zLaTeX/zTikZ 系列

Eureka

2024 年 3 月 29 日

目录

CHAPTER 1

zLaTeX 系列

1.1 简介

1.1.1 为何叫 z?

也不知道为什么这个系列名称要加以 'z' 的前缀，可能是因为个人爱好，或是因为觉得这个字母对自己而言有着一些别的意味。最开始此系列中此包含一个基本的文档类，叫做 πLaTeX，但是后面自己想开发一个用于绘图的宏包，主要基于 TiKZ. 用于常见平面图形的绘制以及外部程序的交互. 也许是看到了 `tikz` 库名称中的 "z"，于是便以 'z' 为前缀，产生了 zLaTeX系列。

1.1.2 项目地址

目前本项目已经在 GitHub, Gitlab, Gitee 上开源，地址如下:

- GitHub: https://github.com/zongpingding/ZLaTeX_ZTikZ
- Gitlab: https://gitlab.com/zongpingding/ZLaTeX_ZTikZ
- Gitee: https://gitee.com/zongpingding/ZLaTeX_ZTikZ

项目中包含 zLaTeX 文档类源码 `zlatex.cls`，zTikZ 宏包源码 `zTikZ.sty`，以及二者的说明文档. 后续在开发过程中，可能会保证 Github 的同步更新，至于 Gitlab 与 Gitee 则不一定会同步本系列的最新版.

1.1.3 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
- zTikZ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，但是在折腾了许久的 LaTeX 之后，现在这个模板才是最对我胃口的；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] 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

1.2 模板设计

1.2.1 设计历程

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，但是用了一段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。

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。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}
```

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没有，需要自己去声明；但是结果往往是自己设计的命令和别的宏包还不协调，冲突. 其中最重要的就是 hyperref 宏包了，初代模板中它的跳转功能是不正常的，由于自己定义的计数器不正确，在使用\label 命令时，激活的章节元素 (跳转位置) 根本不对。当初的目录结构也是自己设计，但是也有着同样的跳转为题. 初代 zLaTeX 文档类全部采用 LaTeX2ε 进行构建，很多的宏展开的地方都写的很繁琐，而且大部分的实现方案都是在 TeX-StackExchange 上找到的，很多时候都是处于一种能跑就行的状态，并不知道其背后的原理.

后来自己便把 zTiKZ 从中 zLaTeX 文档类中剥离出来，同时使用 LaTeX3 对原始文档类和 zTikZ 进行重构. 其中 zLaTeX 文档类继承自 book 文档类，之后几乎所有命令几乎都自己书写，知道它们的具体作用，对其他的宏包的影响。于是 zLaTeX 系列就诞生了，果然，在使用 LaTeX3 对原始项目进行重构之后，整个项目的代码清爽了许多，比如下面的 zLaTeX 文档类选项声明:

```
\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独立开发，在开发过程中参考了诸多优秀文档类/模板，参考最多的 CTeXart 文档类,几乎是本项目的大部分代码思路来源。此文档类完全采用 LaTeX3

[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

语法写成,本文档类中的**选项配置**模块主要参见 TEX-StackExchange 上的讨论,采用了 LATEX3 的 `key-value` 模块；这样的好处有: 选项配置简洁，符合人们习惯，模板维护方便.

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之后，压缩了你的版心大小，那么此时你的正文字体一定得做相应的改变. 那么一行多少个字合适呢？去查了一下 TEX.SE, 针对于英文, 一行的字母个数在 65-90 是比较合适的，并且字体尺寸一般为 `10pt,11pt,12pt`；页边距到底设置多少呢？自己去比对了 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，可以在生成的 PDF 中查看页面布局尺寸等信息, 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.

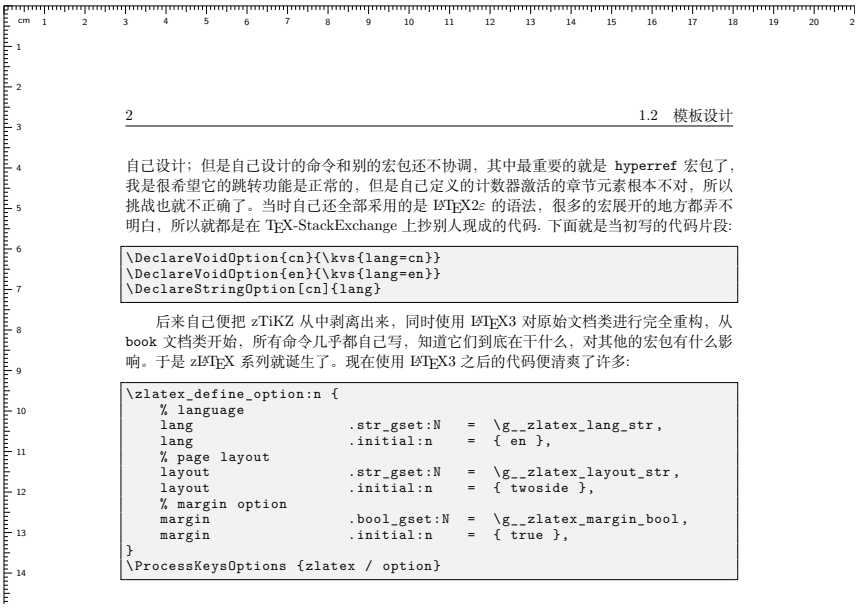


图 1.1: 页面布局示意图

在设计本模板的时，我也一直在纠结字体的问题，我应该把字体打包进入模板吗？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:

- 用户需要这个字体吗, 增加的字体会变成这个模板的负担吗?
- 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.

最后参考这这些标准，一步一步的调整，使得整体的页面布局稍微的合理些. 在设计这个模板时，还要考虑行距等各种元素。但是设计一个模板，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

1.3 兼容性

目前本系列已经实现 Windows 和 Linux 下的兼容; 但是 MacOS 下: 目前仅支持 zLaTeX 文档类. zTikZ 还未进行适配 (参见下文了解具体原因)，所以不保证本系列中的 zTikZ 文档类可以在 MacOS 下正常运行. 具体的兼容情况请参见后续的兼容性章节.

CHAPTER 2

zLaTeX 文档类

2.1 基本介绍

本文档类 zLaTeX 基于 book 类，主要用于满足和方便使用 LaTeX 使用者进行书籍和笔记的排版需求。zLaTeX 全部由 LaTeX3 进行编写，采用 key-value 的方式进行选项配置，方便后续的模板拓展和维护. 如果使用者熟悉 LaTeX，那么花费不到 10min 的时间，你便可以轻松使用本文档类用于日常的笔记撰写或者是正常的书籍的排版.

zLaTeX 的源代码完全开放，欢迎各位对源代码的修改以及二次分发.

2.2 Set Up zLaTeX

2.2.1 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 文档类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 ： 兼容 MacTeX2024(老版也应兼容)

2.2.2 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.

2.2.3 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.

