TENSORSTYLES

A package to provide generic customizable displays of tensors (including vector, matrix, etc.) separating them from their forms.

Romain NOEL
romainoel@free.fr

v1.0.1.4 — 2026-04-28

Abstract

They are different notation conventions for tensors. And according to the context (or the local rules) one would like to switch between conventions: keeping the meaning of each tensor, just changing the display. So, this package is here to print tensors according to their order yet independently from their meaning in a very customizable way.

Contents

1 Introduction

tensorstyles was created to solve a problem we regularly faced: switching from one tensor notation to another. Such an action should be trivial but mainstream tensors libraries do not separate content and form, and only offer one or two notations each.

Therefore, if a user wants to switch from one convention to another (let's say when publish in several journals), they will need to manually change all the command related to tensor by hand without changing the content of their documents.

This is why we propose **tensorstyles**, a package that allows the representation of tensors in many notations, and to switch between them seamlessly.

2 Getting started

2.1 Installing from CTAN

The latest stable version of **tensorstyles** is available on CTAN and should now be part of the usual TEX distributions (TEX Live, MacTEX , MikTEX), under the name **tensorstyles**. It means that if your distribution is kept up-to-date, the package should

normally be already installed on your system. If this is not the case, consider updating the packages of your Tex distribution.

For TEX Live and MacTEX users, this usually means running

```
tlmgr update --all
```

or if administrative privileges are required

```
sudo tlmgr update --all
```

For MikTEX users, please refer to the official MikTEX documentation.

2.2 Installing from Gitlab

If you want to use the cutting-edge development version of **tensorstyles**, you can install it manually by following these steps:

Download the source from **tensorstyles** repository using `git clone` or as a zip archive of the latest development version.

Compile the style files by running `l3build unpack` inside the downloaded directory. (Or run LATEX directly on `source/tensorstyles.ins`.)

Move the resulting `*.sty` files to the folder containing your presentation. To use **tensorstyles** with many presentations, run `l3build install` or move the `*.sty` files to a folder in your TEX path instead.

Use the package by declaring `\usepackage{tensorstyles}` in the preamble of your document.

tensorstyles uses the l3build system to offer the following installation options for advanced users:

`l3build unpack` builds the theme style files.

`l3build doc` builds this documentation manual and the examples.

`l3build check` builds the theme and manual.

`l3build clean` removes the files generated by `l3build`.

`l3build install` installs the theme into your local texmf folder.

`l3build uninstall` removes the theme from your local texmf folder.

2.3 A minimal example

The following code shows a minimal example **tensorstyles** usage .

```
\documentclass{article}
\usepackage{tensorstyles}
\begin{document}
  $\tensor[preset=einstein]{4}{A}$
\end{document}
```

Figure 1: Result of a simple example.

2.4 Dependencies

tensorstyles depends on the following standard packages:

- `expl3`
- `amsmath`
- `amssymb`
- `mathtools`

3 Creating and using commands

3.1 Creating a command

The **tensorstyles** package does not only provide a macro to create tensors. The users can create their own.

To do so the package provides a command that acts in a manner similar to a constructor in object-oriented programming; this command is `\DeclaretensorstylesCmd`

It takes two arguments, the first is the name of the macro to create, including the backslash, and the second, optional argument, is a list of options to customize the display.

For example:

```
\DeclareTensorsCmd{\myTensorCmd}[preset=einstein]
\DeclareTensorsCmd{\myOtherTensorCmd}[preset=math, arrows
   =true]
\DeclareTensorsCmd{\thatTensorCmd}[preset=einstein,
   default-indices=contra]
```

```
\DeclareTensorsCmd{\anotherTensorCmd}[preset=full]
```

Afterward the newly created command can be used.

3.2 Using a custom command

Once a command is created it can be used anywhere in the document afterward. All the commands take the same arguments.

`\tensor[<option>]{<rank>}{<symbol>}[<indices>]_{sub}^{sup}`

or `\myTensorCmd[<option>]{<rank>}{<symbol>}[<indices>]_{sub}^{sup}`

`#1` : `options`: an optional list of options to override for this call
`#2` : `rank`: the rank of the tensor
`#3` : `symbol`: the symbol that represents the tensor
`#4` : `indices`: an optional list of Einstein indices
`#5` : `sub` and `sup`: optional sub- and super-scripts to add after the tensor

The options argument accepts any and all of the options that can be defined during command creation. The options given will override the ones that were defined when the command was created, but only for this call.

