

SPECTRALSEQUENCES

Hood Chatham
hood@mit.edu

Version 1.3.0
2021/7/19

The SPECTRALSEQUENCES package is a specialized tool built on top of PGF/TikZ for drawing spectral sequence charts. It provides a powerful, concise syntax for specifying the data of a spectral sequence, and then allows the user to print various pages of a spectral sequence, automatically choosing which subset of the classes, differentials, structure lines, and extensions to display on each page. It also handles most of the details of the layout. At the same time, SPECTRALSEQUENCES is extremely flexible. It is closely integrated with TikZ to ensure that users can take advantage of as much as possible of its expressive power. It is possible to turn off most of the automated layout features and draw replacements using TikZ commands. SPECTRALSEQUENCES also has a carefully designed error reporting system intended to ensure that it is as clear as possible what is going wrong.

Many thanks to the authors of TikZ for producing such a wonderful package with such thorough documentation. I would have needed to spend a lot more time reading the TikZ code if the documentation weren't so excellent. I took ideas or code or both from tikzcd (part of the code for turning quotes into class or edge labels), PGFPLOTS (axes labels), and sseq (the grid types, the stack). I lifted a fair amount of code from TeXstack exchange. Thanks to Eva Belmont for tons of helpful suggestions, bug reports, and productive conversations. Talking to her has helped to clarify many design concepts for the package. Thanks to Eric Peterson for being a very early adopter and reporting many bugs. Also thanks to all my friends, family, and acquaintances listened to me talk about L^AT_EX programming even though they probably found it dreadfully boring.

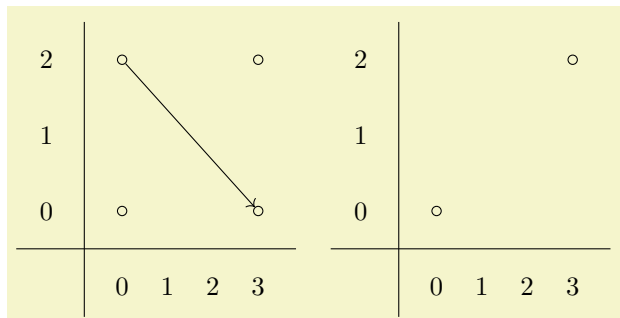
Contents

1	Introduction	3
1.1	Installation	3
1.2	Memory Constraints	3
1.3	A warning about fragile macros	4
2	Package Options and Environments	4
3	The Main Commands	4
4	Options for the main commands	10
4.1	Universal options	10
4.2	Options for <code>\class</code>	13
4.3	Options for <code>\d</code> , <code>\structline</code> , and <code>\extension</code>	18
4.4	Options for <code>\circleclass</code>	20
4.5	Options for TikZ primitives	21
5	Miscellaneous Commands	24
5.1	Settings	24
5.2	Code reuse commands	25
5.3	Families	29
5.4	Utilities	30
5.5	Coordinate parsers and related	31
5.6	The class stack	33
6	Styles	35
6.1	Style-like options	38
7	Global Options	40
7.1	Global coordinate transformations	46
7.2	Plot options and axes style	46
7.3	Layout	50

1 Introduction

The SPECTRALSEQUENCES package consists of two main environments – the `{sseqdata}` environment, which specifies the data for a named spectral sequence, and the `{sseqpage}` environment, which prints a single page of a spectral sequence. The `\printpage` command is also available as a synonym for a `{sseqpage}` environment with an empty body.

Here is a basic example:



```
\begin{sseqdata}[ name = basic, xscale = 0.6,
                  cohomological Serre grading ]
\class(0,0)
\class(0,2)
\class(3,0)
\class(3,2)
\d3(0,2)
\end{sseqdata}
\printpage[ name = basic, page = 3 ] \quad
\printpage[ name = basic, page = 4 ]
```

`\begin{sseqdata}[name = basic, cohomological Serre grading]` starts the declaration of the data of a spectral sequence named *basic* with cohomological Serre grading – that is, the page r differentials go r to the right and down $r - 1$. Then we specify four classes and one page 3 differential, and we ask SPECTRALSEQUENCES to print the third and fourth pages of the spectral sequence. Note that on the fourth page, the source and target of the differential have disappeared.

1.1 Installation

In both MiKTeX and TeX Live installation should be automatic – your TeX distribution should automatically install the package the first time you include `\usepackage{spectralsequences}` in a document and compile it. However, in 2016, TeX Live made an incompatible change to their database, so no new packages will run on versions of TeX Live from before 2016. This includes SPECTRALSEQUENCES. If you have an old version of TeX Live, you can either perform a manual install, or, better, you should install an up to date version of TeX Live. If you want to do a manual install, see [this TeXstack exchange post](#) for instructions.

1.2 Memory Constraints

In a default TeX install, PDFLaTeX has small static memory caps that prevent it from using more than about 60 megabytes of total ram. However, SPECTRALSEQUENCES and PGF/TikZ use a large amount of memory. For this reason, using PDFLaTeX with a default install, you cannot draw more than about 2500 classes across all of your diagrams (fewer if you include differentials, structure lines, and other features). There are a few solutions to this.

The easiest solution is to run LuaLaTeX. LuaLaTeX dynamically allocates memory and so is unlikely to run out of it. Using LuaLaTeX on my computer, I can compile a document that draws two copies of a diagram with 20,000 classes in it (so a total of 40,000 classes). This takes about 50 seconds and 250 megabytes of ram. I expect any real-world use case will compile fine on a modern computer using LuaLaTeX. This option has the advantage that any modern TeX install comes with a copy of LuaLaTeX, and that LuaLaTeX is the designated successor to PDFLaTeX. It has the disadvantage that there are some incompatibilities between LuaLaTeX and PDFLaTeX so if your document depends on PDFLaTeX-specific features, it might be a pain to switch to LuaLaTeX.

Another option is to increase the static memory caps for PDFLaTeX. See [this TeXstack exchange post](#) for instructions on how to do this.

1.3 A warning about fragile macros

All the data in a SPECTRALSEQUENCES environment is stored and used later. As a result, most of the SPECTRALSEQUENCES commands currently cannot tolerate fragile macros. Unfortunately, it is impossible for SPECTRALSEQUENCES to warn you about this situation – if you use a fragile command in a place that it doesn't belong, the result will be an incomprehensible error message. If you are getting nonsense error messages, this might be why. The solution is to convert fragile macros into robust ones. Common examples of fragile macros include `\widehat` and `\underline`. My suggested solution to this is to add the following code to your preamble for each fragile macro (example given for `\mathbb`):

```
\let\oldwidehat\widehat
\protected\def\widehat{\oldwidehat}
```

2 Package Options and Environments

Draft Mode

The drawings that SPECTRALSEQUENCES produces can be quite slow, especially if they are large. Draft mode skips drawing the content of the spectral sequence, but still takes up exactly the same amount of space in the document, so that you can deal with formatting issues. To active draft mode, load the package by saying `\usepackage[draft]{spectralsequences}`.

```
\begin{sseqdata}[\langle options \rangle]
  \langle environment contents \rangle
\end{sseqdata}
```

The `{sseqdata}` environment is for storing a spectral sequence to be printed later. This environment is intended for circumstances where you want to print multiple pages of the same spectral sequence. When using the `{sseqdata}` environment, you must use the `name` option to tell SPECTRALSEQUENCES where to store the spectral sequence so that you can access it later.

```
\begin{sseqpage}[\langle options \rangle]
  \langle environment contents \rangle
\end{sseqpage}
```

This environment is used for printing a page of existing spectral sequence that was already specified using the `{sseqdata}` environment. The body of the environment adds local changes – classes, differentials, structure lines, extensions, and arbitrary TikZ options that are by default only printed on this particular page. The `{sseqpage}` environment can also be used to print a stand-alone page of a spectral sequence – that is, if you only want to print a single page of the spectral sequence, you can skip using the `{sseqdata}` environment.