2.2.4 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>}
\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}
```

2.3 宏包机制

zLaTeX 文档类会根据用户指定的选项自动处理和加载对应的宏包, 所以 zLaTeX 文档类在不同的导言区选项声明下加载的宏包和命令是不同的。本文档类内置导言区选项输出命令:`\zlatexOptions` , 用于打印此时文档类 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:

[1] 可能需要根据自己的实际情况加以调整

2.3.1 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
titlesec	tocloft		

表 2.1: zLaTeX 文档类基本宏包

2.3.2 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

2.3.3 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指定了选项 `math-alias=true`，那么 zLaTeX 此时还会额外加载宏包:`amssymb, mathtools, bm`.

2.4 文档类选项

本模板具有丰富的配置选项，包含页面设置，页边距，边注，数学字体，字体大小，模板语言; 采用键值对 `[<key 1>=<value 1>, <key 2>=<value 2>]` 的形式对各个选项就行指定, 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:

2.4.1 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
toc	rename, 2column	<none>

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.

2.4.2 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,如果原来含有\marginpar 命令的文档,在指定 margin=false 后,对应的\marginpar 环境，会被替换为 framed 宏包提供的 leftbar 环境.
- 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 :(§, ≪).
- toc选项可以同时指定 rename，2column, 分别代表重定义目录名 (居中加粗), 以及使用双栏目录排版.

2.4.3 数学指令

关于文档类选项 math-alias的进一步说明, 默认的自定义命令可能并不一定符合每一个人的习惯, 所以请谨慎加载此选项. 在 math-alias=true 后,zLaTeX 会进行如下命令的声明/重定义, 以及宏包的加载:

```
\RequirePackage{amssymb, mathtools}
\RequirePackage{bm}
% Math Font
```

```
\newcommand{\dd}{\mathrm{d}}
\newcommand{\C}[1]{\ensuremath{\mathcal{#1}}}
\let\ss\S
\renewcommand{\S}[1]{\ensuremath{\mathscr{#1}}}
\newcommand{\B}[1]{\ensuremath{\mathbb{#1}}}
\newcommand{\FF}[1]{\ensuremath{\mathbf{#1}}}
\newcommand{\F}[1]{\ensuremath{\bm{#1}}}
\newcommand{\R}[1]{\ensuremath{\mathrm{#1}}}
\newcommand{\K}[1]{\ensuremath{\mathfrak{#1}}}
% Math Arrow
\newcommand{\lr}{\ensuremath{\longrightarrow}}
\let\LL\ll
\renewcommand{\ll}{\ensuremath{\longleftarrow}}
\newcommand{\equ}{\ensuremath{\Longleftrightarrow}\,}
\newcommand{\sr}{\ensuremath{\longmapsto}}
\newcommand{\lrr}[2][]{\ensuremath{\xRightarrow[#1]{#2}}}
\renewcommand{\lll}[2][]{\ensuremath{\xLeftarrow[#1]{#2}}}
\newcommand{\ns}{\ensuremath{\varnothing}}
\newcommand{\A}{\ensuremath{\forall}}
% Math Operator
\newcommand{\alt}{\ensuremath{\mathrm{Alt}\;}}
\newcommand{\sgn}{\ensuremath{\mathrm{sgn}\;}}
\newcommand{\curl}{\ensuremath{\mathrm{curl}\;}}
\newcommand{\grad}{\ensuremath{\mathrm{grad}\;}}
\newcommand{\trace}{\ensuremath{\mathrm{trace}\;}}
\renewcommand{\div}{\ensuremath{\mathrm{div}\;}}
```

2.5 章节命令

2.5.1 计数器

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

目前 zLaTeX 提供了一个命令`\zlatexUpdateCounterAfter` 用于设置计数器的更新，使用格式为:

```
\zlatexUpdateCounterAfter{<child>}{<father>}
```

也就是让上述的 `<child>` 计数器随着 `<father>` 父计数器的更新而更新, 本命令的实现原型为:

```
\NewDocumentCommand{\zlatexUpdateCounterAfter}{mm}{
    \@addtoreset{#1}{#2}
}
```

关于本文档类的公式计数器的说明, 本文档类公式计数器默认跟随 `section` 计数器更新，在 zLaTeX 的源码中的声明为:

```
\counterwithin{equation}{section}
```

2.5.2 章节格式

目前还不支持指定章节格式，等后续在添加. 使用者可以加载 `titlesec` 等宏包进行自定义. zLaTeX 文档类默认加载了 `titlesec, titletoc, tocloft` 宏包用于章节格式和目录的格式定制. 如果使用者想要自定义章节格式，直接使用 `titlesec` 宏包的`\titleformat` 命令覆盖本模板的原始定义即可，或者是其他的命令.

注记 2.5.1 但是本文档类默认不加载 `tikz, pgf` 宏包，想要使用这两个宏包定义更加复杂章节样式，请手动加载，并设置自己喜欢的章节格式.

2.5.3 高亮环境

目前本系列提供命令`\zlatexFramed {<name>}[<color>]`用于创建类似 MarkDown 的彩色引用环境. 参数中的 `<name>` 表示声明环境的名称,`<color>` 表示此环境的背景颜色. 一个简单的使用样例如下:

```
% 环境 'refer' 声明
\zlatexFramed{refer}[orange]
% 使用环境 'refer'
\begin{refer}%
% something wrting here
\end{refer}
```

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;
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 2.5.2 在上面的 refer 环境开始时插入一个% 可以用于消除多余的空格。

2.5.4 封面样式

本文档类并没有内建复杂的封面格式，只是简单的重定义了`\maketitle` 命令用于生成封面. 声明如下:

```
% title page
\renewcommand{\maketitle}{
\begin{titlepage}
    \vfill\vspace*{40pt}
    \noindent\hspace*{134pt}\rule[-75pt]{6pt}{95pt}{\hspace*{10pt}\
        ↪ fontsize{25}{25}\selectfont\bfseries\@title}\par
    \vspace*{-15pt}
    \noindent\hspace*{150pt}{\Large\bfseries\@author}\par
    \vspace*{400pt}
    \noindent\hspace*{150pt}{\Large\bfseries\textcolor{gray}{\@date}}
    \vfill
\end{titlepage}}
```

如果使用者想要使用更加美观的封面，请手动加载 `tikz` 宏包，自己定义.

2.6 数学环境

2.6.1 常用数学环境

本文档类使用宏包 `amsthm` 定义了如下数学环境；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 `leftbar`. 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。

2.6.2 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:test}
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 (2.6.1), 可以看到，这个是可以精确跳转到对应的定理处的. 同时本模板中的`\cref` 命令会自动根据计数器的类别和文档的语言选项决定具体的引用格式. 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. 相关的详细信息还请参见本文档后面部分的标签与引用.

定理 2.6.1 (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;

$$\mathbf{v} \bigotimes \mathbf{w} = \mathbf{v} \otimes \mathbf{w} = \sum_{i=1}^{3} \sum_{j=1}^{3} a_{ij} u^i v^j \tag{2.6.1}$$

$$= \sum_{i=1}^{3} \left(a_{i1} u^i v^1 + a_{i2} u^i v^2 + a_{i3} u^i v^3 \right) \tag{2.6.2}$$

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 2.6.1 (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;

$$\mathbf{v} \bigotimes \mathbf{w} = \mathbf{v} \otimes \mathbf{w} = \sum_{i=1}^{3} \sum_{j=1}^{3} a_{ij} u^i v^j \tag{2.6.3}$$

$$= \sum_{i=1}^{3} \left(a_{i1} u^i v^1 + a_{i2} u^i v^2 + a_{i3} u^i v^3 \right) \tag{2.6.4}$$

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 2.6.1 (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;

$$\mathbf{v} \bigotimes \mathbf{w} = \mathbf{v} \otimes \mathbf{w} = \sum_{i=1}^{3} \sum_{j=1}^{3} a_{ij} u^i v^j \tag{2.6.5}$$

$$= \sum_{i=1}^{3} \left(a_{i1} u^i v^1 + a_{i2} u^i v^2 + a_{i3} u^i v^3 \right) \tag{2.6.6}$$

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 2.6.1 (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;

$$\mathbf{v} \bigotimes \mathbf{w} = \mathbf{v} \otimes \mathbf{w} = \sum_{i=1}^{3} \sum_{j=1}^{3} a_{ij} u^i v^j \tag{2.6.7}$$

$$= \sum_{i=1}^{3} \left(a_{i1} u^i v^1 + a_{i2} u^i v^2 + a_{i3} u^i v^3 \right) \tag{2.6.8}$$

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 2.6.1 (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;

$$\mathbf{v} \bigotimes \mathbf{w} = \mathbf{v} \otimes \mathbf{w} = \sum_{i=1}^{3} \sum_{j=1}^{3} a_{ij} u^i v^j \tag{2.6.9}$$

$$= \sum_{i=1}^{3} \left(a_{i1} u^i v^1 + a_{i2} u^i v^2 + a_{i3} u^i v^3 \right) \tag{2.6.10}$$

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 2.6.1 (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;

$$\mathbf{v} \bigotimes \mathbf{w} = \mathbf{v} \otimes \mathbf{w} = \sum_{i=1}^{3} \sum_{j=1}^{3} a_{ij} u^i v^j \tag{2.6.11}$$

$$= \sum_{i=1}^{3} \left(a_{i1} u^i v^1 + a_{i2} u^i v^2 + a_{i3} u^i v^3 \right) \tag{2.6.12}$$

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。

2.6.3 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；一般建议空一行再开始此类环境，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 proof 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;

$$\mathbf{v} \bigotimes \mathbf{w} = \mathbf{v} \otimes \mathbf{w} = \sum_{i=1}^{3} \sum_{j=1}^{3} a_{ij} u^i v^j \tag{2.6.13}$$

$$= \sum_{i=1}^{3} \left(a_{i1} u^i v^1 + a_{i2} u^i v^2 + a_{i3} u^i v^3 \right) \tag{2.6.14}$$

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;

$$\mathbf{v} \bigotimes \mathbf{w} = \mathbf{v} \otimes \mathbf{w} = \sum_{i=1}^{3} \sum_{j=1}^{3} a_{ij} u^i v^j \tag{2.6.15}$$

$$= \sum_{i=1}^{3} \left(a_{i1} u^i v^1 + a_{i2} u^i v^2 + a_{i3} u^i v^3 \right) \tag{2.6.16}$$

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。

2.6.4 注意事项

默认的数学类环境均采用正体`\upshape` ，如果使用者不喜欢前者默认的“正体”字体样式，可以直接在数学类环境开始时使用字体命令`\itshape` 进行原有字体样式的覆盖，示例如下:

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

注记 2.6.2 *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;*

$$\mathbf{v} \bigotimes \mathbf{w} = \mathbf{v} \otimes \mathbf{w} = \sum_{i=1}^{3} \sum_{j=1}^{3} a_{ij} u^i v^j \tag{2.6.17}$$

$$= \sum_{i=1}^{3} \left(a_{i1} u^i v^1 + a_{i2} u^i v^2 + a_{i3} u^i v^3 \right) \tag{2.6.18}$$

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。

同时，本文档类中数学类环境和前文的自定义高亮环境`\zlatexFramed` 均默认首行不缩进，需手动添加缩进.

2.6.5 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.

2.7 标签与引用

2.7.1 footnote

可能有人不喜欢默认的脚注没有在页脚的位置，而是在页脚偏上的位置，用户可以独立加载宏包 `footmisc` 用于强制脚注位于页面底部，本文档类不打算添加此宏包，用户可以自行在导言区添加如下命令:

```
\usepackage[bottom]{footmisc}
```

2.7.2 Cleveref

zLaTeX 文档类加载了 cleveref 宏包来构建标签-引用系统。常规的\label {} 操作并没有什么变化，区别主要在引用标签功能上。对于普通的模板你可能会看到如下的说明: 使用\eqref 进行公式标签的索引, 使用\figref 进行图片的索引, 使用\tabref 进行表格的索引... 使用此命令可以避免书写如下格式的引用代码:

```
定理:\ref{thm:test}
% or
\newcommand\thmref{定理:\ref{#1}}
```

在 zLaTeX 中，引用格式预设值如下 (至于多个标签引用时，只有 lang=en 时采用部分变化，对应的前缀变为复数):

语言	公式	图片	表格
lang=en	equation	figure	table
lang=cn	方程	图	表

表 2.4: cref 引用格式

对于 cleveref 中的其它命令，如\Cref ，\crefrange ，\Crefrange 等等，本文档类未对其进行修改，所以以上命令均是兼容的，详细的使用说明请参见 cleveref 宏包的官方文档.

2.7.3 图片与 (列) 表

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\cref 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.

2.7.4 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

2.8 文献引用

2.8.1 基本设置

本模板采用的文献引擎是 biber, 这说明，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}
```

2.8.2 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}
```

2.9 索引

2.9.1 使用方法

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]
```

使用\index 命令时在此命令中的名词是不会显示在 PDF 文档中的，所以如果你要添加一个 “函数” 的 index 项目时，在你的 TeX 文档中应该这样写:

```
函数\index{函数}是从集合到 ...
```

2.9.2 Bug

目前的 index 生成工具 `indextools` 宏包和 tikz 的 `external` 库有冲突，具体表现为：当 `indextools` 和 `external` 库同时使用时，在第一次编译此文档时会抛出如下错误信息：

```
===== 'mode=convert with system call': Invoking 'pdflatex -halt-on-
    ↪ error -inter
action=batchmode -jobname "tikzdatamain-figure0" "\def\
    ↪ tikzexternalrealjob{release
}\input{release}"' ========

! Package tikz Error: Sorry, the system call 'pdflatex -halt-on-error -
    ↪ interact
ion=batchmode -jobname "tikzdata/release-figure0" "\def\
    ↪ tikzexternalrealjob{release}\i
nput{release}"' did NOT result in a usable output file 'tikzdatamain-
    ↪ figure0' (exp
ected one of .pdf:.jpg:.jpeg:.png:). Please verify that you have
    ↪ enabled system
     calls. For pdflatex, this is 'pdflatex -shell-escape'. Sometimes it
        ↪  is also na
med 'write 18' or something like that. Or maybe the command simply
    ↪ failed? Erro
r messages can be found in 'tikzdata/release-figure0.log'. If you
    ↪ continue now, I'l
l try to typeset the picture.
```

关于此问题我已经在 Github 上给作者提了Issue, 同时也在 TEX-SE 上发出了提问. 可以关注上述的问题找到解决方法.

目前的解决方法有两个：

- 取消加载 indextools 宏包，改用传统的 `makeidx` 宏包.(需自行去修改 `zlatex.cls` 中的加载项)
- 仍然使用此宏包，但是在第一遍 (tikz 图片还没有缓存时) 取消导言区以及文档末尾的如下命令：

  ```
  % 导言区
  \makeindex[title=Test Title, columns=3]
  % 文末
  \addcontentsline{toc}{chapter}{部分名词索引}
  \printindex
  ```

  然后在文档的第二次编译时取消两处命令的注释, 以此达到正常编译的目的.

