The odesandpdes package*

Anakin
anakin@ruc.dk

Released 2024/01/11

Abstract

Put text here.

Contents

*This document corresponds to odesandpdes v0.9.5, dated 2024/01/11.

My funny little ODE/PDE package

Start by first having `odesandpdes.sty` downloaded in an accessible directory, or in the same directory as your overleaf main.tex, using it by inserting;

```
\usepackage{odesandpdes}
```

into the preamble, Ideally after amsmath/mathtools. There are a couple notation options, which can be set document-wide with;

```
\usepackage[notation=<option>]{odesandpdes}|
```

1 Usage

The options included are based off of the three most common notations (according to Wikipedia), Lagrange, Leibniz, and Newton. However, if you wish to change it on a section to section basis, the command `\setDE{`*notation=⟨option⟩*`}` will change the form of the subsequent uses.

`\ode` `\ode*` The command(s) are approached with the philosophy of of an intuitive and modular usage. The general form can be understood as;

```
\ode[<variable>]<exponent>{<function>}
\ode*[<variable>]<exponent>
```

`\pde` `\pde*` The general form can be understood as;

```
\pde[<variable>]<exponent>{<function>}
\pde*[<variable>]<exponent>
```

While this should be sufficient for most use, there are a couple of tricks incorporated into the mechanism that makes this package better than the generic `\newcommand*\ode[2][t]{⟨...⟩}` Including automatic assignment of degree and starred variants. Example;

```
\begin{align*}
   \ode N(t) = N(t) * t && \ode N(t,x) = N(t,x) + \ode[x]^2 N(t,x) && \pde^2 f^2
\end{align*}
```

$$\frac{\mathrm{d}N(t)}{\mathrm{d}t} = N(t) * t \qquad \frac{\mathrm{d}N(t,x)}{\mathrm{d}t} = N(t,x) + \frac{\mathrm{d}^2N(t,x)N(t,x)}{\mathrm{d}x^2} \qquad \frac{\partial^2 f^2}{\partial t^2}$$

The `\ode` command will scan for an exponent between `[`⟨*variable*⟩`]` and `{`⟨*function*⟩`}`. Should there is indeed an exponent, that exponent gets 'yoinked' away and processed in accordance to the style of notation.

`\setDE` In the case of Lagrange or Netwon notation, there is the `\setDE{`⟨*maxprimes=integer*⟩`}` option for either the package or the `\setDE{`⟨*option*⟩`}` command;

`\usepackage[maxprimes=<integer>]{odesandpdes}` *and/or*
`\setDE{maxprimes=<integer>}`

3 being the default.

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^2{f(x)}` as opposed to `\ode{f(x)}^2`. As well, with the "proper" spacing, there is little need for the use of the braces, so as to help promote an easier workflow without always needed to worry about the damn brace. Not that one can not use the brace for personal taste. For the sake of parity, the `\pde` command will also take its variable in brackets.