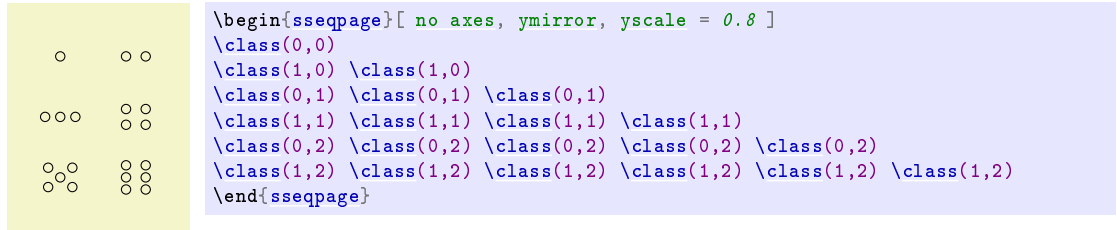
```
\printpage[\langle options \rangle]
```

This command prints a single page of an existing spectral sequence as-is. This is equivalent to a `{sseqpage}` environment with an empty body.

3 The Main Commands

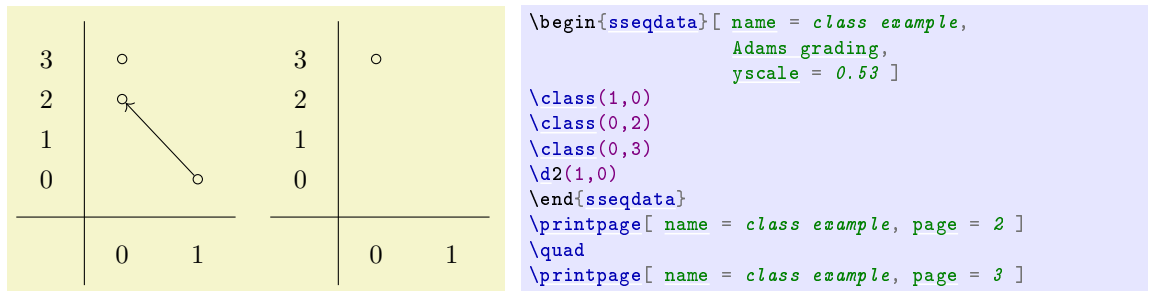
```
\class[\langle options \rangle](\langle x \rangle, \langle y \rangle)
```

This places a class at (x, y) where x and y are integers. If multiple classes occur at the same position, SPECTRALSEQUENCES will automatically arrange them in a pre-specified pattern. This pattern may be altered using the `class pattern` option.



The effect of the `\class` command is to print a TikZ node on a range of pages. Any option that would work for a TikZ `\node` command will also work in the same way for the `\class`, `\replaceclass`, and `\classoptions` commands.

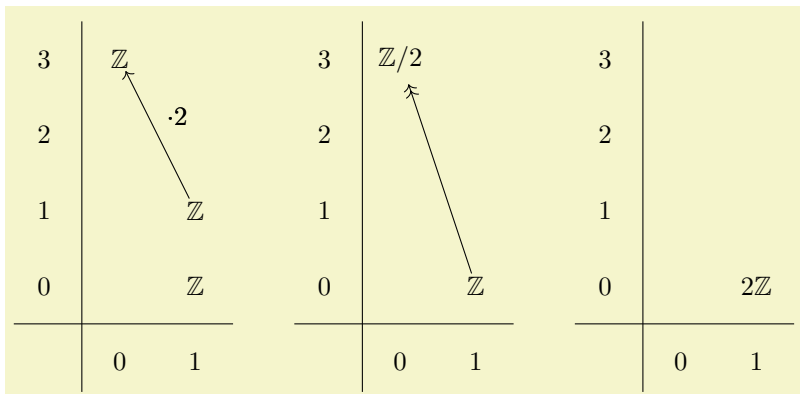
If a class is the source or the target of a differential on a certain page, then the page of the class is set to that page, and the class is only rendered on pages up to that number:



See the class options section for a list of the sort of options available for classes.

`\replaceclass[<options>](<x>,<y>,<n>)`
`\replaceclass[<options>](<classname>)`
`\replacesource[<options>]`
`\replacetarget[<options>]`

After a class is the source or target of a differential, it disappears on the next page. However, some differentials are not injective or not surjective. Using the command `\replaceclass` causes a new symbol to appear on the page after a class supported or accepted a differential (or both). If there are multiple classes at the coordinate (x,y) you may specify which using an integer or a `tag n`. By default, this command will affect the first class placed in that position. You can also provide the `class:name` of a class. The variants `\replacesource` and `\replacetarget` replace the source and target respectively of the most recent differential.



```

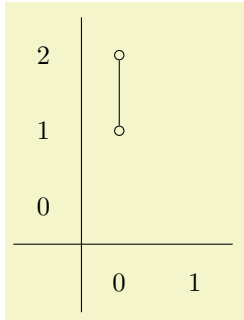
\begin{sseqdata}[name = replace class example, Adams grading, classes = {draw = none} ]
\class["\mathbb{Z}"](0,3)
\class["\mathbb{Z}"](1,1)
\class["\mathbb{Z}"](1,0)
\d["\cdot 2"]2(1,1)
\replacetarget["\mathbb{Z}/2"] %\replaceclass["\mathbb{Z}/2"](0,3)
\d[->]3(1,0)
\replacesource["2\mathbb{Z}"] %\replaceclass["2\mathbb{Z}"](1,0)
\end{sseqdata}
\printpage[ name = replace class example, page = 2 ] \qqquad
\printpage[ name = replace class example, page = 3 ] \qqquad
\printpage[ name = replace class example, page = 4 ]

```

Note that this will not restore any structure lines coming into or off of the class. If you want to restore all structlines on the class use `\replacestructlines`. If you want to selectively replace some of the structure lines, you must use `\structline` again (or use the `structline:page` option).

`\replacestructlines(<source coordinate>)`

This command replaces all structlines touching a class that has been replaced using `\replaceclass`, `\replacesource`, or `\replacetarget`.



```

\begin{sseqdata}[name=replacestructlines]
\class(0,1)
\class(0,2)
\structline
\class(1,0)
\d2(1,0)(0,2)
\replacetarget\replacestructlines
\end{sseqdata}
\printpage[name=replacestructlines, page=3]

```

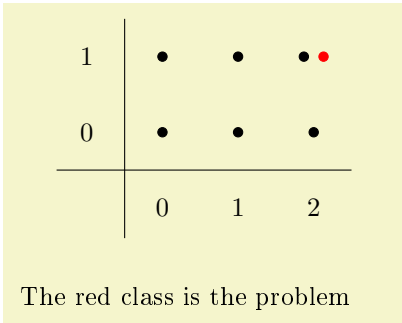
`\classoptions[<options>](<x>,<y>,<n>)`

`\classoptions[<options>](<classname>)`

`\classoptions[<options>]`

This adds options to an existing class. This can be used in a `{sseqpage}` environment to modify the appearance of a class for just one drawing of the spectral sequence, for instance to highlight it for discussion purposes.

If there are multiple classes at the coordinate (x,y) you may specify which using an integer or a `tag n`. By default, this command will affect the first class placed in that position. You can also provide the `class:name` of a class. If no coordinate is indicated at all, then `\lastclass` is used.

