and the other is a rectangle, which is labelled `r`. When user draws a line using the command `\draw (c) -- (r);` then the resulting line segment will not join the centres of the two nodes. The actual line segment will be shorter because the line segment starts at the circle shape and ends at the rectangle shape. In most cases this is the desired behaviour. If one needs a line between the centres then `.center` notation must be used. The following code provides an example.

```
\begin{tikzpicture}[thick]
  \draw[help lines] (0,0) grid (3,3);
  \path (1,1) node(a)[draw,shape=circle]
    {$a$};
  \path (1,2) node(b)[shape=rectangle] {$
    b$};
  \path (2,2) node(c)[shape=circle] {$c$
    };
  \path (2,1) node(d)[draw,shape=
    rectangle] {$d$};
  \draw (a) -- (b) -- (c.center) -- (d)
    -- (a.center);
\end{tikzpicture}
```



6.17 Coordinate systems

Specifying coordinates is the key to effective, efficient, and maintainable picture creation. Coordinates may be specified in different ways each coming with its own specific coordinate system. Within a coordinate system you specify coordinates using *explicit* or *implicit* notation.

Explicit — Explicit coordinate specifications are verbose. To specify a coordinate, users write `(⟨system⟩ cs: ⟨coord⟩)`, where ⟨*system*⟩ is the name of the coordinate system and where ⟨*coord*⟩ is a coordinate whose syntax depends on ⟨*system*⟩. For example, to specify the point having x-coordinate ⟨*x*⟩ and y-coordinate ⟨*y*⟩ in the canvas coordinate system one writes `(canvas cs:x=⟨x⟩, y=⟨y⟩)`.

Implicit — Implicit coordinates specifications are shorter than explicit coordinate specifications. Users specify coordinates using some coordinate system-specific notation inside parentheses. Most examples so far have used the implicit notation for the canvas coordinate system.

Canvas coordinate system — The most widely used coordinate system is the `canvas` coordinate system. It defines coordinates in terms of a horizontal and a vertical offset relative to the origin. The implicit notation `(⟨x⟩, ⟨y⟩)` is the point with x-coordinate ⟨*x*⟩ and y-coordinate ⟨*y*⟩.

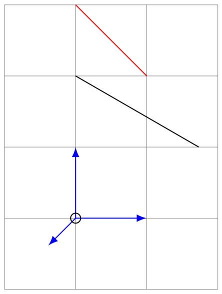
Xyz coordinate system — The `xyz` coordinate system defines coordinates in terms of a linear combination of an x-, a y-, and a z-vector. By default, the x-vector points 1 cm to the right, the y-vector points 1 cm up, and the z-vector points to $(-\sqrt{2}/2, -\sqrt{2}/2)$. However, these default settings can be changed. The implicit notation `(⟨x⟩, ⟨y⟩, ⟨z⟩)` is used to define the point at ⟨*x*⟩ times the x-vector plus ⟨*y*⟩ times the y-vector plus ⟨*z*⟩ times the z-vector.

Polar coordinate system — The canvas polar coordinate system defines coordinates in terms of an angle and a radius. The implicit notation $(\alpha{:}r)$ corresponds to the point $(r\cos\alpha, r\sin\alpha)$. Angles in this coordinate system, as all angles in `tikz`, should be supplied in degrees.

Node coordinate system — The node coordinate system defines coordinates in terms of a label of a node or coordinate. The implicit notation `(⟨label⟩)` is the position of the node or coordinate that was given the label ⟨*label*⟩.

The following example demonstrates the previous four coordinate systems in action. The optional argument of the `tikzpicture` sets the arrow head style to the predefined style named `latex`.

```
\begin{tikzpicture}[>=latex]
  \draw[help lines] (-1,-1) grid (2,3);
  \draw[red] (canvas cs:x=1cm,y=2cm) --
    (0,3);
  \draw[blue,->] (0,0) -- (xyz cs:x=1,y
    =0,z=0);
  \draw[blue,->] (0,0) -- (0,1,0);
  \draw[blue,->] (0,0) -- (0,0,1);
  \draw (canvas polar cs:radius=2cm,angle
    =30)
    -- (90:2);
  \path (0,0) coordinate (origin);
  \draw (origin) circle (2pt);
\end{tikzpicture}
```



Users can freely mix the coordinate systems. For example `\draw (0,0) -- (0,1);` and `\draw (0,0) -- (90:1);` are equivalent.

6.18 Relative and incremental coordinates

Specifying diagrams in terms of absolute coordinates is cumbersome and prone to errors. What is worse, diagrams defined in terms of absolute coordinates are difficult to maintain. For example, changing the position of an n-agon that is defined in terms of absolute coordinates requires changing n coordinates. Fortunately, `tikz` provides a coordinate computation mechanism based on previously defined coordinates. Used intelligently, this reduces the maintenance costs of diagrams.

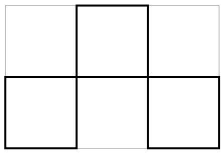
Relative and *incremental* coordinates are computed from the current coordinate in a path. The first doesn't change the current coordinate whereas the second does change it.

Relative coordinate — A relative coordinate constructs a new coordinate at an offset from the current coordinate without changing the current coordinate. The notation +⟨ *offset* ⟩ specifies the relative coordinate that is located at offset ⟨ *offset* ⟩ from the current coordinate.

Incremental coordinate — An incremental coordinate also constructs a new coordinate at an offset from the current coordinate. This time, however, the new coordinate becomes the current coordinate. One uses the implicit notation ++⟨ *offset* ⟩ for incremental coordinates.

The following example draws three squares. The first square is drawn with absolute coordinates, the second with relative coordinates, and the last with incremental coordinates.

```
\begin{tikzpicture}[thick]
  \draw[help lines] (0,0) grid +(3,2);
  \draw (0,0) -- (+1,0) --
    (1,1) -- (+0,1) -- cycle;
  \draw (1,1) -- +(+1,0) --
    +(1,1) -- +(+0,1) -- cycle;
  \draw (2,0) -- ++(+1,0) --
    ++(0,1) -- ++(-1,0) -- cycle;
\end{tikzpicture}
```



Clearly, the relative and incremental coordinates should be preferred because they improve the maintenance of the picture. For example, moving the first square requires changing four coordinates, whereas moving the second or third square requires changing only the start coordinate. The relative coordinate in the grid also improves the maintainability.

6.18.1 Complex coordinate calculations

Finally, tikz offers complex coordinate calculations. However, these calculations are only available if the tikz extension library calc is loaded in the preamble.

Generally, coordinate computations based on previously defined points are enclosed in the special syntax

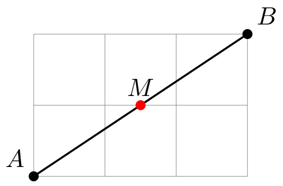
($ ⟨*coordinate modifiers*⟩ $)

where *coordinate modifiers* are a set of possible constructs that usually manipulate two existing points to produce a new one.

The following examples present several coordinate computations involving *distance modifiers*.

The code

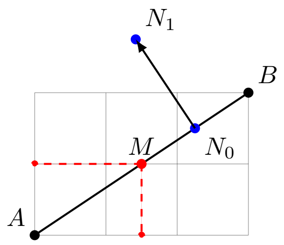
```
% in preamble
\usetikzlibrary{calc}

\begin{tikzpicture}[thick]
  \draw[help lines] (0,0) grid +(3,2);
  \path (0,0) coordinate (A)
    [fill]circle (2pt) node[anchor=south
   east] {$A$};
  \path +(3,2) coordinate (B)
    [fill]circle (2pt) node[anchor=south
   west] {$B$};
  \draw (A) -- (B);
  \path ($(A)!0.5!(B)$) coordinate (M)
    [fill=red]circle (2pt) node[anchor=
   south] {$M$};
\end{tikzpicture}
```



calculates the halfway point (M) between two coordinates, (A) and (B), using the special syntax ($(A)!0.5!(B)$).

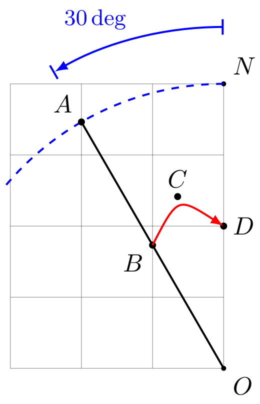
The last example can be elaborated further by extracting the x- and y-coordinate of (M) using the special path operation \let, see the package documentation (Tantau, 2016) for a detailed explanation of this handy feature. Finally, a point N_0 is taken on segment AB at $\frac{3}{4}|AB|$ from A and a point N_1 is calculated such that segment N_0N_1 is normal to AB and $|N_0N_1| = 1.5\,\mathrm{cm}$.

```
% in preamble
\usetikzlibrary{calc}

\begin{tikzpicture}[thick,>=latex]
  \draw[help lines] (0,0) grid +(3,2);
  \path (0,0) coordinate (A)
    [fill]circle (2pt) node[anchor=south
   east] {$A$};
  \path +(3,2) coordinate (B)
    [fill]circle (2pt) node[anchor=south
   west] {$B$};
  \draw (A) -- (B);
  \path ($(A)!0.5!(B)$) coordinate (M)
    [fill=red]circle (2pt) node[anchor=
   south] {$M$};
  % extract M.x
  \draw[dashed,red]
    % point register <-- M coordinates
    let \p{M}=(M) in
    (M) -- (\x{M},0) % extract M.x
      [fill=red]circle(1pt)
    (M) -- (0,\y{M}) % extract M.y
      [fill=red]circle(1pt);
  % point N0
  \path ($(A)!0.75!(B)$) coordinate (N0)
    [fill=blue]circle (2pt) node[anchor=
   north west] {$N_0$};
  % point N1: N0--N1 normal to N0--B
  \path ($(N0)!1.5cm!90:(B)$) coordinate
    (N1);
  \path (N1) [fill=blue]circle (2pt) node
    [anchor=south west] {$N_1$};
  \draw[->] (N0) -- (N1);
\end{tikzpicture}
```



