The fixdif Package

Zhang Tingxuan

2022/7/19　Version 1.3a*

简介

fixDif 宏包在 LaTeX 中重定义了\d命令，并提供来定义微分算子命令的接口。

本宏包不仅可用 pdfTeX, XeTeX, LuaTeX编译，还兼容 XeTeX 和 LuaTeX下的 unicode-math 宏包。

目录

第 1 节　背景

为求美观，我们通常会在微分算子和它前面的表达式之间保留一定的空白[1]. 比如以下情况：

$$f(x)\mathrm{d}x \qquad \text{和} \qquad f(x)\,\mathrm{d}x.$$

我们通常会认为左边比右边好看，在 $f(x)$ 和 $\mathrm{d}x$ 之间的小空白可以视为 $f(x)$ 和 $\mathrm{d}x$ 的乘积符号。

因此，有些用户会喜欢定义这样的命令:

```
\renewcommand\d{\mathop{\mathrm{d}}\!}
```

虽然这个命令在“行间公式”和“行内公式”都很有效，但是依然存在以下三个问题:

1. d 前面的空白在行内分式中依旧出现. 比如, `$\d y/\d x$` 会呈现为 $\mathrm{d}y/\mathrm{d}x$;
2. \d 不能用于数学模式以外的地方. 即 \d{o} 不能用于在文本模式下产生类似 “o̩” 的效果;

*https://github.com/AlphaZTX/fixdif

[1]See https://tex.stackexchange.com/questions/14821/whats-the-proper-way-to-typeset-a-differential-operator.

3. d 和它前面表达式之间的空白被视作乘积符号，而一个算符通常应该是二元的。拿 \cdot (·) 来举例. 当在“行间公式”或“行内公式”下 $x\cdot y$ 间的算符会保留，但在“角标公式”或“脚本脚本公式”则该消失，例如 $a^{x\cdot y}$。因此，该空白也应该在角标下消失，但是 $a^{f(x)\d x}$ 出来的 $a^{f(x)\,\mathrm{d}x}$ 依旧会存在空白，而不是期望中的 $a^{f(x)\mathrm{d}x}$。

如果想解决以上问题，你可以试试本宏包。

第 2 节　引言

在导言区使用以下命令即可加载本宏包

```
\usepackage{fixdif}
```

在文档区使用以下命令

```
\[ f(x)\d x,\quad\frac{\d y}{\d x},\quad\d y/\d x,\quad a^{y\d x}. \]
```

将会出现

$$f(x)\,\mathrm{d}x,\quad \frac{\mathrm{d}y}{\mathrm{d}x},\quad \mathrm{d}y/\mathrm{d}x,\quad a^{y\mathrm{d}x}.$$

2.1　兼容 unicode-math

如果你已经在文档里使用 XeTeX/LuaTeX 下的 unicode-math 宏包，那你得注意下面的问题:

- 如果要使用 amsmath 宏包，请确保 unicode-math 在 amsmath *之后* 被加载。
- 最好使用 unicode-math 提供的 \setmathfont 命令指定数学字体以避免在行内分式情况下出现多余空白的问题，如 $\mathrm{d}y/\mathrm{d}x$.
- fixdif 宏包一定要在 unicode-math *之后* 加载。

因此，正确的顺序应该是

```
\usepackage{amsmath}
\usepackage{unicode-math}
\setmathfont{...}[...]
\usepackage{fixdif}
```

2.2　兼容 hyperref

如果你还想同时使用 hyperref 宏包，那 hyperref 必须在 fixdif *之前* 被加载，否则 hyperref 宏包会报冲突。

2.3　基础命令以及宏包参数

\d fixdif 宏包提供的\d命令，既可以用作数学模式下的微分算子d，也可以用作文本模式下的重音标记命令，就像 LaTeX 或 plain TeX 里的 \d 命令一样。比如

```
$\d x$ 和 \d x
```

将会显示为 “$\mathrm{d}x$ 和 x̣”。

2.3.0.1　改变 \d 的字体

数学模式下\d有两种基本的宏包选项风格 — rm 和 normal. 默认为将$f(x)\d x$ 显示为 $f(x)\,\mathrm{d}x$ 的 rm。如果想用 normal 选项，那么

```
\usepackage[normal]{fixdif}
```

