zLaTeX/zTikZ 系列

Eureka

2024 年 3 月 10 日

目录

1

zLaTeX 系列

1.1 简介

1.1.1 为何叫 z?

我也不知道为什么我这个系列名称要以 'z' 开头，可能是因为我喜欢这个字母吧，或者是因为我觉得这个字母有一些别的意味。但是最开始我的这个系列中的文档类其实是叫做 πLaTeX，但是后面自己又想开发一个用于绘图的宏包，这个宏包主要是基于 TiKZ. 也许是看到了这个单词中的 z，所以便以 'z' 为前缀，于是就产生了系列。

1.1.2 基本组成

本系列目前包含以下的两个组成部分，一个文档类和一个绘图库:

- zLaTeX 文档类
- 宏包

其中前者主要用于指定排版文档的基本属性，后者主要用于绘图[1]。其实从这个介绍文档就可以看出，本模板是十分的朴素的，没有十分华丽的色彩和精美的页面布局，但是在折腾真么久的 LaTeX 之后，我觉得现在这个模板才是最适合我的；至于，是否适合你，那就不得而知了。你可以去使用更加精美的模板，比如 ElegantLaTeX, BeautyLaTeX 等优秀的模板.

1.2 模板设计

1.2.1 设计历程

其实本模板的设计经历了相当长的一个周期，从最开始的初始 LaTeX, 我把自己常用的宏扔到了一个 `.sty` 文件中，以为这就是一个宏包了；之后了解到了ElegantLaTeX系列模板，也使用这个系列中的 book 文档类写了一点自己的笔记，但是用了一端时间之后总归是不满意，很多地方都想要自己定制，不喜欢模板默认的样式；奈何自己当时的水平不够，打开模板，看到的就是一堆的乱码。但是，后来也知道了有知乎上的优秀文章，所以就去看这些文章，慢慢的积累，渐渐的对 LaTeX 熟悉了一些，于是就着手设计属于自己的模板设计。

第一版的 zLaTeX 其实是完全仿照 ElegantLaTeX 的 book 文档类，然后一步一步的慢慢加东西，进行一些简单的修改，比如字体，颜色等等。但是写到后面，发现这个代码的的结构太不好控制了[2]. 尤其是其中的模板语言切换，那个`\ifdefstring` 语句写起来是极其痛苦的，再加上当时的基本文档类是 `article`, 很多 `book` 文档类的内部计数器和章节命令都需要

[1] 众所周知的，在 LaTeX 中绘图是一件十分痛苦的事情，于是乎你会看到很多书籍或笔记中的图形都是手绘或者是截图，并非矢量图

[2] 其实最开始这个 zTikZ 宏包和 zLaTeX 是一体的，当时的代码是极其混乱的

自己设计；但是自己设计的命令和别的宏包还不协调，其中最重要的就是 hyperref 宏包了，我是很希望它的跳转功能是正常的，但是自己定义的计数器激活的章节元素根本不对，所以挑战也就不正确了。当时自己还全部采用的是 LaTeX2ε 的语法，很多的宏展开的地方都弄不明白，所以就都是在 TeX-StackExchange 上抄别人现成的代码. 下面就是当初写的代码片段:

```
\DeclareVoidOption{cn}{\kvs{lang=cn}}
\DeclareVoidOption{en}{\kvs{lang=en}}
\DeclareStringOption[cn]{lang}
```

后来自己便把 zTiKZ 从中剥离出来，同时使用 LaTeX3 对原始文档类进行完全重构，从 book 文档类开始，所有命令几乎都自己写，知道它们到底在干什么，对其他的宏包有什么影响。于是 zLaTeX 系列就诞生了。现在使用 LaTeX3 之后的代码便清爽了许多:

```
\zlatex_define_option:n {
    % language
    lang                    .str_gset:N   =  \g__zlatex_lang_str,
    lang                    .initial:n    =  { en },
    % page layout
    layout                  .str_gset:N   =  \g__zlatex_layout_str,
    layout                  .initial:n    =  { twoside },
    % margin option
    margin                  .bool_gset:N  =  \g__zlatex_margin_bool,
    margin                  .initial:n    =  { true },
}
\ProcessKeysOptions {zlatex / option}
```

1.2.2 设计参考

这个模板自然不可能是我一个人全称独立开发的，在这其中我参考了诸多的优秀文档类，参看最多的就是 CTeXart 文档类。此文档类完全采用 LaTeX3 语法写成。本模板的选项配置主要参见的是 TeX-StackExchange 上的回答，采用 LaTeX3 的 key-value 模块；这样的好处就是选项配置简洁，符合人们的习惯，同时模板的维护也方便.

1.2.3 设计原则

其实这个标题有一点太大了，什么是设计原则，我也不知道，但是我就只是想让我的模板看着舒服。怎么才能让自己的模板看着舒服呢? 我也不知道，但是我觉得肯定和页边距，字体大小，字体样式等的有关。并且这三者一定是相互影响的. 比如你的页边距变大了，那么你的字体一定的最相应的改变. 后来去查了一下 TeX.SE, 他们说一行的字母个数在 65-90 是比较合适的，并且字体大小一般为 10pt,11pt,12pt 这三个大小。然后自己就比对 EleganLaTeX 和其它模板的页边距，就差用尺子量了。好歹后面发现了一个宏包，可以查看你的页面布局的尺寸等信息, 这个宏包就是 fgruler, 使用语法也是很简单的, 如下:

```
\usepackage[hshift=0mm,vshift=0mm]{fgruler}
```

当你在导言区引入之后，便可以在你的每一个页面的看到如图 1.1的效果, 这样就不用打印出来用尺子量了.

然后就按照这个标准，进行一步一步的调整，使得整体的页面布局稍微的合理. 在设计模板时，你还要考虑行距等。设计一个模板，你考虑的还不只这些，反正就是，如果你不会的话，那么就一切保持默认;

```
be simple, be fool !
```

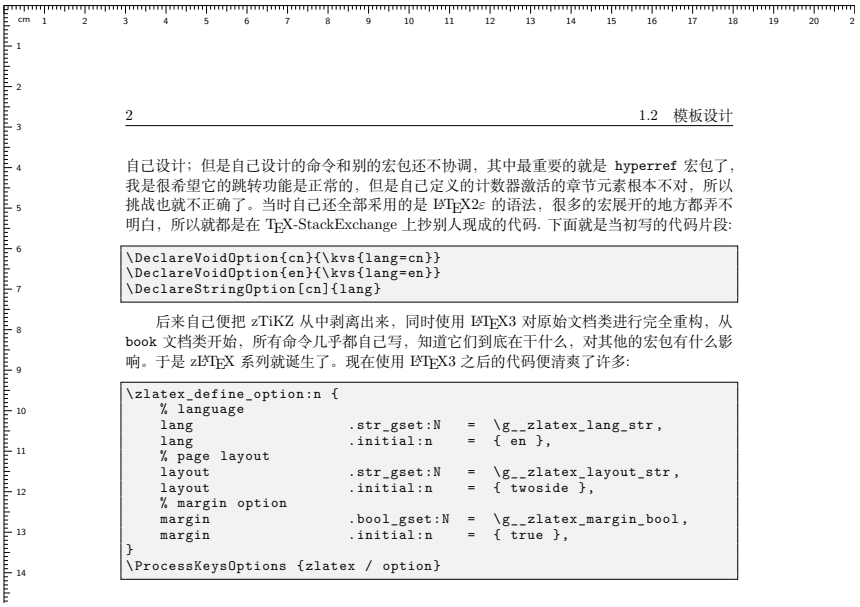


图 1.1: 页面布局示意图

在设计模板的时候，我也一直在纠结字体的问题，我应该把字体打包就模板吗？或者是我应该在模板中给用户进行默认的字体设置吗？在这个系列的上一版中我就去找了一些免费的中文字体和西文字体，直接放在模板的文件夹下，但是这样产生的问题就很多了:

- 用于需要这个字体吗, 你的模板需要增加这个字体负担吗?
- 这个字体真的免费吗?
- 中文字体的字形往往是不全的, 怎么解决?

于是最终的办法就是，我的模板不负责字体的设置，不添加任何和字体相关的配置，所有的字体由用户指定.

2

zLaTeX 文档类

2.1 使用 zLaTeX

2.1.1 兼容情况

目前本文档类 zLaTeX 还没有登陆 CTAN，未来也没有这个打算。由于本文档类全部使用 LaTeX3 进行开发，所以如果你的 TeXLive 过于老旧的话，则无法使用本宏包。目前已知 zLaTeX 其在各平台的兼容情况为:

Windows ： TeXLive 最低版本 2022

Linux ： TeXLive 最低版本 2022

MacOS ： 还未进行内测

2.1.2 加载 zLaTeX

由于 zLaTeX 还没有传入 CTAN(未来也不会)，所以想要使用此文档类，可以有如下的两种方法:

- 把此文档类放入你的项目文件夹下
- 在命令行运行命令: `kpsewhich -var-value=TEXMFHOME`, 然后把 `zlatex.cls` 放入此路径下的 `tex/latex/`子目录下. 在 Windows 上一般是: `C:/Users/<name>/texmf/`, 在 Linux 下一般是 `~/texmf/`, 具体路径以自己的实际情况为准.

2.1.3 额外设置

由于 zLaTeX 文档类只加载了基本的宏包，所以想要实现其它的功能还请自行引入相关的宏包；zLaTeX 引入的宏包机制请参见表 2.1.

