The `physica` package

Leedehai
[GitHub](#) | [Typst](#)

physica *noun*. Latin, study of nature.

Contents

1. Introduction

Typst is typesetting framework aiming to become the next generation alternative to LATEX. It excels in its friendly user experience and performance.

The `physica` package provides handy Typst typesetting functions that make academic writing for natural sciences simpler and faster, by simplifying otherwise very complex and repetitive expressions in the domain of natural sciences.

This manual itself was generated using the Typst CLI and the `physica` package, so hopefully this document is able to provide you with a sufficiently self evident demonstration of how this package shall be used.

2. Using `physica`

With `typst`'s package management:

```
#import "@preview/physica:0.9.0": *
```

```
$ curl (grad f), pdv(,x,y,z,[2,k]), tensor(Gamma,+k,-i,-j) = pdv(vb(e_i),x^j)vb(e^k) $
```

$$\nabla \times (\nabla f), \frac{\partial^{k+3}}{\partial x^2 \partial y^k \partial z}, {\Gamma^k}_{ij} = \frac{\partial \boldsymbol{e_i}}{\partial x^j} \boldsymbol{e^k}$$

3. The symbols

Some symbols are already provided as a Typst built-in. They are listed here just for completeness with annotation like typst `this`, as users coming from LATEX might not know they are already available in Typst out of box.

All symbols need to be used in **math mode** `$...$`.

3.1. Braces

Symbol	Abbr.	Example	Notes
typst `abs(`*content*`)`		`abs(phi(x))` $\longrightarrow \lvert\varphi(x)\rvert$	absolute
typst `norm(`*content*`)`		`norm(phi(x))` $\longrightarrow \lVert\varphi(x)\rVert$	norm
`order(`*content*`)`		`order(x^2)` $\longrightarrow \mathcal{O}(x^2)$	order of magnitude
`Set(`*content*`)`		`Set(a_n), Set(a_i, forall i)` $\longrightarrow \{a_n\}, \left\{a_i \middle\vert \forall i\right\}$ `Set(vec(1,n), forall n)` $\longrightarrow \left\{\begin{pmatrix}1\\n\end{pmatrix} \middle\vert \forall n\right\}$	math set, use `Set` not `set` since the latter is a Typst keyword
`evaluated(`*content*`)`	`eval`	`eval(f(x))_0^infinity` $\longrightarrow f(x)\Big\vert_0^\infty$ `eval(f(x)/g(x))_0^1` $\longrightarrow \left.\frac{f(x)}{g(x)}\right\vert_0^1$	attach a vertical bar on the right to denote evaluation boundaries
`expectationvalue`	`expval`	`expval(u)` $\longrightarrow \langle u\rangle$ `expval(f/N)` $\longrightarrow \left\langle\frac{f}{N}\right\rangle$	expectation value

3.2. Vector notations

Symbol	Abbr.	Example	Notes
typst `vec(...)`		`vec(1,2)` $\longrightarrow \begin{pmatrix}1\\2\end{pmatrix}$	column vector
`vecrow(...)`		`vecrow(1,2)` $\longrightarrow (1, 2)$ `vecrow(sum_0^n a_i, b)` $\longrightarrow \left(\sum_0^n a_i, b\right)$	row vector
`TT`		`v^TT, A^TT` $\longrightarrow v^{\mathsf{T}}, A^{\mathsf{T}}$	transpose, also see Section 3.7.2
`vectorbold(`*content*`)`	`vb`	`vb(a),vb(mu_1)` $\longrightarrow \boldsymbol{a}, \boldsymbol{\mu}_1$	vector, bold
`vectorunit(`*content*`)`	`vu`	`vu(a),vu(mu_1)` $\longrightarrow \hat{\boldsymbol{a}}, \hat{\boldsymbol{\mu}}_1$	unit vector
`vectorarrow(`*content*`)`	`va`	`va(a),va(mu_1)` $\longrightarrow \vec{a}, \vec{\mu}_1$	vector, arrow (not bold: see ISO 80000-2:2019)
`gradient`	`grad`	`grad f` $\longrightarrow \boldsymbol{\nabla} f$	gradient
`divergence`	`div`	`div vb(E)` $\longrightarrow \boldsymbol{\nabla} \cdot \boldsymbol{E}$	divergence
`curl`		`curl vb(B)` $\longrightarrow \boldsymbol{\nabla} \times \boldsymbol{B}$	curl
`laplacian`		`diaer(u) = c^2 laplacian u` $\longrightarrow \ddot{u} = c^2 \nabla^2 u$	Laplacian, different from typst `laplace` Δ
`dotproduct`	`dprod`	`a dprod b` $\longrightarrow a \cdot b$	dot product
`crossproduct`	`cprod`	`a cprod b` $\longrightarrow a \times b$	cross product
`innerproduct`	`iprod`	`iprod(u, v)` $\longrightarrow \langle u, v \rangle$ `iprod(sum_i a_i, b)` $\longrightarrow \left\langle \sum_i a_i, b \right\rangle$	inner product

