The odesandpdes package*

Anakin
anakin@ruc.dk

Released 2024/01/20

Abstract

This package is the solution no one asked for, to a problem nobody had. Have you ever thought to yourself "wow, I sure do dislike having to remember *multiple* macros for my odes (ordinary differential equation) and pdes (partial differential equation)" and the author of this package has to agree, wholeheartedly. In the modern world of "tik-toking" and "family guy surfing", our brains have rotted beyond salvage for even basic levels of cognitive recall. This package aims to fix this, through two macros that have been set to each have an identical form and function, with an emphasis on intuitive use. Thile there are multiple commonly used notational style, they are easily swapped between, all by a single option.
You're welcome.

Contents

1 Usage **3**
1.1 Options . . . 3
1.2 The Meat and Potatoes . . . 3
1.2.1 Notation Style . . . 4
1.2.2 Variable and Function Arguments . . . 4
1.2.3 Degree of Derivative . . . 4
1.2.4 Defining Where the Derivative is . . . 5

2 Examples of use **6**
2.1 Common Use Examples . . . 6
2.2 "at x;" Usage Examples . . . 7
2.3 Prime Count Limits . . . 8

3 Package Implementation **9**
3.1 Set-up . . . 9
3.1.1 Package Options . . . 9
3.2 Package Configuration . . . 10
3.2.1 To not conflict with amsmath . . . 10
3.3 Foundational macros . . . 11
3.3.1 The 'Yoinkers' . . . 12
3.3.2 Macro 'Checkpoints' . . . 13
3.4 Ancilliary Functions . . . 13

*This document corresponds to odesandpdes v1.1.0, dated 2024/01/20.

3.4.1 Variable Macronames 13
3.4.2 Determing the next token 14
3.5 Notational Morphology 15
3.5.1 Starred Forms 16
3.5.2 Unstarred Forms 16
3.5.3 “At Position” Forms 17
3.5.4 Foundational forms 17
3.6 Notational Shaping Tools 18

4 Index **21**

1 Usage

Start by first having `odesandpdes.sty` downloaded in an accessible directory,[1] using it by inserting;

`\usepackage[⟨options⟩]{odesandpdes}`

into the preamble, Ideally *after* any font changing packages you use.

If the reader does not wish to be gradually introduced to the package and its features, feel free to skip directly to section 2.

1.1 Options

`notation`
`maxprimes`

The options included are based off of the three most common notations (according to Wikipedia), Lagrange, Leibniz, and Newton. They can be accessed through the `[`⟨*options*⟩`]` when importing the package;

`\usepackage[notation=⟨option⟩]{odesandpdes}`

In the case of Lagrange or Newton notation, there is the `maxprimes` option for determination of how many physical markings are allowed to be made before the notation switches to a symbolic version;

`\usepackage[maxprimes=⟨integer⟩]{odesandpdes}`

`\setDE`

However, if one might wish to change it on a section to section basis, the command `\setDE{`⟨*options*⟩`}` is able to take any package option as an argument and will apply the new option going forward.

Option list	Default Value	Valid Arguments
notation	Leibniz	default, Lagrange, Leibniz, Newton
maxprimes	3	maxprimes = $n, n \in \mathbb{N}_+$

1.2 The Meat and Potatoes

The command(s) are approached with the philosophy of of an intuitive and modular usage. The full extent of its usage can look like;

`\ode*[x]^2 X(x) =\ode T_{\eta} at 0; -\alpha` $\Rightarrow \dfrac{d^2}{dx^2}X(x) = \left.\dfrac{dT_\eta}{dt}\right|_{t=0} - \alpha$

very quickly, and very easily building complex interactions of differentials. The quick functional break down of each element that comprises the macro;

`\ode`⟨*star*⟩`[`⟨*variable*⟩`]^`⟨*degree*⟩ `{`⟨*function*⟩`}` at␣⟨*position*⟩;

Argument	Usage
`[`⟨*variable*⟩`]`	The variable being derived
`^`⟨*degree*⟩	The order/degree of the derivative
`{`⟨*function*⟩`}`	The function being derived
at␣⟨*point*⟩;	Where the function is being derived

All arguments are conditionally optional, only the function is mandatory, but the command can forgo needing a function if a star is placed.

[1] if the reader is using Overleaf, it should already be accessible.

1.2.1 Notation Style

\LagrODE
\LeibODE
\NewtODE
\LagrPDE
\LeibPDE
\NewtPDE

There are 3 distinct notational styles one can choose between. This choice can be made as a package option in the preamble, in the text with `\setDE{⟨options⟩}`, or if one only needs to use a notation style once, through its respective macro.

In essense, all the `\ode` or `\pde` commands do are call the respective notational variant aligned with the currently set option. This makes it simple enough to just use one of the notational varients, should one wish to do so:

`\LagrODE[x] c = \LeibODE[x] c = \NewtODE[x] c` $\Rightarrow \quad c' = \frac{\mathrm{d}c}{\mathrm{d}x} = \dot{c}$

`\LagrPDE[x] c = \LeibPDE[x] c = \NewtPDE[x] c` $\Rightarrow \quad c_x' = \frac{\partial c}{\partial x} = \dot{c}$

This also means that all these functions are identical in what arguments they take.

1.2.2 Variable and Function Arguments

\ode
\ode*

The most barebone form can be understood as:

`\ode[⟨variable⟩]{⟨function⟩}`
`\ode*[⟨variable⟩]`

\pde
\pde*

and for the sake of parity, the PDE usage is identical:

`\pde[⟨variable⟩]{⟨function⟩}`
`\pde*[⟨variable⟩]`

Any value you give to the *optional* `[⟨variable⟩]` argument will be represented as the variable being derived. While the *mandatory* `{⟨function⟩}` argument will be the function you are deriving. Say you wish to indicate you are deriving $X(t)$, simple as writing `\ode[t]{X}`, however, its worth noting that t is the default variable so writing `\ode{X}` will produce identical results. Hence `\ode[t]{X}=\ode{X}` will produce;

`\ode[t]{X} =\ode{X}` $\Longrightarrow \frac{\mathrm{d}X}{\mathrm{d}t} = \frac{\mathrm{d}X}{\mathrm{d}t}$

While the `{⟨function⟩}` argument is mandatory using the non-starred command, using the starred varient omits the need for the `{⟨function⟩}` argument. Therefor, writing the exact same equation, just starred `\ode*[t]{X} = \ode*{X}` will instead produce;

`\ode*[t]{X} =\ode*{X}` $\Longrightarrow \frac{\mathrm{d}}{\mathrm{d}t}X = \frac{\mathrm{d}}{\mathrm{d}t}X$

Effectively one can rewrite the 'bare-bones' display as:

`\ode⟨star⟩[⟨variable⟩]{⟨function⟩}`

1.2.3 Degree of Derivative

The previously shown stated section is something the reader has likely encountered before, made themselves. This is where this package begins to differentiate[2] itself. Consider:

[2]Calculus Pun!