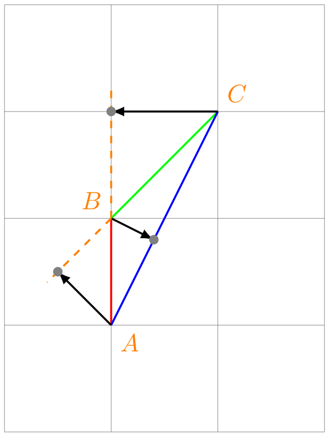
The following example demonstrates more computations with *partway* and distance modifiers.

```
% in preamble
\usetikzlibrary{calc}

\begin{tikzpicture}[thick,>=latex]
  \draw[help lines] (-3,0) grid +(3,4);
  % define origin O
  \path (0,0) coordinate (O)
    [fill] circle (1pt)
      node[anchor=north west] {$O$};
  % define point N
  \path (0,4) coordinate (N)
    [fill] circle (1pt)
      node[anchor=south west] {$N$};
  % point computation helpers
  \draw[dashed,blue] (0,4) arc (90:140:4)
    ;
  \draw[|->|,blue] (0,4.8) arc
    (90:120:4.8)
    node[pos=0.5,anchor=south east]
      {\small 30\,deg};
  % compute A, B, C, D
  \draw (O)
    % connect the origin
    -- % with next point
    % define A:
    %  on segment ON,
    %  at N,
    %  then rotate of 30deg about O
    ($(O)!1.0!30:(N)$)
      coordinate (A)
      [fill] circle (1pt)
      node[anchor=south east] {$A$}
    % define B:
    %  on segment OA,
    %  at 2cm from O
    ($(O)!2.0cm! (A)$)
      coordinate (B)
      [fill] circle (1pt)
      node[anchor=north east] {$B$}
    % define C:
    %  on segment OA,
    %  at 2.5cm from O
    %  then rotate of -15deg about O
    ($(O)!2.5cm!-15:(A)$)
      coordinate (C)
      [fill] circle (1pt)
      node[anchor=south] {$C$}
    % define D:
    %  on segment OA,
    %  at 2cm from O,
    %  then rotate of -30deg about O
    ($(O)!2cm!-30:(A)$)
      coordinate (D)
      [fill] circle (1pt)
      node[anchor=west] {$D$};
  % draw a bezier
  \draw[-latex,red]
    (B) .. controls (C) .. (D);
\end{tikzpicture}
```



Finally, next example demonstrate coordinate computations with projection modifiers.

```
% in preamble
\usetikzlibrary{calc}

\begin{tikzpicture}[scale=1.5,
  thick,>=latex,line join=round]
  \draw[help lines] (0,0) grid +(3,4);
  \draw[red] (1,1) coordinate (A)
    node[anchor=north west,orange] {$A$}
    % segment AB
    -- (1,2) coordinate (B)
      node[anchor=south east,orange] {$B$
    };
  \draw[green] (B)
    % segment BC
    -- (2,3) coordinate (C)
      node[anchor=south west,orange] {$C$
    };
  % segment CA
  \draw[blue] (C) -- (A);
  % connect points with their projections
  % on opposite segments
  % B on segment AC
  \draw[->] (B) -- ($(A)!(B)!(C)$)
    coordinate (Bp);
  \draw[fill,gray] (Bp) circle (1pt);
  % C on segment BA
  \draw[->] (C) --
    ($(B)!(C)!(A)$) coordinate (Cp);
  % construction helpers
  \draw[dashed,orange] (B)
    -- ($(B)!1.2!(Cp)$);
  \draw[fill,gray] (Cp) circle (1pt);
  % A on segment CB
  \draw[->] (A) --
    ($(C)!(A)!(B)$) coordinate (Ap);
  % construction helpers
  \draw[dashed,orange] (B)
    -- ($(B)!1.2!(Ap)$);
  \draw[fill,gray] (Ap) circle (1pt);
\end{tikzpicture}
```



In the last drawing three vertices of a triangle ABC are first defined, annotated, and connected. Successively, some helper lines and dots are represented as aids to the reader. Finally, each vertex V is projected onto the opposite segment S_1S_2 according to the sintax `($(`S_1`)!(`V`)!(`S_2`)$)`.

6.19 What else?

Making graphics with pgf is a huge subject. Therefore, this section does not claim to serve as an exhaustive tutorial on programmed illustrations with tikz. There are several aspects that have been left out of this presentation to save space and remain on the essential commands and options.

For a comprehensive explanation of tikz styles, scopes, options, advanced path operation, customization possibilities, and extension libraries the interested readers are referred to the excellent user guide (Tantau, 2016) and to the book *LaTeX and Friends* (van Dongen, 2012).

All tikz examples given in previous subsections are viewable on Overleaf website.[13] These are provided mainly to facilitate new users in their further explorations.

13. `https://www.overleaf.com/read/mgskyfdpttzt`

6.20 Advanced examples

In this final subsection on tikz three advanced examples are reported to demonstrate the possibilities of the package. The examples are adapted from the website `http://texample.net`, which exhibits a gallery containing a large number of high quality illustrations and graphics made with tikz and pgf.

The LaTeX code reported in Figure 10 produces the example of Figure 11, a nice block diagram obtained with tikz. The diagram is constructed with the aid of the tikz library positioning, which facilitates the relative positioning of the various nodes on the canvas. The code also demonstrates the definition of custom node styles.

Figure 12 shows the geometry of hydrogen and oxygen atoms in the water molecule. The example demonstrates the use of shading options to obtain a three-dimensional effect.

Figure 13 demonstrates the way a submatrix can be highlighted within a mathematical formula. In this example some advanced features of pgf are used in order to produce rectangle nodes that fit the desired areas. The drawing requires a double compilation.

7 LaTeX-aware graphic software

The approach to graphic work production discussed in this section relies on available visual tools and is very different from the ‘programmed graphics’ approach presented in the previous section.

There are many ‘LaTeX-aware’ computer programs, with sophisticated graphical user interfaces (GUI), not included in standard TeX distributions, which are capable of producing professional-quality graphics. The following is a list of the most popular ones:

- Xfig,[14] is one of the first visual tools of this type, an X Window drawing software available for Unix and Linux that saves graphics in its own format (`.fig` files), but exports to many other formats, including Encapsulated PostScript (EPS). An improved version named WinFig[15] is available for MS Windows. This software has been traditionally used in conjunction with the LaTeX package psfrag that can remove labels and other text from `.eps` graphics and replace them with LaTeX labels. Although this approach still works perfectly, it has become somewhat obsolete with respect to other recently introduced workflows.
- Ipe,[16] is a powerful vector graphics editor, with several snapping modes that make it especially suitable for a variety of technical illustrations. The application saves graphics in its own `.ipe` file format, but outputs PDF and EPS for inclusion in LaTeX documents. Ipe uses LaTeX to typeset text,

14. `http://xfig.org`
15. `http://winfig.com`
16. `http://ipe7.sourceforge.net`

```
%in preamble
\usepackage{relsize,calc,paralist,tikz}
\usetikzlibrary{calc,arrows,decorations.pathmorphing,backgrounds,fit,positioning,shapes.symbols,chains}

\definecolor{mydarkgreen}{rgb}{0.03,0.47,0.03} \definecolor{mydarkblue}{rgb}{0.07,0.08,0.4}
\definecolor{mylightblue}{rgb}{.8, .8, 1} \definecolor{mylightgray}{rgb}{0.95,0.95,0.95}
\definecolor{mydarkgray}{rgb}{0.35,0.35,0.35} \definecolor{myblue}{rgb}{.4,.4,1}

\tikzstyle{line} = [draw,>=latex', shorten >=0pt, shorten <=0pt,line width=2pt]

% in document ------------------------------------------
\begin{tikzpicture}
   [node distance = 1cm, auto, font=\footnotesize,
   % STYLES
   every node/.style={node distance=3cm},
   % The comment style is used to describe the characteristics of each force
   comment/.style={
      rectangle, inner sep= 2pt, text width=5cm, node distance=0.25cm,
      font=\relsize{0}\sffamily
   },
   % The discipline style is used to draw the disciplines' name
   discipline/.style={
      rectangle, draw, fill=black!10, inner sep=5pt, text width=4cm, text badly centered,
      minimum height=1.2cm, font=\relsize{0}\bfseries
   },
   % The topic style
   topic/.style={
      rectangle, draw, top color=white, bottom color=mylightblue,very thick,
      inner sep=5pt, text width=3.5cm, text badly centered,
      minimum height=1.7cm, font=\relsize{0}\bfseries
   },
   % the mycircled node type
   mycircled/.style={
      circle, draw, fill=black!10, inner sep=2pt,font=\relsize{0}\bfseries
   }
]% end of tikzpicture global options

% Draw forces
\node [discipline] (FD) {\relsize{1}Flight Dynamics};
\node [mycircled, left of=FD] (plus) {$\boldsymbol{+}$};

\node [discipline, left of=plus] (MechElasStru) {Mechanics of Elastic Structures};
\node [discipline, above of=MechElasStru,yshift=-1cm] (MechRigiBodi) {Mechanics of Rigid Bodies};
\node [discipline, above of=MechRigiBodi,yshift=-1cm] (Aero) {Aerodynamics};
\node [discipline, below of=MechElasStru,yshift=+1cm] (HumaPiloDyna) {Human Pilot Dynamics};
\node [discipline, below of=HumaPiloDyna,yshift=+1cm] (ApplMathMachComp) {Applied Mathematics Machine Computation};

\node [discipline, right of=FD,xshift=3cm] (VehiOper) {Vehicle Operation};
\node [discipline, above of=VehiOper,yshift=-1cm] (VehiDesi) {Vehicle Design};
\node [discipline, below of=VehiOper,yshift=+1cm] (PiloTrai) {Pilot Training};

\node [topic, below of=FD,xshift=-5.98cm, yshift=-3.8cm] (P) {Performance (trajectory, maneuverability)};
\node [topic, right of=P,xshift=1cm] (SC) {Stability \& Control (handling qualities, airloads)};
\node [topic, right of=SC,xshift=1cm] (AE) {Aeroelasticity (control, structural integrity)};
\node [topic, right of=AE,xshift=1cm] (NG) {Navigation and Guidance};

% Comments
\node [comment, above=0.25 of FD] (comment-FD) {
  \begin{compactitem}% needs paralist
    \item Flight Simulator mathematical model \item Aircraft representation
  \end{compactitem}
};

% Draw the links between nodes
\path[line,->] (plus) edge (FD);
\path[line,->] (MechElasStru) edge (plus);
\path[line,->] (MechRigiBodi.east) -- ++(0.3cm,0) -- (plus.120);
\path[line,->] (Aero) -| (plus);
\path[line,->] (HumaPiloDyna.east) -- ++(0.3cm,0) -- (plus.240);
\path[line,->] (ApplMathMachComp) -| (plus);

\path[line,->] (FD) -- (VehiOper);
\path[line,->] (FD.east) ++(1cm,0) |-  (VehiDesi);
\path[line,->] (FD.east) ++(1cm,0) |-  (PiloTrai);

\path[line,<-] (P.north) -- ++(0,0.6cm) -|  (FD);
\path[line,<-] (SC.north) -- ++(0,0.6cm) -|  (FD);
\path[line,<-] (AE.north) -- ++(0,0.6cm) -|  (FD);
\path[line,<-] (NG.north) -- ++(0,0.6cm) -|  (FD);
\end{tikzpicture}
```

Figure 10: Source code of diagram reported in Figure 11.