3.3. Matrix notations

Symbol	Abbr.	Example	Notes
`TT`		`v^TT, A^TT` $\longrightarrow v^{\mathsf{T}}, A^{\mathsf{T}}$	transpose, also see Section 3.7.2
typst `mat(...)`		`mat(1,2;3,4)` $\longrightarrow \begin{pmatrix}1 & 2\\3 & 4\end{pmatrix}$	matrix
`matrixdet(...)`	`mdet`	`mdet(1,x;1,y)` $\longrightarrow \begin{vmatrix}1 & x\\1 & y\end{vmatrix}$	matrix determinant
`diagonalmatrix(...)`	`dmat`	`dmat(1,2)` $\longrightarrow \begin{pmatrix}1 & \\ & 2\end{pmatrix}$ `dmat(1,a,xi,delim:"[",fill:0)` $\longrightarrow \begin{bmatrix}1 & 0 & 0\\0 & a & 0\\0 & 0 & \xi\end{bmatrix}$	diagonal matrix
`antidiagonalmatrix(...)`	`admat`	`admat(1,2)` $\longrightarrow \begin{pmatrix} & 1\\2 & \end{pmatrix}$ `admat(1,a,xi,delim:"[",fill:dot)` $\longrightarrow \begin{bmatrix}\cdot & \cdot & 1\\\cdot & a & \cdot\\\xi & \cdot & \cdot\end{bmatrix}$	anti-diagonal matrix
`identitymatrix(...)`	`imat`	`imat(2)` $\longrightarrow \begin{pmatrix}1 & \\ & 1\end{pmatrix}$ `imat(3,delim:"[",fill:*)`	identity matrix

		$\longrightarrow$ $\begin{bmatrix}1 & * & * \\ * & 1 & * \\ * & * & 1\end{bmatrix}$	
`zeromatrix(...)`	`zmat`	`zmat(2)` $\longrightarrow$ $\begin{pmatrix}0 & 0 \\ 0 & 0\end{pmatrix}$ `zmat(3,delim:"[")` $\longrightarrow$ $\begin{bmatrix}0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0\end{bmatrix}$	zero matrix
`jacobianmatrix(...)`	`jmat`	See below	Jacobian matrix
`hessianmatrix(...)`	`hmat`	See below	Hessian matrix
`xmatrix(...)`	`xmat`	See below	Matrix built with an element building function

Jacobian matrix: `jacobianmatrix(...)`, i.e. `jmat(...)`.

	`jmat(f_1,f_2; x,y)`	`jmat(f,g; x,y,z; delim:"[")`
Typst (like LaTeX) cramps fractions in a matrix...	$\begin{pmatrix}\frac{\partial f_1}{\partial x} & \frac{\partial f_1}{\partial y} \\ \frac{\partial f_2}{\partial x} & \frac{\partial f_2}{\partial y}\end{pmatrix}$	$\begin{bmatrix}\frac{\partial f}{\partial x} & \frac{\partial f}{\partial y} & \frac{\partial f}{\partial z} \\ \frac{\partial g}{\partial x} & \frac{\partial g}{\partial y} & \frac{\partial g}{\partial z}\end{bmatrix}$
	`jmat(f_1,f_2;x,y;big:#true)`	`jmat(f,g;x,y,z;delim:"|",big:#true)`
...but you can uncramp them using argument `big:#true` here	$\begin{pmatrix}\dfrac{\partial f_1}{\partial x} & \dfrac{\partial f_1}{\partial y} \\ \dfrac{\partial f_2}{\partial x} & \dfrac{\partial f_2}{\partial y}\end{pmatrix}$	$\begin{vmatrix}\dfrac{\partial f}{\partial x} & \dfrac{\partial f}{\partial y} & \dfrac{\partial f}{\partial z} \\ \dfrac{\partial g}{\partial x} & \dfrac{\partial g}{\partial y} & \dfrac{\partial g}{\partial z}\end{vmatrix}$

Hessian matrix: `hessianmatrix(...)`, i.e. `hmat(...)`.