`\ode`⟨*star*⟩`[`⟨*variable*⟩`]^`⟨*degree*⟩`{`⟨*function*⟩`}`

A feature of this family of commands, is that it can '*easily*' recognize a following exponent should one be placed. There was rational in choosing to check for the exponent immediately after the macro command opposed to checking for the exponent at the end after the function. As, often you would want add a higher degree very quickly as opposed to *after* defining the function.

`\ode^2f(x)` as opposed to `\ode{f(x)}^2`

This was one of the main motivations of creating a package to begin with as instead of needing, maybe, two personalized commands, such as "`\ddt{f}` and `\ddxx{f}`", or "`\dd{x}{f}` and `\dd[2]{x}{f}`". One simply needs to treat the `\ode` macro itself as being raised to a higher degree.

`\ode* \left(\ode{f} \right)=\ode^2{f}` $\Rightarrow \frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\mathrm{d}f}{\mathrm{d}t}\right) = \frac{\mathrm{d}^2 f}{\mathrm{d}t^2}$

1.2.4 Defining Where the Derivative is

Imagine you, as the reader, are trying to quickly and easily write up the boundry conditions of your problem. One could always make another macro, in what is no doubt an impressive display of differential shortcuts. *Or*:

`\ode`⟨*star*⟩`[`⟨*variable*⟩`]^`⟨*degree*⟩`{`⟨*function*⟩`}` at␣⟨*postion*⟩;

See, TeX does something very interesting when it uses '*glue*', which is partially replicated by packages such as `TikZ`, where it will happily take 'soft' modifiers written directly in plain english. If one wishes to strictly define paragraph spacing in TeX, they would use '`\parskip=1ex`'. If one would rather give it a range of tolerance the following construct '`\parskip=1ex plus 0.5ex minus 0.5ex`' then allows a spacing of 1 ± 0.5 `ex`.

Glue is of course something special, but that does not mean that the author can not gain inspiration. Say one wishes to define Neumann boundries;

`\ode[x]{c} at 0;=0\land\ode[x]{c} at L;=1` $\Rightarrow \left.\frac{\mathrm{d}c}{\mathrm{d}x}\right|_{x=0} = 0 \land \left.\frac{\mathrm{d}c}{\mathrm{d}x}\right|_{x=L} = 1$

`\ode[x]{c} at 0 = L;=1` $\Rightarrow \left.\frac{\mathrm{d}c}{\mathrm{d}x}\right|_{x=0=L} = 1$

Literally could not be easier.[3]

Those reading til this point may have recalled that the first example did not contain many braces. This is because with the "proper" spacing, there is little need for the use of the braces, so as to help promote a more fluid, (and readable), workflow without always needing to worry about the f***ing brace. Not that one can not use the brace for personal taste. In the following section, many examples of use will be illustrated to show the range and versitility of the functions.

The most important thing to always remember. *Just because* the author of this package has done as much as they can to '*~~idiot~~ user proof*' its functions does not mean the user does not still need to be cautious. This is LaTeX we are talking about. There are likely many scenarios that the author did not think of, nor accidentally came across.

[3] My source is that I made it up

2 Examples of use

To show the generality of use. The following examples all take identical form in the TEX/LATEX itself. Additionally, in order to illustrate the functional boundries of the command with respect to each of the notational styles. There is a variety of spacing and bracketing to help highlight these features, and will be shown in the following `verbatim` enviroment;

```
\begin{align*}
\ode A(x)      && \ode[x]{B(x)} && \ode^1 C(x)     && \ode[x]^5 {D(x)} \\
\ode* {E(x)}   && \ode*[x] F(x) && \ode*^2 {G(x)}  && \ode*[x]^6H(x)   \\
\pde[t] I(x)   && \pde[x] {J(x)}&& \pde[t]^3K(x)   && \pde[x]^7 {L(x)} \\
\pde*[t] {M(x)}&& \pde*[x]N(x)  && \pde*[t]^4 O(x) && \pde*[x]^8 P(x)
\end{align*}
```

`\setDE{notation=`⟨*Lagrange*⟩`}` *and/or* `\usepackage[notation=`⟨*Lagrange*⟩`]{odesandpdes}`

$$\begin{aligned}
&A'(x) && B(x)' && C'(x) && D(x)^{(5)} \\
&f'(t)E(x) && f'(x)F(x) && f''(t)G(x) && f^{(6)}(x)H(x) \\
&I_t'(x) && J(x)_x' && K_t'''(x) && L(x)_x^{(7)} \\
&f_t'(t)M(x) && f_x'(x)N(x) && f_t^{(4)}(t)O(x) && f_x^{(8)}(x)P(x)
\end{aligned}$$

`\setDE{notation=`⟨*Leibniz*⟩`}` *and/or* `\usepackage[notation=`⟨*Leibniz*⟩`]{odesandpdes}`

$$\begin{aligned}
&\frac{\mathrm{d}A(x)}{\mathrm{d}t} && \frac{\mathrm{d}B(x)}{\mathrm{d}x} && \frac{\mathrm{d}C(x)}{\mathrm{d}t} && \frac{\mathrm{d}^5D(x)}{\mathrm{d}x^5} \\
&\frac{\mathrm{d}}{\mathrm{d}t}E(x) && \frac{\mathrm{d}}{\mathrm{d}x}F(x) && \frac{\mathrm{d}^2}{\mathrm{d}t^2}G(x) && \frac{\mathrm{d}^6}{\mathrm{d}x^6}H(x) \\
&\frac{\partial I(x)}{\partial t} && \frac{\partial J(x)}{\partial x} && \frac{\partial^3 K(x)}{\partial t^3} && \frac{\partial^7 L(x)}{\partial x^7} \\
&\frac{\partial}{\partial t}M(x) && \frac{\partial}{\partial x}N(x) && \frac{\partial^4}{\partial t^4}O(x) && \frac{\partial^8}{\partial x^8}P(x)
\end{aligned}$$