> 为何我一再坚持使用 `indextools` 宏包? 相较于传统的 `makaidx` 宏包需要在命令行中先使用 LATEX 引擎编译，然后使用 `makeindex` 命令编译，最后再使用 LATEX 引擎编译两遍。`indextools` 宏包可以在不超过两次的 LATEX 引擎编译下直接生成对应的 index，方便了许多.

CHAPTER 3

zTikZ 宏包

目前 zTikZ 宏包能够在 Linux 下编译运行, 在 Windows 上还未进行完整的测试. 但是对于 TEXLive 版本 ≥2023 的用户来说，本宏包应该是可以在 Linux 和 Windows 下运行的. 实际的兼容情况应该得等到足够的用户反馈时才能够确定.

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 Set Up zTikZ

3.2.1 兼容情况

目前 `zTikZ` 模块已经可以做到跨平台, 在各个平台中的兼容性如下:

[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

Windows : TEXLive 最低版本 2023

Linux : TEXLive 最低版本 2022

MacOS : MacTEX 由于缺少 `l3sys-shell` 宏包 (或者是不适配), 所以并不兼容

在 Linux 平台上并没有什么需要注意的事项，重点是 Windows 平台上的兼容性; 使用 `zTikZ` 正常运行，那么在里的系统中 (默认添加了环境变量) 须有以下软件:

- `python`: 用于运行 python 脚本进行符号计算与绘图，需要 Python 库 `sympy, scipy`.
- `sed`: 主要用于绘图脚本中的函数以及绘制样式替换，后续可能会考虑去除 `sed` 依赖.
- `wolframscript`(可选): 如果需要使用本模块的 Mathematica 功能，那么需要安装 WolframScript 以及对应的 Mathematica 软件. 或者只安装 wolframscript，但是在执行命令时选择在云端执行，这样就不用本地的 Mathematica 计算内核了.

3.2.2 环境配置

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.

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

[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

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
  - \Plot : 绘制函数
  - \ParamPlot : 绘制参数方程
  - \ContourPlot : 绘制等高线图
  - \PolarPlot : 绘制极坐标图
  - \PlotPrecise : 函数绘制精度
- 欧几里得几何
  - \ShowPoint : 绘制点
  - \ShowGrid : 绘制网格
  - \ShowAxis : 绘制坐标轴
  - \ShowIntersection : 绘制交点
  - \gnudata : 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>][<marker options>]{<function>}
```

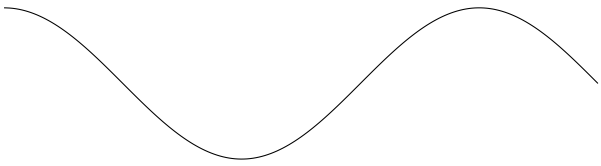
其中 <plot domain> 就是绘制的定义域, 比如-3:4; <plot style> 为绘制函数的样式，包括图形的颜色，线型，粗细等等;<marker option> 表示是否绘制散点图 (意味着此时你要把对应绘制命令的精度降低，不然会十分的耗时，或者是造成结果的不满意)；当你没有指定这个可选参数的值时，默认绘制连续函数的图像.marker 参数采用键值对的方式进行指定，见后文\ShowPoint 中的 <marker option>参数详细解释，二者用法完全一致. <function> 就是你要绘制的函数, 比如 sin(x). 以下为一个具体的例子, 首先创建一个 tikzpicture 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:

```
\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):

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

[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

```
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
│   └── float_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:

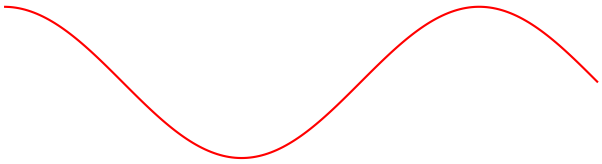
```
\def \tikzexternallastkey {AE7F2539E81C96848ADCCEE3994993D1}%
```

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. 顺便这里说明命令`\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，“2” 表示此数据是在第 1 个 tikzpicture 环境中的第二个绘图数据; 后面我们再解释这个文件夹中其他文件的作用，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\ShowAxis 的参数格式为:\ShowAxis [<plot style>]{(start coordinate);(end coordinate)}. 这里的 <plot style> 参数采用键值对的形式指定, 可用键值对列表以及不同键的默认值可参见如下源码声明:

```
% basic tick args
tickStart       .fp_set:N   = \l__start_fp,
tickStart       .initial:n  = { -5 },
tickEnd         .fp_set:N   = \l__end_fp,
tickEnd         .initial:n  = { 5 },
axisRotate      .fp_set:N   = \l__axis_rotate_angle,
axisRotate      .initial:n  = { 0 },
% tick dimension spec
mainStep        .fp_set:N   = \l__main_step_fp,
mainStep        .initial:n  = { 1.0 },
subStep         .fp_set:N   = \l__sub_step_fp,
subStep         .initial:n  = { 0.1 },
tickLabelShift  .dim_set:N  = \l__tick_label_shift_dim,
tickLabelShift  .initial:n  = { 0pt },
mainTickLenght  .dim_set:N  = \l__main_tick_length_dim,
mainTickLenght  .initial:n  = { 4pt },
subTickLenght   .dim_set:N  = \l__sub_tick_length_dim,
subTickLenght   .initial:n  = { 2pt },
mainTickLabelPosition .tl_set:N  = \l__main_tick_label_position_tl,
mainTickLabelPosition .initial:n = { below },
% color spec
axisColor       .tl_set:N   = \l__axis_color_tl,
axisColor       .initial:n  = { black },
mainTickColor   .tl_set:N   = \l__main_tick_color_tl,
mainTickColor   .initial:n  = { black },
subTickColor    .tl_set:N   = \l__sub_tick_color_tl,
subTickColor    .initial:n  = { black },
mainTickLabelColor .tl_set:N  = \l__main_tick_label_color_tl,
mainTickLabelColor .initial:n = { black },
% tick cross type spec
tickStyle       .choice:,
```

```
tickStyle/cross .code:n     = \tl_set:Nn \l__tick_spec_tl { cross },
tickStyle/above .code:n     = \tl_set:Nn \l__tick_spec_tl { above },
tickStyle/below .code:n     = \tl_set:Nn \l__tick_spec_tl { below },
```

一个自定义\ShowAxis 命令示例如下:

```
\NewDocumentCommand{\xAxis}{O{-2}O{8}}{
    \ShowAxis[
        tickStart=\fp_eval:n {#1+1}, tickEnd=\fp_eval:n {#2-0.75},
        mainStep=1, subStep=.25,
        axisRotate=0, axisColor=black,
        mainTickColor=black, subTickColor=black,
        mainTickLenght=10pt, subTickLenght=5pt,
        tickLabelShift=0pt, tickStyle=below,
        mainTickLabelPosition=below
    ]{(#1, 0); (#2, 0)}
}
```

从上述 zTikZ 内置的\xAxis 命令可以看出，我们可以指定坐标轴的如下属性:

- 主 (子) 刻度绘制起点/终点
- 主 (子) 刻度颜色设置
- 主刻度标签颜色/位置，可选位置有:above，below，left，right
- 主 (子) 刻度长度
- 主 (子) 刻度间隔
- 主刻度坐标偏移量
- 主 (子) 刻度旋转角度，请注意调整旋转后标签的位置.
- 主 (子) 刻度样式:cross，above，below, 分别代表 ticks 在坐标轴的两侧还是某一侧.

\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 “;”). zTikZ 内置\xAxis ,\yAxis 命令，用于绘制两条标准的坐标轴. 命令的参数格式为:

```
\xAxis[<start>][<end>]
\yAxis[<start>][<end>]
```

上面的 <start>，<end> 分别表示 x, y 轴对应的坐标轴绘制的起始, 终止点. 对应 x 轴即为:(<start>，0) -- (<end>，0). y 轴即交换坐标.

注记 3.3.1 如果在使用\ShowAxis 命令时, 没有指定可选参数中键 tickStyle 的值时，那么此时并不会绘制任何的刻度.

在绘制完坐标轴之后，便可以绘制网格; 使用\ShowGrid 命令.\ShowGrid 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\ShowPoint 命令的参数格式为:

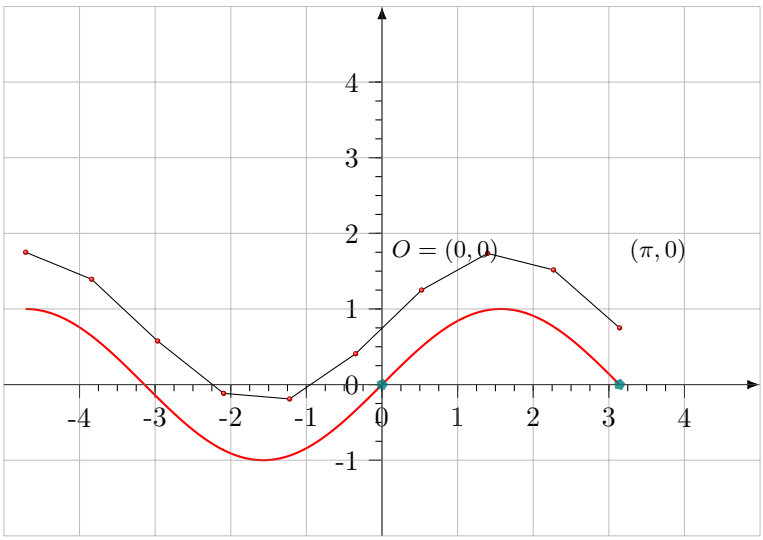
```
\ShowPoint[<marker option>]{(coordinate 1); (coordinate 2); ...}[<label
    ↪  1>; <label2>; ...][<position>]
```

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

- type: zTikZ 库已经加载 pgfmarkers 库，所以任何在此库中的形状均为有效值, 默认为实心 circle.type=*. 可以参见图 3.2, 截取自 pgf 手册.
- 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.
- rotate: marker 的旋转角度, 默认为 0.

终于，现在我们可以给出一个相对完整的代码 (包括 <marker option> 对应的用法):

```
\begin{tikzpicture}[>=Latex]
    \xAxis[-5][5]
    \yAxis[-2][5]
    \Plot[-1.5*pi:pi][red, thick]{sin(x)}
    \ShowGrid[gray, step=1, opacity=.5]{(-5, -2); (5, 5)}
    % marker option
    \PlotPrecise{plot}{10}
    \Plot[-1.5*pi:pi][red, thick][type=ball, color=red]{sin(x)+.75}
    % show point
    \ShowPoint[color=teal, radius=2pt, type=pentagon*, opacity=.8,
        ↪ rotate=60]
        {(0, 0); (3.1415926, 0)}[$O=(0, 0)$; $(\pi, 0)$][above right=4
            ↪ em and 0em, font=\small]
\end{tikzpicture}
```



```
\pgfuseplotmark{-}
\pgfuseplotmark{|}
\pgfuseplotmark{o}
\pgfuseplotmark{asterisk}
\pgfuseplotmark{star}
\pgfuseplotmark{10-pointed star}
\pgfuseplotmark{oplus}
\pgfuseplotmark{oplus*}
\pgfuseplotmark{otimes}
\pgfuseplotmark{otimes*}
\pgfuseplotmark{square}
\pgfuseplotmark{square*}
\pgfuseplotmark{triangle}
\pgfuseplotmark{triangle*}
\pgfuseplotmark{diamond}
\pgfuseplotmark{diamond*}
\pgfuseplotmark{halfdiamond*}
\pgfuseplotmark{halfsquare*}
\pgfuseplotmark{halfsquare right*}
\pgfuseplotmark{halfsquare left*}
\pgfuseplotmark{pentagon}
\pgfuseplotmark{pentagon*}
\pgfuseplotmark{Mercedes star}
\pgfuseplotmark{Mercedes star flipped}
\pgfuseplotmark{halfcircle}
\pgfuseplotmark{halfcircle*}
\pgfuseplotmark{heart}
\pgfuseplotmark{text}
```

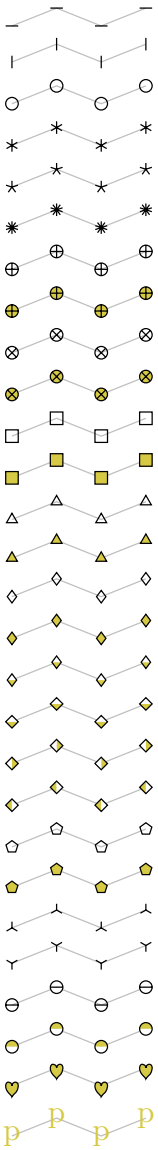


图 3.2: Point Marker

注意:zTikZ 中的命令都不需要你使用 “;” 去结束绘制.

更多的关于 Marker 的选项和用法，请参见如下示例:

```
\begin{tikzpicture}[>=Latex]
    \xAxis[-5][5]
    \yAxis[-5][5]
    \ShowGrid[gray, step=1, opacity=.5]{(-5, -2); (5, 5)}
    \PlotPrecise{plot}{10}
    \Plot[-1.5*pi:pi][red, thick][type=ball, color=red]{sin(x)}

    \PlotPrecise{polar}{20}
    \PolarPlot[0:10*pi][orange][type=ball, color=red]{0.1*t}

    \PlotPrecise{contour}{40}
```

```
    \ContourPlot[-4:4][green][type=ball, color=red]{x**2/9+y**2/4-1}

    \PlotPrecise{param}{40}
    \ParamPlot[0:2*pi][red, name path=ellipse][type=ball, color=red]{2*
        ↪ sin(t), 3*cos(t)}
\end{tikzpicture}
```

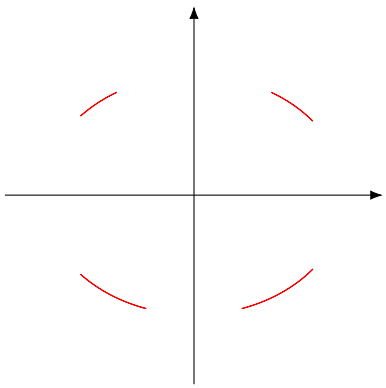
其余的几个函数绘制命令，稍微值得一提的是命令\ContourPlot ，其参数格式为:

```
\ContourPlot[<plot domain>][<plot style>][<marker options>]{<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):

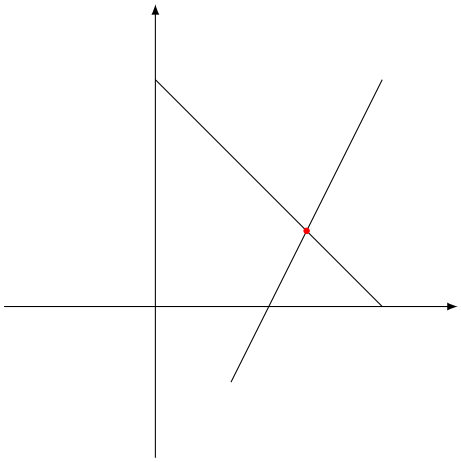
```
\ContourPlot[0:2; -1:1][red, dashed]{x-1}
```

我们还有两个命令没有讲到:\ShowIntersection , \PlotPrecise ; 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.