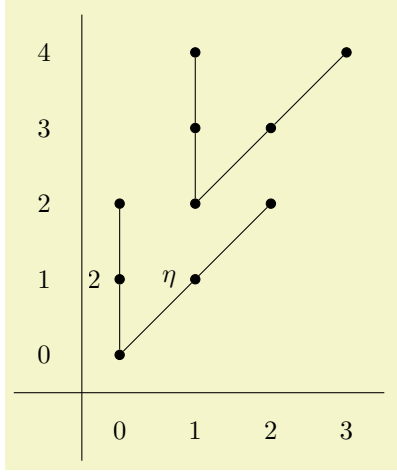


```

\begin{sseqdata}[ name = class options example,
                  classes = fill ]
\class(2,1)
\foreach \x in {0,...,2} \foreach \y in {0,1} {
  \class(\x,\y)
}
\end{sseqdata}
\begin{sseqpage}[ name = class options example,
                  right clip padding = 0.6cm ]
\classoptions[red](2,1,2) % Only is red on this page!
\node[ background ] at (0.3,-2.2)
  {\textup{The red class is the problem}};
\end{sseqpage}

```

Another reason to use this is to give a label to one instance of a class that shows up in a loop or a command defined using `\NewSseqGroup`:



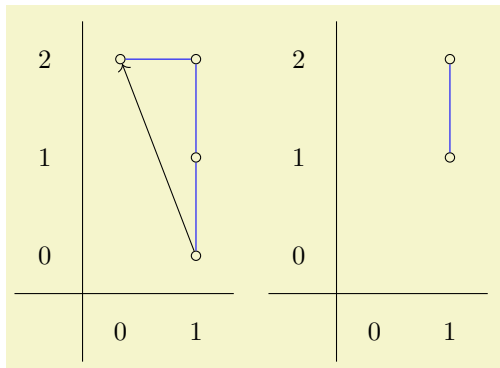
```
\NewSseqGroup\mygroup {} {
  \class(0,0)
  \class(0,1)
  \class(0,2)
  \class(1,1)
  \class(2,2)
  \structline(0,0)(0,1)
  \structline(0,1)(0,2)
  \structline(0,0)(1,1)
  \structline(1,1)(2,2)
}
\begin{sseqpage}[ classes = fill, class labels = { left = 0.3em } ]
\mygroup(0,0)
\mygroup(1,2)
\classoptions["2"](0,1)
\classoptions["\eta"](1,1)
\end{sseqpage}
```

See the class options section for a list of the sort of options available for classes.

```
\d[options](page)
\d[options](page)(x,y,source n,target n)
\d[options](page)(source name,target n)
\d[options](page)(source coordinate)(target coordinate)
```

Calling `\d\meta{page}(x,y)` creates a differential starting at (x,y) of length determined by the specified page. In order to use the `\d` command like this, you must first specify the **degree** of the differentials as an option to the `{sseqdata}` or `{sseqpage}` environment. The degree indicates how far to the right and how far up a page r differential will go as a function of r . If there is a page r differential, on page $r + 1$, the source, target, and any structure lines connected to the source and target of the differential disappear. If no class is specified, the default is to use `\lastclass`.

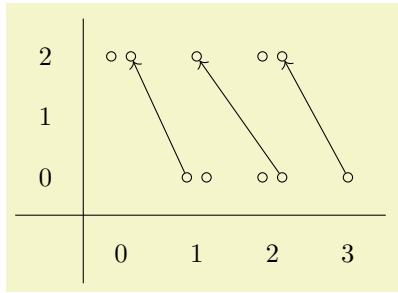
If there are multiple nodes in the source or target, you may specify which one the differential should go to using an index or tag for $\langle source\ n \rangle$ or $\langle target\ n \rangle$. It is also possible to provide the name of the source coordinate and an optional target, or to separately provide the source and target coordinate, either as names or as $(\langle x \rangle, \langle y \rangle, \langle n \rangle)$. Using `\d` with explicit source and target coordinates works even if you did not provide a **degree** to the spectral sequence. If you did provide a **degree**, then SPECTRALSEQUENCES will check whether the difference between the source and target is appropriate for a differential of a given page, and if not it will throw an error. If this is undesirable, you can use the `lax degree` option.



```
\begin{sseqdata}[ name = d example, degree = {-1}{#1},
  struct lines = blue, yscale = 1.3 ]
\class(0,2)
\class(1,2)
\class(1,1)
\class(1,0)
\structline(1,2)(0,2)
\structline(1,2)(1,1)
\structline(1,1)(1,0)
\d2(1,0)
\end{sseqdata}
\printpage[ name = d example, page = 2 ] \quad
\printpage[ name = d example, page = 3 ]
```

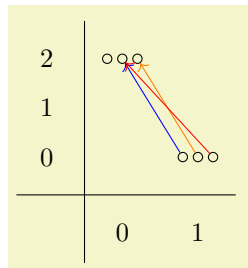
If there are multiple nodes in the source or target coordinate, then there is a funny syntax for indicating which one should be the source and target:

```
\d\meta{page}\pars{\meta{x},\meta{y}}\opt{\,source n,\target n}}
```



```
\begin{sseqpage}[ Adams grading, yscale = 0.8 ]
\class(1,0) \class(1,0)
\class(0,2) \class(0,2)
\d2(1,0,1,2)
\class(2,0) \class(2,0)
\class(1,2)
\d2(2,0,2)
\class(3,0)
\class(2,2) \class(2,2)
\d2(3,0,,2)
\end{sseqpage}
```

Negative indices will count from the most recent class in the coordinate (so the most recent is -1, the second most recent is -2, etc). You can also use a **tag**, which works better if the situation is complicated.



```
\begin{sseqpage}[ Adams grading, yscale = 0.65 ]
\class(1,0)
\class(0,2) \class(0,2)
\d[blue]2(1,0,-1,-1)
\class(1,0)
\class(0,2)
\d[orange]2(1,0,-1,-1)
\class(1,0)
\d[red]2(1,0,-1,-2)
\end{sseqpage}
```

\doptions[*<options>*](*<page>*)(*<x>*,*<y>*),(*<source n>*),(*<target n>*)

\doptions[*<options>*](*<page>*)(*<source name>*),(*<target n>*)

\doptions[*<options>*](*<page>*)(*<source coordinate>*)(*<target coordinate>*)

This command adds options to an existing differential, just like **\classoptions** except for differentials. Its syntax is identical to that of **\d**.

\kill(*<page>*)[*<coord>*]

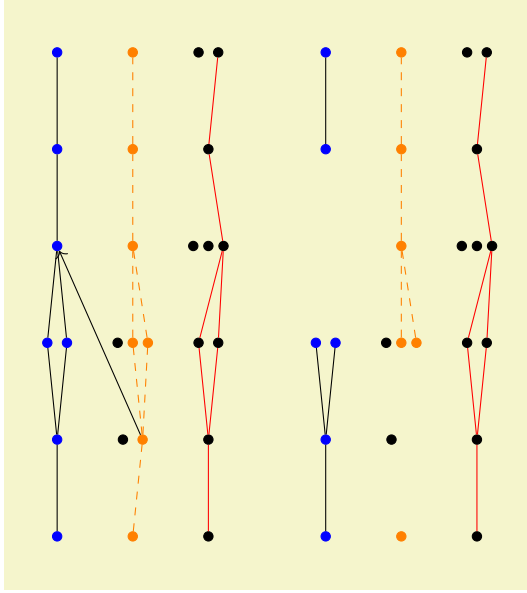
This command sets the indicated coordinate to die on the indicated page, but does not establish a target for the differential. This is useful if you want to draw your own differential using tikz (see **\getdtarget**) or if you are not drawing the class on the other side of the differential for clutter reasons. As usual, if no coordinate is provided, the default argument is **\lastclass**.