此时 $f(x)\d x$ 将会是 $f(x)\,dx$。

\resetdfont 除了以上两种字体，你还可以在导言区使用 \resetdfont 命令来改变 \d 的字体：

```
\resetdfont{\mathsf}
```

此时 \d x 将会是 $\mathsf{d}x$。

2.3.0.2　控制 \partial 的行为

\partial 默认情况下，\partial 也会被判定为一个数学模式下的微分算子。如果你想要改变这种行为，你可以用 nopartial 选项：

```
\usepackage[nopartial]{fixdif}
```

第 3 节　微分算子的定义

注意！本节涉及到的命令只能在导言区使用！

3.1　单命令定义

\letdif　\letdif{⟨*cmd*⟩}{⟨*csname*⟩}　(preamble only)

\letdif 命令有两个参数 — 第一个是被新定义的命令名，第二个是数学模式下一个字符的 名字（不包含\），这可以让该字符产生像\d一样的前置自适应空白，比如使用

```
\letdif{\vr}{delta}
```

后，\vr 就将会在 δ (\delta) 前加上自适应空白。

借助 \letdif 命令，我们可以轻松地用数学字符的代码名进行重定义。比如

```
\letdif{\delta}{delta}
```

此时，\delta 自己也会拥有像微分算子那样的前置自适应空白。

\letdif 命令的的第二个参数 ⟨*csname*⟩ 还可以反复使用。

\letdif*　\letdif*{⟨*cmd*⟩}{⟨*csname*⟩}　(preamble only)

这个命令与 \letdif 基本相同，但是它会在微分算子之后进行校正。这在当数学字体通过 unicode-math 宏包设置的情况下非常友好，比如

```
\usepackage{unicode-math}
\setmathfont{TeX Gyre Termes Math}
\usepackage{fixdif}
\letdif{\vr}{updelta}
```

会导致 \vr 之后追加不应有的空白，但如果你将上面的最后一行代码改成

```
\letdif*{\vr}{updelta}
```

就会得到正常的反馈。

3.2　多命令或字符串定义

\newdif　\newdif{⟨*cmd*⟩}{⟨*multi-cmd*⟩}　(without correction, preamble only)

\newdif*　\newdif*{⟨*cmd*⟩}{⟨*multi-cmd*⟩}　(with correction, preamble only)

这些命令的第一个参数依旧是待定义的命令名；且第二个参数必须包含 不止一个 LaTeX 命令。比如，在已经加载 xcolor 宏包的情况下你可以使用以下代码，

```
\newdif{\redsfd}{\textsf{\color{red}d}}
```

然后就会得到一个名为 \redsfd 的微分算子命令。再比如，使用

```
\newdif{\D}{\mathrm{D}}
```

你会得到一个名为 \D 的 D 微分算子。

如果你的第二个参数只有一个 LaTeX 命令，比如 \Delta，建议使用 \letdif 或 \letdif* 替代。

\newdif 和 \newdif* 会对 ⟨*cmd*⟩ 做是否已有定义的检查，如果有则会报错。

\renewdif　\renewdif{⟨*cmd*⟩}{⟨*multi-cmd*⟩}　(without correction, preamble only)

\renewdif*　\renewdif*{⟨*cmd*⟩}{⟨*multi-cmd*⟩}　(with correction, preamble only)

这两个命令和 \newdif、\newdif* 基本相同。唯一的区别是 \renewdif and \renewdif* 会对 ⟨*cmd*⟩ 进行 未 定义检查。如果没有定义，则会报错。

第 4 节　暂时使用微分算子

\mathdif　\mathdif{⟨*symbol*⟩}　(without correction, in math mode only)

\mathdif*　\mathdif*{⟨*symbol*⟩}　(with correction, in math mode only)

这两个命令只能在数学模式下使用，且只能用于文档区，即 \begin{document} 之后。比如 $x\mathdif{\Delta}\psi$ 会得到 $x\,\Delta\psi$。