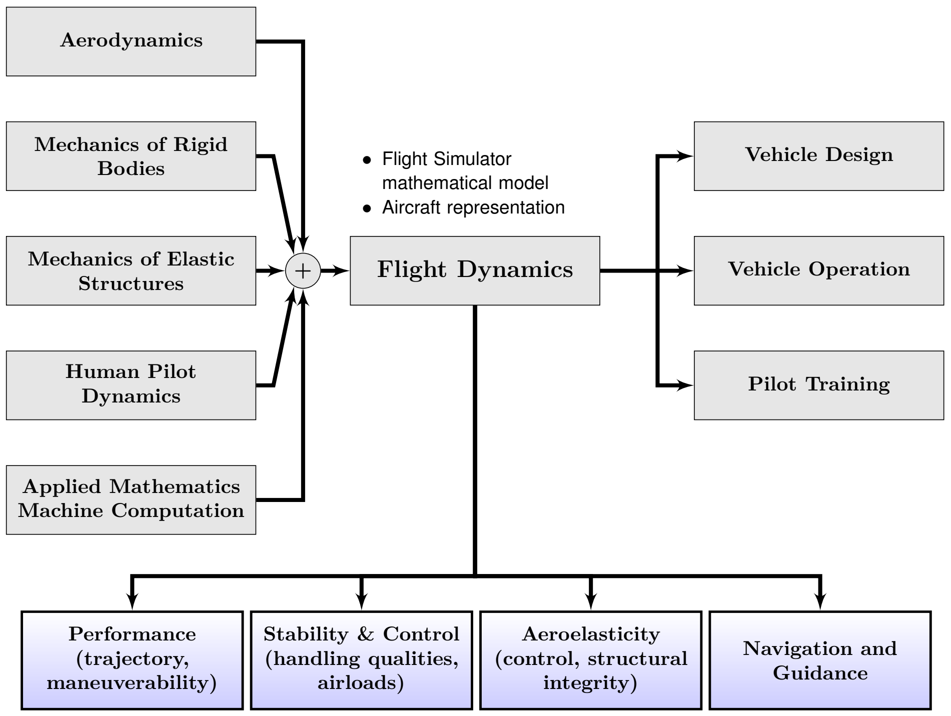

Aerodynamics
Mechanics of Rigid Bodies
Mechanics of Elastic Structures
Human Pilot Dynamics
Applied Mathematics Machine Computation
Flight Simulator mathematical model
Aircraft representation
Flight Dynamics
Vehicle Design
Vehicle Operation
Pilot Training
Performance (trajectory, maneuverability)
Stability & Control (handling qualities, airloads)
Aeroelasticity (control, structural integrity)
Navigation and Guidance

Figure 11: Block diagram of disciplines involved in Flight Dynamics. All graphic elements are placed on the canvas with intuitive tikz commands. See code in Figure 10.

both simple labels and larger paragraphs. Supports layers and views, which make it possible to 'build' illustrations incrementally in a presentation.

- Asymptote,[17] is a vector graphics language and compiler. This software has been mentioned earlier in this article because code snippets in Asymptote language can be embed in LaTeX sources. Asymptote compiler can be used as a standalone tool as well (comes also with its own GUI) for generating both 2D and 3D figures. 3D figures can be included in a pdf file in the PRC (Product Representation Compact) format which allows them to be manipulated when viewed in Adobe Reader.
- LaTeXPiX,[18] is a Windows GUI capable of exporting pgf/LaTeX code.
- TPX,[19] is a Windows GUI similar to LaTeX-PiX, but more flexible.
- Sweave,[20] is a tool that allows users to include R code directly into their LaTeX files. It does much more than just generate graphics, but it makes inclusion of R generated graphics into LaTeX document very easy.
- KtikZ/QtikZ,[21] is a pgf/tikz real-time open source compiler that runs on Linux and Windows.

17. http://asymptote.sourceforge.net
18. http://latexpix.comyr.com/latexpix.htm
19. http://tpx.sourceforge.net
20. http://www.stat.uni-muenchen.de/~leisch/Sweave
21. http://www.hackenberger.at/blog/ktikz-editor-for-the-tikz-language

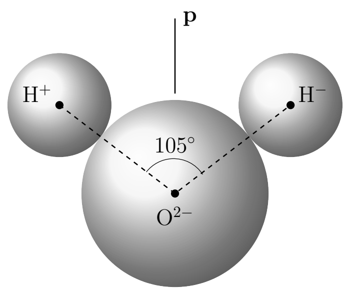
It can speed up the drawing effort while at the same time allowing to code directly in tikz language. It has a template option which allows to define user commands in an easy way as well as a menu with many common (and not so common) tikz constructs.

- LatexDraw,[22] is a very useful open source multiplatform GUI capable of generating pstricks code.
- Dia,[23] is a multiplatform open source GUI that supports both pgf/tikz and pstricks output.
- Sketch,[24] is a language and compiler that allows users to create vector drawings of 3D scenes. It generates pgf/tikz or pstricks code. A detailed presentation of this software can be found in De Marco (2007).
- Inkscape,[25] is an open source and well-supported vector graphics/SVG editor available for all major operating systems. Due to its popularity, and being by far the most powerful and important application among those mentioned here, we will discuss the LaTeX-related capabilities of this graphics software in the rest of this section.

7.1 Using Inkscape

Inkscape is an open source vector graphics editor using the W3C standard Scalable Vector Graphics (SVG) file format, with capabilities similar to

22. http://latexdraw.sourceforge.net
23. http://live.gnome.org/Dia
24. http://www.frontiernet.net/~eugene.ressler
25. http://www.inkscape.org



(a) The diagram by Jimi Oke shows the geometry of hydrogen and oxygen atoms in the water molecule, and the position of the dipole (p).

```
\begin{tikzpicture}[>=latex,scale=1.3]
  \shade[ball color=gray!10!] (0,0)
    coordinate(Hp) circle (0.9);
  \shade[ball color=gray!10!] (2,-1.53)
    coordinate(O) circle (1.62);
  \shade[ball color=gray!10!] (4,0)
    coordinate(Hm) circle (0.9);
  \draw[thick,dashed] (0,0) -- (2,-1.53)
    -- (4,0) ;
  \draw[thick] (2,.2) -- (2,1.5) node[
    right]{$\mathbf{p}$};
  \draw (2.48,-1.2) arc (33:142:.6);
  \draw (2,-.95) node[above]{$105^{\circ}
    $};
  \draw (0,.2) node[left]{H$^+$};
  \draw (4,.2) node[right]{H$^-$};
  \draw (2,-1.63) node[below]{O$^{2-}$};
  \foreach \point in {O,Hp,Hm}
    \fill [black] (\point) circle (2pt);
\end{tikzpicture}
```

(b) The tikz code of the above drawing.

Figure 12: Example of graphics made with tikz.

commercial applications such as Adobe Illustrator, CorelDRAW, or Xara X.

Inkscape supports many advanced SVG features, moreover, developers took great care in designing a streamlined interface, that allows user to edit nodes, perform complex path operations, trace bitmaps, and much more in a very easy way. A well written documentation and many tutorials are available online. For a guide on this application the reader is referred to Bah (2011).

Inkscape provides a large API (Application Programming Interface) and a Python scripting capability. These features have encouraged the development of several third-party extension plugins. One of these plugins is TexText,[26] a particularly important extension for TEX users because it gives

26. https://textext.github.io/textext

them the possibility to add and re-edit (multi-line) LATEX/XELATEX/LuaLATEX generated SVG elements to a drawing. It offers a multi-line editor, optionally with syntax highlighting.

SVG elements created with TexText are enriched with special additional information containing the LATEX code used to generate all text and symbols prior to be traced into SVG vector graphics. The additional information allow users to re-edit the original LATEX commands.

TexText is written in Python and uses either pdf2svg (or a combination of pstoedit and ghostscript) as converter for producing SVG code from the generated PDF (or PostScript). Detailed installation instructions for all major platforms are found on the project website.

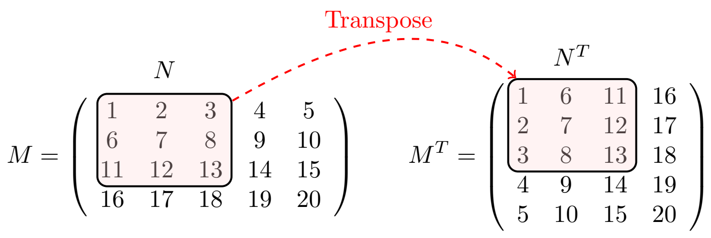
Once TexText and its dependencies are correctly installed, a menu entry Extensions → Tex Text will appear in Inkscape. See Figure 15a. When this menu item is selected, a TexText dialog window appears to assist the user to input the desired LATEX content.

The TexText input dialog window is shown in Figure 15b. LATEX code is entered into the edit box 5. In the case PyGTK is installed, it will show line and column numbers. If PyGTKSourceView has been additionally installed, the edit box will also highlight the syntax with colors. The user can add any valid and also multi-line LATEX code. The plugin provides additional settings which can be adjusted to the user's needs:

- The group box 1 controls the TEX command to be used for compiling the code. Possible options are: pdflatex, xelatex, lualatex.
- The group box 2 points to a custom preamble file that the user might need in order to have his LATEX input compiled successfully.
- The group box 3 regulates a scale factor to be applied to the final SVG element.
- The button 4 becomes active only when the user re-edits the code of the enriched SVG element, and controls the alignment relative to the previous state of the same graphic object.
- Menu items 7 control the default math environment in new nodes and the appearance of the editor.