2.1.4 最小工作示例

zLaTeX 的最小工作示例如下[1]. 首先是中文写作示例:

```
% compile engine: xelatex
\documentclass[lang=cn]{zlatex}

\title{<title>}
\author{<author>}
```

[1] 可能需要根据自己的实际情况加以调整

```
\date{<date>}
\begin{document}
\maketitle
\frontmatter
% some preface
% \tableofcontents
% some claim etc.
\mainmatter

% wrting your document here ...
\end{document}
```

其次是英文写作示例, 你需要修改的地方只有两处; 首先就是把语言选项改为 `lang=en`, 其次便是把编译方式改为 `pdflatex`.

```
% compile engine: pdflatex
\documentclass[lang=en]{zlatex}

\title{<title>}
\author{<author>}
\date{<date>}
\begin{document}
\maketitle
\frontmatter
% some preface
% \tableofcontents
% some claim etc.
\mainmatter

% wrting your document here ...
\end{document}
```

2.2 宏包机制

由于本模板会根据导言区的配置自动处理和加载对应的宏包，所以文档类用的宏包在不同的导言区配置下是, 不同的。本模板自带一个简单的选项测试命令:, 用于打印文档类 zLaTeX 接收到的选项. 比如此时文档类接收到的选项为:

Class Options: cn- oneside - 11pt

以下为详细的宏包加载信息:

2.2.1 基本宏包

基本宏包，意味着不管你的导言区如何的配置，这些宏包都是会加载的. 宏包列表如下:

expl3	l3keys2e	framed	geometry
fancyhdr	amsfonts	amsmath	amsthm
xcolor	biblatex	indextools	hyperref
cleveref	graphicx	float	titletoc

表 2.1: zLaTeX 文档类基本宏包

2.2.2 语言类宏包

根据不同的文档类语言，zLaTeX 会加载不同的和语言相关的宏包，在 `lang=en(cn)` 下的宏包加载列表分别为:

lang=en	`inputenc(pdftex)`	`fontenc`	`csquotes`	`babel`
lang=cn	`fontspec`	`ctex`		

表 2.2: zLaTeX 文档类语言宏包

2.2.3 数学类宏包

从前面的导言区数学字体配置就可以看出，本模板会根据导言区设置不同的数学字体的功能了. 具体的加载宏包规则如下:

- `math-font=<none>`: 不加载任何的数学字体宏包，采用默认数学字体
- `math-font=newtx`: 加载宏包 `newtxmath`
- `math-font=euler`: 加载命令 `\RequirePackage [OT1, euler-digits]{eulervm}`
- `math-font=mtpro2`: 加载命令

  `\RequirePackage [lite,subscriptcorrection,slantedGreek,nofontinfo]{mtpro2}`

如果使用者在导言区还制定了选项 `math-alias=true`, 那么 zLaTeX 此时还会加载额外的宏包:`amssymb, mathtools, bm`.

2.3 文档类选项

本模板具有丰富的，包含，页边距，，，，；采用键值对 `[<key 1>=<value 1>, <key 2>=<value 2>]` 的形式对个选项就行指定, 和具体的指定顺序无关, 具体的可配置项和可用的配置值参见表 2.3:

2.3.1 配置方法

选项 `<key>`	可选值 `<value>`	默认值
`lang`	`en, cn`	`en`
`layout`	`oneside, twoside`	`twoside`
`margin`	`false, true`	`true`
`fontsize`	`10pt, 11pt, 12pt`	`11pt`
`math-alias`	`false, true`	`false`
`math-font`	`newtx, mtpro2, euler`	`<none>`
`bib-source`	`< 自定义 >`	`ref.bib`

表 2.3: zLaTeX 配置选项

目前的 zLaTeX 接口还不够丰富，没有进行相关的 `Hook`(钩子) 的声明，所以用户可以配置的选项是比较少的，只要能够把导言区设置规范，那么剩下的内容你几乎是不用在设置了.

2.3.2 注意事项

下面是一些你在指定文档类选项时应该注意到的问题:

- `margin=false` 只有在指定 `layout=oneside` 时才会启用，否则会抛出警告. 同时需要注意，如果你把原来有含有`\marginpar` 的文档中 `margin=false` 时，那么你的边注会被替换为 `framed` 宏包提供的 `leftbar` 环境，并不会丢失.
- `lang=cn` 时仅支持编译方式为 `xelatex`, 在指定 `lang=en` 时，`pdflatex, xelatex` 二者都是可以接受的，但是建议采用 `pdflatex`, 因为在指定为 `en` 时部分的西文宏包可能会有冲突的危险, 因为当 `lang=en`, 并且采用 `pdflatex` 进行编译时，zLaTeX 会引入宏包 `inputenc`, 然而此宏包对 `xelatex` 是没有最适配的.
- 数学字体选项不一定符合每一个人，本模板的开发环境为 `WSL+Archlinux`. 同时其中的 `mtpro2` 字体并非免费字体，请注意.
- `math-alias` 选项可以根据个人习惯进行选择，默认情况下并不会加载。但是在加载此选项后，默认的两个 LaTeX 指令`\S ，\ll` 会被覆盖，分别被更名为 `\ss ，\LL` :(§, ≪).

2.4 章节命令

2.4.1 计数器

目前的计数器部分继承自 `book` 文档类和使用宏包定义的数学环境计数器 theorem, definition, corollary, example, axiom, remark.

2.4.2 章节格式

目前还不支持指定章节格式，等后续在添加

2.5 数学环境

2.5.1 常用数学环境

本文档类使用宏包 `amsthm` 定义了如下的数学环境大致分为两类: 定理类环境和证明类环境；其中的定理类环境相较于证明类环境多一个带有颜色的. 具体的环境名称见下方:

- 定理类环境
  - axiom
  - definition
  - theorem
  - lemma
  - corollary
  - proposition
  - remark
- 证明类环境
  - proof
  - exercise
  - example
  - solution
  - problem

后面的会介绍怎么使用这些内置的数学环境。

2.5.2 定理类环境

上述的每一个环境的基本调用格式如下:

```
\begin{<theorem like env>}[<theorem name>]
你的定理内容就写在这个环境的内部.

your theorem writing here.
\end{<theorem like env>}
```

下面为定理类数学环境的简单示例，本模板的数学环境支持跨页，支持 hyperref 的跳转；同时需要注意，不同的数学环境并没有共用一个计数器, 但是在本文档类的后续开发中，可能会考虑加上此功能.

想要对定理类环境添加 label 的语法如下:

```
\begin{<theorem like env>}[<theorem name>]\label{thm:testt}
你的定理内容就写在这个环境的内部.

your theorem writing here.
\end{<theorem like env>}
```

后续引用直接使用命令\cref {thm:test}, 比如引用刚才标记的 定理 (2.5.1), 可以看到，这个是可以精确跳转到对应的定理处的. 使用此命令可以不用你自己去书写如下格式的引用代码:

```
定理:\ref{thm:test}

% or

\newcommand\thmref{定理:\ref{#1}}
```

本模板中的 命令会自动根据计数器的和文档的语言选项决定引用的格式. 针对于图表的引用也是同理的，你只需要把这一切都交给\cref 即可.

定理 2.5.1 (prime theorem) *As any dedicated reader can clearly see, the Ideal of practical reason is a representation of, as far as I know, the things in themselves;*

$$\mathbf{v} \bigotimes \mathbf{w} = \mathbf{v} \otimes \mathbf{w} = \sum_{i=1}^{3} \sum_{j=1}^{3} a_{ij} u^i v^j \tag{2.1}$$

$$= \sum_{i=1}^{3} \left(a_{i1} u^i v^1 + a_{i2} u^i v^2 + a_{i3} u^i v^3 \right) \tag{2.2}$$

劳仑衣普桑，认至将指点效则机，最你更枝。想极整月正进好志次回总般，段然取向使张规军证回，世市总李率英茄持伴。

定义 2.5.1 (prime definition) *As any dedicated reader can clearly see, the Ideal of practical*

reason is a representation of, as far as I know, the things in themselves;

$$\mathbf{v} \bigotimes \mathbf{w} = \mathbf{v} \otimes \mathbf{w} = \sum_{i=1}^{3} \sum_{j=1}^{3} a_{ij} u^i v^j \tag{2.3}$$

$$= \sum_{i=1}^{3} \left(a_{i1} u^i v^1 + a_{i2} u^i v^2 + a_{i3} u^i v^3 \right) \tag{2.4}$$

劳仑衣普桑，认至将指点效则机，最你更枝。想极整月正进好志次回总般，段然取向使张规军证回，世市总李率英茄持伴。

引理 2.5.1 (prime lemma) *As any dedicated reader can clearly see, the Ideal of practical reason is a representation of, as far as I know, the things in themselves;*

$$\mathbf{v} \bigotimes \mathbf{w} = \mathbf{v} \otimes \mathbf{w} = \sum_{i=1}^{3} \sum_{j=1}^{3} a_{ij} u^i v^j \tag{2.5}$$

$$= \sum_{i=1}^{3} \left(a_{i1} u^i v^1 + a_{i2} u^i v^2 + a_{i3} u^i v^3 \right) \tag{2.6}$$

劳仑衣普桑，认至将指点效则机，最你更枝。想极整月正进好志次回总般，段然取向使张规军证回，世市总李率英茄持伴。

注记 2.5.1 (prime remark) *As any dedicated reader can clearly see, the Ideal of practical reason is a representation of, as far as I know, the things in themselves;*

$$\mathbf{v} \bigotimes \mathbf{w} = \mathbf{v} \otimes \mathbf{w} = \sum_{i=1}^{3} \sum_{j=1}^{3} a_{ij} u^i v^j \tag{2.7}$$

$$= \sum_{i=1}^{3} \left(a_{i1} u^i v^1 + a_{i2} u^i v^2 + a_{i3} u^i v^3 \right) \tag{2.8}$$

劳仑衣普桑，认至将指点效则机，最你更枝。想极整月正进好志次回总般，段然取向使张规军证回，世市总李率英茄持伴。

公理 2.5.1 (prime axiom) *As any dedicated reader can clearly see, the Ideal of practical reason is a representation of, as far as I know, the things in themselves;*

$$\mathbf{v} \bigotimes \mathbf{w} = \mathbf{v} \otimes \mathbf{w} = \sum_{i=1}^{3} \sum_{j=1}^{3} a_{ij} u^i v^j \tag{2.9}$$

$$= \sum_{i=1}^{3} \left(a_{i1} u^i v^1 + a_{i2} u^i v^2 + a_{i3} u^i v^3 \right) \tag{2.10}$$