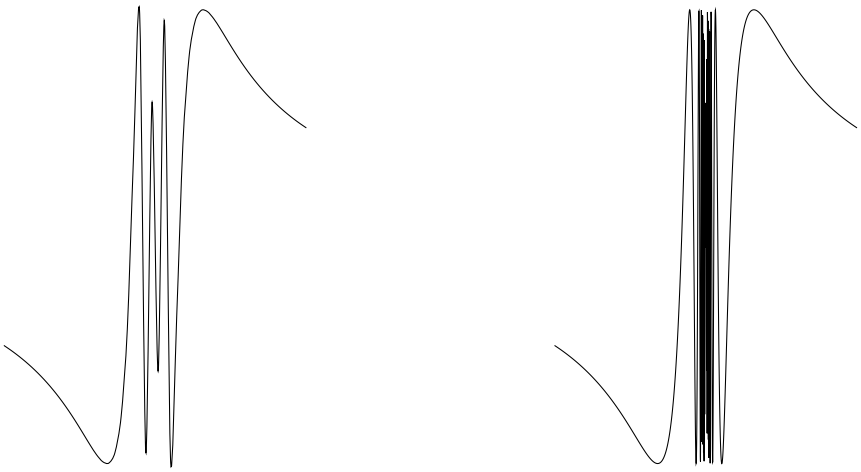
```
\begin{tikzpicture}[>=Latex]
    \ShowAxis{(-2, 0); (4, 0)}
    \ShowAxis{(0, -2); (0, 4)}
    \Plot[1:3][name path=line1]{2*x-3}
    \Plot[0:3][name path=line2]{-x+3}
    \ShowIntersection[color=red]{line1; line2}{1}

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



\PlotPrecise 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:



下面我们给出几个运用到 zTikZ 这部分命令的一些综合绘图案例:

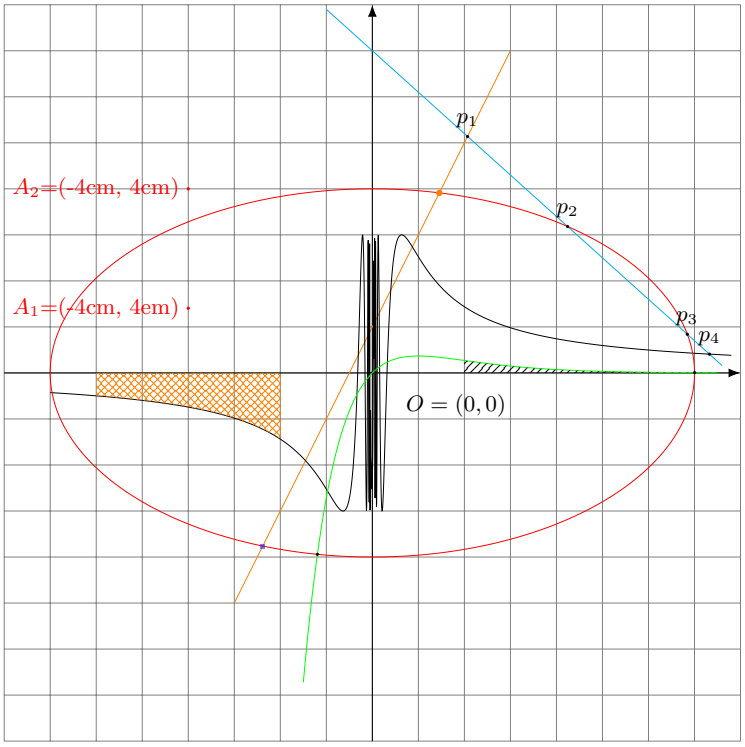


图 3.3: 绘制示例 1

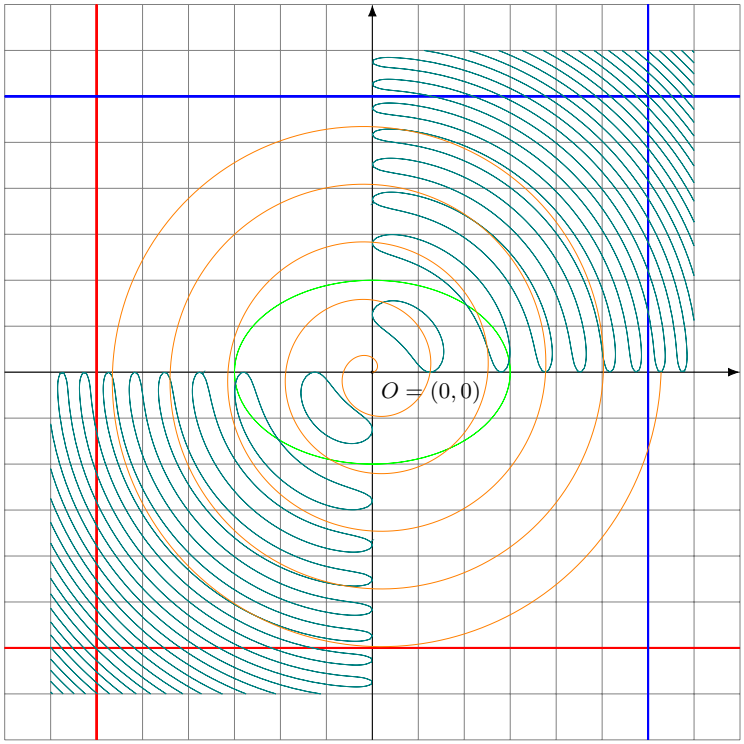


图 3.4: 绘制示例 2

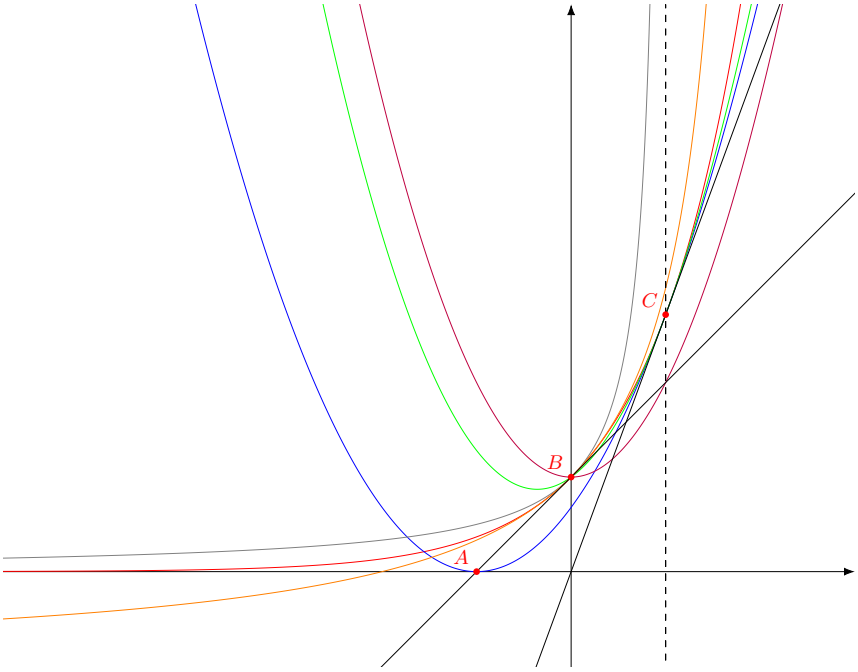


图 3.5: 绘制示例 3

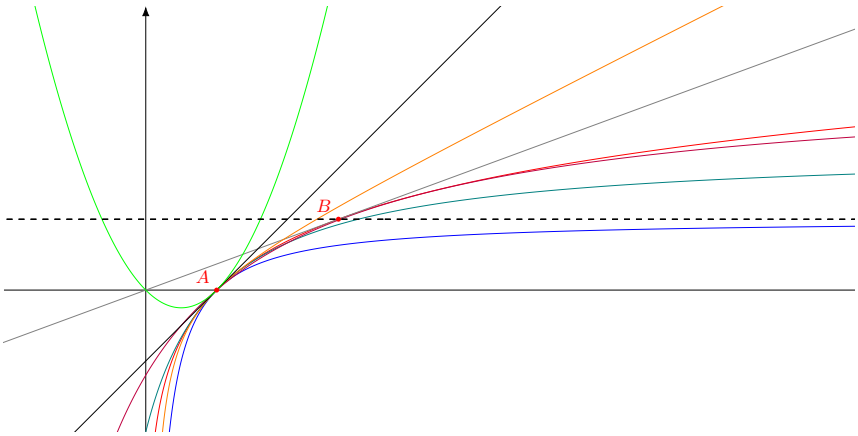


图 3.6: 绘制示例 4

> 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，有可能你在编译时并不会抛出这个错误) 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

python 绘图是比较就简单的，zTikZ 提供了用于 python 绘图的 `pyfig`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>]{}` 中的参数, 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:

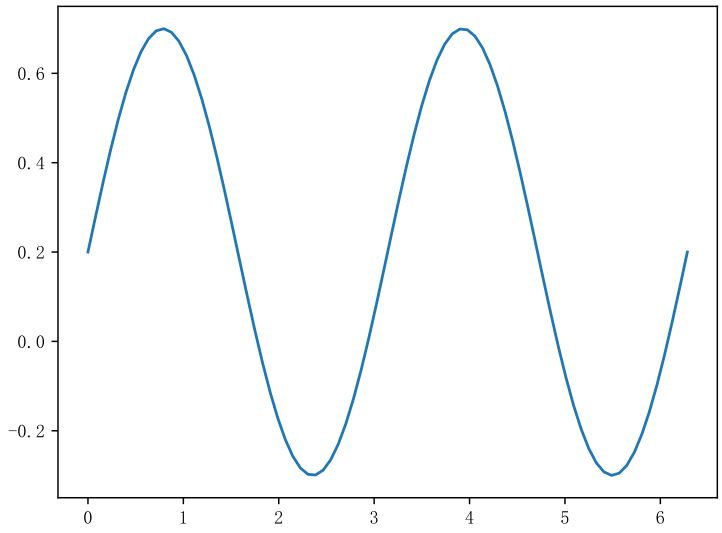


图 3.7: Python 绘图示例 1

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 官方的三维图像. 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:

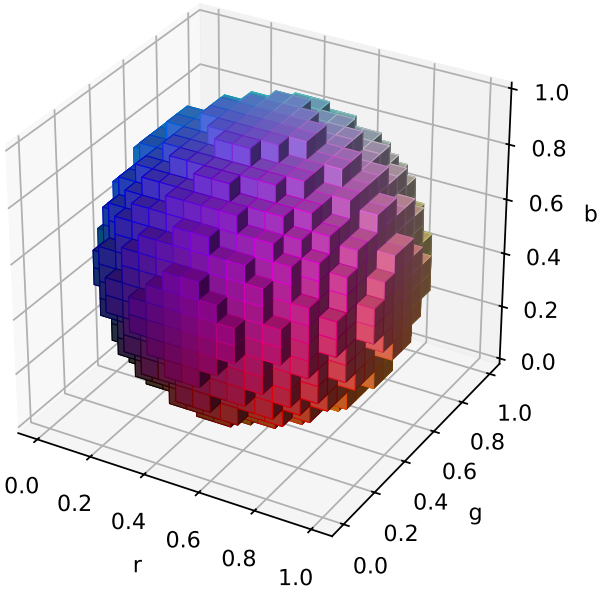


图 3.8: Python 绘图示例 2