2 Examples of use

The following examples all take identical form, shown in the following;

```
\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}`

$$\begin{aligned}
a'(x) && b(x)' && c'(x) && d(x)^{(5)} \\
f'(t)e(x) && f'(x)f(x) && f''(t)g(x) && f^{(6)}(x)h(x) \\
i'_t(x) && j(x)'_x && k'''_t(x) && l(x)^{(7)}_x \\
f'_t(t)m(x) && f'_x(x)n(x) && f^{(4)}_t(t)o(x) && f^{(8)}_x(x)p(x)
\end{aligned}$$

`\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}`

$$\begin{aligned}
\dot{a}(x) && \dot{b}(x) && \dot{c}(x) && \overset{5}{\dot{d}}(x) \\
\dot{t}\,e(x) && \dot{x}f(x) && \ddot{t}\,g(x) && \overset{6}{\dot{x}}h(x) \\
\dot{i}\,(x) && \dot{j}(x) && \dddot{k}(x) && \overset{7}{\dot{l}}(x) \\
\dot{t}\,m(x) && \dot{x}n(x) && \overset{4}{\dot{t}}\,o(x) && \overset{8}{\dot{x}}p(x)
\end{aligned}$$

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

$$\begin{aligned}
&f' && f'' && f''' && f'''' && f''''' && f'''''' && f''''''' && f^{(8)} && f^{(9)} \\
&\dot{f} && \ddot{f} && \dddot{f} && \ddddot{f} && \overset{\cdot\cdot}{\cdot\cdot}f && \overset{\cdot\cdot\cdot}{\cdot\cdot\cdot}f && \overset{\cdot\cdot\cdot\cdot}{\cdot\cdot\cdot\cdot}f && \overset{8}{\dot{f}} && \overset{9}{\dot{f}}
\end{aligned}$$

2.2 Examples of "at x;"

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` ⟨*value*⟩`;`',[1] and the representation will shown according to notational convention.

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

`\setDE{notation=Lagrange}`	`\setDE{notation=Leibniz}`	`\setDE{notation=Newton}`
$c'(23\pi) = i$	$\left.\frac{\mathrm{d}c}{\mathrm{d}x}\right\|_{x=23\pi} = i$	$\dot{c}(23\pi) = i$
$c'''(69) = i$	$\left.\frac{\mathrm{d}^3 c}{\mathrm{d}x^3}\right\|_{x=69} = i$	$\dddot{c}(69) = i$
$c^{(69)}(L) + t = i$	$\left.\frac{\mathrm{d}^{69} c}{\mathrm{d}x^{69}}\right\|_{x=L} + t = i$	$\overset{69}{\dot{c}}(L) + t = i$
$c^{(9)} af 420; = i$	$\frac{\mathrm{d}^9 c}{\mathrm{d}x^9} af 420; = i$	$\overset{9}{\dot{c}} af 420; = i$
$c^{(6)} 13 = i$	$\frac{\mathrm{d}^6 c}{\mathrm{d}x^6} 13 = i$	$\overset{6}{\dot{c}} 13 = i$

Important to note, due to how the notational styles are defined, only the Leibniz notation can take arguments for the function that involve subscripts and superscripts without delimiters.

```
\begin{align*}
    \ode f_1           &= i \\
    \ode f_1 at L;     &= i \\
    \ode f   at L;     &= i \\
    \ode {(f_1)}       &= i \\
    \ode {(f_1)} at L; &= i
\end{align*}
```

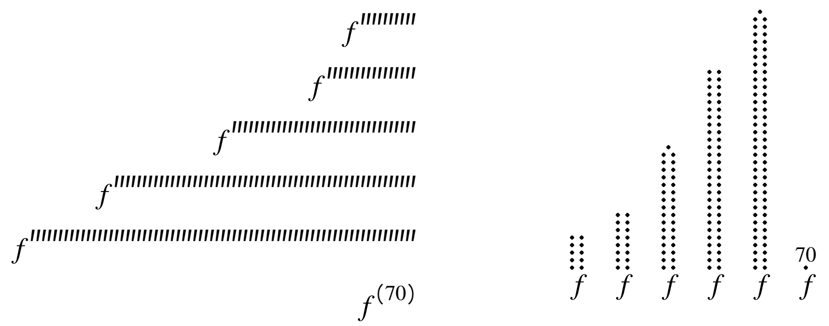
[1] heavily inspired by Tikz

`\setDE{notation=Lagrange}`	`\setDE{notation=Leibniz}`	`\setDE{notation=Newton}`
$f'_1 = i$	$\frac{\mathrm{d}f_1}{\mathrm{d}t} = i$	$\dot{f}_1 = i$
$f'_1 atL; = i$	$\left.\frac{\mathrm{d}f_1}{\mathrm{d}t}\right\vert_{t=L} = i$	$\dot{f}_1 atL; = i$
$f'(L) = i$	$\left.\frac{\mathrm{d}f}{\mathrm{d}t}\right\vert_{t=L} = i$	$\dot{f}(L) = i$
$(f_1)' = i$	$\frac{\mathrm{d}(f_1)}{\mathrm{d}t} = i$	$\dot{(f_1)} = i$
$(f_1)'(L) = i$	$\left.\frac{\mathrm{d}(f_1)}{\mathrm{d}t}\right\vert_{t=L} = i$	$\dot{(f_1)}(L) = i$

2.3 Prime Count Limit

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

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@rkings` `\expo@de` `\@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@rkings%
\newcount\expo@de%
\countdef\@detempv@l=255%
```