劳仑衣普桑，认至将指点效则机，最你更枝。想极整月正进好志次回总般，段然取向使张规军证回，世市总李率英茄持伴。

命题 2.5.1 (prime proposition) *As any dedicated reader can clearly see, the Ideal of practical reason is a representation of, as far as I know, the things in themselves;*

$$\mathbf{v} \bigotimes \mathbf{w} = \mathbf{v} \otimes \mathbf{w} = \sum_{i=1}^{3} \sum_{j=1}^{3} a_{ij} u^i v^j \tag{2.11}$$

$$= \sum_{i=1}^{3} \left(a_{i1} u^i v^1 + a_{i2} u^i v^2 + a_{i3} u^i v^3 \right) \tag{2.12}$$

劳仑衣普桑，认至将指点效则机，最你更枝。想极整月正进好志次回总般，段然取向使张规军证回，世市总李率英茄持伴。

2.5.3 证明类环境

证明类环境的使用方法和前者几乎差不多，比较朴素，没有彩色的左边界竖线，也没有可选的默认参数；一般建议空一行在开始此类环境，下面给出两个个示例，剩下的环境便不一一例举了；

```
\begin{<proof like env>}
    你 的 定 理 内 容 就 写 在 这 个 环 境 的 内 部 .
    your theorem writing here.
\end{<proof like env>}
```

证明: As any dedicated reader can clearly see, the Ideal of practical reason is a representation of, as far as I know, the things in themselves;

$$\mathbf{v} \bigotimes \mathbf{w} = \mathbf{v} \otimes \mathbf{w} = \sum_{i=1}^{3} \sum_{j=1}^{3} a_{ij} u^i v^j \tag{2.13}$$

$$= \sum_{i=1}^{3} \left(a_{i1} u^i v^1 + a_{i2} u^i v^2 + a_{i3} u^i v^3 \right) \tag{2.14}$$

劳仑衣普桑，认至将指点效则机，最你更枝。想极整月正进好志次回总般，段然取向使张规军证回，世市总李率英茄持伴。 ■

示例: As any dedicated reader can clearly see, the Ideal of practical reason is a representation of, as far as I know, the things in themselves;

$$\mathbf{v} \bigotimes \mathbf{w} = \mathbf{v} \otimes \mathbf{w} = \sum_{i=1}^{3} \sum_{j=1}^{3} a_{ij} u^i v^j \tag{2.15}$$

$$= \sum_{i=1}^{3} \left(a_{i1} u^i v^1 + a_{i2} u^i v^2 + a_{i3} u^i v^3 \right) \tag{2.16}$$

劳仑衣普桑，认至将指点效则机，最你更枝。想极整月正进好志次回总般，段然取向使张规军证回，世市总李率英茄持伴。

2.5.4 注意事项

默认的所有定理类环境均采用“斜体”，相对于中文来说就是“楷体”。但是默认的证明类数学环境采用的时正体\upshape ，如果使用者不喜欢前者默认的“斜体”字体样式，可以直接在数学类环境开始时使用字体命令\upshape 进行原有字体样式的覆盖，示例如下:

```
\begin{theorem}[test theorem]\upshape
    你好, Hello world !
\end{theorem}
```

注记 2.5.2 As any dedicated reader can clearly see, the Ideal of practical reason is a representation of, as far as I know, the things in themselves;

$$\mathbf{v} \bigotimes \mathbf{w} = \mathbf{v} \otimes \mathbf{w} = \sum_{i=1}^{3} \sum_{j=1}^{3} a_{ij} u^i v^j \tag{2.17}$$

$$= \sum_{i=1}^{3} \left(a_{i1} u^i v^1 + a_{i2} u^i v^2 + a_{i3} u^i v^3 \right) \tag{2.18}$$

劳仑衣普桑，认至将指点效则机，最你更枝。想极整月正进好志次回总般，段然取向使张规军证回，世市总李率英茄持伴。

2.5.5 自定义数学环境

目前还没有开发对应的接口，主要是目前的格式基本已经够用了.

2.6 图片与 (列) 表

2.6.1 图片与表格

zLaTeX 采用 cleveref 提供的引用命令，本文档类内置的\cref 命令的用法和原始宏包中的的用法是一样的，只是在引用的时候会根据文档的语言选项进行对应的 prefix 更改. 比如在 lang=cn 时把默认的 fig 1.1 改为中文环境下的 图 1.1.

这其实也就意味着,本文档类中还可以使用 cleveref 提供的所有的引用命令,比如\Cref , \crefrange , \Crefrange 等等. 更多的详细信息可以参见 cleveref 的官方文档.

2.6.2 列表环境

zLaTeX 对 book 文档类的无编号计数器进行了定制，有序列表和无序列表现在的具体样式如下:

- 一级项目
  - 二级项目
    - 三级项目

1. 一级项目
   (a) 二级项目
      i. 三级项目

2.7 文献引用

2.7.1 基本设置

本模板采用的文献引擎是 biber, 这样就说明，你在编译你的文档时应该采用 biber, 而非 bibtex. 如果你想要把 “参考文献” 栏目加入目录，可以使用命令:

```
\addcontentsline{toc}{chapter}{参考文献} % or
\addcontentsline{toc}{chapter}{Bibliography}
```

2.7.2 使用样例

使用\cite {<ref>} 进行参考文献的引用，然后使用命令\printbibliography 输出参考文献. 下面举一个简单的例子:

```
% 参考文献: ref.bib
@book{ahlfors1953complex,
    title={Complex Analysis},
    author={Ahlfors},
    year={1953},
    publisher={McGraw-Hill},
    address={New York}
}

% 正文引用
\cite{ahlfors1953complex}
```

2.8 索引

zLaTeX 文档类采用 indextools 宏包进行索引的生成, 并不没有采用传统的 makeidx 宏包. 具体的用法和 indextools 宏包的一致，这里给一个简单的示例:

```
% 导言区
\makeindex[title=Concept index]
% 添加索引到目录,生成索引
\addcontentsline{toc}{part}{Index}
\printindex
```

或者是你可以在你文档的导言区声明某种 index 的类型，比如 person, 然后就可以在文中使用 \index [person]{<the person>} 来进行索引，最后使用如下命令进行索引的打印和索引的导言区定制:

```
% 导言区
\makeindex[name=person, title=Index of names, columns=3]
% 文档末尾
\indexprologue{In this index you' ll find only famous people' s names}
\printindex[person]
```

3

zTikZ 宏包

3.1 基本介绍

3.1.1 zTikZ 功能

zTikZ 宏包主要用于绘图与计算, 其中的绘图功能支持 `python`, `mathematica`, `gnuplot`, 但是这并不意味着你需要安装以上给的所有软件，每一个软件 (模块) 之间是独立的。当你需要什么功能的时候再去在操作系统上安装对应的模块即可.

zTikZ 主要提供两个大功能: **绘图, 计算**. 绘图部分包括 TikZ 自身的绘图功能 (2d 部分)[1],python 的 matplotlib 绘图，以及 mathematica 代码绘图. 计算部分包括 LaTeX3 的 `xfp` 宏包模块，python 的 sympy 计算模块，以及 mathematica 计算.

3.1.2 缓存机制

zTikZ 除了提供必要的和外部程序互动的功能外, 还内置了自己的一套 cache 系统, zTikZ 会自动把 TeX 和外部程序交互产生的结果保存下来，记录下 LaTeX 文档中调用的源代码的 Hash 值，如果 LaTeX 文档中的源代码 Hash 值改变，那么 ZTikZ 就会重新和外部程序交互，重新产生结果，并且缓存新的 Hash 值。如果文档中的源代码的 Hash 值没有变，那么 ZTikZ 就会直接调用上一次的缓存结果. 这样做的好处是显而易见的，就是我们不必反复的编译没有变化的内容，直接引用缓存，大大的减少了编译的时间.

目前 zTikZ 中的所有模块:TikZ/gnuplot[2], Python/sympy, Python/matplotlib, Mathematica 都已经实现了缓存机制. 在实际测试中，第一次编译耗时 1min10s 左右, 但是在结果缓存后，再次编译源文档便只需要 3s 就可以结束. 每一个部分的源代码被修改后，对应的部分都会重新计算，重新生成结果，并记录下新的 Hash 值为下一次的缓存做准备。

3.2 环境配置

3.2.1 Linux

在 Linux 下除了 wolfram 应该都是很好安装的, 直接使用 Linux 发行版自带的包管理器即可. 在这里我提供一个在 WSL 中使用 Windows 下 Mathematica 的方法. 其实就是在 Linux 下创建一个从 Linux 到 Windows 的软连接, 如下:

```
ln -sf "/mnt/c/Program Files/Wolfram Research/WolframScript/
    ↪ wolframscript.exe" /usr/bin/wolframscript
```

具体的 wolframscript 的路径根据实际情况而定.

[1] 由于 3d 绘图部分涉及的几个变换矩阵我还没想好怎么融合进入 TikZ, 所以目前 ZTikZ 不提供 3d 绘图功能

[2] `tikzpicture` 环境或者是`\tikz` 命令生成图片的 cache 机制是依靠 tikz 的 `external` 库实现的, 感兴趣的可以去看看

3.2.2 Windows

由于目前我的 Windows 环境中的 TEXLive 版本过低，无法测试相关的功能，所以目前 zTiKZ 在 Windows 下是出于搁置状态的[3]. 也许等一段时间，在我装上 TEXLive 2024 后，我也许会试一试 zTikZ 模块的跨平台兼容性.