The LATEX code and the accompanying settings will be stored within the new SVG node in the Inkscape document. This allows the user to re-edit the 'LATEX node' later by selecting it and running the TexText extension (which will then show the dialog containing the saved values).

The TexText dialog window provides also a preview button 6 as well, which shortens the feedback cycle from entry to result considerably. Once the user is happy with the previewed LATEX object the Save button can be clicked to get the resulting SVG object of Figure 16. The final traced result of the intermediate temporary PDF produced with the



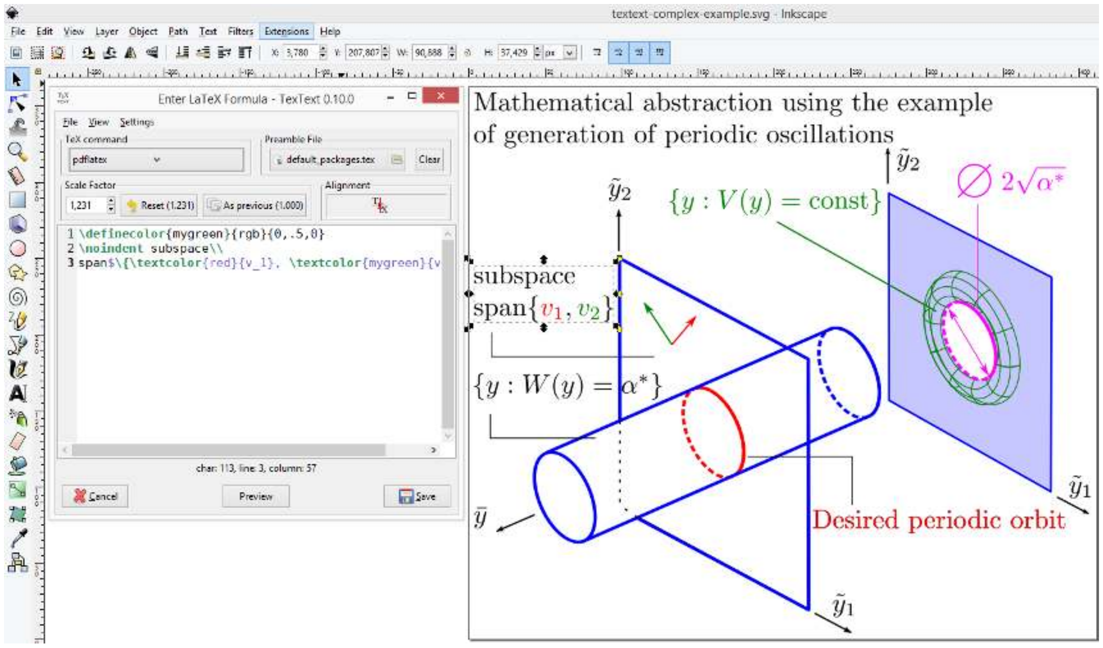
(a) An example by Stefan Kottwitz. A submatrix within a matrix is highlighted with tikz. The same is done in the transposed matrix. tikz and some advanced features of pgf are used in order to produce rectangle nodes that fit the desired areas. The drawing requires a double compilation.

```
% in preamble
\usepackage{tikz}
\usetikzlibrary{fit}
% ...
\tikzset{highlight/.style={rectangle
    ,rounded
    corners,fill=red!15,draw,fill
    opacity=0.3,thick,inner
    sep=0pt}}
\newcommand{\tikzmark}[2]{\tikz[
    overlay,remember
    picture,baseline=(#1.base)] \
    node (#1) {#2};}
\newcommand{\Highlight}[1][
    submatrix]{\tikz[overlay,
    remember
    picture]{
    \node[highlight,fit=(left.north
    west) (right.south east)] (#1)
    {};}}
```

```
\begin{document}
\[
  M = \left(\begin{array}{*5{c}}
    \tikzmark{left}{1} & 2 & 3 & 4 &
     5\\ 6 & 7 & 8 & 9 & 10 \\
    11 & 12 & \tikzmark{right}{13} &
     14 & 15 \\
    16 & 17 & 18 & 19 & 20 \end{
    array}\right)
  \Highlight[first]
  \qquad
  M^T = \left(\begin{array}{*5{c}}
    \tikzmark{left}{1} & 6 & 11 & 16
     \\ 2 & 7 & 12 & 17 \\
    3 & 8 & \tikzmark{right}{13} &
    18 \\ 4 & 9 & 14 & 19 \\
    5 & 10 & 15 & 20 \end{array}\
    right)
\]
\Highlight[second]

\tikz[overlay,remember picture] {
  \draw[->,thick,red,dashed] (first)
     to[out=30,in=140]
     node[above] {Transpose} (second
    );
  \node[above of=first] {$N$};
  \node[above of=second] {$N^T$};
}
\end{document}
```

(b) The tikz code of the above drawing.

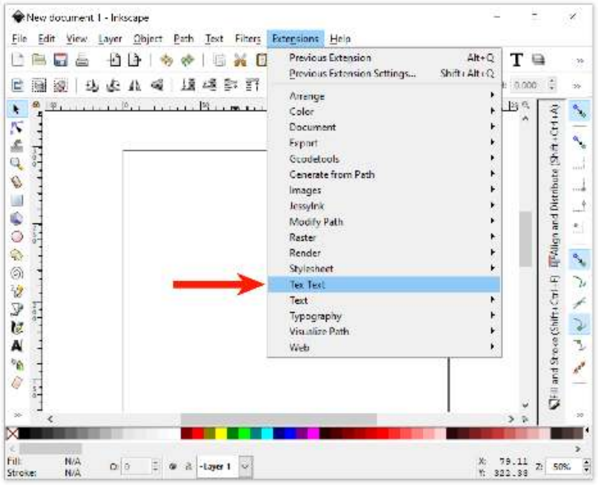
Figure 13: Example of nice graphics made with tikz.


textext-complex-example.svg - Inkscape
File Edit View Layer Object Path Text Filters Extensions Help
Enter LaTeX Formula - TexText 0.10.0
File View Settings
TeX command
pdflatex
Preamble File
default_packages.tex
Clear
Scale Factor
1,231
Reset (1,231)
As previous (1,000)
Alignment
Cancel
Preview
Save
Mathematical abstraction using the example
of generation of periodic oscillations
$\tilde{y}_2$
$\{y : V(y) = \text{const}\}$
$\varnothing\ 2\sqrt{\alpha^*}$
subspace
span$\{v_1, v_2\}$
$\{y : W(y) = \alpha^*\}$
$\bar{y}$
Desired periodic orbit
$\tilde{y}_1$

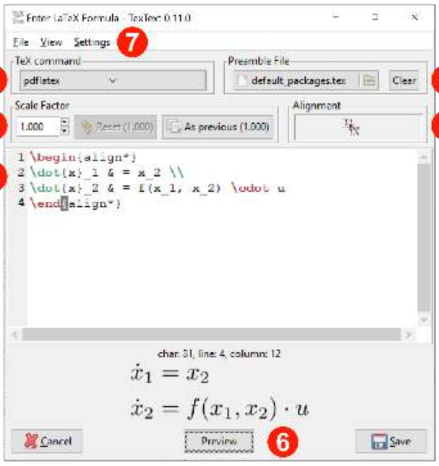
```
\definecolor{mygreen}{rgb}{0,.5,0}
\noindent subspace\\
span$\{\textcolor{red}{v_1}, \textcolor{mygreen}{v
```

char: 113, line: 3, column: 57

Figure 14: A screenshot of Inkscape with TexText extension in use.



(a) Selecting TexText from Inkscape Extensions menu.



```
\begin{align*}
\dot{x}_1 & = x_2 \\
\dot{x}_2 & = f(x_1, x_2) \cdot u
\end{align*}
```

char: 81, line: 4, column: 12

$$\dot{x}_1 = x_2$$
$$\dot{x}_2 = f(x_1, x_2) \cdot u$$

(b) The TexText dialog window.

Figure 15: Using TexText extension plugin in Inkscape.

input LATEX code is visible in Figure 17 where the highlighted nodes of the SVG element are shown.

TexText is a very powerful tool when coupled with the potential of Inkscape itself. Yet the user have to be aware of including the required packages in the preamble file if special commands are used in LATEX code that rely on such packages. The preamble file can be chosen by the selector mentioned above. The default preamble file shipped with TexText is the following:

```
% default_packages.tex
\usepackage{amsmath,amsthm,amssymb,
    amsfonts}
\usepackage{color}
```

Basically, user's LATEX code will be inserted into this template:

```
\documentclass{article}
% ===> preamble file content <===
% default:
%   \input{default_packages}
\pagestyle{empty}
\begin{document}
% ==> User's code <===
\end{document}
```

This will be typeset in a separate system thread, the PDF result will be converted to SVG and

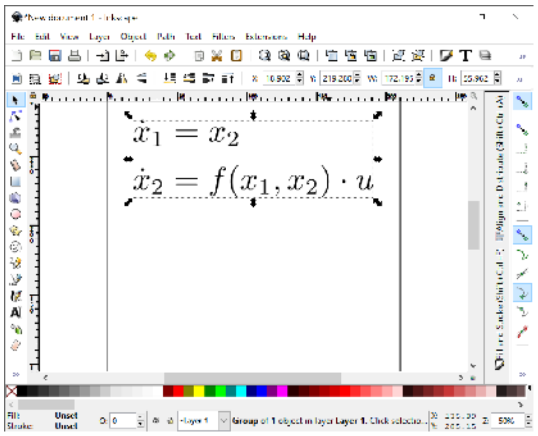


Figure 16: SVG element resulting from input of Figure 15b.