`\setDE{notation=`⟨*Newton*⟩`}` *and/or* `\usepackage[notation=`⟨*Newton*⟩`]{odesandpdes}`

$$\begin{aligned}
&\dot{A}(x) && B\dot{(x)} && \dot{C}(x) && D\overset{5}{\dot{(x)}} \\
&\dot{t}E(x) && \dot{x}F(x) && \ddot{t}G(x) && \overset{6}{\dot{x}}H(x) \\
&\dot{I}(x) && J\dot{(x)} && \dddot{K}(x) && L\overset{7}{\dot{(x)}} \\
&\dot{t}M(x) && \dot{x}N(x) && \overset{4}{\dot{t}}O(x) && \overset{8}{\dot{x}}P(x)
\end{aligned}$$

`\setDE{maxprimes=⟨7⟩}` *and/or* `\usepackage[maxprimes=⟨7⟩]{odesandpdes}`

$$f' \quad f'' \quad f''' \quad f'''' \quad f''''' \quad f'''''' \quad f''''''' \quad f^{(8)} \quad f^{(9)}$$

$$\dot{f} \quad \ddot{f} \quad \dddot{f} \quad \ddddot{f} \quad \overset{\cdot\cdot\cdot\cdot\cdot}{f} \quad \overset{\cdot\cdot\cdot\cdot\cdot\cdot}{f} \quad \overset{\cdot\cdot\cdot\cdot\cdot\cdot\cdot}{f} \quad \overset{8}{\dot{f}} \quad \overset{9}{\dot{f}}$$

2.2 "at x;" Usage Examples

Now, because the author is not an insane person, and went through the effort of learning how TEX deconstructs text into constitute registries and boxes, the way any sane person might. When using a non-starred version of a command, after the function is defined, you can place an '`at␣`⟨*point*⟩`;`', and the representation will shown according to notational convention.

```
\begin{align*}
    \ode[x]  c at 23\pi;   &= 1 \\
    \ode[x]^3 c   at 69;   &= 2 \\
    \ode[x]^{69} c at L;+t &= 3 \\
    \ode[x]^9  c af 420;   &= 4 \\
    \ode[x]^6  c  a t 13;  &= 5
\end{align*}
```

`\setDE{notation=Lagrange}`	`\setDE{notation=Leibniz}`	`\setDE{notation=Newton}`
$c'(23\pi) = 1$	$\left.\frac{\mathrm{d}c}{\mathrm{d}x}\right\rvert_{x=23\pi} = 1$	$\dot{c}(23\pi) = 1$
$c'''(69) = 2$	$\left.\frac{\mathrm{d}^3c}{\mathrm{d}x^3}\right\rvert_{x=69} = 2$	$\dddot{c}(69) = 2$
$c^{(69)}(L) + t = 3$	$\left.\frac{\mathrm{d}^{69}c}{\mathrm{d}x^{69}}\right\rvert_{x=L} + t = 3$	$\overset{69}{\dot{c}}(L) + t = 3$
$c^{(9)}af420; = 4$	$\frac{\mathrm{d}^9c}{\mathrm{d}x^9}af420; = 4$	$\overset{9}{\dot{c}}af420; = 4$
$c^{(6)}at13; = 5$	$\frac{\mathrm{d}^6c}{\mathrm{d}x^6}at13; = 5$	$\overset{6}{\dot{c}}at13; = 5$

As can be seen in the examples, this '*modifier*' is robust enough that one can write effectively any combination of characters after the function, excluding, *verbatim*, '`at␣`' and it will work as intended.

Important to note, due to a slight difference in how the notational styles are defined, only the Leibniz notation can take arguments for the function that involve subscripts and superscripts without delimiters. Mostly easily illustrated in this following example using the \pde command;

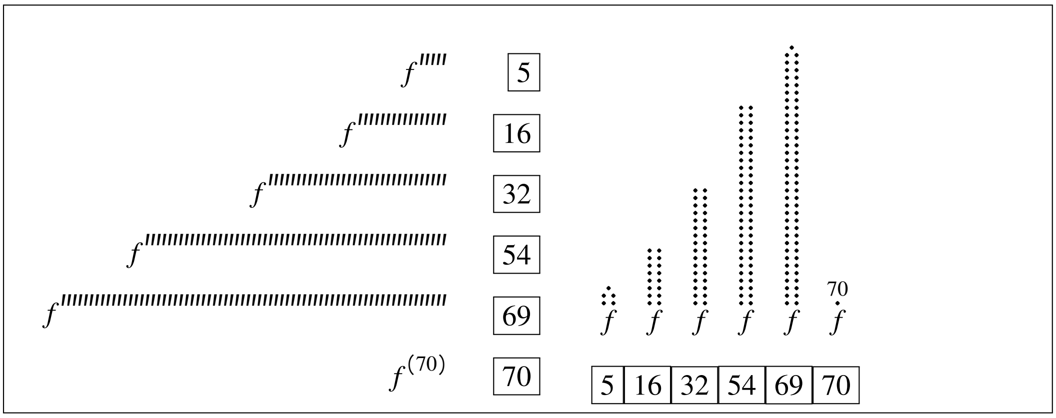
```
\begin{align*}
   \pde[y]   f_1         &= 1 \\
   \pde[y]   f_1   at L; &= 2 \\
   \pde[y]   f     at L; &= 3 \\
   \pde[y] {(f_1)}       &= 4 \\
   \pde[y] {(f_1)} at L; &= 5
\end{align*}
```

\setDE{notation=Lagrange}	\setDE{notation=Leibniz}	\setDE{notation=Newton}
$f'_{y_1} = 1$	$\frac{\partial f_1}{\partial y} = 1$	$\dot{f}_1 = 1$
$f'_{y_1}atL; = 2$	$\left.\frac{\partial f_1}{\partial y}\right\vert_{y=L} = 2$	$\dot{f}_1atL; = 2$
$f'_y(L) = 3$	$\left.\frac{\partial f}{\partial y}\right\vert_{y=L} = 3$	$\dot{f}(L) = 3$
$(f_1)'_y = 4$	$\frac{\partial (f_1)}{\partial y} = 4$	$(\dot{f_1}) = 4$
$(f_1)'_y(L) = 5$	$\left.\frac{\partial (f_1)}{\partial y}\right\vert_{y=L} = 5$	$(\dot{f_1})(L) = 5$

2.3 Prime Count Limits

Because the Newton and Lagrange notation is procedural; the only limit is your imagination, and also the fact that TEX can only have something like 127 unplaced tokens at a time.