\structline[*<options>*](*<source coordinate>*)(*<target coordinate>*)

The **\structline** command creates a structure line from *<source coordinate>* to *<target coordinate>*. The source and target coordinates are either of the form (*<x>*,*<y>*,*<n>*) or (*<class name>*). If there are multiple classes at (*x*,*y*), then *<n>* specifies which of the classes at (*x*,*y*) the structure line starts and ends at – if *n* is positive, then it counts from the first class in that position, if *n* is negative, it counts backwards from the most recent. You can also use a **tag** for *n*. If the *<target coordinate>* is omitted, then **\lastclass** is used, so that **\structline**(*<sourcecoord>*) connects the most recent class to the specified coordinate. If both coordinates are omitted, then **\lastclass** and **\lastclass1** are used, and so **\structline** with no arguments at all will connect the two most recent classes.

If the source or target of a structure line is hit by a differential, then on subsequent pages, the structure line disappears.

If the source or target has had multiple generations (i.e., they got hit and you used **\replaceclass**), then the **\structline** will only appear starting on the first page where the current generation of both the source and target are present. If this is undesirable, you can use the **structline:page** option or the **to** to change it. Also, the structline will disappear the first time after this the source or target has a differential, but this can be changed with the **\replacestructlines** command.



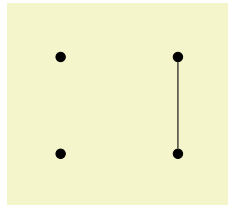
```
\DeclareSseqGroup\tower {} {
  \class(0,0)
  \foreach \y in {1,...,5} {
    \class(0,\y)
    \structline
  }
  \class(0,2)
  \structline(0,1,-1)
  \structline(0,3,-1)
}
\begin{sseqdata}[ name = structline example,
  classes = { circle, fill },
  Adams grading, no axes,
  yscale = 1.28 ]
\class(1,1) \class(1,2)
\class(2,3) \class(2,3) \class(2,5)
\tower[classes = blue](0,0)
\tower[struct lines = dashed,orange](1,0)
\tower[struct lines = red](2,0)
\d2(1,1,2)
\end{sseqdata}
\printpage[ name = structline example, page = 2 ] \quad
\printpage[ name = structline example, page = 3 ]
```

\structlineoptions[*(options)*](*(source coordinate)*)(*(target coordinate)*)

This command adds options to an existing structure line, just like `\classoptions` except for structure lines. Its syntax is identical to `\structline`.

\extension[*(options)*](*(source coordinate)*)(*(target coordinate)*)

The `\extension` command has an identical syntax to the `\structline` command and most of the same options. Instead of adding a structline, it adds an extension. The extensions are only shown on page ∞ or page ranges ending at ∞ .



```
\begin{sseqdata}[ name = extension example,
  classes = { circle, fill },
  Adams grading, no axes,
  yscale = 1.28 ]
\class(0,0) \class(0,1)
\extension
\end{sseqdata}
\printpage[ name = extension example, page = 2 ] \quad
\printpage[ name = extension example, page = \infty ]
```

\extensionoptions[*(options)*](*(source coordinate)*)(*(target coordinate)*)

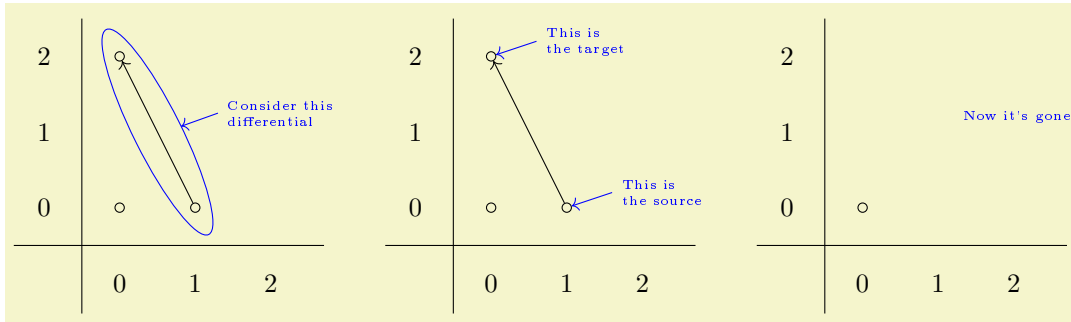
This command adds options to an existing extension. Its syntax is identical to `\extension`.

\circleclasses[*(options)*](*(source coordinate)*)(*(target coordinate)*)

This command is a lot like `\structline` except that it puts a circle around the classes instead of connecting them with a line. It might take a certain amount of fiddling with options to get `\circleclasses` to produce good results. There is no `\circleclassesoptions` command because it doesn't seem necessary.

\draw
\path
\node
\clip

Any code that would work in a `{tikzpicture}` environment will also work unchanged in a `{sseqdata}` or `{sseqpage}` environment, with a few minor differences. This is a very flexible way to add arbitrary background or foreground features to the spectral sequence:



```

\begin{sseqdata}[ name = tikz example, Adams grading, math nodes = false,
tikz primitives = { blue, font = \tiny, <- }, circle classes = tikz primitive style,
x range = {0}{2}, x axis extend end = 2em ]

\class(0,0)
\class(1,0)
\class(0,2)
\d2(1,0)
\end{sseqdata}
%
\begin{sseqpage}[ name = tikz example ]
\circleclasses[ name path = myellipse, inner sep = 3pt, ellipse ratio = 1.6 ] (1,0) (0,2)
\path[ name path = myline ] (1.3,1.25) -- (0.6,1);
\draw[ name intersections = { of = myellipse and myline } ]
(intersection-1) to (1.3,1.25) node[ right, text width = 1.6cm ] {Consider this differential};
\end{sseqpage} \qqquad
%
\begin{sseqpage}[ name = tikz example ]
\draw[ xshift = 1 ] (0,0) to (0.6,0.2) node[ right, text width = 1.1cm ] {This is the source};
\draw[ yshift = 2 ] (0,0) to (0.6,0.2) node[ right, text width = 1.1cm ] {This is the target};
\end{sseqpage} \qqquad
%
\begin{sseqpage}[ page = 3, name = tikz example ]
\circleclasses[ inner sep = 3pt, ellipse ratio = 1.6 ] (1,0)(0,2)
\node[ right, font = \tiny ] at (1.2,1.2) {Now it's gone!};
\end{sseqpage}

```

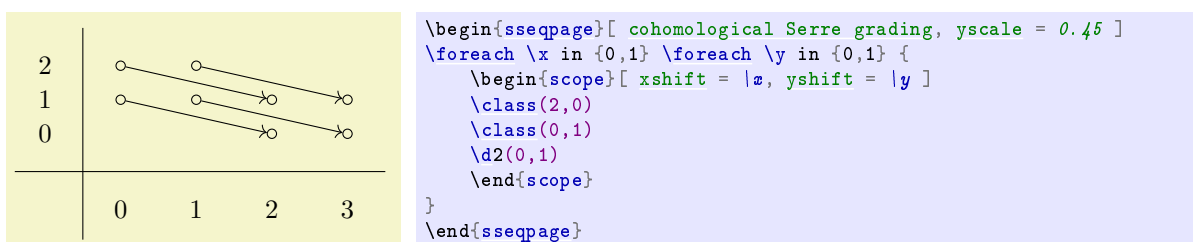
4 Options for the main commands

4.1 Universal options

The following options work with all of the drawing commands in this package, including `\class`, `\d`, and `\structline`, their friends `\replaceclass`, `\classoptions`, `\doptions`, and `\replacestructlines`, as well as with TikZ primitives.