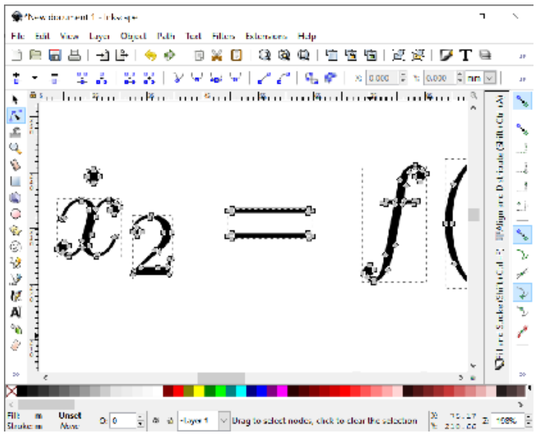


Figure 17: Nodes highlighted in the SVG element of Figure 16.

the vector object will be inserted into the current Inkscape document.

In conclusion of this section, a fairly elaborated illustration is displayed in Figure 18. The vector image has been produced using Inkscape with the TexText extension. The preamble file has been customized to load the package mt2pro and use the commercial font MathTime Professional 2.

8 Presenting data with plots

This section studies the presentation of data with "data plots" using LaTeX. Usually one shall use the word 'graph' instead of data plot. The main focus of this final part of the article is the package pgfplots, which creates astonishingly beautiful data plots in a consistent style with great ease.

The package pgfplots is built on top of pgf and is designed to draw graphs in a variety of formats, with a consistent, professional look and feel. The package also allows to import data stored in files in tabular format via the package pgfplotstable.[27]

27. https://ctan.org/pkg/pgfplotstable

As is usual with the pgf family, their manuals are impressive (Feuersänger, 2018).

8.1 The axis environment

The workhorse of the pgfplots package is an environment called axis, which may define one or several *plots* (graphs). Each plot is drawn with the command \addplot. When the graphs are drawn the environment also draws a 2- or 3-dimensional axis. The axis environment is used inside a tikzpicture environment, so one can also use tikz commands. The options of the axis environment specify the type of the plot, the width, the height, and so on.

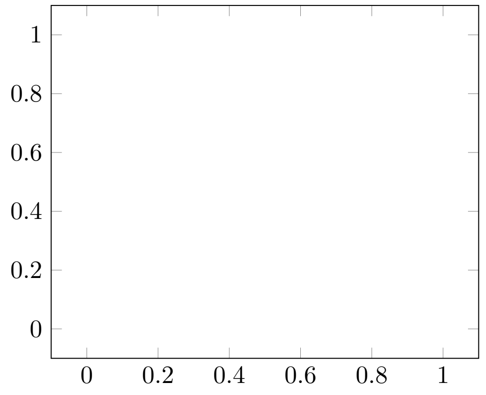
Typically, one or more plots are created in LaTeX following the template:

```
% in preamble
\usepackage{pgfplots}% loads tikz
...
\begin{tikzpicture}
   \begin{axis}[⟨graphic options⟩]
   ...
   ⟨pgfplots or tikz commands⟩
   ...
   \end{axis}
\end{tikzpicture}
```

The simplest possible graph with pgfplots is given by the code

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

that is, an empty axis environment, with default formatting options. The result is:



This can be customized, for example, changing the ranges of x- and y-axis, introducing a grid, and defining axis labels. This is done by passing the following self-explanatory options to the axis environment

```
\begin{axis}[
  xmin = -1, xmax = 1,
  ymin =  0, ymax = 2,
  grid = major,
```

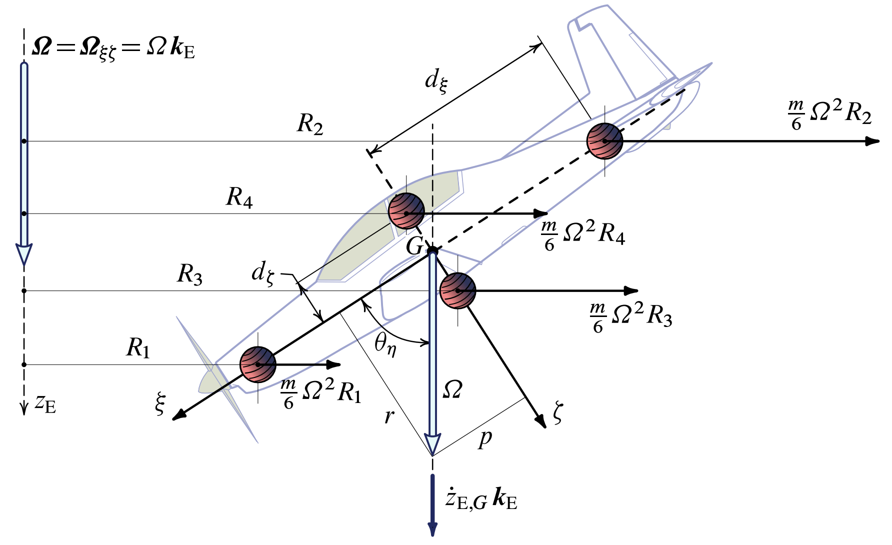
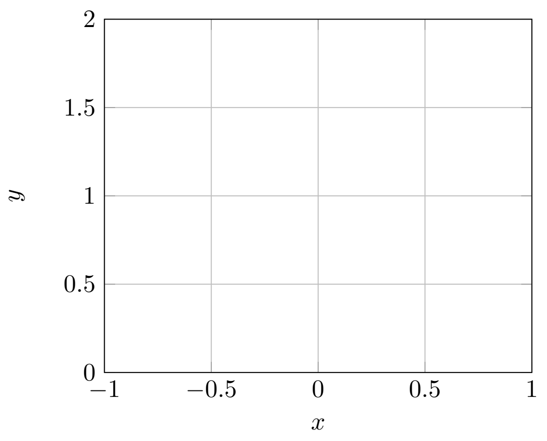

$\boldsymbol{\Omega} = \boldsymbol{\Omega}_{\xi\zeta} = \Omega\,\boldsymbol{k}_{\mathrm{E}}$
d_ξ
R_2
R_4
R_3
R_1
d_ζ
G
θ_η
$\frac{m}{6}\Omega^2 R_2$
$\frac{m}{6}\Omega^2 R_4$
$\frac{m}{6}\Omega^2 R_3$
$\frac{m}{6}\Omega^2 R_1$
z_{E}
ξ
ζ
Ω
r
p
$\dot{z}_{\mathrm{E},G}\,\boldsymbol{k}_{\mathrm{E}}$

Figure 18: A fairly elaborated illustration representing an aircraft in a spin manoeuvre. The image has been made using Inkscape with the TexText extension. The preamble file has been customized to load the package mt2pro and use the commercial font MathTime Professional 2.

```
  xlabel = $x$, ylabel = $y$
]
\end{axis}
```

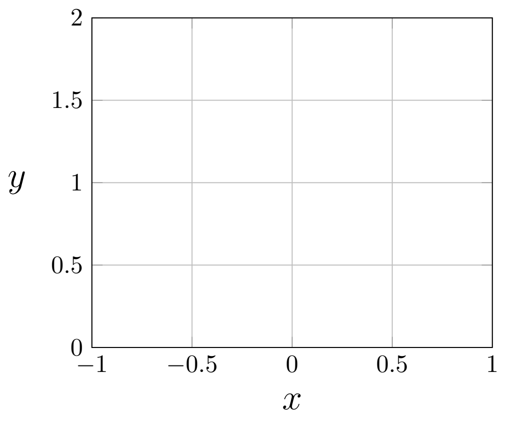
The customized axes now appear as follows:



The package pgfplot, as also pgf and tikz do, provides a way to change default settings at a global level. For plots this feature is given by the command \pgfplotsset. The following example enlarges the default font size in axis labels and rotates the y-axis label.

```
% in preamble
\usepackage{pgfplots,relsize}
...
% pgfplots styles
\pgfplotsset{
  every axis x label/.append style = {
    font = \relsize{2}
  },
  every axis y label/.append style = {
    font = \relsize{2},
    rotate = -90,
    xshift = 0.5em
  }
}
\begin{tikzpicture}
\begin{axis}[
  xmin = -1, xmax = 1,
  ymin =  0, ymax = 2,
  grid, xlabel = $x$, ylabel = $y$
]
\end{axis}
\end{tikzpicture}
```

The above code yields:

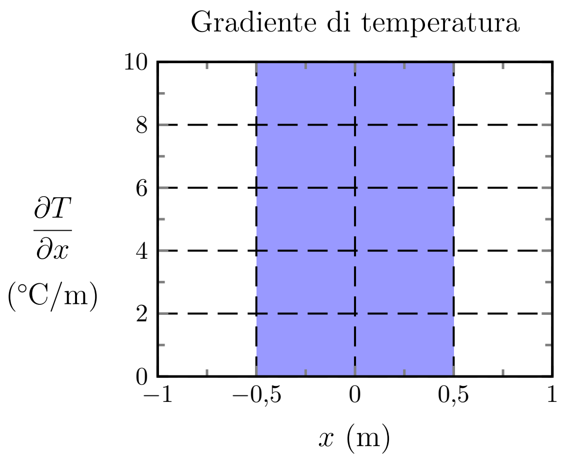


The macro \pgfplotset acts on predefined *styles*, as, for instance, in the last example on those labelled every axis x label as well as every axis y label. The usual approach is to change style parameters by *appending* customized settings to the defaults, such as font, rotate, xshift, and several others. The user provided settings overwrite the default ones. To learn more on style customizations available for pgf and pgfplots the reader might want to look at the remaining examples in this section and at the package documentation.