\setDE{maxprimes=69}



3 Package Implementation

As a fair warning for anyone interested in the implementation of this package, it is documented in what might be considered, *absurd* levels of detail. This comes from the creation of this package being a great learning experience for the author, and the in-depth documentation of that understanding is only beneficial. Futhermore, a lot of the techniques used in this package are not obvious. Some of which, to paraphrase the creator of TeX, his divine emmisary *Donald E. Knuth* himself in the ever holy TeXbook, were prefaced with "*Worthy of being known to, at least a few, wizards able to traverse the nether world of TeXarcana*".

3.1 Set-up

Package options are difficult to deal with, so using the `xkeyval` package alleviates much of the *pain* associated with it,

```
\RequirePackage{xkeyval}
```

`\m@xm@rk` `\exp@c@unt` `\@detempv@l`

Being that there are a lot of minor calculations within the package reserving registries for integer counts feels like a good idea

```
\newcount\m@xm@rk%
\newcount\exp@c@unt%
\countdef\@detempv@l=255%
```

`\v@rr@t@ks` `\func@t@ks` `\@tpost@ks`

As well reserving token registries for tossing arguments around the groups and macros,

```
\newtoks\v@rr@t@ks%
\newtoks\func@t@ks%
\newtoks\@tpost@ks%
```

`\@dev@rb@x` `\@defunb@x` `\@deresb@x`

Reserving box registries for the purpose of collecting the components together in a coherent manner. Starting already here, one might notice the fact that most macros stat with `\@de`, this helps 'protect' the definitions from being used accidentally, and makes the commands easily identifiable.

```
\newbox\@dev@rb@x%
\newbox\@defunb@x%
\newbox\@deresb@x%
```

3.1.1 Package Options

`\@de@option`

Defining the package options for notational styles using the LaTeX `\providecommand` to reloading times. Important to note that defining the command is not the same as using the command, which is useful in conjunction with `\csname` and `\endcsname` for macro defintions.

```
\providecommand\@de@option{Leib}
```

Now using the `keyval` package, it becomes possible to define a family of package options associated with inputing some `notation=#1`. This allows for easily defining the notation for the entire document. The possible options will be defined afterwards,

```
\DeclareOptionX{notation}[default]%
    {\def\@de@option{\csname @de@not@#1\endcsname}}
```

`\@de@not@Lagrange` `\@de@not@Leibniz` `\@de@not@Newton` Once the package option has been declared, now the options can be defined. The options take identical form with the exception of the last part of definition. This is because the `\@de@option` is not the macro used for the notation definitions. Rather, `\@de@option` is an intermediate that expands into one of the defined options, which subsequently expands into one of the four character strings, “`Lagr`”, “`Leib`”, or “`Newt`”

$$\begin{tikzcd} \text{\@de@option} \arrow[r, bend right, "\text{expands to}"'] & \text{\@de@not@``option''} \arrow[r, bend left, "\text{expands to}"] & \text{``string''} \end{tikzcd}$$

```
\def\@de@not@Lagrange{Lagr}
\def\@de@not@Leibniz{Leib}
\def\@de@not@Newton{Newt}
```

`\@de@not@default` The default option for the notation is defined by pointing to the definition of the `Leibniz` notation option,

```
\let\@de@not@default\@de@not@Leibniz
```

A second option is defined to allow freedom in deciding the cut-off point for the Lagrange and Newton notations where it no longer makes more physical marks and uses the symbolic extension instead, with a default of 3 marks before becoming symbolic.

```
\DeclareOptionX{maxprimes}[3]{\m@xm@rk=#1\advance\m@xm@rk\@ne}
```

To ensure that all other options given to the package will be ignored the star is used to indicate that all undefined options will be directed towrds this declared option,

```
\DeclareOptionX*{\PackageWarning{odesandpdes}{`\CurrentOption' ignored}}
```

Finally the declared options are executed as to allow the default options to initialize and be processed,

```
\ExecuteOptionsX{notation,maxprimes}
\ProcessOptionsX\relax
```

3.2 Package Configuration

`\setDE` In addition to being able to use options directly in the `\usepackage` package command, one also gets access to the command `\setDE`. Which can be used at any point in the document to change the style of notation or max prime count. Functionally done in identical manner to how `\DeclareOptionX` is used.

```
\newcommand\setDE[1]{\setkeys[package]{@de}{#1}}
```

3.2.1 To not conflict with amsmath

`\@de@ver` `\@de@top` `\@de@bove` Purely because amsmath is a bitch and doesn’t want anyone enjoying their time in TeX it becomes required to make compatibility checks and work within their abstracted definitions,

```
\@ifpackageloaded{amsmath}{
```

```
    \let\@de@ver=\@@over%
    \let\@de@top=\@@atop%
    \let\@de@bove=\@@above}%
```

Otherwise it just uses the TEX primitive commands for fractions because of increase ease of function and speed of processing,

```
    {\let\@de@ver=\over%
    \let\@de@top=\atop%
    \let\@de@bove=\above}
```

3.3 Foundational macros

\d@@ \d@l

Creating protected macro definitions for increase in speed of processes,

```
\def\d@@{\mathrm d}
\let\d@l=\partial
```

\@dest@red \@den@st@r

In the same vein, strings are defined for the starred and unstarred versions of macro commands,

```
\def\@dest@red{st@r@d}
\def\@den@st@r{n@st@r}
```

\ode \pde

The macro definitions of the ODE and PDE commands

```
\def\ode{\csname \@de@option ODE\endcsname}
\def\pde{\csname \@de@option PDE\endcsname}
```

In essence these two are the same command. This is done for the sake of consistancy in use and effect. As well, in an attempt to not make the alternative notations *inaccesible*, the main macros are themselves stepping stones to the package declared option. As perhaps multiple notational styles might be useful in a single equation, who knows?