	`hmat(f; x,y)`	`hmat(; x,y,z; delim:"[")`
Typst (like LaTeX) cramps fractions in a matrix...	$\begin{pmatrix}\frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} \\ \frac{\partial^2 f}{\partial y \partial x} & \frac{\partial^2 f}{\partial y^2}\end{pmatrix}$	$\begin{bmatrix}\frac{\partial^2}{\partial x^2} & \frac{\partial^2}{\partial x \partial y} & \frac{\partial^2}{\partial x \partial z} \\ \frac{\partial^2}{\partial y \partial x} & \frac{\partial^2}{\partial y^2} & \frac{\partial^2}{\partial y \partial z} \\ \frac{\partial^2}{\partial z \partial x} & \frac{\partial^2}{\partial z \partial y} & \frac{\partial^2}{\partial z^2}\end{bmatrix}$
	`hmat(f;x,y;big:#true)`	`hmat(;x,y,z;delim:"|",big:#true)`
...but you can uncramp them using argument `big:#true` here	$\begin{pmatrix}\dfrac{\partial^2 f}{\partial x^2} & \dfrac{\partial^2 f}{\partial x \partial y} \\ \dfrac{\partial^2 f}{\partial y \partial x} & \dfrac{\partial^2 f}{\partial y^2}\end{pmatrix}$	$\begin{vmatrix}\dfrac{\partial^2}{\partial x^2} & \dfrac{\partial^2}{\partial x \partial y} & \dfrac{\partial^2}{\partial x \partial z} \\ \dfrac{\partial^2}{\partial y \partial x} & \dfrac{\partial^2}{\partial y^2} & \dfrac{\partial^2}{\partial y \partial z} \\ \dfrac{\partial^2}{\partial z \partial x} & \dfrac{\partial^2}{\partial z \partial y} & \dfrac{\partial^2}{\partial z^2}\end{vmatrix}$

Matrix built with an element building function: `xmatrix(`*m, n, func*`)`, i.e. `xmat(...)`. The element building function *func* takes two integers which are the row and column numbers starting from 1.

```
#let g = (i,j) => $g^(#(i - 1)#(j - 1))$
xmat(2, 2, #g)
```

$$\begin{pmatrix} g^{00} & g^{01} \\ g^{10} & g^{11} \end{pmatrix}$$

```
xmat(2, 3, #(r, c) => {
  $"exp"(#r, #c) = #calc.pow(r, c)$
}, delim:"[")
```

$$\begin{bmatrix} \exp(1,1) = 1 & \exp(1,2) = 1 & \exp(1,3) = 1 \\ \exp(2,1) = 2 & \exp(2,2) = 4 & \exp(2,3) = 8 \end{bmatrix}$$

3.4. Dirac braket notations

Symbol	Abbr.	Example	Notes
`bra(`*content*`)`		`bra(u)` $\longrightarrow \langle u\vert$ `bra(vec(1,2))` $\longrightarrow \left\langle \begin{pmatrix}1\\2\end{pmatrix}\right\vert$	bra
`ket(`*content*`)`		`ket(u)` $\longrightarrow \vert u\rangle$ `ket(vec(1,2))` $\longrightarrow \left\vert \begin{pmatrix}1\\2\end{pmatrix}\right\rangle$	ket
`expval(`*content*`)`		`expval(u)` $\longrightarrow \langle u\rangle$ `expval(vec(1,2))` $\longrightarrow \left\langle \begin{pmatrix}1\\2\end{pmatrix}\right\rangle$	expectation
`braket(`a, b`)`		`braket(a), braket(u, v)` $\longrightarrow \langle a\vert a\rangle, \langle u\vert v\rangle$ `braket(vec(1,2), b)` $\longrightarrow \left\langle \begin{pmatrix}1\\2\end{pmatrix}\middle\vert b\right\rangle$	braket
`ketbra(`a, b`)`		`ketbra(a), ketbra(u, v)` $\longrightarrow \vert a\rangle\langle a\vert, \vert u\rangle\langle v\vert$ `ketbra(vec(1,2), b)` $\longrightarrow \left\vert \begin{pmatrix}1\\2\end{pmatrix}\right\rangle\left\langle b\right\vert$	ketbra
`matrixelement(`n, M, m`)`	`mel`	`mel(n, diff_nu H, m)` $\longrightarrow \langle n\vert \partial_\nu H\vert m\rangle$ `mel(vec(U,V),A,m)` $\longrightarrow \left\langle \begin{pmatrix}U\\V\end{pmatrix}\middle\vert A\middle\vert m\right\rangle$	matrix element

3.5. Math functions

Typst built-in math operators: `math.op`.