The rank can be a number or a letter. It is used for determining the font to use, which arrow type to add, and how many indices to write. For fonts and arrows, this is determined by a list, customizable via an option, that associates a given rank with a font and an arrow. For indices, unless they were manually given, the macro will add as many indices as the rank specified if it is a number, or if it is a letter the rank itself will be added instead. There is one exception to this: x and X. When used, the macro will calculate the rank based on the number of indices given, potentially 0.

The symbol can be anything, including another tensor; it is the object to which the notations will be added.

Indices are given with a specific syntax:

- First, write either ^ for contravariant indices or _ for covariant indices.
- Then any character that follows will be added as an index; for instance ^abc adds three contravariant indices a, b, and c.
- Any subsequent ^ and _ is used to switch between covariant and contravariant indices.
- If a group, {...}, is given it is treated as one index and it is expanded, allowing indices to be given with sub- and superscripts, or even passing a tensor as an index.

Every created command also has a star variant. This variant has another, fully customizable, set of options. This allows one to rapidly switch between two conventions.

4 Options

All the options have a star version used for the star variant of the command. They take the same arguments and have the same effects. Their name is the name of the regular options with a star added at the end. For example, `preset` and `preset*`.

The default values of each option depend on the preset.

4.1 Local vs. global options

The options given directly to the command (as described in the section 3.2) are applied locally. So they will only affect the display of this particular command, not the following ones.

For global setting that will last until the next modification, one can use

`\tensorsseet{<command>}[<global␣options>]`

`#1` : `command` for which the options have to be set.
`#2` : `global options` that will be set.

This setting can be called at any moment within the document.

4.2 General options

`preset` *empty, full, math, arrow, einstein, engineer, bra, ket* ..... full

Which preset to use.
`\tensor[preset=empty]{2}{x}` gives x
`\tensor[preset=full]{2}{x}` gives $\left\{ \overleftrightarrow{X}_{\alpha\,\beta} \right|$
`\tensor[preset=math]{2}{x}` gives X
`\tensor[preset=arrow]{2}{x}` gives $\overleftrightarrow{x}$
`\tensor[preset=einstein]{2}{x}` gives $\{x_{ab}|$
`\tensor[preset=engineer]{2}{x}` gives $\underline{\underline{x}}$
`\tensor[preset=bra]{2}{x}` gives $\langle x|$
`\tensor[preset=ket]{2}{x}` gives $|x\rangle$

`switch-*` *true, false* ..... false

Option to invert the star switch and regular version. `\tensor*{2}{x}` is equivalent to `\tensor[switch-*=true]{2}{x}`.
Every option has a star equivalent that controls the display of the star-versioned command. Therefore, one can use a combination of `preset` and `preset*` to switch easily between two displays.
For example, with the global setup: `\tensorsset{\tensor}[preset=math, preset*=einstein]`, the commands `$\tensor{2}{x}$`, `$\tensor[switch-*=true]{2}{x}$` and `$\tensor*{2}{x}$` yields : X, x_{ab} and x_{ab}

4.3 Font options

`font` *true, false* ..... true

Whether a special font should be applied to tensors.

`font-cmd` *{0=\font, 1=\font, ...}* .....

A list containing pairs `rank=font`. It contains the fonts that will be used for a tensor of a given rank. If the user tries to create a tensor with a rank not defined here and font is activated, the package will throw an error.
For example, `font-cmd = {0={}, 1=\boldsymbol, 2=\MakeUppercase, 3=\MakeUpperCal, 4=\MakeUpperBB, 5=\MakeBoldUppercase, n=\mathfrak}` is used for `preset=math`

4.4 Arrows options

`arrows` *true, false* ..... true

Whether arrows should be added to tensors.

`arrow-cmd` *{0=\macro, 1=\macro, ...}* .....

A list containing pairs `rank=macro`. It contains the macro that will be used to add arrows for a tensor of a given rank. If the user tries to create a tensor with a rank not defined here and arrows are activated, the package will throw an error.
For example, `arrow-cmd = {0={}, 1=\overrightarrow, 2=\overleftrightarrow, 3=\overrightleftrightarrow, 4=\overdoubleleftrightarrow, 5=\overrightdoubleleftrightarrow, n=\underleftrightarrow}` is used for `preset=arrow`

`recursive-arrows` *true, false* ..... false

Whether to add as many arrows as the rank.