```
\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 `mmafig`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幅图形的情况，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.

注记 3.3.2 用户如果要使用 zTikZ 的 Mathematica 模块，请务必确保 wolframscript 在命令行中能够正常运行. 可以使用如下文件作为测试用例，检测 wolframscript 是否正常工作;

```
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["works_well.pdf", figure];
```

把这里的源码保存为 test.wls, 然后在命令行运行:

```
wolframscript -script test.wls
```

如果正常工作的话，那么在你的当前工作目录下会产生一个名为 works_well.pdf 的 pdf 文件. 反之，你的 wolframscript 没有正常配置或者是激活, 也就不能够使用本模块.

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，就像另外一个开源矢量图象绘制软件Asymptote 中的.prc 文件一样. 但是要使得能在 PDF 中预览动态图形，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.

3.3.4 matlab

目前 zTikZ 中的 Matlab 模块还在开发中, 但是目前你可以使用 matlab2tikz 这个 Matlab 插件来把你的 Matlab 图形转换为对应的 tikz 代码, 效果也是很好的.

但是目前你可以在命令行中调用 Matlab 运行自己的 Matlab 脚本, 一个测试脚本如下:

[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.

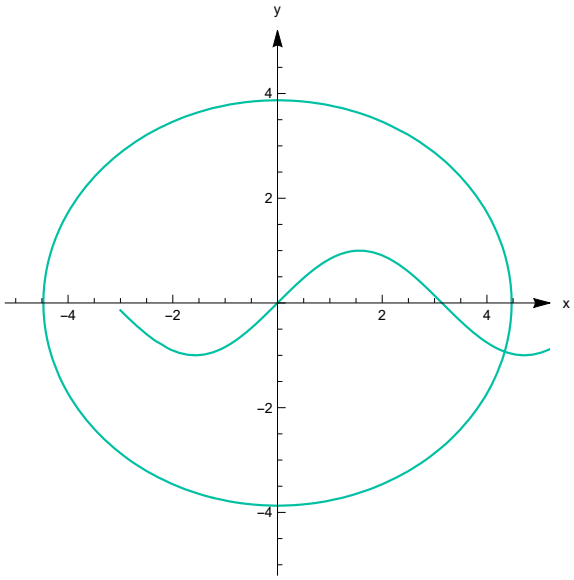


图 3.9: Mathematica 绘图示例

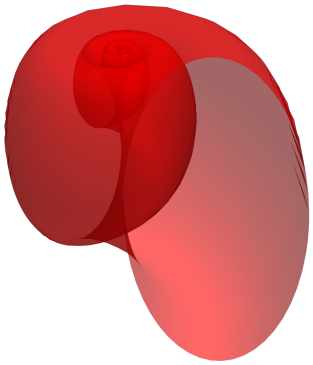


图 3.10: Mathematica 绘图示例 2

```
x = 1:0.1:2*pi;
y = sin(x);

figure('visible','off')
plot(x, y, 'r-');

exportgraphics(gcf, 'myfig.pdf')
```

然后在命令行中使用如下命令进行运行:

```
matlab -batch "run('matlab.m')"
```

运行完后，你便可以在当前目录下看到一个 pdf 文件，名为 `myfig.pdf`; 在运行方式这一点上,Matlab 和 Wolframscriprt 的运行命令:`wolframscript -script mma.wls` 是有一点区别的.