Expressions	Results
`sin(x), sinh(x), arcsin(x), asin(x)`	$\sin(x), \sinh(x), \arcsin(x), \operatorname{asin}(x)$
`cos(x), cosh(x), arccos(x), acos(x)`	$\cos(x), \cosh(x), \arccos(x), \operatorname{acos}(x)$
`tan(x), tanh(x), arctan(x), atan(x)`	$\tan(x), \tanh(x), \arctan(x), \operatorname{atan}(x)$
`sec(x), sech(x), arcsec(x), asec(x)`	$\sec(x), \operatorname{sech}(x), \operatorname{arcsec}(x), \operatorname{asec}(x)$
`csc(x), csch(x), arccsc(x), acsc(x)`	$\csc(x), \operatorname{csch}(x), \operatorname{arccsc}(x), \operatorname{acsc}(x)$
`cot(x), coth(x), arccot(x), acot(x)`	$\cot(x), \coth(x), \operatorname{arccot}(x), \operatorname{acot}(x)$

Expressions	Results	Notes
typst `Pr(x)`	$\Pr(x)$	probability
typst `exp x`	$\exp x$	exponential
typst `log x, lg x, ln x`	$\log x, \lg x, \ln x$	logarithmic
typst `det A`	$\det A$	matrix determinant

`diag(-1,1,1,1)`	$\operatorname{diag}(-1,1,1,1)$	diagonal matrix, compact form (use `dmat` for the "real" matrix form)
`trace A, tr A`	$\operatorname{trace} A, \operatorname{tr} A$	matrix trace
`Trace A, Tr A`	$\operatorname{Trace} A, \operatorname{Tr} A$	matrix trace, alt.
`rank A`	$\operatorname{rank} A$	matrix rank
`erf(x)`	$\operatorname{erf}(x)$	Gauss error function
`Res A`	$\operatorname{Res} A$	residue (complex analysis)
`Re z, Im z`	$\operatorname{Re} z, \operatorname{Im} z$	real, imaginary (complex analysis)
`sgn x`	$\operatorname{sgn} x$	sign function

3.6. Differentials and derivatives

Symbol	Abbr.	Example	Notes
`differential(...)`	`dd`	e.g. $\mathrm{d}f, \mathrm{d}x\mathrm{d}y, \mathrm{d}^3x, \mathrm{d}x \wedge \mathrm{d}y$ See Section 3.6.1	differential
`variation(...)`	`var`	`var(f)` $\longrightarrow \delta f$ `var(x,y)` $\longrightarrow \delta x \delta y$	variation, shorthand of `dd(..., d: delta)`
`difference(...)`		`difference(f)` $\longrightarrow \Delta f$ `difference(x,y)` $\longrightarrow \Delta x \Delta y$	difference, shorthand of `dd(..., d: Delta)`
`derivative(...)`	`dv`	e.g. $\frac{\mathrm{d}}{\mathrm{d}x}, \frac{\mathrm{d}f}{\mathrm{d}x}, \frac{\Delta^k f}{\Delta x^k}, \mathrm{d}f/\mathrm{d}x$ See Section 3.6.2	derivative
`partialderivative(...)`	`pdv`	e.g. $\frac{\partial}{\partial x}, \frac{\partial f}{\partial x}, \frac{\partial^4 f}{\partial x^2 \partial y^2}, \frac{\partial^5 f}{\partial x^2 \partial y^3}, \partial f/\partial x$ See Section 3.6.3	partial derivative, could be mixed order

3.6.1. Differentials

Functions: `differential(`**args*, ***kwargs*`)`, abbreviated as `dd(...)`.

- positional *args*: the variable names, then at the last **optionally** followed by an order number e.g. `2`, or an order array e.g. `[2,3]`, `[k]`, `[m n, lambda+1]`.
- named *kwargs*:
  - `d`: the differential symbol [default: `upright(d)`].
  - `p`: the product symbol connecting the components [default: `none`].

Order assignment algorithm:

- If there is no order number or order array, all variables has order 1.
- If there is an order number (not an array), then this order number is assigned to *every* variable, e.g. `dd(x,y,2)` assigns $x \leftarrow 2, y \leftarrow 2$.
- If there is an order array, then the orders therein are assigned to the variables in order, e.g. `dd(f,x,y,[2,3])` assigns $x \leftarrow 2, y \leftarrow 3$.
- If the order array holds fewer elements than the number of variables, then the orders of the remaining variables are 1, e.g. `dd(x,y,z,[2,3])` assigns $x \leftarrow 2, y \leftarrow 3, z \leftarrow 1$.
- If a variable x has order 1, it is rendered as $\mathrm{d}x$ not $\mathrm{d}^1 x$.