`\l@wert@ks` `\upp@rt@ks` `\@tpost@ks` As well reserving token registries for tossing arguments around the groups and macros,

```
\newtoks\l@wert@ks%
\newtoks\upp@rt@ks%
\newtoks\@tpost@ks%
```

`\@delowb@x` `\@deuppb@x` `\@deresb@x` Reserving box registries for the purpose of collecting the components together in a coherent manner,

```
\newbox\@delowb@x%
\newbox\@deuppb@x%
\newbox\@deresb@x%
```

3.1.1 Package Options

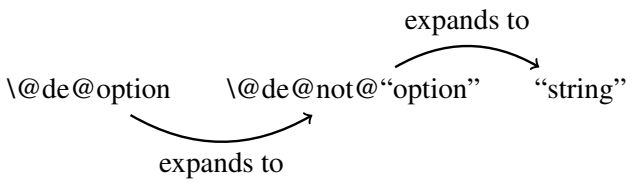
`\@de@option` Defining the package options for notational styles using the LaTeX `\providecommand` to reloading times. Important to note that defining the command is not the same as using the command, which is useful in conjunction with `\csname` and `\endcsname` for macro defintions.

```
\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`"




```
\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@rkings=#1\advance\m@rkings1}%
```

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 excecuted as to allow the default options to initialize and be processed,

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

3.2 Package Configuration

Macro for notation style option to be used by `\setDE` Macro for number of primes to be used by `\setDE`

```
\define@key[package]{@de}{notation}%
    {\def\@de@option{\csname @de@not@#1\endcsname}}
\define@key[package]{@de}{maxprimes}%
    {\m@rkings=#1\advance\m@rkings1}
```

`\setDE` Macro for switching of the style mid-document

```
\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 primitives for ease of function,

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

3.3 Foundational macros

`\d@@` `\d@l` Protected def for the d Protected def for the del

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

\@dem@rkst@red
\@dem@rkn@st@r

These just kind of exist for the moment

```
\def\@dem@rkst@red{st@r@d}
\def\@dem@rkn@st@r{n@st@r}
```

\ode
\pde

The macro definitions of the ODE and PDE commands

```
\def\ode{\let\@de@perat@r=\d@@% sets the d
    \@de@ifst@r}
\def\pde{\let\@de@perat@r=\d@l% sets the del
    \@de@ifst@r}
```

In essence these two are the same command. This is done for the sake of consistancy in use and effect. The only difference comes down to how the operator is defined

3.3.1 Macro Checkpoints

\@de@ifst@r
\@de@ifbr@ck
\@de@ifexpon

Now a group of 'what's next' functions are needed. Each of the major elements, star (*), option ([), and exponent (^) is given a macro. These macros make use of an ancilliariy function \@deifnextch@r, which is defined in the section 3.4.2.

```
\def\@de@ifst@r{\@deifnextch@r * \@dest@red@rg \@den@st@r@rg}
\def\@de@ifbr@ck{\@deifnextch@r [ \@de@ption@l@rg {\@de@ption@l@rg[t]}}
\def\@de@ifexpon{\@deifnextch@r ^ \@de@exponent@rg {\@de@exponent@rg^1}}
\def\@de@ifindex{\@deifnextch@r _ \@de@index@rg \@de@noindex@rg}
```

\@deif@tpos
\@de@tfinder

Macro for authentification of the 'at '

```
\def\@deif@tpos{\@deifnextch@r a \@de@tfinder \@definalchosenform}
\def\@de@tfinder a#1{\ifx#1t\expandafter\@de@tom@at\else
    \@definalchosenform a#1\fi}%
```