\LagrODE \LeibODE \NewtODE

There is unfortunately no way to avoid the process of making an individual macro for each ODE version;

```
\def\LagrODE{\let\@de@perat@r\d@@% sets the d
    \let\@dec@mm@nd\@de@not@Lagrange
    \@de@ifst@r}
\def\LeibODE{\let\@de@perat@r\d@@%
    \let\@dec@mm@nd\@de@not@Leibniz
    \@de@ifst@r}
\def\NewtODE{\let\@de@perat@r\d@@%
    \let\@dec@mm@nd\@de@not@Newton
    \@de@ifst@r}
```

\LagrPDE \LeibPDE \NewtPDE

As well as making a macro for each PDE version;

```
\def\LagrPDE{\let\@de@perat@r\d@l% sets the del
    \let\@dec@mm@nd\@de@not@Lagrange
    \@de@ifst@r}
\def\LeibPDE{\let\@de@perat@r\d@l%
    \let\@dec@mm@nd\@de@not@Leibniz
    \@de@ifst@r}
\def\NewtPDE{\let\@de@perat@r\d@l%
    \let\@dec@mm@nd\@de@not@Newton
    \@de@ifst@r}
```

In terms of usage, these are all the same command, the main differences come from what the operator is defined as, `\d@@` or `\d@l`, and which notational form that `\@dec@mm@nd` points at for further processes down the stream. They are however, given all caps for the *ode* and *pde* in order to enhance visual clarity should one use them.

3.3.1 The 'Yoinkers'

`\@dest@r@rg` `\@de@ption@l@rg` `\@de@exp@n@rg`

Now a group of functions are needed for the processing each of the major elements, the star (`*`), for whether to have a function parameter. The option (`[`), for determining the variable being differentiated. And exponent (`^`), for deteriming what order the differential should be. Whether these functions should be used or not, comes from the use of a macro described in section 3.3.2.

Importantly each of these elements, should they appear, require the relevant token to be 'yoinked' by the macro in question. Should a star appear, `\@dest@r@rg` 'gobbles' said star and propmts the next element, an optional argument, to be checked for.

```
\def\@dest@r@rg*{\expandafter\@de@ifbr@ck}
```

For an optional argument, `\@de@option@l@rg` will yoink the argument, as well as the surrounding brackets,

```
\def\@de@ption@l@rg[#1]{\expandafter\v@rr@t@ks{#1}\relax \@de@ifexp@n}%
```

If an exponent should appear, `\@de@exponent@rg` will yoink the `^`, and the integer following it,

```
\def\@de@exp@n@rg^#1{\exp@c@unt#1\relax \@deifst@rred}
```

`\@dest@r@dy@ink` `\@den@st@ry@ink`

Depending on if one is using the starred version of the command, there is a command that yoinks the following function variable and one that ends the compiling here.

```
\def\@dest@r@dy@ink{\expandafter\@dec@mpf@rm}
\def\@den@st@ry@ink{\expandafter\@dey@inkf@rm}
```

`\@de@func@ther` `\@de@func@Leib` `\@de@func@Lagr` `\@de@func@Newt`

As a consequence of the inherent differences in how the notational styles treat functions, the `\@de@func@Leib` macro has to be treated differently. Whereas both the Lagrange and Newton notations will just accept the first token following the call of the function yoinker. The Leibniz varient will attempt to absorb all the tokens untill the first space token is found. This is not done in the traditional way of denoting an explicit space token at the end of the control sequence, but rather through a special macro defined in section 3.4.2. This had to be done as a consequence of getting the '`at␣x;`' function to work properly.

```
\def\@de@func@ther#1{\expandafter\func@t@ks{#1}\relax
    \expandafter\@de@if@tpos}
\def\@de@func@Leib{\expandafter\@de@ifbrace}
\let\@de@func@Lagr\@de@func@ther
\let\@de@func@Newt\@de@func@ther
```

`\@de@tpos@rg`

Finally, the last element that can be used, is designed to eat all the tokens between its call and the first semi-colon it sees, to ensure a function can be derived anywhere.

```
\def\@de@tpos@rg#1;{\expandafter\@tpost@ks{#1}\relax \@de@tf@rm}
```

3.3.2 Macro 'Checkpoints'

\@de@ifst@r
\@de@ifbr@ck
\@de@ifexp@n

As can be seen in the definitions of the \ode and \pde, there are no explicitely defined \ode* or \pde* macros. A workaround is implemented by making the first step of the macro to check if the first token that appears is a star, or *asterisk*, if one would prefer the technical language. These macros make use of an ancilliariy function \@deifch@r, which is defined in the section 3.4.2.

```
\def\@de@ifst@r{\@deifch@r *
    {\@dest@rgument\@dest@red\@dest@r@rg}
    {\@dest@rgument\@den@st@r\@dest@r@rg*}}
\def\@de@ifbr@ck{\@deifch@r [
    \@de@ption@l@rg
    {\@de@ption@l@rg[t]}}
\def\@de@ifexp@n{\@deifch@r ^
    \@de@exp@n@rg
    {\@de@exp@n@rg^\@ne}}
```

\@de@ifbrace

\@de@ifbrace is a bit more special than the other \@deif conditionals, as it is not a general use conditional. Only the Leibniz notational style function yoinker makes use of it. This is likely not a good long-term solution, but that just means it's going to be this way for at least a few years.

```
\def\@de@ifbrace{\@deifch@r \bgroup
    \@de@func@ther
    \@de@tilsp@ce}
```

\@de@if@tpos
\@de@ttw@f@c

In the same way, there also exist a macro to check for the 'at␣'. The main difference however, is the follow up command that helps *robustify* \@de@if@tpos. This is done through absorbing all the tokens after the 'a' until the next space token, if only a single token is absorbed, and that token is a 't', then success! Otherwise nothing happens.[4]

```
\def\@de@if@tpos{\@deifch@r a \@de@ttw@f@c \@dec@mpf@rm}
\def\@de@ttw@f@c a#1 {\ifx t#1\expandafter
    \@de@tpos@rg\else
    \@dec@mpf@rm a#1\fi}%
```

3.4 Ancilliary Functions

There are a lot of macros or command sequences that need to be used in addendum to the main commands that one would download this package for. As a consequence, there are a plethora of ancilliary functions to pull from defined in this section.

3.4.1 Variable Macronames

\@dest@rgument
\@deifst@rred
\@dec@mpf@rm

It becomes useful to be able to freely define which macro to be used when going through the option tree. Subsequently, three macros are defined to fufill that purpose.

[4] There is a way to make this function in a far more generalized way using \csname and \endcsname. However, as this package makes use of this feature exactly *once*, there is no benefit to generalizing the functionality.