Examples

(1) `dd(f), dd(x,y)`

$$\mathrm{d}f, \mathrm{d}x\mathrm{d}y$$

(2) `dd(x,3), dd(f,[k]), dd(f,[k],d:delta)`

$$\mathrm{d}^3x, \mathrm{d}^k f, \delta^k f$$

(3) `dd(f,2), dd(vb(x),t,[3,])`

$$\mathrm{d}^2 f, \mathrm{d}^3\boldsymbol{x}\mathrm{d}t$$

(4) `dd(x,y,[2,3]), dd(x,y,z,[2,3])`

$$\mathrm{d}^2x\mathrm{d}^3y, \mathrm{d}^2x\mathrm{d}^3y\mathrm{d}z$$

(5) `dd(x, y, z, [[1,1],rho+1,n_1])`

$$\mathrm{d}^{[1,1]}x\mathrm{d}^{\rho+1}y\mathrm{d}^{n_1}z$$

(6) `dd(x,y,d:Delta), dd(x,y,2,d:Delta)`

$$\Delta x\Delta y, \Delta^2 x\Delta^2 y$$

(7) `dd(t,x_1,x_2,x_3,p:and)`

$$\mathrm{d}t \wedge \mathrm{d}x_1 \wedge \mathrm{d}x_2 \wedge \mathrm{d}x_3$$

(7) `dd(t,x_1,x_2,x_3,d:upright(D))`

$$\mathrm{D}t\mathrm{D}x_1\mathrm{D}x_2\mathrm{D}x_3$$

3.6.2. Ordinary derivatives

Function: `derivative(`*f*, **args*, ***kwargs*`)`, abbreviated as `dv(...)`.
- *f*: the function, which can be `#none` or omitted,
- positional *args*: the variable name, then at the last **optionally** followed by an order number e.g. `2`,
- named *kwargs*:
  - `d`: the differential symbol [default: `upright(d)`].
  - `s`: the “slash” separating the numerator and denominator [default: `none`], by default it produces the normal fraction form $\frac{\mathrm{d}f}{\mathrm{d}x}$. The most common non-default is `slash` or simply `\/`, so as to create a flat form $\mathrm{d}f/\mathrm{d}x$ that fits inline.

Order assignment algorithm: there is just one variable, so the assignment is trivial: simply assign the order number (default to 1) to the variable. If a variable x has order 1, it is rendered as x not x^1.

Examples

(1) `dv(,x), dv(,x,2), dv(f,x,k+1)`

$$\frac{\mathrm{d}}{\mathrm{d}x}, \frac{\mathrm{d}^2}{\mathrm{d}x^2}, \frac{\mathrm{d}^{k+1}f}{\mathrm{d}x^{k+1}}$$

(2) `dv(, vb(r)), dv(f, vb(r)_e, 2)`

$$\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{r}}, \frac{\mathrm{d}^2}{\mathrm{d}\boldsymbol{r}_e^2}$$

(3) `dv(f,x,2,s:\/), dv(f,xi,k+1,s:slash)`

$$\mathrm{d}^2 f/\mathrm{d}x^2, \mathrm{d}^{k+1}f/\mathrm{d}\xi^{k+1}$$

(4) `dv(, x, d:delta), dv(, x, 2, d:Delta)`

$$\frac{\delta}{\delta x}, \frac{\Delta^2}{\Delta x^2}$$

(5) `dv(vb(u), t, 2, d: upright(D))`

$$\frac{\mathrm{D}^2\boldsymbol{u}}{\mathrm{D}t^2}$$

(6) `dv(vb(u),t,2,d:upright(D),s:slash)`

$$\mathrm{D}^2\boldsymbol{u}/\mathrm{D}t^2$$

3.6.3. Partial derivatives (incl. mixed orders)

Function: `partialderivative(`*f*, **args*, ***kwargs*`)`, abbreviated as `pdv(...)`.
- *f*: the function, which can be `#none` or omitted,
- positional *args*: the variable names, then at last **optionally** followed by an order number e.g. `2`, or an order array e.g. `[2,3]`, `[k]`, `[m n, lambda+1]`.
- named *kwargs*:
  - `s`: the “slash” separating the numerator and denominator [default: `none`], by default it produces the normal fraction form $\frac{\partial f}{\partial x}$. The most common non-default is `slash` or simply `\/`, so as to create a flat form $\partial f/\partial x$ that fits inline.
  - `total`: the user-specified total order.