`recursive-arrows-cmd` *\macro* .....

The macro that will be used to add arrows recursively, for example `recursive-arrows-cmd=\underline` is use for `preset=engineer`.

4.5 Einstein options

`contra-variant` *true, false* ..... true

Whether indices should be added to tensors.

`default-indices` *contra, covariant* ..... covariant

Whether indices added automatically should be covariant or contravariant.
For example: `\tensor[default-indices=contra]{2}{x}` gives $\left\{\overleftrightarrow{X}^{ab}\right|$

`contra-style` *greek, alph, arabic* ..... alph

The alphabet to use when adding contravariant indices automatically. Either Greek or Latin alphabets, or numbers.
For example: `\tensor[contra-style=arabic,default-indices=contra]{2}{x}` gives $\left\{\overleftrightarrow{X}^{12}\right|$

`covariant-style` *greek, alph, arabic* ..... greek

The alphabet to use when adding covariant indices automatically. Either Greek or Latin alphabets, or numbers.
For example: `\tensor[covariant-style=arabic]{2}{x}` gives $\left\{\overleftrightarrow{X}_{12}\right|$

`specific-indices` *1,2,3,...* .....

When adding indices automatically, which ones should not be added as `default-indices`. For instance, if default-indices is contra, which indices to add as covariant indices.
For example: `\tensor[specific-indices=2]{3}{x}` gives $\left\{\widetilde{\mathcal{X}}_{\alpha}{}^{b}{}_{\gamma}\right|$

`collapsed-indices` *true, false* ..... false

Whether indices are spaced horizontally, *i.e.* there will not be a covariant index under a contravariant index and vice versa, or not.
For example: `\tensor[specific-indices=2,collapsed-indices=true]{3}{x}` gives $\left\{\widetilde{\mathcal{X}}^{b}_{\alpha\,\gamma}\right|$

`index-marker` *\␣, ., ;, ...* ..... \␣

If indices are not collapsed, what symbol to use to fill in the blanks.
For example: `\tensor[specific-indices=2,index-marker=.]{3}{x}` gives $\left\{\widetilde{\mathcal{X}}_{\alpha\,.\,\gamma}^{\,b\,.}\right|$

4.6 Dirac options

`bra-ket` *true, false* ..... true

Whether delimiters should be added to tensors.

`scale-var` *auto, none, big, bigg, Big, Bigg* ..... auto

How the delimiters should scale with the tensor.

`delims-var` *\rvert\lbrace, \rbrace\lvert, ...* ..... \lbrace\rvert

Which delimiters to add. If scale-var is auto, the delimiters must be usable with \left and \right, else any symbol can be used.

`mix-delims` *true, false* ..... true

If two tensors are following each other, and the delimiters between them are identical, whether to display one or two.
For example: `\tensor[preset=bra]{1}{x}\tensor[preset=ket]{1}{y}` gives $\langle x|y\rangle$ while `\tensor[preset=bra, mix-delims=false]{1}{x}\tensor[preset=ket]{1}{y}` gives $\langle x||y\rangle$

5 Using the package

In this section, we will take the example of a user trying to write an article in which they will need several tensor notations.

Let's say that the user wants to represent tensors using an Einstein convention. More specifically they want to represent purely covariant and purely contravariant tensors with this convention.

There are two ways to do that with **tensorstyles**. The first is to define two new commands to represent tensors, one for covariant tensors and one for contravariant tensors.

```
\DeclareTensorsCmd{\covariantTensor}[preset=einstein]
\DeclareTensorsCmd{\contravariantTensor}[preset=einstein,
    default-indices=contra]
```

The second is to take advantage of the star variant of the commands and to define only one that does both.

```
\DeclareTensorsCmd{\myTensor}[preset=einstein, preset*=
   einstein, default-indices*=contra]
```

We will take the second approach for this example.

Now whenever they want to represent a tensor in their article, all they have to do is call this new command:

`\myTensor{4}{e}` $\rightarrow e_{a\,b\,c\,d}$ `\myTensor*{4}{e}` $\rightarrow e^{a\,b\,c\,d}$

Unfortunately, the user now needs to represent a tensor with both covariant and contravariant indices, something they did not anticipate. Fortunately **tensorstyles** offers two ways to do just that. First, the user could use an option to specify which indices should be contravariant or covariant:

`\myTensor{4}{e}[specific-indices={1,4}]` $\rightarrow e^{a}{}_{b\,c}{}^{d}$ Which is equivalent to
`\myTensor*{4}{e}[specific-indices*={2,3}]` $\rightarrow e^{a}{}_{b\,c}{}^{d}$

Or they can use an optional argument of the command to pass the indices directly.