\@dest@rgument takes an argument and defines two macros \@deifst@rred which defines whether the function 'yoinker' exists or not, and \@dec@mpf@rm which works with \@de@option,defined in subsection 3.2, to define the final ODE or PDE form.

```
\def\@dest@rgument#1{%
    \def\@deifst@rred{\csname @de#1y@ink\endcsname}%
    \def\@dec@mpf@rm{\csname#1@\@dec@mm@nd\endcsname}}
```

\@de@tf@rm
\@dey@inkf@rm

Additional macros are also defined for determining intermediate forms during the construction process of the resulting ODEs and PDEs

```
\def\@de@tf@rm{\csname @de@t@\@dec@mm@nd\endcsname}%
\def\@dey@inkf@rm{\csname @de@func@\@dec@mm@nd\endcsname}%
```

3.4.2 Determing the next token

An integral part of the '*mastication*' process is the identification of the proceeding token in the oncoming token stream. Therefore, a macro is defined to streamline this process instead of needing to create a unique \futurelet sequence for each token type.

The use of \futurelet is a strange and arcane process that better described by occult terminology than the proper scientific terms one would use in daily life. However, it is important to understand at least a little bit for the implementation of the \@deifch@r macro.

\@deifch@r
\@det@stt@k
\@de@tmpA
\@de@tmpB

\@deifch@r takes in three tokens as arguments, the first argument will assign \@detesttoken and be what the macro looks out for, while the other two arguments are for storage to be executed later. Building off this, there are two main elements that compose the macro, the namesake \@deifch@r, and its supplement macro \@denext@rg. This is because \futurelet is a primitive that will act as the \let primitive, just one token removed.

\let token1 token3

\let token1 ← token2 token3 \futurelet token1 token2 token3

The most important consequence is that, should \futurelet be enacted upon a stream of three tokens, "\futurelet token1 token2 token3"; token1 will be \let to point at token3 *before* token2 is expanded. What this means, is one is able to have token3 *act upon the unexpanded* token2.[5]

```
\def\@deifch@r#1#2#3{%
    \let\@dew@tcht@k=#1\relax
    \def\@de@tmpA{#2} \def\@de@tmpB{#3}
    \futurelet\@det@stt@k\@denext@rg}
```

Using this *enlightenment*, define the token representing an 'if-then-else' control sequence \@denext@rg. In \@deifch@r, \@dew@tcht@k becomes a macro for the token we want to check against. Using this to our advantage, before TeX expands \@denext@rg, it will assign \@detesttoken to point to a third, currently, unknown token after \@denext@rg. This is where the magic happens; because \@denext@rg only expands *after* the assignment of \@detesttoken, meaning it becomes possible to compare \@detesttoken and \@dew@tcht@k against eachother to determine which outcome should be executed.

[5] If this means something to you, it's too late. You've lost your chance of escaping TeX.

\@denext@rg \@de@nextact The first half of `\@denext@rg` ensures that a space tokens does not get in the way of assignment, as unfortunate as it is, the `\futurelet` primitive *does* consider a space token to be a valid token to point to.

```
\def\@denext@rg{%
    \ifx\@det@stt@k\@sptoken\relax
        \let\@de@nextact\@desp@cegobbler\else
```

The second half of `\@denext@rg` is what does the actual comparison. Should the comparison be positive, `\@detesttoken = \@dew@tcht@k`, then the code stored in `\@de@tmpA` will be executed, otherwise, `\@de@tmpB` will be executed

```
        \ifx\@det@stt@k\@dew@tcht@k\relax  % if
            \let\@de@nextact\@de@tmpA\else   % ifn't
            \let\@de@nextact\@de@tmpB\fi\fi
    \@de@nextact}
```

\@desp@cegobbler Ensuring that the space(s), explicit or implicit, trailing after `\@deifch@r` requires some TEX *tomfoolary*. By defining the function with a non-character token, the trailing space will matter for the macro definition, thereby, creating a macro that gobbles one space token on use.

```
\def\<{\@desp@cegobbler}
\expandafter\def\< {\futurelet\@det@stt@k\@denext@rg}
```

These three macros work together as a three point cycle discarding spaces until the first non-space token is found, in which case the `\if`-`\else` will be executed.

\@de@tilsp@ce While the previous macro gobbles space tokens until it finds a non-space token `\@de@tilsp@ce` gobbles non-space tokens until it finds a space token. There is a difference however, in that `\@de@tilsp@ce` stores the gobbled tokens until it finds that space token, subsequently ~~ejaculating~~ *returning* the the tokens as a registry list.

```
\def\@de@tilsp@ce#1 {%
    \beginnext%
    \toks0={#1}
    \edef\next{\func@t@ks=\expandafter{\the\toks0}}
    \endnext \@de@if@tpos}
```

\beginnext \endnext The `\beginnext`, `\endnext` construct is a relatively common construct one finds when working with variable macros and subsequently working with `\edef` commands. Using the explicit `\begingroup` and `\endgroup` group denotions means that one can play all sorts of registry based games, that can not be broken by implicit groupings. By `\edef`'ing `\next` inside this construct, whatever finalized product you have assigned to `\next`, will be a fully expanded assortment of values from those registries.

```
\def\beginnext{\begingroup
    \let\next\undefined}
\def\endnext{\expandafter\endgroup\next}
```

3.5 Notational Morphology

There is nothing particularly interesting about the methodology behind preparing the output forms, just using the classical TEX methods of exponents and fractions. So while these macro definitions will be left in, there won't be much commenting on them directly. The follow-up section will be illustrating the macros used *within* the ode replacement text, those will be explained.

One thing of note, is that these macros make *heavy* use of the '`\the\registry`' commands to expand registries previously used for storing tokens, and integers. Another hugely important element in these macros are the `\box` commands for arranging and subsequently storing said arrangement into a *box* which can then float to the top of the groupings like a message in a bottle.