- If it is absent, then (1) if the orders assigned to all variables are numeric, the total order number will be **automatically computed**; (2) if non-number symbols are present, computation will be attempted with minimum effort, and a user override with argument `total` may be necessary.

Order assignment algorithm:
- If there is no order number or order array, all variables has order 1.
- If there is an order number (not an array), then this order number is assigned to *every* variable, e.g. `pdv(f,x,y,2)` assigns $x \leftarrow 2, y \leftarrow 2$.
- If there is an order array, then the orders therein are assigned to the variables in order, e.g. `pdv(f,x,y,[2,3])` assigns $x \leftarrow 2, y \leftarrow 3$.
- If the order array holds fewer elements than the number of variables, then the orders of the remaining variables are 1, e.g. `pdv(f,x,y,z,[2,3])` assigns $x \leftarrow 2, y \leftarrow 3, z \leftarrow 1$.
- If a variable x has order 1, it is rendered as x, not x^1.

Examples

(1) `pdv(,x), pdv(,t,2), pdv(,lambda,[k])`

$$\frac{\partial}{\partial x}, \frac{\partial^2}{\partial t^2}, \frac{\partial^k}{\partial \lambda^k}$$

(2) `pdv(f,vb(r)), pdv(phi,vb(r)_e,2)`

$$\frac{\partial \varphi}{\partial \mathbf{r}}, \frac{\partial^2 \varphi}{\partial \mathbf{r}_e^2}$$

(3) `pdv(,x,y), pdv(,x,y,2)`

$$\frac{\partial^2}{\partial x \partial y}, \frac{\partial^4}{\partial x^2 \partial y^2}$$

(4) `pdv(f,x,y,2), pdv(f,x,y,3)`

$$\frac{\partial^4 \varphi}{\partial x^2 \partial y^2}, \frac{\partial^6 \varphi}{\partial x^3 \partial y^3}$$

(5) `pdv(,x,y,[2,]), pdv(,x,y,[1,2])`

$$\frac{\partial^3}{\partial x^2 \partial y}, \frac{\partial^3}{\partial x \partial y^2}$$

(6) `pdv(,t,2,s:\/), pdv(f,x,y,s:slash)`

$$\partial^2 / \partial t^2, \partial^2 f / \partial x \partial y$$

(7) `pdv(, (x^1), (x^2), (x^3), [1,3])`

$$\frac{\partial^5}{\partial (x^1) \partial (x^2)^3 \partial (x^3)}$$

(8) `pdv(phi,x,y,z,tau, [2,2,2,1])`

$$\frac{\partial^7 \varphi}{\partial x^2 \partial y^2 \partial z^2 \partial \tau}$$

(9) `pdv(,x,y,z,t,[1,xi,2,eta+2])`

$$\frac{\partial^{\eta+\xi+5}}{\partial x \partial y^{\xi} \partial z^2 \partial t^{\eta+2}}$$

(10) `pdv(,x,y,z,[xi n,n-1],total:(xi+1)n)`

$$\frac{\partial^{(\xi+1)n}}{\partial x^{\xi n} \partial y^{n-1} \partial z}$$

(11) `integral_V dd(V) (pdv(cal(L), phi) - diff_mu (pdv(cal(L), (diff_mu phi)))) = 0`

$$\int_V \mathrm{d}V \left(\frac{\partial \mathcal{L}}{\partial \varphi} - \partial_\mu \left(\frac{\partial \mathcal{L}}{\partial (\partial_\mu \varphi)} \right) \right) = 0$$

3.7. Miscellaneous

3.7.1. Reduced Planck constant (hbar)

In the default font, the Typst built-in symbol `planck.reduce` $\hslash$ looks a bit off: on letter “h” there is a slash instead of a horizontal bar, contrary to the symbol’s colloquial name “h-bar”. This package offers `hbar` to render the symbol in the familiar form: $\hbar$. Contrast:

Typst's `planck.reduce`	$E = \hbar\omega$	$\frac{\pi G^2}{\hbar c^4}$	$A e^{\frac{i(p x - E t)}{\hbar}}$	$i \hbar \frac{\partial}{\partial t} \psi = -\frac{\hbar^2}{2m} \nabla^2 \psi$
this package's `hbar`	$E = \hbar\omega$	$\frac{\pi G^2}{\hbar c^4}$	$A e^{\frac{i(p x - E t)}{\hbar}}$	$i \hbar \frac{\partial}{\partial t} \psi = -\frac{\hbar^2}{2m} \nabla^2 \psi$

3.7.2. Matrix transpose

Matrix transposition can be simply written as `..^T`, where the `T` will be formatted properly to represent transposition instead of a normal letter T. This conversion is disabled if the base is integral symbol.