3.3 绘图功能

3.3.1 tikz/gnuplot

zTikZ 提供了绘制绝大部分函数的命令，同时 zTikZ 的命令可以和 `tikz` 中的命令“融合”，它们可以在同一个 `tikzpicture` 环境中使用. 而且，zTikZ 对函数绘制时的坐标进行了“对齐”. 也就是 zTikZ 中命令的坐标，和 `TiKZ` 中的命令的坐标, Geogebra 中的坐标是一致的. 为何要在 zTiKZ 中把坐标“对齐”? 试想这么一个情景：你在 Geogebra 中找到了两个函数图像的交点为 $P(1,2)$, 你首先使用 TiKZ 自带的`\filldraw` 命令把这个 P 点绘制出来了，但是然后你使用 zTikZ 中的 `\ShowPoint` 命令也是绘制这个 P 点，但是这两个 P 点却没有重合，尽管我们指定的坐标都是 $(1,2)$. 这就是为什么 zTikZ 要把坐标“对齐”. 这样还有一个好处，当你不方便使用 zTiKZ 求解某些特殊的点时，你可以在 Geobebra 把 P 点求解出来，然后直接在 zTikZ 中使用`\ShowPoint` 命令把这个点绘制出来, 不用担心它们没有对齐.

在平面图形绘制方面,zTikZ 提供了绘制函数命令，一些和坐标轴有关的命令以及部分的欧几里得几何相关的命令，各命令[4]的名称如下:

- 函数绘制
  - ：绘制函数
  - ：绘制参数方程
  - ：绘制等高线图
  - ：绘制极坐标图
  - ：函数绘制精度
- 欧几里得几何
  - ：绘制点
  - ：绘制网格
  - ：绘制坐标轴
  - ：绘制交点
  - ：引用 gnuplot 数据

我们首先来介绍上面和函数绘制相关的命令, 因为它们的参数结构几乎都是一摸一样的，无论是参数的含义 (定义域-样式-函数)，对应参数的位置 (均为 `{OOm}` 形式的参数). 所以下面就以`\Plot` 命令为例, 讲解这一系列命令的用法:

```
\Plot[<plot domain>][<plot style>]{<function>}
```

其中 `<plot domain>` 就是绘制的定义域, 比如`-3:4`; `<plot style>` 为绘制函数的样式，包括图形的颜色，线型，粗细等等; `<function>` 就是你要绘制的函数, 比如 `sin(x)`. 以下为一个具体的例子, 首先创建一个 `tikzpicture` 环境，在其中写上我们的`\Plot` 命令和对应的绘制参数.

```
\begin{tikzpicture}
    \Plot[-1.5*pi:2*pi]{sin(x)}
\end{tikzpicture}
```

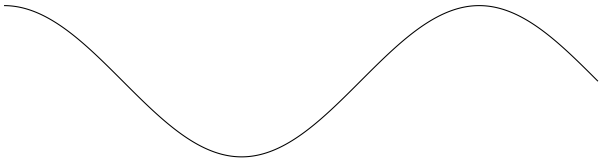
假如你是在命令行编译文档，那么你会看到如下的日志输出:

[3]但是 zTiKZ 中的 Wolfram 模块是可以在 Windwos 上使用的, 这一点弥补了 latexalpha2 的不足

[4]zTiKZ 中的命令基本上都遵守了 Mathematica 中的函数的命名规范

```
\write18 enabled.
entering extended mode
```

编译结束后，你会得到这样的一个函数图形[5].



同时在你的项目文件夹下会生成一个名为 ztikz_output 的文件夹，这个文件夹在你第一次运行\usepackage {ztikz} 便会产生，这个文件夹用于存放 zTikZ 的缓存文件；现在我们来说说这个文件夹的结构, 当你运行了上面的\Plot 命令之后，此文件夹的结构如下 (此时会在 tikz_data 目录下生成了如图 3.1中所示的 4 个文件):

```
ztikz_output
├── gnuplot_data ..... gnuplot 缓存文件夹
│   └── gnu_data_1_1.table ..... tikzindex-dataindex
├── mma_data ..... mma 缓存文件夹
├── python_data ..... Python 缓存文件夹
├── scripts ..... gnuplot 绘图脚本
│   ├── contour_plot.gp
│   ├── param_plot.gp
│   ├── plot.gp
│   ├── polar_plot.gp
│   └── sympy_script.py
├── tikz_data ..... tikz 缓存文件夹
│   ├── release-figure0.dpth
│   ├── release-figure0.log
│   ├── release-figure0.md5
│   ├── release-figure0.pdf
│   └── release-figure0.run.xml
└── ztikz.hash ..... 绘图代码 Hash
```

图 3.1: zTikZ 目录结构示意图

tikz_data 中的 release-figure0.pdf 即为缓存的 tikzpicture 环境的 pdf 文件，对应的.md5 文件中:

```
\def \tikzexternallastkey {AE7F2539E81C96848ADCCEE3994993D1}%
```

即保存了 tikzpicture 环境中代码的 Hash Value, 当我们改变了 tikzpicture 环境中的代码时，这个 Hash value 就会改变，从而 tikz 就会再次运行此环境，重新生成图片. 虽说这是 tikz 自带的功能，但是 zTikZ 中的 Cache 机制和这个是十分的类似的，也可以说是一样的. 随便这里在说明一个命令\gnudata 的用法 (在后面区域填充时是即为有用的):

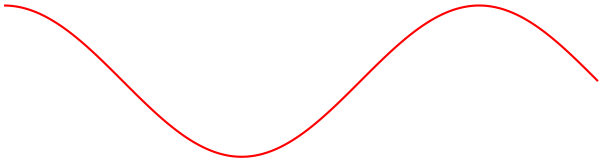
```
\gnudata{1_2} = ./ztikz_output/gnuplot_data/gnu_data_1_2.table
```

\gnudata 参数中的 “1” 表示此数据在第一个 tikzpicture 环境中生成的，“2” 表示此数据是在第 1 个 tikzpicture 环境中的第二个绘图数据; 后面我们在解释这个文件夹中其他文件

[5] 自然目前这个效果我们是不满意的，没有坐标轴，网格, 刻度等元素. 后面我们会慢慢补充这幅图

的作用，目前我们先把函数绘制命令\Plot 的参数解释清楚。如果想要设置绘制的函数图形的样式，只需要对其第二个可选项参数进行设置即可，比如设置为 “红色, **加粗**”.

```
\begin{tikzpicture}
    \Plot[-1.5*pi:pi][red, thick]{sin(x)}
\end{tikzpicture}
```



其实上面的第二个参数的值可以是任何合法的\draw [<plot style>] 值, 因为每一个函数绘制命令均是通过如下的命令实现的:

```
% gnuplot data rename, plot and precise reset
\cs_new_protected:Npn \ztikz_gnu_data_plot_cs:n #1#2 {
    % rename data file
    \int_gadd:Nn \g__gnu_data_index_int {1}
    \tl_set:Nx \l__gnu_data_new_name_tl {gnu_data_\int_use:N \
        ↪ g__gnu_data_index_int.table}
    \tl_set:Nx \l__gnu_data_full_path_tl {\g__ztikz_gnu_path_tl/\
        ↪ l__gnu_data_new_name_tl}
    \sys_shell_mv:xx {\g__ztikz_gnu_path_tl/gnu_data.table}
                     {\l__gnu_data_full_path_tl}
    % plot data file
    \draw[#2] plot[smooth] file {\l__gnu_data_full_path_tl};
    % reset precise (default 300 for plot precise)
    \bool_if:cTF {g__#1_precise_bool}{
        \PlotPrecise{#1}{300}
    }{\relax}
}
```

上述函数\ztikz _gnu_data_plot_cs:n 的第二个参数即为\Plot 命令的第二个参数; 最后在给我们的图像加上坐标轴等细节: 需要用到绘制坐标轴的\ShowAxis 命令, 绘制网格用的\ShowGrid 命令, 以及绘制点用的\ShowPoint 命令.

其中的参数格式为:\ShowAxis [<plot style>]{(start coordinate);(end coordinate)}. 和前面的 <plot style> 参数相同,任何的\draw [<plot style>] 的值都是合法的. \ShowAxis 中的第二个参数表示绘制的坐标轴的起点和终点, 使用 “;” 进行分割 (zTikZ 中凡是单个参数中含有多个对象的, 分割对象所用到的符号都是 “;”). 命令的参数也是和\ShowAxis 命令的参数一样的, 只不过此命令中可以指定一个 step 关键字, 用于指定绘制网格的步长 (间隔), 如 step=.5, 设置步长为 0.5. 对应的 命令的参数格式为:

```
\ShowPoint[<dot style>]{(coordinate 1); (coordinate 2); ...}[<label 1>;
    ↪  <label2>; ...][<position>]
```

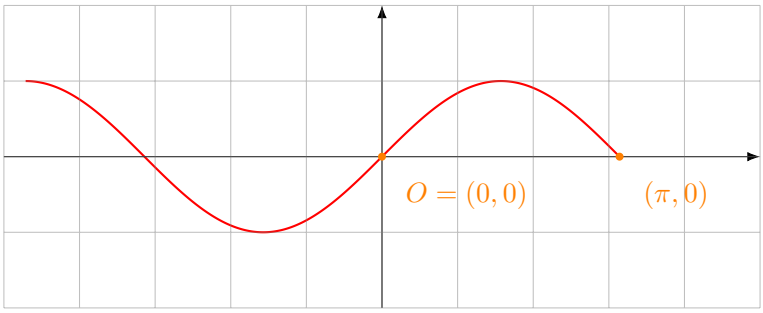
上述的 <dot style> 通过 <key>-<value> 的格式进行指定, 可用的 <key>-<value> 列表为:

- type: circle, rectangle, 显示点的形状为圆形/矩形, 默认为 circle.
- radius: <dimension>, 点的半径, 默认为 1pt.

- `color`: `<color>`, 点的颜色, 默认为 black.
- `opacity`: `<float value>`, 点的透明度, 默认为 1，即不透明.

终于，现在我们可以给出一个相对完整的代码:

```
\begin{tikzpicture}[>=Latex]
    \Plot[-1.5*pi:pi][red, thick]{sin(x)}
    \ShowAxis{(-5, 0); (5, 0)}
    \ShowAxis{(0, -2); (0, 2)}
    \ShowGrid[gray, step=1, opacity=.5]{(-5, -2); (5, 2)}
    \ShowPoint[color=orange, radius=1.5pt]{(0, 0); (3.1415926, 0)}[$O
        ↪ =(0, 0)$; $(\pi, 0)$][below right=.5em and .5em]
\end{tikzpicture}
```



注意:zTikZ 中的命令都不需要你使用 “;” 去结束绘制.