3.5.1 Starred Forms

`\st@r@d@Lagr` Macro for Lagr+star

```
\def\st@r@d@Lagr{%
    \setbox\@deresb@x\hbox{$
        {f^{\mkern1mu\@dedr@wm@rk\lagr@prime\lagr@prime\br@ced@xpon}
        _{\m@kep@rtLagr}}\mkern-\tw@ mu\left(\the\v@rr@t@ks\right)
        $}%
    \@derele@se}%
```

`\st@r@d@Leib` Macro for Leib+star

```
\def\st@r@d@Leib{%
    \setbox\@defunb@x\hbox{$\@de@perat@r^{\@deem@rex}$}%
    \b@se@Leib}%
```

`\st@r@d@Newt` Macro for Newt+star

```
\def\st@r@d@Newt{%
    \setbox\@dev@rb@x\hbox{$\the\v@rr@t@ks$} \b@se@Newt}%
```

3.5.2 Unstarred Forms

`\n@st@r@Lagr` Macro for Lagr

```
\def\n@st@r@Lagr{%
    \setbox\@deresb@x\hbox{$
        {\the\func@t@ks
        ^{\mkern\@ne mu\@dedr@wm@rk\lagr@prime\lagr@prime\br@ced@xpon}
        _{\m@kep@rtLagr}}\mkern\m@ne mu$}%
    \@derele@se}%
```

`\n@st@r@Leib` Macro for Leib

```
\def\n@st@r@Leib{%
    \setbox\@defunb@x\hbox{$
        \@de@perat@r^{\@deem@rex}\mkern0.40mu\the\func@t@ks$}
        \b@se@Leib}
```

`\n@st@r@Newt` Macro for Newt

```
\def\n@st@r@Newt{%
    \setbox\@dev@rb@x\hbox{$\the\func@t@ks$} \b@se@Newt}%
```

3.5.3 "At Position" Forms

\@de@t@Lagr Macro for Lagr at point

```
\def\@de@t@Lagr{%
    \noexpand\hbox{$
        \n@st@r@Lagr\mkern-\thr@@ mu\left(\the\@tpost@ks\right)
        $}}%
```

\@de@t@Leib Macro for Leib at point

```
\def\@de@t@Leib{%
    \noexpand\hbox{$
        \left.\n@st@r@Leib\mkern\@ne mu\right|
        _{\mkern1mu\displaystyle\the\v@rr@t@ks\mkern2mu
        \rlap{$\scriptstyle=\mkern\thinmuskip\the\@tpost@ks$}}
        $}%
    }%
```

\@de@t@Newt Macro for Newton at point

```
\def\@de@t@Newt{%
    \noexpand\hbox{$
        \n@st@r@Newt\mkern-\tw@ mu\left(\the\@tpost@ks\right)
        $}}%
```

3.5.4 Foundational forms

\m@kep@rtLagr Macro for Lagr partial notations

```
\def\m@kep@rtLagr{\ifx\@de@perat@r\d@l\the\v@rr@t@ks\else\empty\fi}
```

\b@se@Leib Macro for the base Leibniz form

```
\def\b@se@Leib{%
    \setbox\@dev@rb@x\hbox{$
        \@de@perat@r\mkern0.40mu\the\v@rr@t@ks^{\@deem@rex}$}%
    \setbox\@deresb@x\hbox{\kern0.5\p@%
        $\raise2\p@\box\@defunb@x\@de@ver\lower5\p@\box\@dev@rb@x$%
        \kern0.5\p@}%
    \@derele@se}%
```

\b@se@Newt Macro for the base Newton form

```
\def\b@se@Newt{%
    \setbox\@defunb@x\hbox{\vbox{\baselineskip=\z@\lineskip=\m@ne\p@%
        \@dedr@wm@rk\@de@ned@ts\@detw@d@ts\@denewt@nd@t}}%
    \setbox\@deresb@x\hbox{\vbox{\baselineskip=\z@\lineskip=-0.5\p@%
        \hbox to\wd\@dev@rb@x{\hss\raise\z@\box\@defunb@x\hss}%
        \hbox{\raise\z@\box\@dev@rb@x}}}%
    \@derele@se}
```

\m@kep@rtNewt Macro for Newt partial notations

```
\def\m@kep@rtNewt{\ifx\@de@perat@r\d@l\empty\fi}
```

3.6 Notational Shaping Tools

Here's where some spice comes back into play. One of the major challenges[6] was ensuring that the appropriate number of primes or dots were placed when changing the maxprimes option.

Did the author realistically need to make it so one could have a procedural number of primes/dots? Nope. Would there ever be a realistic use-case for a derivative of order 3 or higher in which one would use markings? Of course not. Did the author do it anyways? Absolutely.

\lagr@prime \br@ced@xpon

The macro for the Lagrangian prime is very straightforward each time \lagr@prime is used, a prime mark will be placed, and the exponent count will reduce by one. The function does this repeatedly until the exponent count is reduced to 1.

```
\def\lagr@prime{\mkern0.35mu\prime\global\advance\exp@c@unt\m@ne}
```

Should the exponent count be greater than the maximum allowed prime markings, \br@ced@xpon will be used instead, which will display the general form of an integer enclosed by parenthesis.

```
\def\br@ced@xpon{\left(\the\exp@c@unt\right)}
```

\@detw@d@ts \@de@ned@ts

The dots for the Newtonian notation are more complicated than just incrementing a counter by one for each placed mark. Because Newtonian notation is built with a point at the top, it requires the initial dot to be place prior the rest of the dots as the \vbox primitive builds top down

In order to deal with that, this set of macros, \@detw@d@ts and \@de@ned@ts will take the exponent count, and determine if the number is ≡ mod2 if it is congruent. There is no initial dot created, if it is not congruent *and* a greater value than the set maxprimes, an initial dot is placed into the token stream to become the star on top.

The reason for these macros to be so complicated, is that TEX only has addition, and multiplication with integer registries. There is no divsion or float value functionality.

```
\def\@detw@d@ts{\ifnum\exp@c@unt>\@ne%
        \advance\exp@c@unt-\tw@\hbox to 5\p@{\hss$\cdot\cdot$\hss}\fi}%
\def\@de@ned@ts{\@detempv@l=\the\exp@c@unt%
    \loop\ifnum\@detempv@l>\tw@%
        \advance\@detempv@l-\tw@\repeat%
    \ifnum\@detempv@l<\tw@%
        \advance\exp@c@unt\m@ne\hbox to 5\p@{\hss$\cdot$\hss}\fi}%
```

\@denewt@nd@t

The generalized form of the the Newtonian derivative notation is is just a glorified fraction, with a dot as the denominator, and a number as the numerator.

```
\def\@denewt@nd@t{\hbox{\vbox{%
    \hbox to 5\p@{\hss\raise\thr@@\p@\hbox{$\scriptstyle\@deem@rex$}\hss}%
    \hbox to 5\p@{\hss\hbox{$\displaystyle\cdot$}\hss}}}}%
```

\@deem@rex

For the Leibniz notation, there is no reason to display the exponent should it be an integer value less than 2, therefor, any exponent count less than two will be replaced with \empty.

```
\def\@deem@rex{\ifnum\tw@>\exp@c@unt\empty\else\the\exp@c@unt\fi}
```

[6] Aside from my mental challenges.