3.3.2 The 'Yoinkers'

\@dest@red@rg
\@den@st@r@rg
\@de@ption@l@rg
\@de@exponent@rg
\@de@index@rg
\@dest@r@dy@ink
\@den@st@ry@ink

Macros for starred and unstarred varients

```
\def\@dest@red@rg*{\@de@isst@r{st@r@d} \@de@ifbr@ck}
\def\@den@st@r@rg {\@de@isst@r{n@st@r} \@de@ifbr@ck}
```

Macro for optional and no optional args

```
\def\@de@ption@l@rg[#1]{\expandafter\l@wert@ks{#1}\relax \@de@ifexpon}%
```

Macro for yoinking the exponent

```
\def\@de@exponent@rg^#1{\expo@de#1\relax \@deisitorisntitastar}
```

Macro for yoinking the index

```
\def\@de@index@rg_#1{#1\relax \@deisitorisntitastar}
\def\@de@noindex@rg{\relax \@deisitorisntitastar}
```

Yoinks the function variable

```
\def\@dest@r@dy@ink{\expandafter\@definalchosenform}
\def\@den@st@ry@ink{\expandafter\@de@y@inkf@rm}
```

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

Possibly make a function to keep eating tokens till a space is found? Could work on account of the \upp@rt@ks

```
\def\@de@func@Leib{\expandafter\@defuncg@bbler}
\def\@de@func@ther#1{\expandafter\upp@rt@ks{#1}\relax \@deif@tpos}
\let\@de@func@Lagr\@de@func@ther
\let\@de@func@Newt\@de@func@ther

\def\@defuncg@bbler#1{\beginnext%
    \toks0={\the\upp@rt@ks#1}
    \edef\next{\upp@rt@ks=\expandafter{\the\toks0}}
    \endnext\@deifnotsp@ce}
```

\@deifnotsp@ce
\@deleibsp@cemucher

In order for the Leibniz notation to be able to differentiate the way it applies to functions, it needs to be able to process a stream of tokens one by one until a space token is found.

```
\def\@deifnotsp@ce{%
    \futurelet\@detesttoken\@deleibsp@cemucher}
\def\@deleibsp@cemucher{%
    \ifx\@detesttoken\@sptoken%
        \expandafter\@deif@tpos\else%
        \expandafter\@defuncg@bbler\fi}
```

3.4 Ancilliary Functions

toksloadcsexpansion
\tokscat

```
\def\toksloadcsexpansion#1\to#2{%
    #2=\expandafter{#1}}
\def\tokscat#1&#2\to#3{%
    \beginnext
    \edef\tokscat@a{\the#1\the#2}%
    \toks2={#3}%
    \toksloadcsexpansion\tokscat@a
    \to{\toks4}%
    \edef\next{\the\toks2={\the\toks4}}%
    \endnext}
```

\beginnext
\endnext
\next

If one wants to take advatage of the \edef primitive, it becomes worthwhile to be able to make an explicit group to allow all sorts of of register-based games and finally use \edef to compute an appropriate replacement text for \next

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

3.4.1 Variable Macronames

\@de@isst@r
deisitorisntitastar
\@definalchosenform

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. \@de@isst@r takes an argument and defines two macros \@deisitorisntitastar which defines whether the function 'yoinker' exists or not, and \@definalchosenform

which works with \@de@option,defined in subsection 3.2, to define the final ODE or PDE form.

```
\def\@de@isst@r#1{%
    \def\@deisitorisntitastar{\csname @de#1y@ink\endcsname}%
    \def\@definalchosenform{\csname#1@\@de@option\endcsname}}
```

\@de@t@posf@rm
\@de@y@inkf@rm

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

```
\def\@de@t@posf@rm{\csname @de@tpl@ce@\@de@option\endcsname}%
\def\@de@y@inkf@rm{\csname @de@func@\@de@option\endcsname}%
```

\@de@tom@at

Used for choosing which notational form to take

```
\def\@de@tom@at#1;{\expandafter\@tpost@ks{#1}\relax \@de@t@posf@rm}
```

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 \@deifnextch@r macro.