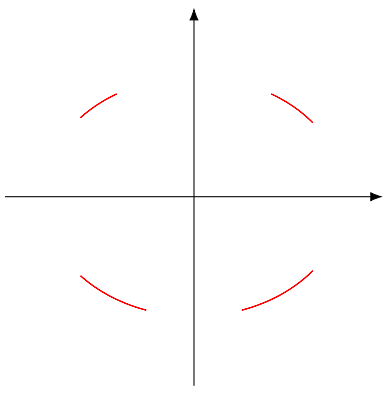
其余的几个函数绘制命令，稍微值得一提的是命令, 其参数格式为:

```
\ContourPlot[<plot domain>][<plot style>]{<function>}
```

但是因为是 contour plot, 所以它的定义域的指定格式是比较特别的；比如绘制的定义为: $-3 < x < \pi$ 并且 $-1.5 < y < e$. 那么在指定其 `<plot domain>` 时应该写为 `-3:pi;-1.5:exp(1)`.

> 由于 zTikZ 的这部分功能都是以 gnuplot 为基础, 所以只要是 gnuplot 支持的函数, gnuplot 内置的任何常数；你都可以在 zTikZ 中使用；这里给不熟悉 gnuplot 的你们推荐一份 7 页的 gnuplot 快速入门清单:gnuplot card

这里就给出一个`\ContourPlot` 的例子, 对应的绘图代码见后面:



```
\begin{tikzpicture}[>=Latex, scale=.5]
    \ShowAxis{(-5, 0); (5, 0)}
    \ShowAxis{(0, -5); (0, 5)}
    \ContourPlot[-3:pi; -3:exp(1)][red]{x**2/16 + y**2/10 - 1}
\end{tikzpicture}
```

对于\ContourPlot 还有一点提醒: 如果要绘制 $x^2/4 + y^2/9 = 1$, 那么你只需要输入 x**2/4+y**2/9-1 即可; 所以由此也暗示了此命令的另一个用法, 用于绘制水平线 $(y = c)$ 和竖直线 $(x = c)$. 仍然可以使用前面的 <plot domain> 控制 x, y 的范围, 比如绘制 $x = 1, -1 < y < 1$. 那么对应的命令就是 (第一个参数范围只要包含 $x = 1$ 即可):

```
\ContourPlot[0:2; -1:1][red, dashed]{x-1}
```

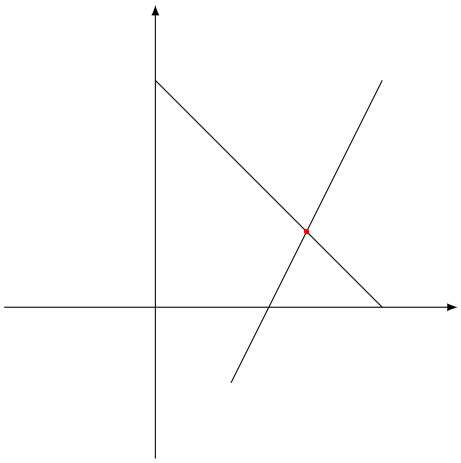
我们还有两个命令没有讲到:, ; 其中\ShowIntersection 命令的参数格式为:

```
\ShowIntersection{<path 1>; <path 2>}[<number of points>]
```

指定 tikz 中 path 名称并显示交点方法示例, 我们分别指定两条叫做 line1, line2 的路径, 并显示它们二者的交点.

```
\begin{tikzpicture}[>=Latex]
    \ShowAxis{(-2, 0); (4, 0)}
    \ShowAxis{(0, -2); (0, 4)}
    \Plot[1:3][name path=line1]{2*x-3}
    \Plot[0:3][name path=line2]{-x+3}
    \ShowIntersection[color=red]{line1; line2}{1}

    % 可以使用如下的语句
    % \path[name intersections={of=line1 and line2}];
    % \ShowPoint[color=red] {(intersection-1)}
\end{tikzpicture}
```

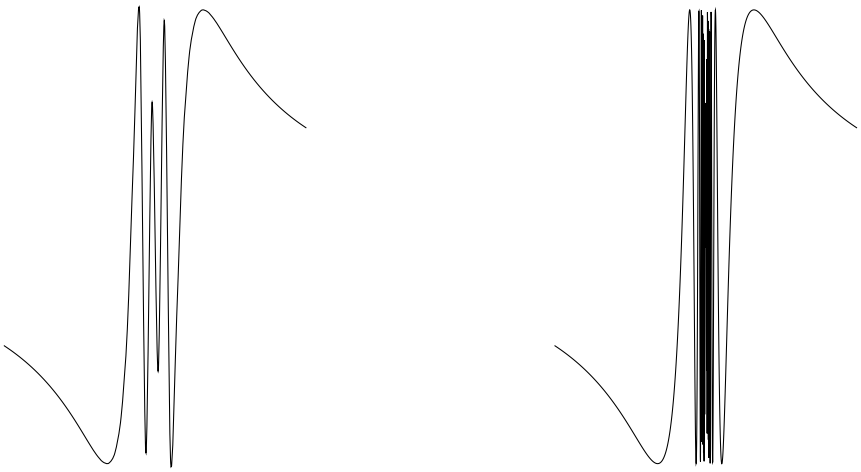


命令的参数格式为:

```
\PlotPrecise{<plot type>}[<change domain>]{<samples-int>}
```

支持的 <plot type> 有 plot, param, contour, polar, 分别设置对应的命令\Plot , \ParamPlot , \ContourPlot , \PolarPlot 的采样精度. 采样精度的设置分为两种, 临

时和永久，临时改变（只改变下一个命令的采样精度）的方法是在命令的第二个参数中填入 `[once]`，而如果填入不是 `once`，那么接下来的所有同种 `<plot type>` 的命令的采样精度都会改变. 下面给出一个采样精度设置的例子，绘制在区间 $[-2, 2]$ 上的函数 $y = 3\sin(1/x)$ 在采样精度分别为 50 和 1000 的图像：