`\myTensor{4}{e}[^a_bc^d]` $\rightarrow e^{a}{}_{bc}{}^{d}$ Which is equivalent to
`\myTensor*{4}{e}[^a_bc^d]` $\rightarrow e^{a}{}_{bc}{}^{d}$

This is more powerful since the user will have full control over the positions of the indices and their values. This is especially useful to pass indices with sub- or superscripts or even an entire command as an index. For example

`\myTensor{4}{e}[^{a_i}_bc^{\myTensor{2}{a}}]` $\rightarrow e^{a_i}{}_{bc}{}^{a_{a\,b}}$

Our user is now quite far into their document, but now they want to represent a tensor with another convention, let's say an engineer notation with recursive underlines. Perhaps they want to present their document to students, and to show them that other notations exist.

They can do this in two ways. First they can define a new command

```
\DeclareTensorsCmd{\tensorEngineer}[preset=engineer]
```

This works quite well and it preserves the separation of content and form. But it adds a new command that will only be used once, and so it is not really optimal in terms of memory or computation.

Alternatively they can change the behavior of their already existing command locally:

`\myTensor{4}{e}[preset=engineer]` $\rightarrow \underline{\underline{\underline{e}}}$

This avoids the declaration of a new command but it partially links content and form, so our user may have to be careful if they want to change the notation of their document.

For the last part of their document our user wants to use another convention entirely. Moreover it is a complex one: they want to use delimiters, like a Dirac convention, and to change the font of the tensor depending on the rank. Since they want this behavior for the rest of their document, redefining the display at each call is out of the question. So they have two options:

They can define a new command for this convention, or, the approach we will follow, they can use the `\tensorsset` command to override the options globally after its call.

```
\myTensor{2}{a}

\tensorsset{\myTensor}[preset=math, bra-ket=true, delims-
    var = \lbrace\rvert]

\myTensor{4}{e} and \myTensor{2}{a}
```

Gives us:

a_{ab}

$\{\mathbb{E}|$ and $\{A|$

Now, let us say that users want to switch the notation convention within the whole document. Thanks to **tensorstyles** this is trivial, all they need to do is go back to the function declaration and change the options at the beginning of the document they gave:

```
\DeclareTensorsCmd{\myTensor}[preset=arrow, preset*=math]
```

Plus they also used an engineer preset once, which will not be changed either, but since this call was to explicitly show this notation there is no need for a change. However, if many different presets have been used through the document, the previous approach remains tedious. Therefore, another approach can be to redefine presets themselves.

6 Sources of inspiration

Many packages have been used as sources of inspiration to build **tensorstyles**:

- https://gitlab.com/RomainNOEL/latex3_template_pkg for LateX3 template.
- mattens package from Danie Els for complex vector notations.
- hhtensor package from Harald Harders to print tensors with arrow, bold, or uline.
- tensor package from Philip G. Ratcliffe for Einstein tensor notation.
- tensind package from Javier Bezos for Einstein tensor notation.
- overarrows package from Julien Labbé to draw arrows that stretch.
- braket package from Donald Arseneau for Dirac bra-ket notation.
- physics package from Sergio C. de la Barrera for vec, braket and too many things.

7 Known issues

Some limitations have been faced.

- when using `\dots` with `collapsed-indices`, this is due to the tentative of context determination employed by `\dots` which interferes with the **tensorstyles**. A simple workaround is to use `\ldots` instead.
- if you are mixing delimiters using two variables of different sizes with automatic scale, *e.g.* `\tensor[preset=bra, mix-delims=true, scale-var=auto]{n}{\frac{1}{A}}\tensor*[preset*=ket, mix-delims*=true, scale-var*=auto]{n}{B}`, then the delimiters will not have the same size. A simple solution is to set the desired scale for the "wrong" delimiter.

7.1 Contributors

- Maelwen THOMAS
- Laurent NAVARRO

8 License

tensorstyles is licensed under the terms of the LaTeX project public license (LPPL) 1.3c license.