To enable this feature, users need to first import this and call

```
#import "...(this physica package)...": super-T-as-transpose
#show: super-T-as-transpose
```

Examples

(1) `(A_n B_n)^T = B_n^T A_n^T`

$$(A_n B_n)^\mathsf{T} = B_n^\mathsf{T} A_n^\mathsf{T}$$

(2) `integral_0^T A^T f(x) dif x`

$$\int_0^T A^\mathsf{T} f(x) \,\mathrm{d}x$$

3.7.3. Tensors

Tensors are often expressed using the [abstract index notation](), which makes the contravariant and covariant "slots" explicit. The intuitive solution of using superscripts and subscripts do not suffice if both upper (contravariant) and lower (covariant) indices exist, because the notation rules require the indices be vertically separated: e.g. ${T^a}_b$ and ${T_a}^b$, which are of different shapes. "T_b^a" is flatly wrong, and `T^(space w)_(i space j)` produces a weird-looking "$T_{i\ j}^{\ w}$" (note w, j vertically overlap).

Function: `tensor(`*symbol*, **args*`)`.

- *symbol*: the tensor symbol,
- positional *args*: each argument takes the form of $+ \ldots$ or $- \ldots$, where a `+` prefix denotes an upper index and a `-` prefix denotes a lower index.

Examples

(1) `tensor(u,+a), tensor(v,-a)`

$$u^a, v_a$$

(2) `tensor(h,+mu,+nu), tensor(g,-mu,-nu)`

$$h^{\mu\nu}, g_{\mu\nu}$$

(3) `tensor(T,+a,-b), tensor(T,-a,+b)`

$${T^a}_b, {T_a}^b$$

(4) `tensor(T, -i, +w, -j)`

$${{T_i}^w}_j$$

(5) `tensor((dd(x^lambda)),-a)`

$$(\mathrm{d}x^\lambda)_a$$

(6) `tensor(AA,+a,+b,-c,-d,+e,-f,+g,-h)`

$${{{{\mathbb{A}^{ab}}_{cd}}^e}_f}^g{}_h$$

(7) `tensor(R, -a, -b, -c, +d)`

$${R_{abc}}^d$$

(8) `tensor(T,+1,-I(1,-1),+a_bot,-+,+-)`

$${{{T^1}_{I(1,-1)}}^{a_\bot}}_+{}^-$$

(9) `grad_mu A^nu = diff_mu A^nu + tensor(Gamma,+nu,-mu,-lambda) A^lambda`

$$\nabla_\mu A^\nu = \partial_\mu A^\nu + {\Gamma^\nu}_{\mu\lambda} A^\lambda$$

3.7.4. Isotopes

Function: `isotope(`*element*, *a*: ..., *z*: ...`)`.
- *element*: the chemical element (use `".."` for multi-letter symbols)
- *a*: the mass number A [default: `none`].
- *z*: the atomic number Z [default: `none`].

Change log: Typst merged my PR, which fixed a misalignment issue with the surrounding text.

Examples

(1) `isotope(I, a:127)`

$$^{127}\mathrm{I}$$

(2) `isotope("Fe", z:26)`

$$_{26}\mathrm{Fe}$$

(3) `isotope("Bi",a:211,z:83) --> isotope("Tl",a:207,z:81) + isotope("He",a:4,z:2)`

$$^{211}_{83}\mathrm{Bi} \longrightarrow {}^{207}_{81}\mathrm{Tl} + {}^{4}_{2}\mathrm{He}$$

(4) `isotope("Tl",a:207,z:81) --> isotope("Pb",a:207,z:82) + isotope(e,a:0,z:-1)`

$$^{207}_{81}\mathrm{Tl} \longrightarrow {}^{207}_{82}\mathrm{Pb} + {}^{0}_{-1}\mathrm{e}$$

3.7.5. The n-th term in Taylor series

Function: `taylorterm(`*func*, *x*, *x0*, *idx*`)`.
- *func*: the function e.g. `f`, `(f+g)`,
- *x*: the variable name e.g. `x`,
- *x0*: the variable value at the expansion point e.g. `x_0`, `(1+a)`,
- *idx*: the index of the term, e.g. `0`, `1`, `2`, `n`, `(n+1)`.

If *x0* or *idx* is in a parenthesis e.g. `(1+k)`, then the function automatically removes the outer parenthesis where appropriate.