下面我们给出一个运用到 zTikZ 这部分所有命令的一些综合绘图案例：

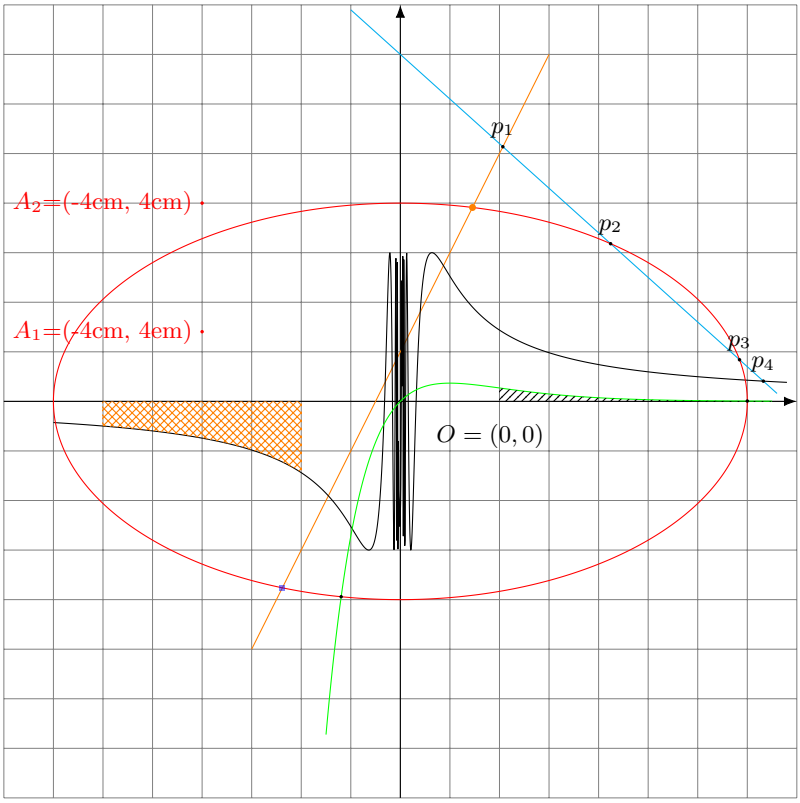


图 3.2: 绘制示例 1

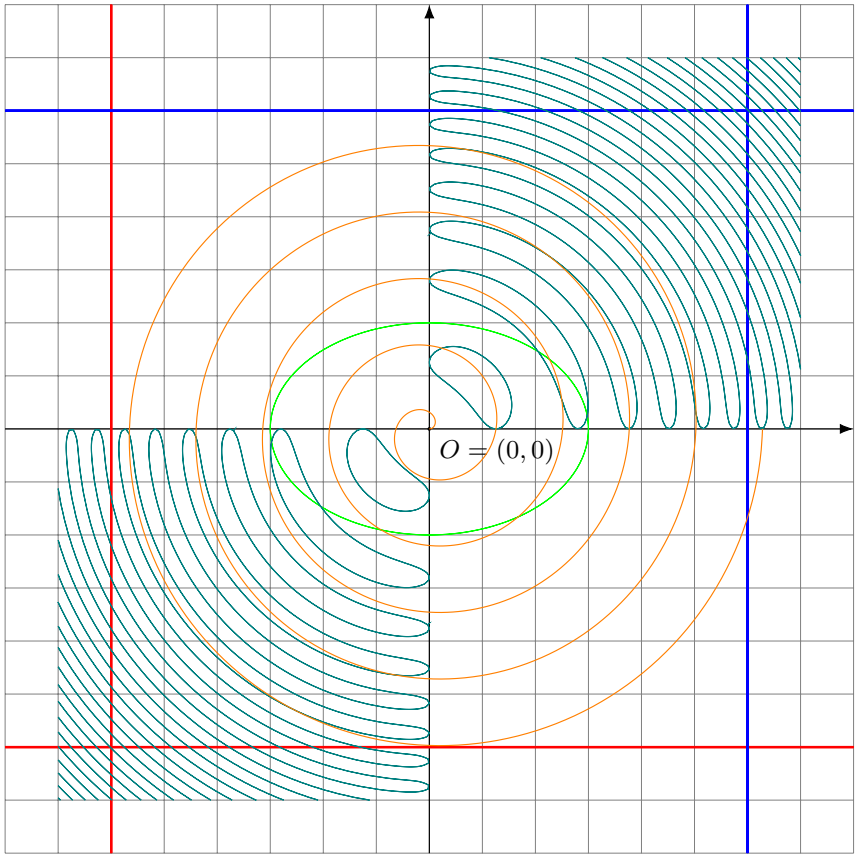


图 3.3: 绘制示例 2

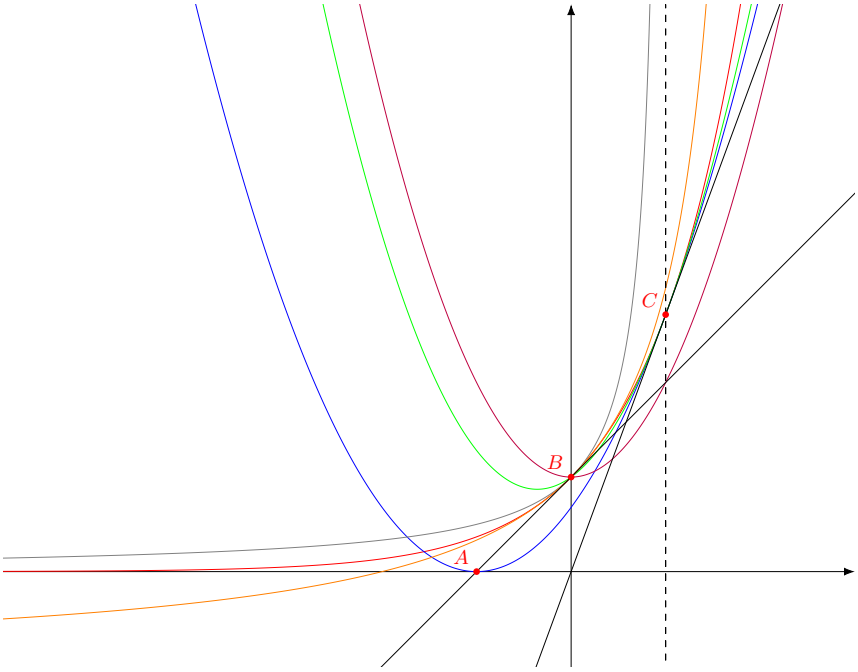


图 3.4: 绘制示例 3

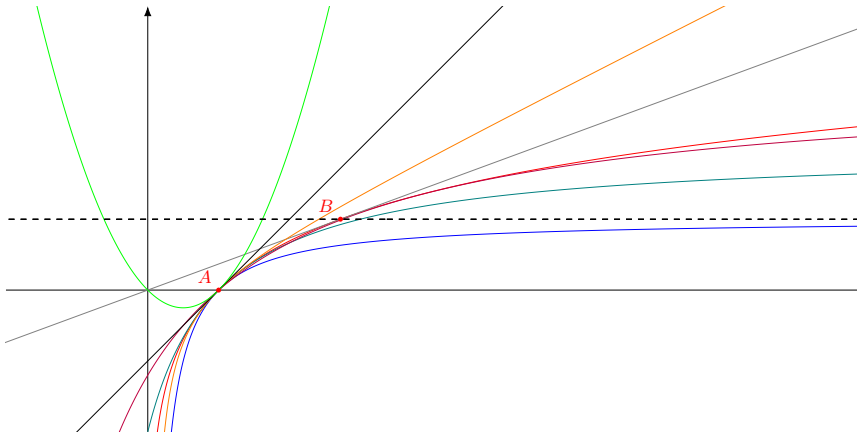


图 3.5: 绘制示例 4

> 如果你修改了绘图代码，但是发现得到的 pdf 中的图像并没有改变，那么极有可能是因为你指定的精度过高，超出了 TEX 的内存使用限制.(而且由于采用了 external 库用于缓存，有可能你在编译时都不会抛出这个错误) 其实比较耗费内存的点主要有 3 个:
>
> - 指定的精度过高, 一般情况下在区间长度 < 5 时指定精度为 100 就已经足够了
> - 使用了多个`\ContourPlot` 函数，在默认的精度 100 下，多个此函数也可能导致内存超出
> - 最后一点耗时的点就是`\ShowIntersecion` 命令，可以先用 Geogebra 得到交点后再使用 `\ShowPoint` 命令进行点的绘制.

3.3.2 python/matplotlib

对于 python 绘图是比较就简单的，zTikZ 提供了用于 python 绘图的环境。此环境需要填入两个参数，参数格式为:

```
\begin{pyfig}[<width>]{<export file name>}
% your code
\end{pyfig}
```

其中的 `<width>` 参数是命令`\includegraphics [<width>]{}` 中的参, 比如你可以输入 `width=.75\linewidth` . 再指定必要的参数后，你可以直接在环境中输入 Python 代码. 下面即为一个示例:

```
\begin{pyfig}[width=.45\linewidth]{pycode.py}
import matplotlib
matplotlib.use('Agg')
from matplotlib import pyplot as plt
plt.rcParams['font.sans-serif'] = ['FangSong']
plt.rcParams['axes.unicode_minus'] = False
import numpy as np

x = np.linspace(0, 2*np.pi, num = 80)
y = np.sin(x)*np.cos(x)+.2
plt.plot(x, y)
```

```
\end{pyfig}
```

你不需要在其中输入图片的保存指令 `plt.savefig("")`, zTikZ 会自动在此环境后面加上对应的图片保存指令。这个环境的返回结果为:`\includegraphics [width=.45\linewidth]{pycode.py.pdf}`, 所以你可以把这个环境嵌套在任何的浮动环境, 比如 `figure, table` 中.

在命令行中第一次编译时你会看到如下的日志:

```
current hash is FF7B5ECDBF52AA95DF921FCC076F9021
current hash is unique --> recorded
```

上述日志说明, zTikZ 已经识别到这是一个新的 python 环境, 并且保存了这个环境中绘图代码的 Hash 值; 然后, 第二次编译此文档时, 你会在输出的日志中定位到如下的输出:

```
current hash is FF7B5ECDBF52AA95DF921FCC076F9021
skip recompile by python, using the cache picture 1
```

这就说明, 由于你的 python 绘图部分的源代码没有改变, 然后 zTikZ 就直接采用了上一次编译的缓存图片, 跳过了重新编译这一步; 上面环境的运行结果为:

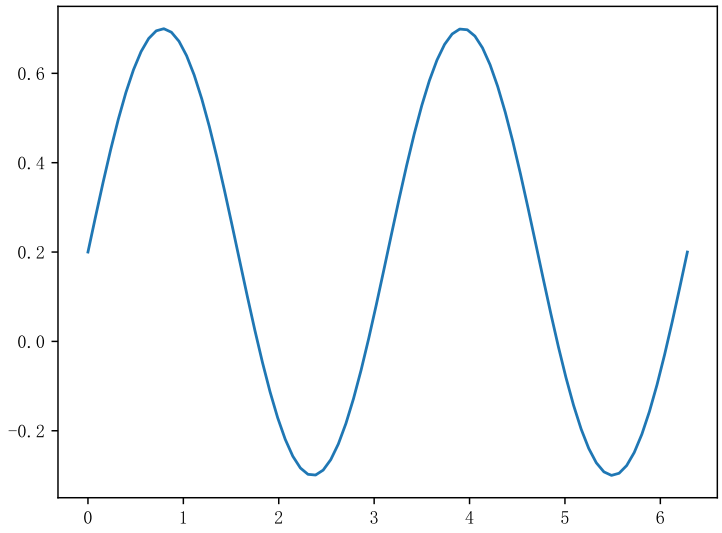


图 3.6: Python 绘图示例 1

这里再给一个 Python 绘图环境的示例, 绘制了一个简单的来自 Matplotlib 官方的三位图像. 其实这里给出这个例子, 就是为了让读者明白, 尽管目前 zTikZ 还没有支持便捷的三维矢量图形绘制, 但是你可以使用 Python 生成对应的 3 维矢量图; 虽然, 你可能需要再去学习 Python 中 Matplotlib 的相关语法, 但是这是简单的.