The following example demonstrates further customizations. The style of grid lines is changed and axis labels are formatted with the help of macro \si provided by the package siunitx.

```
% in preamble
\usepackage{pgfplots,relsize,siunitx}
...
% pgfplots styles
\pgfkeys{
  /pgf/number format/.cd,
  use comma
}
\pgfplotsset{
  every axis/.append style={
    font=\relsize{0},
    line width=1.0pt,
    tick style={line width=1.0pt}
    },
  every axis x label/.append style={
    font=\relsize{1},
    yshift=0pt,
    xshift=0em
  },
  every axis y label/.append style={
    font=\relsize{1},
    rotate=-90,
    xshift=-0.7em,
    yshift=-1.4em,
  },
  major grid style={
    line width = 0.8pt,
    black,
    dash pattern=on 8pt off 4pt
  },
  every axis title/.append style={
    font=\relsize{1}
  }
}
\begin{tikzpicture}
\begin{axis}[
  xmin=-1, xmax=1,
  ymin=0, ymax=10,
  xtick={-1,-0.5,...,1},
  ytick={0,2,...,10},
  minor x tick num = 1,
  minor y tick num = 1,
  grid=major,
  xlabel={$x$ (\si{\meter})},
  ylabel={
    \parbox{2cm}{%
      \centering
      $\dfrac{\partial T}{\partial x}$
      \\[0.7em]
      \centering
      (\si{\celsius/\meter})
    }
 },
 title=Gradiente di temperatura,
 axis on top=true
]
% the shaded rectangle
\fill[blue!40]
    (axis cs:-0.5,0) --
    (axis cs:0.5,0) --
    (axis cs:0.5,10) --
    (axis cs:-0.5,10) --
    cycle;

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

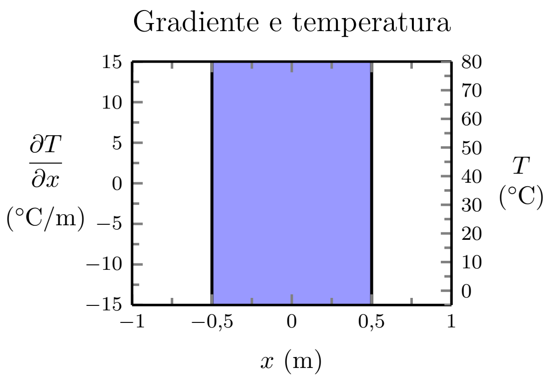
The macro pgfkeys, similar to pgfplotsset, is provided by pgf and is used to change the default decimal separator in numbers from '.' (dot) to ',' (comma). This example demonstrates also the use of tikz drawing commands inside the axis environment. The above code yields:



Next example evolves from the previous one. It demonstrates the use of a second y-axis on the right side of the plot bounding box. The second axis has different range and scaling with respect to the default one on the left side, and is used typically when multiple sets of data with values in different ranges have to be represented in the same plot.

```
\begin{tikzpicture}
% same initial settings
% of previous example
\begin{axis}[
  height=6cm,% <==
  % same initial settings
  % of previous example
]
% same drawing commands
% of previous example
% add plot data #1 here <==
\end{axis}
% second axis
\begin{axis}[
  height=6cm,% <==
  xmin=-1, xmax=1,
  axis x line=none,  % <==
  axis y line*=right,% <==
  ymin=-5, ymax=80,
  ytick={-10,0,...,80},
  minor y tick num = 1,
  ylabel={
    \parbox{2cm}{%
      \centering
      $T$
      \\[0.0em]
      \centering
      (\si{\celsius})
    }
  },
]
% add plot data #2 here
\end{axis}
\end{tikzpicture}
```

The new y-axis is obtained by superimposing a second reference frame on the first one. This is done by giving a second axis environment in the same tikzpicture. The second environment has an hidden x-axis and an `axis y line` set to `right`. All data whose y-values are conveniently represented with the new axis range should be provided in the second axis environment. The above code yields:



The code of a more elaborated multiple axis example is shown in Figure 19. The product is that of Figure 20 showing three y-axes conveniently positioned on the left- and right-hand sides of the main area of the plot. Various tikz drawing commands are used in this case to annotate and decorate the graph for a refined visual result.

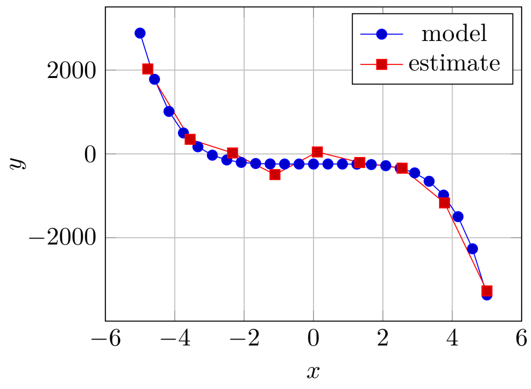
8.2 The macro \addplot

The command `\addplot` is used within an axis environment to define the lines in a graph. The command accepts a number of options and an argument that specifies the set of data to be represented on canvas.

The following example contains two line graphs with two different markers and a legend. No options are passed to the two `\addplot` commands to customize their behaviour. Yet, two different types of data sources are chosen: the first is a mathematical function, $f(x) = -x^5 - 242$; the second is a discrete set of (x, y)-coordinates.

```
\begin{tikzpicture}
\begin{axis}[
  grid=major,
  xlabel={$x$}, ylabel={$y$},
  y tick label style={
    /pgf/number format/.cd,
    set thousands separator={},
  /tikz/.cd}
  ]
  % a function of x
  \addplot {-x^5 - 242};
  \addlegendentry{model}
  % a discrete set of coordinates
  \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}
```

The above code yields:



The mathematical function is defined intuitively at high-level as `-x^5 - 242` and is parsed by the powerful low-level `\pgfmathparse` feature of pgf. By default, the function is evaluated at 25 points equally spaced between two automatically calculated x-axis limits — in this example $[-5, 5]$. Data points are connected with a blue solid line and marked by default with dots of the same color. The second set of data is manually given with a `coordinates`

```
%in preamble
\usepackage{pgfplots}
\usetikzlibrary{calc,arrows,decorations.pathmorphing,
     backgrounds,fit,positioning,shapes.symbols,shapes.
     geometric,shapes.misc,chains}
% ...
\begin{tikzpicture}
\pgfplotsset{compat=1.3}
\pgfkeys{
   /pgf/number format/.cd,
       set decimal separator={,{\!}},
      set thousands separator={}}
\pgfplotsset{
   every axis/.append style={
      font=\relsize{2},
      line width=1.0pt,
      tick style={line width=1.0pt}},
   every axis x label/.append style={
      font=\relsize{4},
      yshift=0pt,
      xshift=0em},
   every axis y label/.append style={
      font=\relsize{3},
      rotate=-90,
      xshift= 0.8em,
      yshift=-1.4em},
   major grid style={
      line width = 0.8pt,
      black,
      dash pattern=on 8pt off 4pt},
   every axis title/.append style={
      font=\relsize{3}},
   no markers
}
% the left y-axis #1
\begin{axis}[
   clip=false,
   scale only axis,
   width=2cm, xshift=-0.4cm,
   xmin=-1, xmax=1,
   hide x axis,
   axis y line*=left,
   ymin=-15, ymax=15,
   ytick={-15,-10,...,15},
   minor y tick num = 1]
\node [above, yshift=6pt] at (rel axis cs:0,1)
   {$\dfrac{\partial T}{\partial x}$ (\si{\celsius/\meter
     })};
\end{axis}
% the unique x-axis
\begin{axis}[
   scale only axis,
   height=2cm, yshift=-0.4cm,
   xmin=-1, xmax=1,
   xtick={-1,-0.5,...,1},
   minor x tick num = 1,
   xlabel={$x$ (\si{\meter})},
   axis x line*=bottom,
   hide y axis,
   ymin=-15, ymax=15,
]
\end{axis}
% the curve #1
\begin{axis}[
   scale only axis,
   xmin=-1, xmax=1,
   hide x axis,
   ymin=-15, ymax=15,
   hide y axis,
   title=\parbox{8cm}{\centering Trasmissione del calore
     attraverso una parete},
]
\fill[blue!40]
    decorate [decoration={random steps,segment length=2mm
     }] { [very thick] (axis cs:-0.5,-14.8) -- (axis cs
     :0.5,-14.8) } -- (axis cs:0.5,14.8)
    decorate [decoration={random steps,segment length=2mm
     }] { [very thick] -- (axis cs:-0.5,14.8)}
    -- cycle;
\draw[very thick] (axis cs:-0.5,-15) -- (axis cs:-0.5,15)
     ;
\draw[very thick] (axis cs:0.5,-15) -- (axis cs:0.5,15);
\node [rounded rectangle, minimum size=6mm, very thick,
     draw=black!50, top color=white, bottom color=black
     !20, font=\ttfamily] at (rel axis cs:0.125,0.94)
     {1};
\node [rounded rectangle, minimum size=6mm, very thick,
     draw=black!50, top color=white, bottom color=black
     !20, font=\ttfamily] at (rel axis cs:0.50,0.94) {2};
\node [rounded rectangle, minimum size=6mm, very thick,
     draw=black!50, top color=white, bottom color=black
     !20, font=\ttfamily] at (rel axis cs:0.875,0.94)
     {3};
% \addplot of temperature gradients here
\end{axis}
\pgfplotsset{
   every axis y label/.append style={
     xshift= -2.4em
   }
}
% the right y-axis 1
\begin{axis}[clip=false, scale only axis,
   xshift=0.4cm, xmin=-1, xmax=1, hide x axis,
   axis y line*=right,
   ymin=-5,ymax=80, ytick={-10,0,...,80},
   minor y tick num = 1,
]
\node [above, yshift=6pt] at (rel axis cs:1,1)
   {$T$ (\si{\celsius})};
% \addplot of temperatures here
\end{axis}
\pgfplotsset{every axis y label/.append style={xshift
     =-1.3em}}
% the right y-axis 2
\begin{axis}[clip=false, scale only axis,
   xshift=2.4cm,
   xmin=-1, xmax=1,
   axis x line=none, hide x axis,
   axis y line*=right,
   ymin=-5, ymax=500,
   ytick={0,50,...,500},
   minor y tick num = 1,
]
\node [above, yshift=6pt, xshift=16pt] at (rel axis cs
     :1,1) {$q$ (\si{\kcal/\meter^2})};
% \addplot of heat fluxes here
\end{axis}
\end{tikzpicture}
```

Figure 19: Source code of diagram reported in Figure 20.

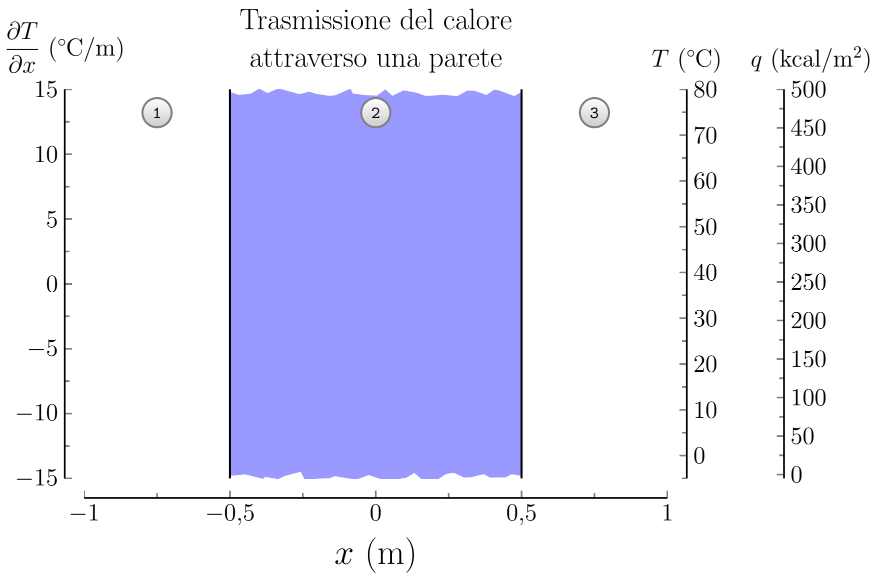


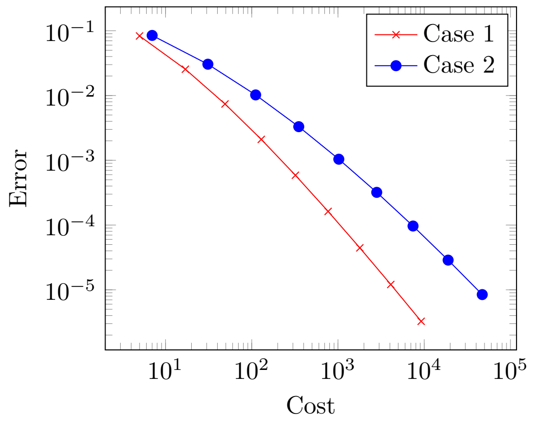
Figure 20: Example of multiple y-axes obtained with pgfplots. See source code in Figure 10.

directive to `\addplot` as a sequence of couples (⟨*x*⟩,⟨*y*⟩). This second set of data points are connected with a red solid line and marked by default with boxes filled with the same color.

The following example demonstrates the use of logarithmic scales for the axes within the environment loglogaxis. The markers of the line graphs are controlled by specific settings passed as options to the `\addplot` commands.