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羸弱的，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 xfp也许可以解决的痛点:

- 在 TikZ 绘图中，常常是需要坐标运算的，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 在调用此命令时默认载入 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\py {sin(2.345)} 即可, 不用写成\py {np.sin(2.345)}. 对于库 SciPy 中的函数使用方法同理.

python command

zTikZ 提供了命令\py 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. \py 的参数说明如下:

```
\py[<return type>]{<expression>}
```

上述的第一个默认参数值为:hold, 可选值有 str, 二者的区别可以简单的认为，返回的 Token 的类别码不同. 比如当外部文件中的内容为:

```
\[ a^2 + b^2 = c^2 \]
```

默认情况才下，\py 返回此命令的结果为:

$$a^2 + b^2 = c^2$$

但是如果你指定返回的类型为 str 时,那么在文档中的显示结果就会变为:\[a^2 + b^2 = c^2 \]. 而不是默认情况下的排版公式.

值得说明的是，\py 命令和 xfp 宏包提供的\inteval , \fpeval 是类似的；也就是你可以把 \py 命令嵌套到你自己定义的一个命令中. 同样是使用 ##1 来表示接收到的参数. 比如你可以创建下面这几个命令:

```
\newcommand{\pypow}[1]{\py{#1}}
\newcommand{\pyreverse}[1]{\py{'#1'[::-1]}}
\newcommand{\pyuppercase}[1]{\py{'#1'.upper()}}
```

分别用于数值计算 (乘方计算), 字符串反转输出, 字符串大写输出. 使用效果如下:

- Power Calculation: $2^{64} = 1024$
- Reverse a string using Python: XeTaL-olleH
- Uppercase a string: HELLO-LATEX
- Modulus: $102 = 6 \mod 8$
- Return string Options: 1024

注记 3.4.1 如果你想要使用 Python 中的求模运算需要输入% 时, 在\py 命令中你应该写为:

```
\py{102\%8}
```

或者是如果你需要在\py 命令中传入 $, 请像下面这样书写:

```
\py{'\\$\\$'+str(2**10)+'\\$\\$'}
```

注记 3.4.2 目前由于 Windows 上的 sed 命令 (又或者是平台差异) 和 Linux 下的差异, 所以可能导致在 Windows 上使用时, \py 中的单引号' 不能正确的输入到目标文件中, 从而导致字符串的声明失败. 请一定注意!

python environment

zTikZ 同时也提供了一个用于自由书写 Python 代码的环境 pycode, 可以用于生成复杂且规律的表格代码等排版元素. 比如下面的示例:

```
\begin{pycode}{pycode_1.py}
import numpy as np


# write file
with open ('./ztikz_output/python_data/pycode_1.py.out', 'w') as file:
    file.write("\\begin{tabular}{p{3cm}ccc}\n")
    file.write("\\hline\n")
    file.write("number/function & $\\sin$ & $\\cos$ & $\\tan$\\\\\n")
    file.write("\\hline\n")
    for i in range(1, 21):
        file.write(
            f"${i}$ & ${np.around(np.sin(i), decimals=4)}$ &  ${np.
                ↪ around(np.cos(i), decimals=4)}$ & ${np.around(np.tan
                ↪ (i), decimals=4)}$\\\\\n"
        )

    file.write("\\hline\n")
    file.write("\\end{tabular}\n")
\end{pycode}
```

那么在运行此命令后，在 zTikZ 的缓存文件夹中会生成一个名为 `pycode_1.py.out` 的文件，其内容为:

```
\begin{tabular}{p{3cm}ccc}
\hline
number/function & $\sin$ & $\cos$ & $\tan$\\
\hline
$1$ & $0.8415$ &  $0.5403$ & $1.5574$\\
$2$ & $0.9093$ &  $-0.4161$ & $-2.185$\\
$3$ & $0.1411$ &  $-0.99$ & $-0.1425$\\
$4$ & $-0.7568$ &  $-0.6536$ & $1.1578$\\
$5$ & $-0.9589$ &  $0.2837$ & $-3.3805$\\
$6$ & $-0.2794$ &  $0.9602$ & $-0.291$\\
$7$ & $0.657$ &  $0.7539$ & $0.8714$\\
$8$ & $0.9894$ &  $-0.1455$ & $-6.7997$\\
$9$ & $0.4121$ &  $-0.9111$ & $-0.4523$\\
$10$ & $-0.544$ &  $-0.8391$ & $0.6484$\\
$11$ & $-1.0$ &  $0.0044$ & $-225.9508$\\
$12$ & $-0.5366$ &  $0.8439$ & $-0.6359$\\
$13$ & $0.4202$ &  $0.9074$ & $0.463$\\
$14$ & $0.9906$ &  $0.1367$ & $7.2446$\\
$15$ & $0.6503$ &  $-0.7597$ & $-0.856$\\
\hline
\end{tabular}
```

所以这段代码的具体效果如下:

number/function	sin	cos	tan
1	0.8415	0.5403	1.5574
2	0.9093	−0.4161	−2.185
3	0.1411	−0.99	−0.1425
4	−0.7568	−0.6536	1.1578
5	−0.9589	0.2837	−3.3805
6	−0.2794	0.9602	−0.291
7	0.657	0.7539	0.8714
8	0.9894	−0.1455	−6.7997
9	0.4121	−0.9111	−0.4523
10	−0.544	−0.8391	0.6484
11	−1.0	0.0044	−225.9508
12	−0.5366	0.8439	−0.6359
13	0.4202	0.9074	0.463
14	0.9906	0.1367	7.2446
15	0.6503	−0.7597	−0.856

表 3.1: Using Python to generate Table

注记 3.4.3 本环境 (pycode) 目前还不够成熟, 请谨慎使用, 也欢迎各位提出宝贵的改进意见. 当然, 本环境目前具有缓存机制.

注记 3.4.4 推荐用户使用最新的由 LaTeX3 编写的宏包:csvsimple-l3, 或者是tabularray用于在 LaTeX 中进行表格的排版.

3.4.3 mathematica

使用 Mathematica 进行数值计算这一部分和后面的`\wolfram` 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.

```
\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>`，不用再次指定后续参数.(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`\wolfram` 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 中有着诸多和数值运算相关的函数，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

`\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. 但是目前还在读取文件时的 catcode 问题中挣扎.

wolframdsolve

命令`\wolframdsolve` 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:

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

CHAPTER 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.

部分命令/名词索引