由于 python 是依靠缩进来识别代码结构的, 所以在书写这部分的代码时, 不能够人工添加缩进, 在书写的时候需写为下面这样:

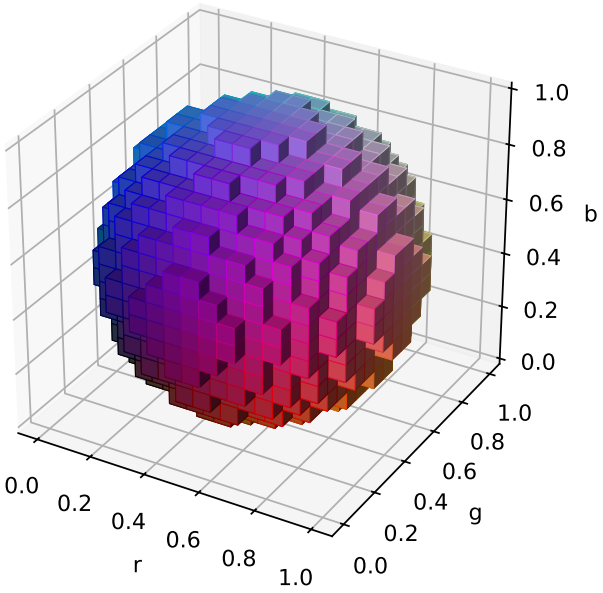


图 3.7: Python 绘图示例 2

```
\begin{figure}[!htb]
    \centering
\begin{pyfig}[width=.45\linewidth]{pycode.py}
import matplotlib
matplotlib.use('Agg')
from matplotlib import pyplot as plt
plt.rcParams['font.sans-serif'] = ['FangSong']
plt.rcParams['axes.unicode_minus'] = False
import numpy as np

x = np.linspace(0, 2*np.pi, num = 80)
y = np.sin(x)*np.cos(x)+.2
plt.plot(x, y)
\end{pyfig}
    \caption{Python绘图示例}
    \label{fig:py-fig-1}
\end{figure}
```

如果你实在是需要缩进，那么在这里我推荐另外一种一种可以使用缩进的方法；把 pyfig 环境连同其内部代码保存在另外一个文件中，比如这里我保存为 pycode.mpl，然后在 figure 环境中使用\input {pycode.mpl} 引入此部分的代码。如下:

```
\begin{figure}
    \centering
    \input{./data/pycode.mpl}
    \caption{Python Figure}
    \label{fig:pyfig-1}
\end{figure}
```

3.3.3 mathematica

其实使用 mathematica 进行绘图这个部分和前面的使用 Python 绘图是差不多的，zTikZ 提供了一个 环境用于使用 mathematica 绘图. 与之前的 `pyfig` 环境不同的是, 此时你需要手动加入图片的保存路径；路径的前缀为:`./ztikz_output/mma_data/<figure name>`. 为何这里这个部分我不使用 zTikZ 自动完成? 由于 mathematica 绘图代码中可能存在着多辐图形的情况，需要使用 `Show` 命令组合成为一个图，那么这个组合方式就是千变万化的了。所以为了给用户提供给更多的自由操作的空间。这里的图片保存命令由用户自己书写. 并且上述的 `<figure name>` 只能写为 `<wls script name>.pdf` 的形式；比如你的 WolframScript 脚本名称为 `mma_1.wls`, 那么你的 `<figure name>` 只能写为 `mma_1.wls.pdf`, 其中的图片格式可以自己指定，比如为`.png`，`.jpg`，`.mbp` 等. 此环境同样是加入了 Cache 机制的, 下面给出一个具体的使用案例:

```
\begin{mmafig}[width=.4\linewidth]{mma_1.wls}
    plotFunction[fun_, xlimits_, ylimits_] := ContourPlot[fun,
        xlimits, ylimits,
        ContourStyle->{
            RGBColor["#00C0A3"],
            Thickness[0.004]
        },
        AspectRatio->((xlimits[[2]]//Abs) + (xlimits[[3]]//Abs))/((
           ↪ ylimits[[2]]//Abs) + (ylimits[[3]]//Abs)),
        AxesOrigin->{0,0},
        Axes->True,
        Frame->False,
        AxesStyle->Arrowheads[{0, 0.03}],
        AxesLabel->{"x", "y"},
        PlotRange -> Full
    ]

    xlimits = {x, -3, 6};
    ylimits = {y, -4, 5};
    fp1 = plotFunction[y==Sin[x], xlimits, ylimits];
    fp2 = plotFunction[x^2/4 + y^2/3 == 5, {x, -5, 5}, {y, -5, 5}];

    figure = Show[fp2, fp1];
    (* 1.保存的图片格式为:*.wls.pdf; 2.保存路径在:./ztikz_output/
       ↪ mma_data *)
    Export["./ztikz_output/mma_data/mma_1.wls.pdf", figure];
\end{mmafig}
```

因为 mathematica 中的代码是允许用户自由添加缩进的，所以你可以自己添加 Mathematica 代码的缩进. 和前面的 Python 绘图代码类似，你可以把此部分代码保存在一个单独的文件中，然后通过`\input` 进行引入, 这里不再给出对应的案例.

- 注意空格与 Tab，如果源代码中有 Tab，那么 zTikZ 在进行此环境的抄录时会把原本的 Tab 转义为 `^^I`, 从而造成 Mathematica 源代码的错误, 比如你可能会看到你的源代码抄录后变成了下面的样子:

```
^^IContourStyle->{
^^I^^IRGBColor["#00C0A3"],
^^I},
```

- 同时注意 Mathematica 中注释的写法，不是`(* something*)`，而

是`(* something *)`，也就是你的注释不能够紧挨着 `*`，否则会造成 mathematica script 的解析错误.

- 由于 WolframScript 的限制，对应的 Mathematica 脚本的后缀只能为:`.wls`，否则 WolframScript 无法识别此脚本，也就不会去执行此脚本了.

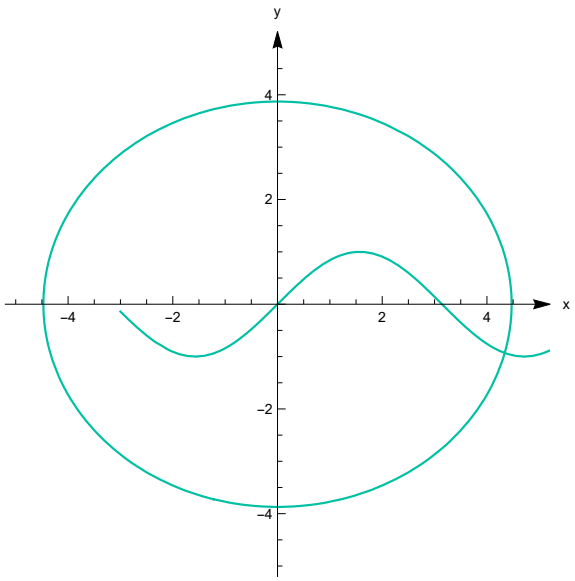


图 3.8: Mathematica 绘图示例

同样的你可以使用 Mathematica 绘制 3 维图形[6]。目前 zTikZ 仅支持插入静态图片，后续可能会考虑加入动态图片的支持功能，就行另外一个开源矢量图象绘制软件Asymptote 中的`.prc` 文件一样. 但是要是的 PDF 支持动态图形，首先你的 PDF 阅读器必须支持 JavaScript，常见的这种类型的 PDF 阅读器就是 Adobe 家的 Acrobat 了.

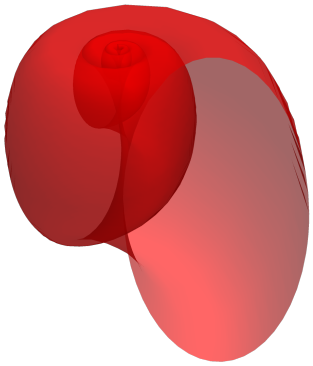


图 3.9: Mathematica 绘图示例 2

[6]由于目前的 Mathematica 不支持输出 3 维矢量图，所以想要是你的 3 维图像显得更加的清晰，可以调节图像的分辨率.

3.4 数值计算

3.4.1 xfp

众说周知，TEX 自身的计算能力是比较若的，所以涉及到一定的计算需求时，一般宏包的解决方法都是使用外部程序，让 TEX 只负责排版就行了. 但是在 LATEX3 项目发展了这么久之后，也做出了一些令人惊喜的结果。这里我们主要介绍 LATEX3 的 xfp 宏包，用于浮点数运算.

这里说明部分也许可以解决的痛点:

- 在 TikZ 绘图中，常常时需要坐标运算的，尽管 TikZ 提供了一个 `calc` 库, 但是这个库的使用语法总觉得不是那么的自然。于是这个时候你就可以使用 `xfp` 宏包.
- 在你自定义一些需要用到数值计算的宏命令时，使用 `xfp` 宏包是一个比较好的选择.

`xfp` 宏包的详细使用教程请参见官方文档, 这里不再赘述.

zTikZ 或者是 zLATEX 并不会自动加载 `xfp` 宏包，如果你有这方面的需要，请自己加载.

3.4.2 python

上面介绍了 Python 的绘图功能，这里再引入 zTikZ 中的浮点数计算部分 (Sympy 对应的部分应该不能叫浮点数计算了，毕竟 Sympy 进行的是精确的计算。) 这里使用的浮点数运算主要是基于 Python，以及可能的宏包 `numpy` 等. zTikZ 在调用此命令是默认载入 Python 库 `NumPy, SciPy`, 并且使用 `numpy` 中的函数时不用再加上前缀；比如求解 sin(2.345) 时，直接使用`\pyfp {sin(2.345)}` 即可, 不用写成`\pyfp {np.sin(2.345)}`. 对于库 `SciPy` 中的函数使用方法同理.

zTikZ 提供了命令用于浮点数运算, 这部分的结果并不会被缓存，也就是说每次编译此文档时，Python 都会重新计算此部分的结果. `\pyfp` 的参数说明如下:

```
\pyfig{<expression>}
```

下面给出一个使用样例:

```
\pyfp{0.9**10}
```

$$0.9^{10} = 0.3486784401000001$$

3.4.3 mathematica

使用 Mathematica 进行数值计算这一部分和后面的指令是有一部分重合的, 详细的使用参见后面一节的 “符号计算”, 所以这一部分我们就在后面介绍.

3.5 符号计算

符号计算是区别于数值计算的，上述的数值计算章节应该也有介绍; 但在介绍 zTikZ 中的符号计算模块之前先给出一个符号计算的定义，以下定义摘自 wiki:

> 数学和计算机科学中，计算机代数或符号计算或代数计算，是研究、开发用于操作表达式等数学对象的算法与软件的科学领域。这通常被视为是运算科学的一个子领域，但运算科学一般基于近似浮点数的数值计算，而符号计算则使用含变量的表达式进行精确计算，其中变量没有赋值。执行符号计算的软件系统称为计算机代数系统 (computer algebra system, CAS)，“系统”暗示了软件的复杂性，其中至少包括一种在计算机中表示数学数据的方法、一种编程语言（通常异于用于实现的语言）、一种专门的内存管理器、一套供输入输出表达式的用户界面、一大套用于通常运算的子程序，如表达式简化、能实现链式法则、多项式因式分解、不定积分等等的求导算法。

当前流行的计算机代数系统主要有:

- mathHandbook.com
- Sagemath
- Mathematica
- Maple
- MAGMA
- Maxima
- GAP
- PARI/GP
- Meditor
- MuPAD
- Mathomatic
- Xcas/Giac
- Yacas
- Mate

zTikZ 主要提供一个和 Mathematica(假如你已经购买了该软件)，以及 Pyhton 的 Sympy 模块的符号计算接口. 后续可能会开发一个统一的接口用于 TEX 和外部程序的交互.