第 5 节　参考示例

本节给出如何在你的文档里正确使用本宏包的示例。

比如以下两种例子:

```
\letdif{\Delta}{Delta}       % 例 1，在导言区使用
\letdif{\nabla}{nabla}       % 例 2，在导言区使用
```

实际上，第二种情况更加合理。虽然有时候我们会把 “Δ” 用作拉普拉斯算子（等价于 ∇^2），但有时候 “Δ” 又只能被当做变量或者函数使用。因此，推荐为 “Δ” 用作拉普拉斯算子时单独定义一个命令，而 `\Delta` 仅作为一个普通的数学符号 “Δ” 存在。然而，在绝大多数情况下，“∇” 被视为 nabla 运算符，因此无需保留 “∇”。所以可以这样修改以上代码：

```
\letdif{\laplacian}{Delta}   % Example 1, corrected, in preamble
```

我们还可以使用 xparse 宏包定义一个新的命令：

```
\letdif{\nabla}{nabla}
\DeclareDocumentCommand{ \laplacian }{ s }{
  \IfBooleanTF{#1}{\mathdif{\Delta}}{\nabla^2}
}
```

这样 `\laplacian` 会得到 ∇^2，而 `\laplacian*` 会得到 Δ。

5.0.0.1　处理 “+” and “−”

使用 `$-\d x$`，就会得到 “$-\mathrm{d}x$”，但如果你觉得应该是 “$-\mathrm{d}x$” 。那你可以使用 `-{\d x}`。“`\d x`” 在一个 *group* 会被视作 *ordinary*，而不是 *inner*，所以自适应空白会消失不见。当然，个人更推荐使用 “$-\,\mathrm{d}x$”。

第 6 节　源代码

1 ⟨*package⟩

Check the TEX format and provides the package name.

```
2 \NeedsTeXFormat{LaTeX2e}
3 \ProvidesPackage{fixdif}[2022/7/19 Interface for defining differential operators.]
```

6.1　Control the skip between slashes and differential operator

Change the math code of slash (/) and backslash (\) so that the skip between slashes and differential operators can be ignored.

```
4 \@ifpackageloaded{unicode-math}{
```

If the unicode-math package has been loaded, use the XETEX/LuaTEX primitive `\Umathcode` to change the type of slashes. The numeral “4” stands for “open”.

```
5   \Umathcode`\/="4 "0 "002F
6   \Umathcode"2044="4 "0 "2044
7   \Umathcode"2215="4 "0 "2215
8   \Umathcode"2F98="4 "0 "2F98
9   \Umathcode`\\="4 "0 "005C
10  \Umathcode"2216="4 "0 "2216
11  \Umathcode"29F5="4 "0 "29F5
12  \Umathcode"29F9="4 "0 "29F9
13 }{
```

If the unicode-math package has not been loaded, use the TEX primitive `\mathcode` to change the type of slashes. The `\backslash` needs to be redefined through `\delimiter` primitive too.

```
14  \mathcode`\/="413D
15  \mathcode`\\="426E % \backslash
16  \def\backslash{\delimiter"426E30F\relax}
17 }
```

6.2 Patch the skips around the differential operator

\mup@tch The following \mup@tch patches the skip after the differential operator.

```
\def\mup@tch{\mathchoice{\mskip-\thinmuskip}{\mskip-\thinmuskip}{}{}}
```

The \s@beforep@tch patches the commands with star (\letdif*, etc).

```
\def\s@beforep@tch{\mathchoice{}{}{\mbox{}}{\mbox{}}}
```

6.3 Declare the package options

Declare the options of the package and execute them.

```
\DeclareOption{rm}{\@ifpackageloaded{unicode-math}
  {\def\@@dif{\symrm{d}}}{\def\@@dif{\mathrm{d}}}}
\DeclareOption{normal}{\def\@@dif{d}}
\DeclareOption{partial}{\def\fixdif@partial@bool{1}}
\DeclareOption{nopartial}{\def\fixdif@partial@bool{0}}
\ExecuteOptions{rm,partial}
\ProcessOptions\relax
```