Examples

(1) `taylorterm(f,x,x_0,0)`

$$f(x_0)$$

(2) `taylorterm(f,x,x_0,1)`

$$f^{(1)}(x_0)(x-x_0)$$

(3) `taylorterm(f,x,(1+a),2)`

$$\frac{f^{(2)}(1+a)}{2!}(x-(1+a))^2$$

(4) `taylorterm(f,x,x_0,n)`

$$\frac{f^{(n)}(x_0)}{n!}(x-x_0)^n$$

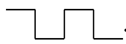
(5) `taylorterm(F,x^nu,x^nu_0,n)`

$$\frac{F^{(n)}(x_0^\nu)}{n!}(x^\nu-x_0^\nu)^n$$

(6) `taylorterm(f_p,x,x_0,(n+1))`

$$\frac{{f_p}^{(n+1)}(x_0)}{(n+1)!}(x-x_0)^{n+1}$$

3.7.6. Signal sequences (digital timing diagrams)

In engineering, people often need to draw digital timing diagrams for signals, like .

Function: `signals(str, step:`..., `style:`...`)`.
- `str`: a string representing the signals. Each character represents an glyph (see below).
- `step` (optional): step width, i.e. how wide each glyph is [default: `#1em`].
- `color` (optional): the stroke color [default: `#black`].

Glyph characters

`HLM` ⟺ `"10-"` full step | `hlm ^v` 1/2 step, 1/10 step | `| ' ,` (edge) 0 step | `= #` empty, shaded

`R` (rise) | `F` (fall) | `C` (charge) | `D` (drain)

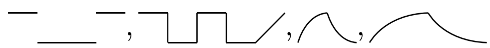
`<` | `>` | `X`

ignore: (blankspace)
separate: &

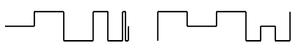
repeat: . (dot)

Examples

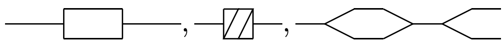
(1) `signals("10.1"), signals("1|0|1|0R"), signals("CD"), signals("CD", step: #2em)`



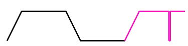
(2) `signals("M'H|L|h|l|^|v,&|H'M'H|l,m,l|")` (the ampersand & serves as a separator)



(3) `signals("-|=|-", step: #2em), signals("-|#|-"), signals("-<=>-<=")`

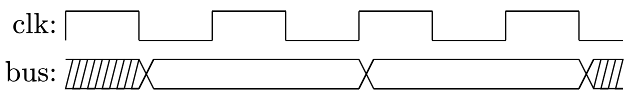


(4) `signals("R1..F0..", step: #.5em)signals("R1.|v|1", step: #.5em, color:#fuchsia)`



(5)

```
"clk:" & signals("|1....|0....|1....|0....|1....|0....|1....|0..", step: #0.5em) \
"bus:" & signals(" #.... X=... ..... ..... X=... ..... ..... X#.", step: #0.5em)
```



3.8. Symbolic addition

This package implements a very rudimentary, **bare-minimum-effort** symbolic addition function to aid the automatic computation of a partial derivative's total order in the absence of user override (see Section 3.6.3). Though rudimentary and unsophisticated, this should suffice for most use cases in partial derivatives.

Function: `BMEsymadd([...])`.
- `...`: symbols that need to be added up e.g. `[1,2]`, `[a+1,b^2+1,2]`.

Examples

(1) `BMEsymadd([1]), BMEsymadd([2, 3])` → $1, 5$

(2) `BMEsymadd([a, b^2, 1])` → $a + b^2 + 1$

(3) `BMEsymadd([a+1,2c,b,2,b])` → $a + 2b + 2c + 3$

(4) `BMEsymadd([a+1,2(b+1),1,b+1,15])` → $a + b + 2(b + 1) + 18$

(5) `BMEsymadd([a+1,2(b+1),1,(b+1),15])`	$\longrightarrow$	$a + 3(b + 1) + 17$
(6) `BMEsymadd([a+1,2(b+1),1,3(b+1),15])`	$\longrightarrow$	$a + 5(b + 1) + 17$
(7) `BMEsymadd([2a+1,xi,b+1,a xi + 2b+a,2b+1])`	$\longrightarrow$	$a\xi + 3a + 5b + \xi + 3$

4. Acknowledgement

Huge thanks to these LATEX packages, for lighting the way of physics typesetting.
- `physics` by Sergio C. de la Barrera,
- `derivatives` by Simon Jensen,
- `tensor` by Philip G. Ratcliffe et al.