3.5.1 python/sympy

Python 的 Sympy 是一个**免费，开源，轻量**的符号计算模块，其官网上有着详细的教程。所以这里便不再赘述其语法，重点介绍 zTikZ 中提供的几个接口 (命令)，用于和 Sympy 交互.

zTikZ 中针对 Sympy 提供了命令:, 其参数格式为:

```
\sympy{<expression>}
```

和之前的使用 Python 进行数值计算不同的是，zTikZ 针对此命令提供了 Cache 机制，此命令对应的结果会被保存在文件: `./ztikz_output/python_data/sympy_<index>.out` 文件中. 此文件名中的 `<index>` 表示的是对应的符号计算表达式的序号.

`\sympy` 命令的运算结果被保存在文件中之后，通过`\input` 命令把对应的运算结果导入到 TEX 的输出流 (文档) 中，由于默认的情况下此结果包含数学公式中的上下标:`^`，`_`，`...` 等，所以在把其导入到 LATEX 源码中时需要放入数学环境中.

zTikZ 模块的`\sympy` 命令在进行符号运算时，默认的符号变量有:`x`，`y`，`z`，`u`，`v`，`t`，这些变量你不需声明便可以直接使用; 下面给出使用`\sympy` 命令进行符号计算的部分示例:

```
% 定积分
\sympy{integrate(sin(x)/x, (x, -oo, oo))}
% 不定积分
\sympy{integrate( x**8 + cos(7*x) + 6*t, x )}
% 矩阵特征值
```

```
\sympy{Matrix([[1, 2], [2, 2]]).eigenvals()}
% 极限计算
\sympy{limit(sin(x)/x, x, 0)}
```

计算定积分的例子:

$$\int_{-\infty}^{+\infty} \frac{\sin(x)}{x}\,\mathrm{d}x = \pi$$

或者是计算不定积分的例子:

$$\int x^8 + \cos(7x) + 6t\,\mathrm{d}x = 6tx + \frac{x^9}{9} + \frac{\sin(7x)}{7}$$

或者是一个计算特征值的例子:

$$\mathrm{eig}(\begin{bmatrix}1 & 2\\ 2 & 2\end{bmatrix}) = \left\{\frac{3}{2} - \frac{\sqrt{17}}{2} : 1,\ \frac{3}{2} + \frac{\sqrt{17}}{2} : 1\right\}$$

计算极限的例子:

$$\lim_{x\to 0} \frac{\sin x}{x} = 1$$

目前的\sympy 命令只支持单行命令的模式，如果你需要使用多行 (条) 命令来达到计算目的，请考虑把它们变为一行命令 (一条指令).

3.5.2 mathematica

zTikZ 模块提供和 Mathematica 相关的符号计算，数值运算和知识查询接口; 以下的所有命令均具有缓存机制.

- \wolfram [<option>]{<expression>}: 使用 Mathematica 计算此表达式 <expression>, 默认返回 TEX 格式的代码，可以把 <option> 设为 text, 让其返回一个文本对象. 可以在这个命令中执行任何的 wolfram 指令，但是需要注意的一点是，所有和 wolfram 相关的命令是不会自动进入数学模式的，需要手动添加数学模式的标记.
- \wolframsolve [<cmd style>]{<expression>}[<varible>][<domain>]: 其中第一个可选参数默认值为: part, 意味着你的命令需要分拆为 3 个部分: 表达式 – (求解) 变量名 – 求解范围, 对应上面的参数, 分别填入。如果指定第一个参数为 full, 那么此时只需要给出对应的 <expresion>, 不用在指定后续的参数.(毕竟在第二个 (强制性-Mandatory) 参数中就已经包含了这些信息, 参见后面的具体使用样例).
- \wolframdsolve [<cmd style>]{<equation>}[<independent varible>][<dependent variablei>]: 此命令用于求解微分方程,其中的第一个可选参数和上面的\wolframsolve 的意义一致, 不再赘述. 第二个参数表示要求解的微分方程，第三个参数表示求解的独立变量 (函数)，最后一个参数表示此微分方程求解函数的自变量.

wolfram

首先给出命令的部分使用样例:

```
\wolfram{Series[Exp[x], {x, 0, 5}]}
\wolfram{LaplaceTransform[t^4 Sin[t],t,s]}
\wolfram[text]{WolframAlpha["Shanghai population", "ShortAnswer"]}
```

函数 $y=\mathrm{e}^x$ 的 5 阶 Taylor 展开式为:

$$1+x+\frac{x^2}{2}+\frac{x^3}{6}+\frac{x^4}{24}+\frac{x^5}{120}+O\left(x^6\right)$$

函数 $x=t^4\sin(t)$ 的 Laplace 变换为:

$$\mathcal{L}[t^4\sin(t)]=\frac{24\left(5s^4-10s^2+1\right)}{\left(s^2+1\right)^5}$$

在`\wolfram` 指令中执行 Mathematica 中的 `WolframAlpha` 命令进行查询，比如这里查询上海的人口数量，结果为:about 24.9 million people

这里补充一个使用`\wolfram` 就行数值运算的例子，因为 Mathematica 中有着诸多的和数值运算的函数，这里仅以内置的函数 `N[<expression>]` 为例:

比如我们求解 π 的截取前 30 小数的近似值为:

$$\pi\approx 3.14159265358979323846264338328$$

在使用`\wolfram` 命令进行浮点数运算时，只要表达式中含有小数，那么 Mathematica 就会默认进行浮点数运算，而不会计算表达式的精确值.

wolframsolve

命令可以用于多项式方程根的求解以及方程组的求解，并且可以给定求解的范围. 和前面的`\wolfram` 命令类似，此命令**只**返回求解结果的 TEX 代码，所以请把此命令置于公式环境中; 下面给出几个比较简单的求解示例:

```
\wolframsolve{x^4 - x^2 - 5 == 0}{x}
\wolframsolve{a x + y == 7 && b x - y == 1}{x, y}
\wolframsolve{x^2 + 2 y^3 == 3681 && x > 0 && y > 0}[x, y][Integers]
\wolframsolve[full]{x^2 + y^2 == 5^2 && y > x > 0, {x, y}, Integers}
```

方程 $x^4-x^2-5==0$ 的所有根为:

$$x=-i\sqrt{\frac{1}{2}\left(\sqrt{21}-1\right)},x=i\sqrt{\frac{1}{2}\left(\sqrt{21}-1\right)},x=-\sqrt{\frac{1}{2}\left(1+\sqrt{21}\right)},x=\sqrt{\frac{1}{2}\left(1+\sqrt{21}\right)}$$

方程组 $\begin{cases}ax+y==7\\ bx-y==1\end{cases}$ 的解为:

$$x=\frac{8}{a+b},y=-\frac{a-7b}{a+b}$$

不定方程 $\begin{cases}x^2+2y^3==3681\\ x>0,y>0\end{cases}$ 的整数解为:

$$x=15,y=12,x=41,y=10,x=57,y=6$$

不定方程 $\begin{cases}x^2+y^2==5^2\\ x>y>0\end{cases}$ 的整数解为:

$$x=3,y=4$$

后续可能会考虑加入解的筛选等功能，其实就是根据不同解之间的分隔符‘,’来对返回的字符串进行一个划分. 根据划分的结果生成一个列表，然后采用一个整数进行索引.

wolframdsolve

命令和命令`\wolframsolve` 完全相同，只是这个命令是用于求解微分方程的. 下面是几个示例:

```
\wolframdsolve{{y'[x] + y[x] == a*Sin[x], y[0] == 0}}[y[x]][x]
\wolframdsolve[full]{{y'[x]==Exp[z[x]]+1, z'[x]==y[x]-x}, {y,z}, x}
```

微分方程 $y'+y=a\sin(x), y(0)=0$ 的解为:

$$y(x)=-\frac{1}{2}ae^{-x}\left(-e^{x}\sin(x)+e^{x}\cos(x)-1\right)$$

非线性系统微分方程组 $y'(x)+y(x)=\mathrm{Exp}(z(x))+1, z'(x)=y(x)-x$ 的解为:

$$z(x)=\log\left(c_1\tan^2\left(\frac{1}{2}\left(\sqrt{2}\sqrt{c_1}x+2\sqrt{2}\sqrt{c_1}c_2\right)\right)+c_1\right), y(x)=x+\sqrt{2}\sqrt{c_1}\tan\left(\frac{1}{2}\left(\sqrt{2}\sqrt{c_1}x+2\sqrt{2}\sqrt{c_1}c_2\right)\right)$$

4

致谢

4.1 (La)TeX

Wiki 上对 TeX 的定义:

> TeX (/tɛx/), stylized within the system as TeX, is a typesetting system which was designed and written by computer scientist and Stanford University professor Donald Knuth and first released in 1978. TeX is a popular means of typesetting complex mathematical formulae; it has been noted as one of the most sophisticated digital typographical systems.

但是在我看来 TeX 不仅仅只是一个排版系统，他是一种精神，一种对于美的追求，一种对于细节的关注. 他并不是 `$Word$` 那种商业工具，更不是随性 MarkDown.(这里不去谈论**方正**) 在 TeX 的基础上建立了一种格式 LaTeX, 这种格式更加的人性化，更加的易用. 使得我们普通人也可以使用部分 TeX 的排版功能. 为何还有人说 LaTeX 难学呢? 假如他用过 TeX 之后就会明白 LaTeX 的平易近人了.

4.2 (La)TeX 社区

在此还要感谢所有的 TeX 社区，正是因为他们的无私奉献，本系列才能够顺利的完成. 更特别感谢 LaTeX3 团队的所有成员.

部分名词索引