\@deifnextch@r
\@detesttoken
\@de@tmpA
\@de@tmpB

\@deifnextch@r takes in three tokens as arguments, the first token will define what the macro should look out for, while the other two arguments are for storage to be excecuted later. \@deifnextch@r composed of two main elements, the namesake \@deifnextch@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.

The most important consequence means 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.[2]

```
\def\@deifnextch@r#1#2#3{
    \let\@dedesiredtoken=#1\relax
    \def\@de@tmpA{#2}%
    \def\@de@tmpB{#3}
    \futurelet\@detesttoken\@denext@rg}
```

Using this *enlightenment*, define the token representing an 'if-then-else' control sequence \@denext@rg. In \@deifnextch@r, \@dedesiredtoken becomes a macro for the token we want to check against. Using this to our advantage, before TEX expands

[2] If that means nothing to you, rest easy knowing that you still have the chance to escape learning TEX before its too late

\@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 \@dedesiredtoken against eachother to determine which outcome should be excecuted.

\@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 which can be pointed at. Therefor, we use a space gobbling token \@degobblespace, which will be defined after \@denextarg.

```
\def\@denext@rg{%
    \ifx\@detesttoken\@sptoken%
        \let\@de@nextact\@degobblesp@ce\else
```

The second half of \@denext@rg is what does the actual comparison. Should the comparison be positive, \@detesttoken = \@dedesiredtoken, then the code stored in \@de@tmpA will be excecuted, otherwise, \@de@tmpB excecutes.

```
        \ifx\@detesttoken\@dedesiredtoken%   if
            \let\@de@nextact\@de@tmpA\else%  ifn't
            \let\@de@nextact\@de@tmpB\fi\fi
    \@de@nextact}
```

\@degobblesp@ce

Ensuring that the space(s), explicit or implicit, trailing after \@deifnextch@r requires some TEX *tomfoolary*.

```
\let\@desavedef\<
```

By defining the function with a non-character token, the space matters

```
\def\<{\@degobblesp@ce}
\expandafter\def\< {\futurelet\@detesttoken\@denext@rg}
```

3.5 Notational Morphology

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

```
\def\st@r@d@Lagr{%
    \setbox\@deresb@x\hbox{$
        f^{\mkern1mu\m@kem@rk\lagr@prime\lagr@prime\br@ced@xpon}
        _{\m@kep@rtLagr}
        \mkern-0.5mu\left(\the\l@wert@ks\right)
        $}%
    \@derele@se}%
```

\n@st@r@Lagr Macro for Lagr

```
\def\n@st@r@Lagr{%
    \setbox\@deresb@x\hbox{$
        \the\upp@rt@ks
        ^{\mkern1mu\m@kem@rk\lagr@prime\lagr@prime\br@ced@xpon}
        _{\m@kep@rtLagr}
        $}%
    \@derele@se}%
```

\@de@tpl@ce@Lagr Macro for Lagr at point

```
\def\@de@tpl@ce@Lagr{%
    \noexpand\hbox{$
        \n@st@r@Lagr\mkern-03mu\left(\the\@tpost@ks\right)
        $}}%
```

\lagr@prime Macro for the prime used by the lagrangian notation

\br@ced@xpon

\m@kep@rtLagr

```
\def\lagr@prime{\mkern0.35mu\prime\global\advance\expo@de-1}
```

Macro for making the exponent in parenthesis

```
\def\br@ced@xpon{\left(\the\expo@de\right)}
```

Macro for Lagrange partial notations

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

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

```
\def\st@r@d@Leib{%
    \setbox\@deuppb@x\hbox{$\@de@perat@r^{\empty@rexpon}$}%
    \b@se@Leib}%
```

\n@st@r@Leib Macro for Leib

```
\def\n@st@r@Leib{%
    \setbox\@deuppb@x\hbox{$
        \mkern0.40mu\@de@perat@r^{\empty@rexpon}\the\upp@rt@ks$}%
    \b@se@Leib}%
```