`\@dedr@wm@rk` Because both the Lagrangian and Newtonian notational styles involve a physical marking being repeated, common macro was made that takes 3 arguments, the first will be for the initial placement, the second argument is fed into a follow-up macro `\@derepe@tdr@w`, and the third argument is what will be placed should the exponent count be higher than the max allowed.

Effectively `\@dedr@wm@rk` is what checks whether it should be a marking or the more symbolic generalized form.

```
\def\@dedr@wm@rk#1#2#3{
    \ifnum\exp@c@unt<\m@xm@rk
        #1\@derepe@tdr@w#2\else
        #3\fi}
```

`\@derepe@tdr@w` While `\@derepe@tdr@w` is what provides the conditional looping enviroment to ensure the markings are placed;

```
\def\@derepe@tdr@w#1{\loop\ifnum\exp@c@unt>\z@#1\repeat}
```

`\@derele@se` Shorthand for allowing the final formed ode or pde to rise to the surface

```
\def\@derele@se{\noexpand{\box\@deresb@x}}
```

Index

Numbers written in italic refer to the page where the corresponding entry is described; numbers underlined refer to the code line of the definition; numbers in roman refer to the code lines where the entry is used.

Symbols

\@@above 26
\@@atop 25
\@@over 24
\@de@bove 23
\@de@exp@n@rg . 54, 72, 73
\@de@func@Lagr 59
\@de@func@Leib 59
\@de@func@Newt 59
\@de@func@ther . . 59, 75
\@de@if@tpos . 60, 77, 103
\@de@ifbr@ck 54, 65
\@de@ifbrace 61, 74
\@de@ifexp@n 55, 65
\@de@ifst@r 38, 41, 44, 47, 50, 53, 65
\@de@ned@ts . . . 155, 163
\@de@nextact 90
\@de@not@Lagrange 14, 37, 46
\@de@not@Leibniz 14, 40, 49
\@de@not@Newton 14, 43, 52
\@de@not@default . . . 17
\@de@option . . 11, 34, 35
\@de@perat@r . 36, 39, 42, 45, 48, 51, 114, 126, 145, 148, 160
\@de@ption@l@rg 54, 69, 70
\@de@t@Lagr 130
\@de@t@Leib 134
\@de@t@Newt 141
\@de@tf@rm 64, 84
\@de@tilsp@ce . . . 76, 99
\@de@tmpA 86, 94
\@de@tmpB 86, 95
\@de@top 23
\@de@tpos@rg 64, 79
\@de@ttw@f@c 77
\@de@ver 23, 150
\@dec@mm@nd 37, 40, 43, 46, 49, 52, 83, 84, 85
\@dec@mpf@rm 57, 77, 80, 81
\@dedr@wm@rk 109, 121, 155, 174
\@deem@rex 114, 126, 148, 171, 173
\@defunb@x . . 8, 114, 125, 150, 154, 157
\@deifch@r 65, 68, 71, 74, 77, 86
\@deifst@rred . . . 56, 81
\@den@st@r 32, 67
\@den@st@ry@ink 57
\@denewt@nd@t . . 155, 170
\@denext@rg 89, 90
\@derele@se . . . 112, 123, 152, 159, 179
\@derepe@tdr@w . 176, 178
\@deresb@x . . 8, 108, 119, 149, 156, 179
\@desp@cegobbler 92, 97
\@dest@r@dy@ink 57
\@dest@r@rg . . 54, 66, 67
\@dest@red 32, 66
\@dest@rgument 66, 67, 81
\@det@stt@k 86, 91, 93, 98
\@detempv@l 2, 165, 166, 167, 168
\@detw@d@ts . . . 155, 163
\@dev@rb@x 8, 117, 129, 147, 150, 157, 158
\@dew@tcht@k 87, 93
\@dey@inkf@rm . . . 58, 84
\@ifpackageloaded . . 23
\@sptoken 91
\@tpost@ks 5, 64, 132, 138, 143

A

\advance 18, 161, 164, 167, 169

B

\b@se@Leib 115, 127, 146
\b@se@Newt 117, 129, 153
\beginnext 100, 104
\br@ced@xpon 109, 121, 161

C

\CurrentOption 19

D

\d@@ 30, 36, 39, 42
\d@l 30, 45, 48, 51, 145, 160
\displaystyle . . 137, 172

E

\empty 145, 160, 173
\endnext 103, 104
\exp@c@unt 2, 56, 161, 162, 163, 164, 165, 169, 173, 175, 178

F

\func@t@ks 5, 59, 102, 120, 126, 129

H

\hbox 108, 114, 117, 119, 125, 129, 131, 135, 142, 147, 149, 154, 156, 157, 158, 164, 169, 170, 171, 172
\hss 157, 164, 169, 171, 172

L

\lagr@prime 109, 121, 161
\LagrODE *4*, 36
\LagrPDE *4*, 45
\left 110, 132, 136, 143, 162
\LeibODE *4*, 36
\LeibPDE *4*, 45
\lower 150

M

\m@kep@rtLagr 110, 122, 145
\m@kep@rtNewt 160
\m@xm@rk 2, 18, 175
\mathrm 30
\maxprimes *3*

N

\n@st@r@Lagr . . . 118, 132
\n@st@r@Leib . . . 124, 136
\n@st@r@Newt . . . 128, 143
\newcommand 22
\NewtODE *4*, 36
\NewtPDE *4*, 45
\next 102, 105, 106
\notation *3*

O

\ode *4*, 34

\ode* *4*

P

\PackageWarning 19
\partial 31
\pde *4*, 34
\pde* *4*
\providecommand 11

R

\raise . 150, 157, 158, 171
\RequirePackage 1
\right 110, 132, 136, 143, 162
\rlap 138

S

\scriptstyle . . . 138, 171
\setDE *3*, 22
\setkeys 22
\st@r@d@Lagr 107
\st@r@d@Leib 113
\st@r@d@Newt 116

T

\thinmuskip 138
\thr@@ 132, 171
\toks 101, 102

U

\undefined 105

V

\v@rr@t@ks 5, 55, 110, 117, 137, 145, 148
\vbox 154, 156, 170

W

\wd 157