Control the behavior of \partial.

```
\def\fixdif@partial@true{1}
\ifx\fixdif@partial@bool\fixdif@partial@true
  \AtEndOfPackage{\letdif{\partial}{partial}}
\fi
```

\resetdfont Define the \resetdfont command.

```
\gdef\resetdfont#1{\let\@@dif\relax%
  \def\@@dif{#1{d}}}
```

6.4 Deal with the \d command

\@dif \@dif is the differential operator produced by \d in math mode. Here we prefer \mathinner to \mathbin to make the skip.

```
\def\@dif{\mathinner{\@@dif}\mup@tch}
```

\d@accent Restore the \d command in text by \d@accent with the \let primitive.

```
\let\d@accent\d
```

\d Redefine the \d command. In text, we need to expand the stuffs after \d

```
\DeclareRobustCommand\d{\ifmmode\@dif\else\expandafter\d@accent\fi}
```

6.5 User's interface for defining new differential operators

\letdif \letdif* Define the \letdif and \letdif* command. The internal version of \letdif is \@letdif, of \letdif* is \s@letdif.

```
\def\@letdif#1#2{\AtBeginDocument{%
```

#1 is the final command; #2 is the "control sequence name" of #1's initial definition. Here we create a command (\csname#2@old\endcsname) to restore #2.

```
  \ifcsname #2@old\endcsname\else%
  \expandafter\let\csname #2@old\expandafter\endcsname
    \csname #2\endcsname%
  \fi%
```

Finally let #1 be the new command.

```
  \gdef#1{\mathinner{\csname #2@old\endcsname}\mup@tch}%
}}
```

The definition of \s@letdif is similar, but with the patch for negative skips.

```
\def\s@letdif#1#2{\AtBeginDocument{%
  \ifcsname #2@old\endcsname\else%
  \expandafter\let\csname #2@old\expandafter\endcsname
    \csname #2\endcsname%
  \fi%
  \gdef#1{\mathinner{\s@beforep@tch\csname #2@old\endcsname\mbox{}}\mup@tch}%
}}
\def\letdif{\@ifstar\s@letdif\@letdif}
```

\newdif
\newdif*

Define the \newdif and \newdif* commands. #1 is the final command; #2 is the "long" argument.

```
\long\def\@newdif#1#2{\AtBeginDocument{%
  \ifdefined#1
    \PackageError{fixdif}{\string#1 is already defined.}
      {Try another command instead of \string#1.}%
  \else
    \long\gdef#1{\mathinner{#2}\mup@tch}%
  \fi%
}}
\long\def\s@newdif#1#2{\AtBeginDocument{%
  \ifdefined#1
  \PackageError{fixdif}{\string#1 is already defined.}
    {Try another command instead of \string#1.}%
  \else
    \long\gdef#1{\s@beforep@tch\mathinner{#2\mbox{}}\mup@tch}%
  \fi%
}}
\def\newdif{\@ifstar\s@newdif\@newdif}
```

\renewdif
\renewdif*

Define the \renewdif and \renewdif* commands.

```
\long\def\@renewdif#1#2{\AtBeginDocument{%
  \ifdefined#1
    \long\gdef#1{\mathinner{#2}\mup@tch}%
  \else
    \PackageError{fixdif}{\string#1 has not been defined yet.}
      {You should use \string\newdif instead of \string\renewdif.}%
  \fi%
}}
\long\def\s@renewdif#1#2{\AtBeginDocument{%
  \ifdefined#1
    \long\gdef#1{\s@beforep@tch\mathinner{#2\mbox{}}\mup@tch}%
  \else
    \PackageError{fixdif}{\string#1 has not been defined yet.}
      {You should use \string\newdif instead of \string\renewdif.}%
  \fi%
}}
\def\renewdif{\@ifstar\s@renewdif\@renewdif}
```

6.6 In-document commands: \mathdif and \mathdif*

```
\def\@mathdif#1{\mathinner{#1}\mup@tch}
\def\s@mathdif#1{\s@beforep@tch\mathinner{#1\mbox{}}\mup@tch}
\DeclareRobustCommand\mathdif{\@ifstar\s@mathdif\@mathdif}
```

End of the package.

```
⟨/package⟩
```