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

```
\def\b@se@Leib{%
    \setbox\@delowb@x\hbox{$
        \@de@perat@r\mkern0.40mu\the\l@wert@ks^{\empty@rexpon}$}%
    \setbox\@deresb@x\hbox{\kern0.50\p@%
        $\raise2\p@\box\@deuppb@x\@de@ver\lower5\p@\box\@delowb@x$%
        \kern0.50\p@}%
    \@derele@se}%
```

\@de@tpl@ce@Leib Macro for specification of where the ode is defined

```
\def\@de@tpl@ce@Leib{%
    \noexpand\hbox{$
        \left.\n@st@r@Leib\mkern0mu\right|
        _{\mkern1mu\displaystyle\the\l@wert@ks\mkern2mu
        \rlap{$\scriptstyle=\mkern2mu\the\@tpost@ks$}}
        $}%
    }%
```

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

```
\def\st@r@d@Newt{%
    \setbox\@delowb@x\hbox{$\the\l@wert@ks$}%
    \b@se@Newt}%
```

\n@st@r@Newt Macro for Newt

```
\def\n@st@r@Newt{%
    \setbox\@delowb@x\hbox{$\displaystyle\the\upp@rt@ks$}%
    \b@se@Newt}%
```

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

```
\def\b@se@Newt{%
    \setbox\@deuppb@x\hbox{\vbox{\baselineskip=\z@\lineskip=-1\p@%
        \m@kem@rk\@ned@ts\tw@d@ts\newt@nd@t}}%
    \setbox\@deresb@x\hbox{\vbox{\baselineskip=\z@\lineskip=-0.5\p@%
        \hbox{\raise0.00ex\box\@deuppb@x}%
        \hbox{\raise0.00ex\box\@delowb@x}}}%
    \@derele@se}
```

\@de@tpl@ce@Newt Macro for Newton at point

```
\def\@de@tpl@ce@Newt{%
    \noexpand\hbox{$%
        \n@st@r@Newt\mkern-02mu\left(\the\@tpost@ks\right)%
        $}}%
```

\newt@nd@t Macro for numbering

```
\def\newt@nd@t{\hbox{\vbox{%
    \hbox to 5\p@{\hss\raise0.50ex\hbox{$\scriptstyle\empty@rexpon$}\hss}%
    \hbox to 5\p@{\hss\hbox{$\displaystyle\cdot$}\hss}}}}%
```

\m@kep@rtNewt Macro for Netwon partial notations

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

\tw@d@ts Macro for dots Tests as "mod2" testing of dot groupings
\@ned@ts

```
\def\tw@d@ts{\ifnum\expo@de>1%
        \advance\expo@de-2\hbox to 5\p@{\hss$\cdot\cdot$\hss}\fi}%
\def\@ned@ts{\@detempv@l=\the\expo@de%
    \loop\ifnum\@detempv@l>2%
        \advance\@detempv@l-2\repeat%
    \ifnum\@detempv@l<2%
        \advance\expo@de-1\hbox to 5\p@{\hss$\cdot$\hss}\fi}%
```

3.6 Notational Shaping Tools

\empty@rexpon Macro for determining if the exponent should be empty

```
\def\empty@rexpon{\ifnum2>\expo@de\empty\else\the\expo@de\fi}
```

\m@kem@rk Macro for checking if marks should be used or something else If not zero, check if less than max allowed Make primes while below limit

```
\def\m@kem@rk#1#2#3{%
    \ifnum\expo@de<\m@rkings
        #1\m@rkrepe@ting#2\else
        #3\fi}
```

`\m@rkrepe@ting` Macro for creating the appropriate number of marks, primes or whatever

```
\def\m@rkrepe@ting#1{\loop\ifnum\expo@de>0#1\repeat}
```

`\@derele@se` Shorthand for allowing the boxes to rise to the serface

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

\let\<\@desavedef
\endinput
```