`xshift` = $\langle integer \rangle$
`yshift` = $\langle integer \rangle$

Shifts by integer values are the only coordinate changes that are allowed to be applied to `\class`, `\d`, `\structline`, their relatives, or to a `{scope}` environment that contains any of these commands. These shift commands help with reusing code. For instance:



This code segment is very useful so `SPECTRALSEQUENCES` has the command `\NewSseqGroup` which to make code like this more convenient. The following code produces the same output as above:

```

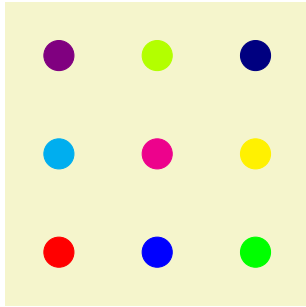
\NewSseqGroup\examplegroup {} {
  \class(2,0)
  \class(0,1)
  \d2(0,1)
}
\begin{sseqpage}
\examplegroup(0,0)
\examplegroup(0,1)
\examplegroup(1,0)
\examplegroup(1,1)
\end{sseqpage}

```

A word of warning: the behavior of `xshift` in `SPECTRALSEQUENCES` is incompatible with the normal behavior of `xshift` in `TikZ`. For some reason, saying `xshift = 1` in `TikZ` does not shift the coordinate $(0,0)$ to the coordinate $(1,0)$ – instead it shifts by 1pt. In `SPECTRALSEQUENCES`, saying `xshift = 1` moves the coordinate $(0,0)$ to the coordinate $(1,0)$. This includes `TikZ` primitives: saying `\draw[xshift = 1] (0,0) -- (1,0);` inside a `{sseqdata}` or `{sseqpage}` environment is the same as saying `\draw(1,0) -- (2,0);` despite the fact that this is not the case in the `{tikzpicture}` environment.

Colors

These come from the `LATEX` color package via `TikZ`, so see the [color package documentation](#) for more information.



```

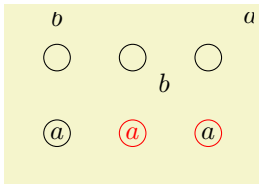
\begin{sseqpage}[ classes = {fill,inner sep = 0.4em},
  no axes, scale = 1.3 ]
\class[red](0,0)
\class[blue](1,0)
\class[green](2,0)
\class[cyan](0,1)
\class[magenta](1,1)
\class[yellow](2,1)
\class[blue!50!red](0,2) % a 50-50 blend of blue and red
\class[green!30!yellow](1,2) % 30% green, 70% yellow
\class[blue!50!black](2,2)
\end{sseqpage}

```

"*text*" "*options*"

Specify a label for a class, a differential, or a structure line. This uses the `TikZ` quotes syntax. If the label text includes an equal sign or comma, you need to enclose the entire label in braces, e.g., `\class["{x = y}"](0,0)`. The options include anything you might pass as an option to a `TikZ` node, including arbitrary coordinate transforms, colors, opacity options, shapes, fill, draw, etc. The behavior is a little different depending on whether you use it on a class or on a differential or structure line.

For a class, the `<text>` is placed in the position inside the node by default – in effect, the `<text>` becomes the label text of the node (so saying `\class["label text"](0,0)` causes a similar effect to saying `\node at (0,0) {label text};`). There are other position options such as `above left`, `below left`, etc which cause the label text to be placed in a separate node positioned appropriately. If the placement is `above`, `left`, etc, then any option that you may pass to a `TikZ` node will also work for the label, including general coordinate transformations. If the placement is “inside”, then the only relevant `<options>` are those that alter the appearance of text, such as opacity and color.



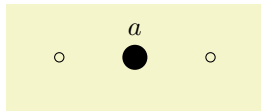
```

\begin{sseqpage}[ classes = { minimum width = width("a") + 0.5em }, no axes ]
\class["a"](0,0)
\class["a", red](1,0)
\class["a" black, red](2,0)
\class["b" above](0,1)
\class["b" { below right, yshift = 0.1cm }](1,1)
\class["a" { above right = {1em} }](2,1)
\end{sseqpage}

```

You can adjust the default behavior of class labels using the `labels` style option or its relatives `class labels`, `inner class labels` or `outer class labels`. Note that it is also possible to give a label to a `\node` this way, although the behavior is slightly different. In particular, the label defaults

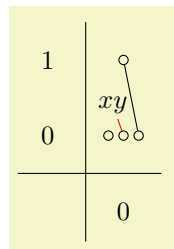
to the `above` position instead of going in the `\node` text by default. Also, this won't respect the various label style options like `labels`, etc.



```
\begin{sseqpage}[ no axes ]
\class(0,0)
\class(2,0)
\node[circle, fill, "a"] at (1,0) {};
\end{sseqpage}
```

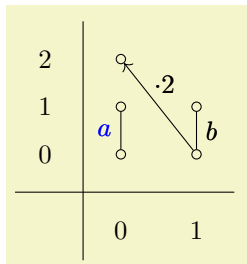
`pin = <style>`

The `pin` key makes SPECTRALSEQUENCES draw a line connecting the label to the relevant class, which can provide necessary clarification in dense diagrams. The `pin` key itself can take options which adjust the way that the line is drawn:



```
\begin{sseqpage}
\class(0,0)
\class["xy" { above, xshift = -4pt, pin = red }](0,0)
\class(0,0)
\class(0,1)
\structline
\end{sseqpage}
```

For either a `\structline` or a `\class` the label normally goes on the right side of the edge. The special option `'` makes it go in the opposite position from the default. I imitated the label handling in the `tikzcd` package, so if you use `tikzcd`, this should be familiar.



```
\begin{sseqpage}[ Adams grading, yscale = 0.63 ]
\class(0,0)
\class(0,1)
\class(0,2)
\structline["a" blue](0,0)(0,1)
\class(1,0)
\class(1,1)
\structline["b"](1,0)(1,1)
\d[ "\cdot 2" { pos = 0.7, yshift = -5pt } ] 2 (1,0)
\end{sseqpage}
```

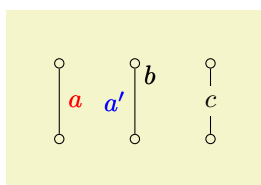
You can use the style options `labels`, `edge labels`, `differential labels`, and `struct line labels` to adjust the styling of edge labels. For instance, if you would prefer for the labels to default to the left hand side of the edge rather than the right hand side, you could say `edge labels = {auto = left}`. You can also use quotes to label edges drawn with TikZ primitives:



```
\begin{sseqpage}[ yscale = 0.58, no axes ]
\class(0,0)
\class(1,1)
\draw (1,0) to["hi" { pos = 0.7, yshift = -0.5em } ] (0,1);
\end{sseqpage}
```

description

The `description` key, stolen from `tikzcd`, places the label on top of the edge. In order to make this option work correctly, if the background color is not the default white, you must inform SPECTRALSEQUENCES about this using the key `background color = <color>`. In this document, the background color is called `graphicbackground`.



```
\begin{sseqpage}[ no axes, background color = graphicbackground ]
\foreach \x in {0,1,2} \foreach \y in {0,1} {
  \class(\x,\y)
}
\structline["a" red](0,0)(0,1)
\structline["a'" blue, "b" {yshift = 1em}](1,0)(1,1)
\structline["c" description](2,0)(2,1)
\end{sseqpage}
```

4.2 Options for `\class`

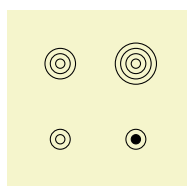
Because the main job of the `\class` command is to print a TikZ `\node` on the appropriate pages of the spectral sequence, most options that would work for a TikZ node also work for the commands `\class`, `\replaceclass`, and `\classoptions`. Here are a few that you might care about:

A TikZ shape

If you give the name of a TikZ shape, the class node will be of that shape. The standard TikZ shapes are `circle` and `rectangle`. `SPECTRALSEQUENCES` defines two new shapes:

`circlen` = $\langle n \rangle$

This draws n concentric circles. It's intended for indicating a \mathbb{Z}/p^n summand. For large values of n the result isn't all that appealing.