```
% in preamble
\usepackage{filecontents}
\begin{filecontents*}{data1.txt}
Level    Cost        Error
1           7 8.47178381e-02
2          31 3.04409349e-02
3         111 1.02214539e-02
4         351 3.30346265e-03
5        1023 1.03886535e-03
6        2815 3.19646457e-04
7        7423 9.65789766e-05
8       18943 2.87339125e-05
9       47103 8.43749881e-06
\end{filecontents*}
% ...
\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]
    {data1.txt};
\legend{Case 1,Case 2}
\end{loglogaxis}
\end{tikzpicture}
```

The second line graph is constructed by reading coordinates from a conveniently formatted text file, `data1.txt`. The file contains three columns of data and a first row that provides the labels for each column. Options `x=` and `y=` in the second `\addplot` command select the desired x- and y-coordinate sets according to the column names. The above code yields:



The example below demonstrates the use of a logarithmic scale for the y-axis only.

```
\begin{tikzpicture}
\begin{semilogyaxis}[
\begin{tikzpicture}
\tikzset{
  every pin/.style={
    fill=yellow!50!white,
    rectangle, rounded corners=3pt,
    font=\tiny},
  small dot/.style={
    fill=black, circle,
    scale=0.3}
}
\begin{axis}[
  clip=false,
  title=How \texttt{axis description cs} works
]
\addplot {x^3};

% annotations
\node[small dot,
  pin={[pin distance=2cm]20:{$(0,0)$}}]
  at (axis description cs:0,0) {};
\node[small dot,
  pin=-30:{$(1,1)$}]
  at (axis description cs:1,1) {};
\node[small dot,
  pin=-90:{$(1.03,0.5)$}]
  at (axis description cs:1.03,0.5) {};
\node[small dot,
  pin=125:{$(0.5,0.5)$}]
  at (axis description cs:0.5,0.5) {};
\end{axis}
\end{tikzpicture}
```

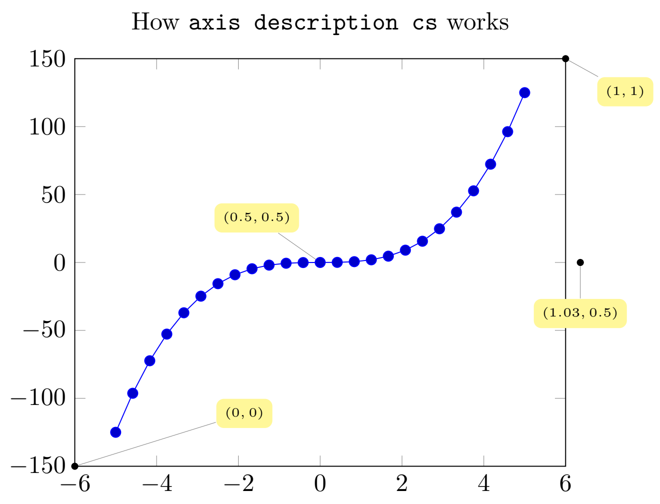
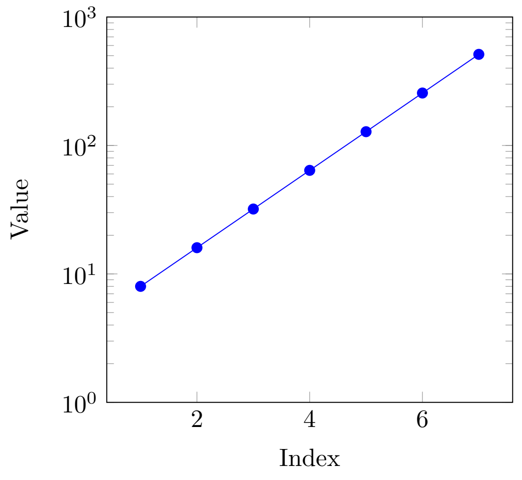


Figure 21: An example showing how, with the special coordinates system named `axis description`, in pgfplots it is possible to define points relative to the bounding box of a plot.

```
  ymin=1, ymax=1000,
  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}
```

The above code yields:



The package pgfplots defines special coordinate systems (`cs`) that make it easy to add annotations to the plots. One of these reference systems is named `axis description`. It has its point of coordinates $(0, 0)$ in the bottom left corner of the plot bounding box, and its point $(1, 1)$ at the top right corner of the frame. A demonstration of this coordinate system is shown in Figure 21. The annotations of the plot are made by defining a style/shape named `small dot`, and using the `pin` option of the tikz `\node` operation.

A fairly elaborated example of line graph annotation is provided by Figure 22. Thanks to the tikz library `intersection`, the pgf macro named `linelabel` is defined in such a way that it can be used as an option to `\addplot`. The option receives three arguments: the first is the normalized ascissa (in range $[0, 1]$) of the point along the path where the annotation is pinned (0 for the leftmost point, 1 for the rightmost); the second argument specifies the angle and the length of the pin line; the third argument is the content of the annotation (a formula or even a multi-line text). This customization, too, is possible thanks to the `node` and `pin` features in tikz.

8.3 The macro \addplot3

The command `\addplot3` within an axis environment creates a three-dimensional plot. According to the option and arguments, it can display a line graph or a shaded surface.

The following example plots a helical curve:

```
\begin{tikzpicture}
\pgfplotsset{
```

```
\begin{tikzpicture}
% needs tikzlibrary: intersections
\pgfkeys{/pgfplots/linelabel/.style args
      ={#1:#2:#3}{
  name path global=labelpath,
  execute at end plot={
    \path [name path global =
      labelpositionline] (rel axis cs:#1,0)
        -- (rel axis cs:#1,1);
    \draw [help lines, text=black,
      inner sep=0pt,
      name intersections={
        of=labelpath and labelpositionline}]
        (intersection-1) -- +(#2)
        node [label={#3}] {};
    }
}}
\pgfplotsset{%
  every axis legend/.append style={
  cells={anchor=west},%
  fill=gray!10,
  font=\relsize{1},
  at={(0.97,0.03)},
  anchor=south east,thin,draw=none},
  every axis title/.append style={font=\
      relsize{1}},
  every axis/.append style={font=\relsize{0}},
  every axis x label/.append style={
    font=\relsize{1},
    yshift=0pt,xshift=0em},
  every axis y label/.append style={
    font=\relsize{2},
    rotate=-90},
  every axis/.append style={
    thick,
    tick style={thick}}
}
\tikzstyle{every pin}=[fill=white,draw=none,
      font=\relsize{2}]
\begin{axis}[xlabel={$x$}]
  \addplot [thick,
    linelabel=0.8:{135:1.75cm}:
      {[black]above left:$x^2$}] {x^2};
  \addplot [thick,
    blue,
    densely dashed,
    linelabel=0.85:{135:0.50cm}:
      $\frac{3}{2}x^2$] {1.5*x^2};
  \addplot [thick,
    red,
    dash pattern=on 5pt off 2pt,
    linelabel=0.7:{135:1.25cm}:
      $\frac{1}{10}x^3$] {0.1*x^3};
  \addplot [thick,
    dash pattern=on 1.2pt off 2pt on 5pt off 2pt,
    linelabel=0.80:{-135:0.75cm}:{left:
      \makebox[0pt][r]{$\left.
      \begin{array}{rl}
        \frac{1}{2}x^2 &\text{if }x\le 0\\[6pt]
        -\frac{1}{5}x^3 &\text{if }x> 0
      \end{array}
      \right\}$}
      }
    ]
    {(x<0)*0.5*x^2 + (x>0)*(-0.20*x^3)};
\end{axis}
\end{tikzpicture}
```

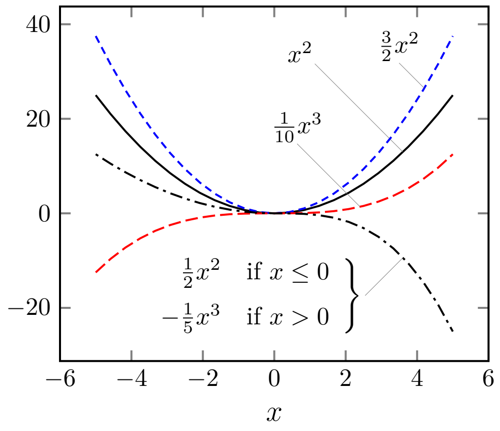


Figure 22: An example of line graph annotations obtained introducing a customized option `linelabel` to the command `\addplot`. The option works as a macro and is based on the `tikz` extension library `intersections`.

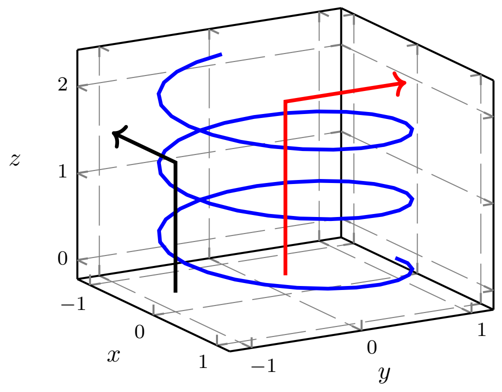
```
  every axis/.append style={
    font=\relsize{-1},
    line width=0.8pt,
    tick style={line width=0.8pt}
  },
  major grid style={
    line width = 0.4pt,
    gray,
    dash pattern=on 16pt off 4pt
  }
}
\begin{axis}[
   view={60}{20},% <== view point
   xmin=-1.2, xmax=1.2,
   ymin=-1.2, ymax=1.2,
   grid = major,
   xlabel=$x$, ylabel=$y$, zlabel=$z$,
   every axis x label/.style={
      at={(rel axis cs:0.5,-0.15,-0.15)},
      font=\relsize{0}},
   every axis y label/.style={
      at={(rel axis cs:1.15,0.5,-0.15)},
      font=\relsize{0}},
   every axis z label/.style={
      at={(rel axis cs:-0.15,-0.15,0.5)},
      font=\relsize{0}},
   variable=\t]
% the helix
\addplot3+[
  domain=0:5.5*pi,
  samples=70,
  samples y=0,
  no marks,
  line width=1.5pt]
  ( {sin(deg(t))},  % <== x(t)
    {cos(deg(t))},  % <== y(t)
    {2*t/(5*pi)} ); % <== z(t)
% a line in 3d
\addplot3 +[->,no marks,line width=1.5pt]
   coordinates {
    (0,0,0)
    (0,0,2)
    (0,1.1,2)};
% a line in 3d with a tikz command
\draw[->,line width=1.5pt]
  (axis cs:0,-1,0)
  -- (axis cs:0,-1,1.5)
  -- (axis cs:-1,-1,1.5);
\end{axis}
\end{tikzpicture}
```

The helix is defined as

$$x(t) = \sin t\,,\ y(t) = \cos t\,,\ z(t) = \frac{2}{5\pi}t$$

with $0 \le t \le \frac{11}{2}\pi$ (see `domain` option). The line graph is made by connecting 70 three-dimensional data points (see `samples` option). The mathematical expression of the curve is passed to `\addplot3` as a triplet of functions of `t` after having declared `variable=\t` as an option of the axis environment. Two more lines are represented in the diagram. One is made with the `\addplot3` command itself (hence, by default is red) by connecting three points: $(0,0,0)$, $(0,0,2)$, and $(0,1.1,2)$. The other line is made using the tikz command `\draw` by connecting three points: $(0,-1,0)$, $(0,-1,1.5)$, and $(-1,-1,1.5)$.

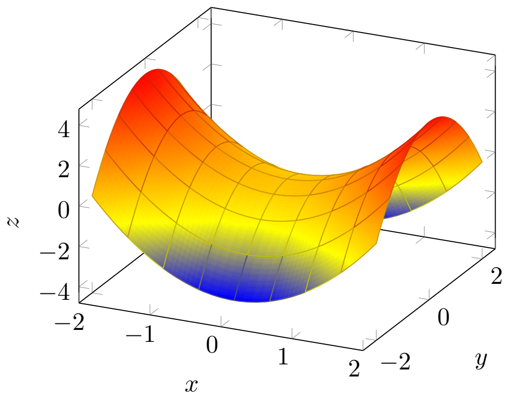
The above code yields:



The following final example creates the plot of a three-dimensional shaded surface. The surface is constructed by evaluating the function $f(x,y) = x^2 - y^2$ in a set of points on the xy-plane. The shading algorithm is provided by the pgfplots extension library `patchplots`.

```
\begin{tikzpicture}
\begin{axis}[width=0.98\linewidth,
  ymin=-2.5, ymax=2.5,
  xlabel={$x$}, ylabel={$y$}, zlabel={$z$
    }
]
\addplot3[
  % needs pgfplotslibrary: patchplots
  patch, patch refines=3,
  shader=faceted interp,
  patch type=biquadratic]
  table[z expr=x^2-y^2] {
     x  y
    -2 -2
     2 -2
     2  2
    -2  2
     0 -2
     2  0
     0  2
    -2  0
     0  0};
\end{axis}
\end{tikzpicture}
```

The above code yields:



8.4 What else?

In this final part we really have only scratched the surface of what can be done with pgfplots. A more in-depth presentation of both basic and advanced features of the package can be found in De Marco and Giacomelli (2011). Examples of quality manuscripts including several fine tuned technical illustrations, diagrams and scientific plots can be found on the author's page of his course on Flight Dynamics and Simulation at the University of Naples Federico II.[28]

Scientific writers nowadays can rely on several different applications and data visualization technologies for producing their own two- and three-dimensional graphs. These include, to name a few: Gnuplot, the Python libraries Matplotlib, Seaborn, ggplot, Bokeh, and Plotly, the R libraries ggplot2 and Lattice, the Javascript libraries D3 and Plotly.js, the numerical computing environments Matlab and Mathematica, and the highly specialized software Tecplot. Some of these tools provide their users with the possibility to export their plots as tikz or pgfplots code, e. g. the Gnuplot `lua tikz` terminal[29] or the Matlab script matlab2tikz.[30]

28. http://wpage.unina.it/agodemar/DSV-DQV/#materiale-didattico
29. http://gnuplot.info/documentation.html
30. https://github.com/matlab2tikz/matlab2tikz

The strength of pgfplots combined with tikz is the excellent typographical quality of their graphical outputs. Apart from those given in this article and the pgfplots online gallery,[31] the reader can find several online examples of publication quality graphs.[32] Being pgfplots able to parse the definition of mathematical functions as well as to import numerical data produced with third-party software, a production work flow based on this package is probably the way to go for the majority of LaTeX users. Moreover, this approach promotes the automation of production processes — from the collection of data, to their import in pgfplots, to the drawing and typesetting of the final image (in raster or vector format), up to the inclusion of the image in the master document. Experience confirms that this approach brings about a tremendous speed up of graphics production.

In cases where a certain third-party technology must be used to produce graphic works, still these can be successively annotated with LaTeX content. The suggested approach is to import them in Inkscape and add LaTeX objects with the TexText plugin. Examples of quality graphics matching the style of websites with a lot of LaTeX content are given by the flight simulation software library JSBSim documentation project,[33] and by the online teaching material of the Flight Mechanics course for the Italian Air Force Academy student pilots.[34]

9 Conclusion

This paper describes the most common scenarios encountered by LaTeX users when they face the problem of producing quality graphics to include in their documents. In cases of diagrams, pictures and more or less complicated illustrations the two approaches based on package tikz and on the Inkscape graphics vector software have been presented. The last part of the article introduces the package pgfplots for making scientific plots.

The paper is example driven and aims at stimulating readers' creativity, providing them as well with several online references.

31. http://pgfplots.sourceforge.net/gallery.html

32. From the author, see also these sample projects on Overleaf: https://www.overleaf.com/read/mgskyfdpttzt , https://www.overleaf.com/read/kqkvsrfjxnmz , https://www.overleaf.com/read/rcbqhpqqhccn .

33. https://jsbsim-team.github.io/jsbsim-reference-manual

34. https://agodemar.github.io/FlightMechanics4Pilots

References

Bah, Tavmjong (2011). *Inkscape. Guide to a Vector Drawing Program.* Prentice-Hall, Upper Saddle River, NJ, USA, 4th edition.

Beccari, Claudio (2011). «The unknown picture environment». *ArsTEXnica*, (11), pp. 57–64. http://www.guitex.org/home/it/numero-11.

De Marco, Agostino (2007). «Illustrazioni tridimensionali con Sketch/LaTeX/PSTricks/TikZ nella didattica della Dinamica del Volo». *ArsTEXnica*, (4), pp. 51–68. http://www.guitex.org/home/numero-4.

— (2009). «Produrre grafica vettoriale di alta qualità programmando asymptote». *ArsTEXnica*, (8), pp. 25–39. http://www.guitex.org/home/numero-8.

De Marco, Agostino and Roberto Giacomelli (2011). «Creare grafici con pgfplots». *ArsTEXnica*, (12), pp. 12–38. http://www.guitex.org/home/it/numero-12.

Feuersänger, Christian (2018). «Manual for package pgfplots». https://ctan.org/pkg/pgfplots.

Goosens, Michel, Frank Mittelbach, Sebastian Rahtz, Denis Roegel and Herbert Voß (2007). *The LaTeX Graphics Companion.* Addison-Wesley Publishing Company, Reading, Mass.

Harris, Robert L. (1996). *Information Graphics. A Comprehensive Illustrated Reference.* Management Graphics, Atlanta, GA, USA.

Lamport, Leslie (1994). *LaTeX, a document preparation system.* Addison-Wesley, Reading, MA, 2nd edition.

Tantau, Till (2016). «The pgf package». http://ctan.org/pkg/pgf.

van Dongen, Marc (2012). *LaTeX and Friends.* Springer-Verlag, Berlin, Heidelberg.

Voß, Herbert (2011). *PSTricks – Graphics and PostScript for TEX and LaTeX.* UIT – Cambridge, Cambridge, UK, 1st edition.

▷ Agostino De Marco
Università degli Studi di Napoli Federico II
agostino dot demarco at unina dot it