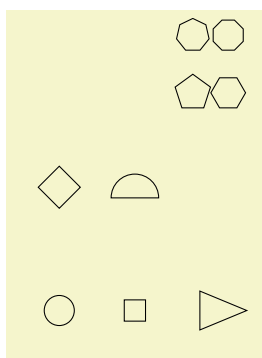
```
\begin{sseqpage}[ no axes ]
\class[circlen = 2](0,0)
\class[circlen = 2,fill](1,0)
\class[circlen = 3](0,1)
\class[circlen = 4](1,1)
\end{sseqpage}
```

`newellipse`

`ellipse ratio` = $\langle ratio \rangle$

This shape is used for `\circleclasses`. It's a variant on the `ellipse` shape that gives more control over the ellipse's aspect ratio.

There are many more TikZ shapes in the shapes library, which you can load using the command `\usetikzlibrary{shapes}`. The following are some examples:



```
\begin{sseqpage}[ no axes, classes = { inner sep = 0.4em },
class placement transform = { scale = 1.8 },
yscale = 1.63 ]

\class(0,0)
\class[isosceles triangle](2,0)
\class[rectangle](1,0)
\class[diamond](0,1)
\class[semicircle](1,1)
\class[regular polygon, regular polygon sides = 5](2,2)
\class[regular polygon, regular polygon sides = 6](2,2)
\class[regular polygon, regular polygon sides = 7](2,2)
\class[regular polygon, regular polygon sides = 8](2,2)
\end{sseqpage}
```

See the [TikZ manual](#) for more information.

`minimum width` = $\langle dimension \rangle$

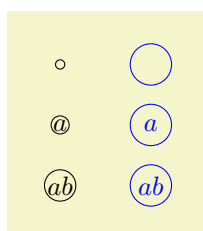
`minimum height` = $\langle dimension \rangle$

`minimum size` = $\langle dimension \rangle$

`inner sep` = $\langle dimension \rangle$

`outer sep` = $\langle dimension \rangle$

These options control the size of a node. This is typically useful to make the size of nodes consistent independent of the size of their label text. For instance:

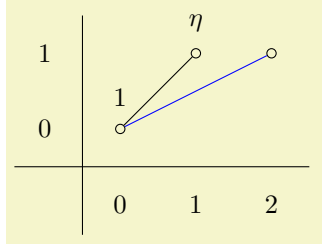


```
\begin{sseqdata}[ name = minimum width example, no axes, yscale = 0.8 ]
\class["ab"](0,0)
\class["a"](0,1)
\class(0,2)
\end{sseqdata}
\printpage[ name = minimum width example ]
\printpage[ name = minimum width example,
change classes = { blue, minimum width = width("ab") + 0.5em } ]
```

name = $\langle node\ name \rangle$

The `\class` command makes a TikZ node on appropriate pages. You can refer to this node using TikZ commands by using its coordinates. Using the `class:name` option, you can give the node a name, which you can use to refer to the class. Using names creates more readable code. The `show name` option can be used to display the names of classes. You can modify the names of classes systematically using the options `class name prefix`, `class name postfix`, and `class name handler`.

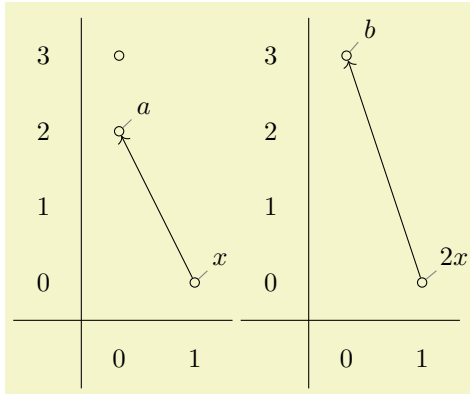
Named classes are immune to coordinate transformations. For example, in the following code, `xshift` does not apply to the nodes specified by `(id)` and `(eta)` but does apply to the coordinate specified by `(1,1)`:



```
\begin{sseqpage}[classes = { show name=above }]
\class[name = 1](0,0)
\class[name = \eta](1,1)
\class(2,1)
\structline[xshift = 1] (1) (\eta)
\structline[xshift = 1,blue] (1) (1,1)
\end{sseqpage}
```

show name = $\langle label\ options \rangle$

This option is like saying "`class name`"`\marg{label options}` if the class has a name, and does nothing if the class has no name. If the class has multiple names, only the most recent is used. This is particularly useful with class styles, . For instance, by saying `this page classes = { show name = above }` you can display names of all of the sources and targets of differentials on each page.

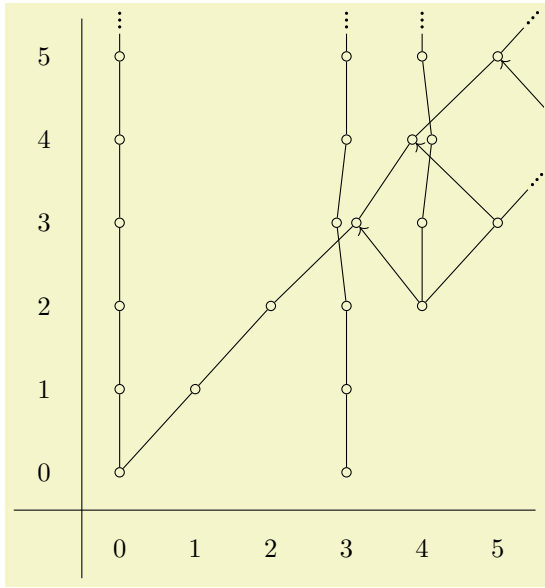


```
\begin{sseqdata}[
  name = show name example,
  this page classes = { show name = { above right, pin } }
]
\class[ name = a ](0,2)
\class[ name = b ](0,3)
\class[ name = x ](1,0)
\d2(x)(a)
\replacesource[name=2x]
\d3(x)(b)
\end{sseqdata}

\printpage[ name = show name example, page = 2]
\printpage[ name = show name example, page = 3]
```

tag = $\langle tag \rangle$

This key adds a tag to the current class. Tags are used for identifying which of multiple classes in the same position you are referring to. They are useful when you have groups of related classes and want a family of differentials connecting them. For instance:

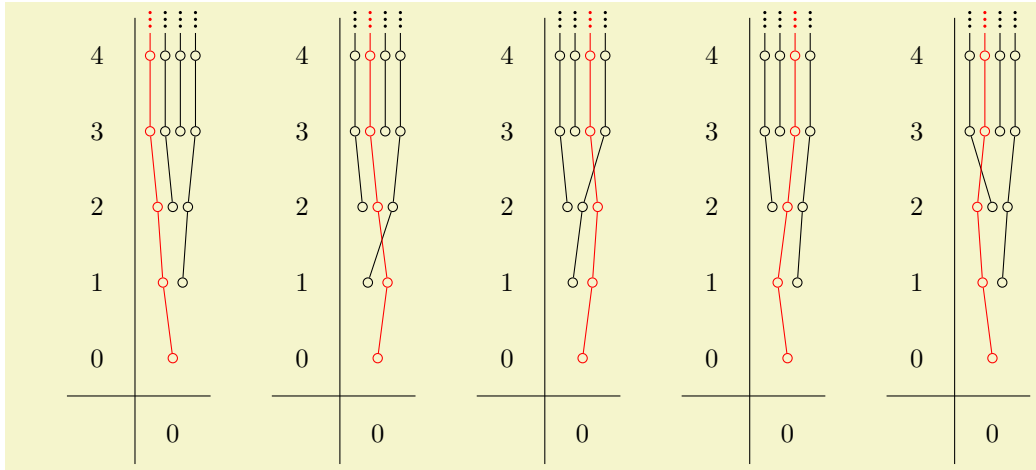


```
\DeclareSseqGroup\tower {} {
  \class(0,0)
  \foreach \i in {1,...,11} {
    \class(0,\i)
    \structline(0,\i-1,-1)(0,\i,-1)
  }
}
\NewSseqGroup\hvee {} {
  \tower(0,0)
  \foreach \i in {1,...,11} {
    \class(\i,\i)
    \structline(\i-1,\i-1,-1)(\i,\i,-1)
  }
}
\begin{sseqpage}[ degree = {-1}{1}, yscale = 1.1,
                  x range = {0}{5}, y range = {0}{5} ]
\tower(3,0)
\hvee[ tag = id ](0,0)
\hvee[ tag = h21 ](4,2)
\foreach \n in {0,...,5} {
  \d2(4+\n,2+\n,h21,id)
}
\end{sseqpage}
```

We want each differential to go from the h21 vee to the id vee, independent of which classes are in the same position of the two vees. The easy way to accomplish this is by giving tags to each of the two vees.

insert = $\langle integer \rangle$

If there are multiple classes in the same position, this option allows you to insert classes later into earlier positions. This is intended to help you put logically related classes next to each other. If the integer is positive, it inserts the class in the specified position, and if the integer is negative, it counts backwards from the end. Providing 0 is the same as omitting the option entirely. Values larger in absolute value than the total number of classes are truncated. Consider:



```

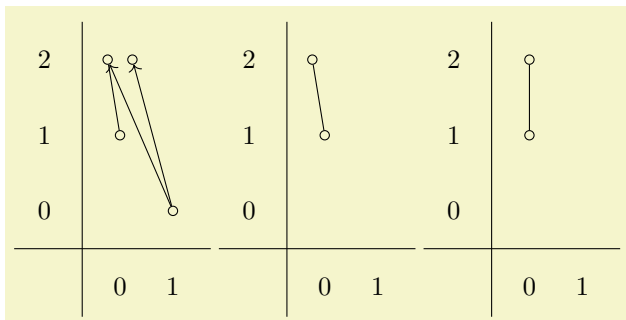
\DeclareSseqGroup \tower {} {
  \class(0,0)
  \DoUntilOutOfBounds {
    \class(\lastx,\lasty+1)
    \structline
  }
}

\begin{sseqdata}[ name = insert-example, y_range = {0}{4}, class pattern = linear ]
\tower(0,2)
\tower(0,3)
\tower(0,1)
\end{sseqdata}
\qqquad
\begin{sseqpage}[ name = insert-example ]
\tower[ red, classes = { insert = 1 } ](0,0)
\end{sseqpage}
\qqquad
\begin{sseqpage}[ name = insert-example ]
\tower[ red, classes = { insert = 2 } ](0,0)
\end{sseqpage}
\qqquad
\begin{sseqpage}[ name = insert-example ]
\tower[ red, classes = { insert = 3 } ](0,0)
\end{sseqpage}
\qqquad
\begin{sseqpage}[ name = insert-example ]
\tower[ red, classes = { insert = -2 } ](0,0)
\end{sseqpage}
\qqquad
\begin{sseqpage}[ name = insert-example ]
\tower[ red, classes = { insert = -3 } ](0,0)
\end{sseqpage}

```

`offset = {(⟨x offset⟩,⟨y offset⟩)}`

By default, a class uses the offset specified by `class pattern`. Occasionally this is undesirable. In this case, you can specify the offset for a particular class by hand. For example if the sum of two classes is hit by a differential, it looks better for the class replacing them to be centered:



```

\begin{sseqdata}[ name = offset example,
  xscale = 0.7,
  Adams grading,
  class placement transform = {scale = 1.8} ]
\class(0,1)
\class(0,2)\class(0,2)
\draw(0,1)--(0,2);
\class(1,0)
\d2(1,0,,1)
\replacetarget
\d2(1,0,,2)
\end{sseqdata}
\printpage[name = offset example, page=2]
\printpage[name = offset example, page=3]
\begin{sseqpage}[name = offset example, page=3]
\classoptions[offset = {(0,0)}](0,2)
\end{sseqpage}

```

`tooltip = ⟨text⟩`

This key generates a “tooltip” over the given class. That is, if you hover your mouse over it, a little window will popup with the tooltip text. This is particularly useful to give the coordinates or names of classes in large charts where it may be hard to tell from looking at the picture what position the class is in, or there may not be room to supply names to classes.

The tooltip is made using the `\pdftooltip` command from the `pdfcomment` package. The `pdfcomment` package generates two extra auxiliary files, so it is not included by default. In order to use the `tooltip` option, you have to use the `tooltips` package option (e.g., load `SPECTRALSEQUENCES` with

`\usepackage[tooltips]{spectralsequences}`). This cannot handle math, but it will print math expressions into \TeX input form. Not all pdf viewers will display the tooltip correctly. If this concerns you, the command `\sseqtooltip` is used to produce the tooltip, and you can redefine it as any other command that takes `\sseqtooltip{<text>}{<tooltip text>}` and produces a tooltip. For instance, on [this stack exchange post](#), there is code that supposedly produces tooltips that work with Evince. I have not tested whether it works by itself or whether it works with my package, but you could. You could potentially figure out how to get math to work in tooltips too – if you find a satisfactory method, please let me know.

Here’s an example:

1	○	○
0	○	○
<hr/>		
	0	1

```
\begin{sseqpage}[classes = {tooltip = {(\xcoord,\ycoord)}}]
\class(0,0)
\class(0,1)
\class(1,0)
\class(1,1)
\end{sseqpage}
```

There’s another example at the beginning of the section on the [class stack](#).

`page = <page>--<page max>`
`generation = <generation>--<generation max>`

These options only work in `\classoptions`. The `page` option gives a range of pages for which the options apply to. If only one page is specified, it is the minimum page and the option applies to all larger pages.

1	2	4
○	●	○

```
\begin{sseqdata}[ name = page_example, no axes,
title = |page, title style = {yshift = -0.5cm} ]
\class(0,0)
\classoptions[page = 2 -- 3, fill, blue](0,0)
\end{sseqdata}

\printpage[ name = page_example, page = 1 ] \qqquad
\printpage[ name = page_example, page = 2 ] \qqquad
\printpage[ name = page_example, page = 4 ]
```

A “generation” of a class is the interval from one call of `\class` or `\replaceclass` to the page on which it next supports or is hit by a differential. By default the `\classoptions` command adds options only to the most recent generation of the class in a `{sseqdata}` environment, or on the generation appropriate to the current page in a `{sseqpage}` environment. Using the `generation` option allows you to provide a single generation or range of generations of the class that the options should apply to. The first generation is generation 0, and the most recent generation is generation -1. Larger negative values count backwards.

3	○	3	○	3	○	3	○	3
2	○	2	○	2	○	2	○	2
1		1		1	○	1	○	1
0	○	0	●	0	○	0	○	0
<hr/>								
	0	1		0	1		0	1

```

\begin{sseqdata}[ name = page_example2, Adams grading, xscale = 0.6, yscale = 0.5 ]
\class(0,2)\class(1,0)
\d2(1,0)
\replacesource
\class(0,3)
\d3(1,0)
\replacesource
\classoptions[fill, red](1,0) % (a) applies to most recent (last) generation.
\end{sseqdata}

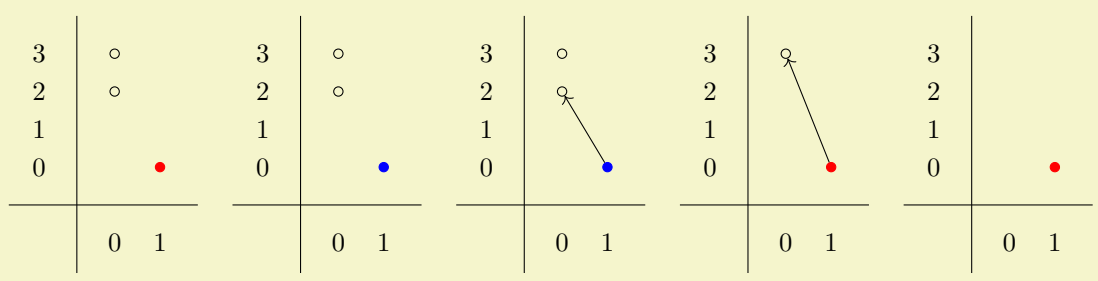
\printpage[ name = page_example2, page = 1 ] % generation 0 of (1,0), not styled
\quad
\begin{sseqpage}[ name = page_example2, page = 1, keep changes ]
\classoptions[fill, blue](1,0) % (b) applies to the generation present on page 1, that is, generation 0.
\end{sseqpage} \quad

% generation 0 of (1,0), so class is blue from (b)
\printpage[ name = page_example2, page = 2 ] \quad

% generation 1 of (1,0), class is not styled
\printpage[ name = page_example2, page = 3 ] \quad

% generation 2 of (1,0), class is red from (a)
\printpage[ name = page_example2, page = 4 ]

```



```

\begin{sseqdata}[ name = page_example2, Adams grading, update existing ]
% (c) applies to all generations, overwrites (b) and (a):
\classoptions[fill, red, generation = 0 - -1](1,0)
\end{sseqdata}

\printpage[ name = page_example2, page = 1 ] % generation 0 of (1,0), so class is red
\quad
\begin{sseqpage}[ name = page_example2, page = 1, keep changes ]
\classoptions[fill, blue](1,0) % (d) applies to the generation present on page 1, that is, generation 0.
\end{sseqpage} \quad

% generation 0 of (1,0), class is blue from (d)
\printpage[ name = page_example2, page = 2 ] \quad

% generation 1 of (1,0), class is red from (c)
\printpage[ name = page_example2, page = 3 ]
\quad
\printpage[ name = page_example2, page = 4 ] % generation 2 of (1,0), class is red from (c)

```

`\xcoord`
`\ycoord`

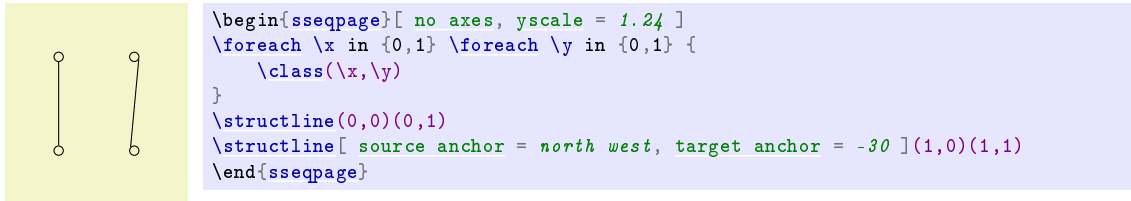
These commands represent the x and y coordinate of the current class when used in class options. The only use I have for them is in the `tooltip` option, but maybe there is some other purpose for them.

4.3 Options for `\d`, `\structline`, and `\extension`

Because the main job of the `\d`, `\structline`, and `\extension` commands is to print an edge on the appropriate pages of the spectral sequence, most TikZ options that you could apply to a TikZ “to” operator (as in `\draw (x1,y1) to (x2,y2);`) can be applied to `\d`, `\structline`, and `\extension`. Some such options are as follows:

`source anchor = $\langle anchor \rangle$`
`target anchor = $\langle anchor \rangle$`

Because you can't use the normal TikZ mechanism for specifying the source and target anchors, SPEC-TRALSEQUENCES has these two keys for `\d` and `\structline`:

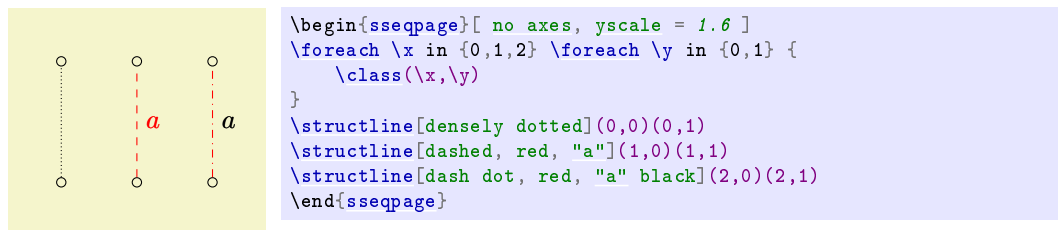


`shorten > = $\langle distance \rangle$`
`shorten < = $\langle distance \rangle$`

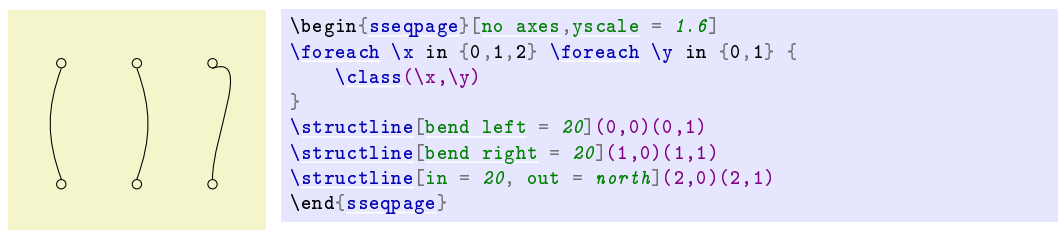
These behave exactly like the corresponding options from TikZ, shortening the end and beginning of the edge respectively. Note that you can lengthen the edge by shortening by a negative amount.

Dash patterns:

See the [TikZ manual](#) for a complete explanation of the dash pattern related options. Some examples:

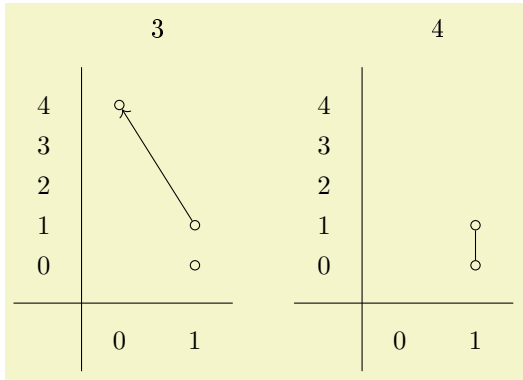


`bend left = $\langle angle \rangle$`
`bend right = $\langle angle \rangle$`
`in = $\langle anchor \rangle$`
`out = $\langle anchor \rangle$`



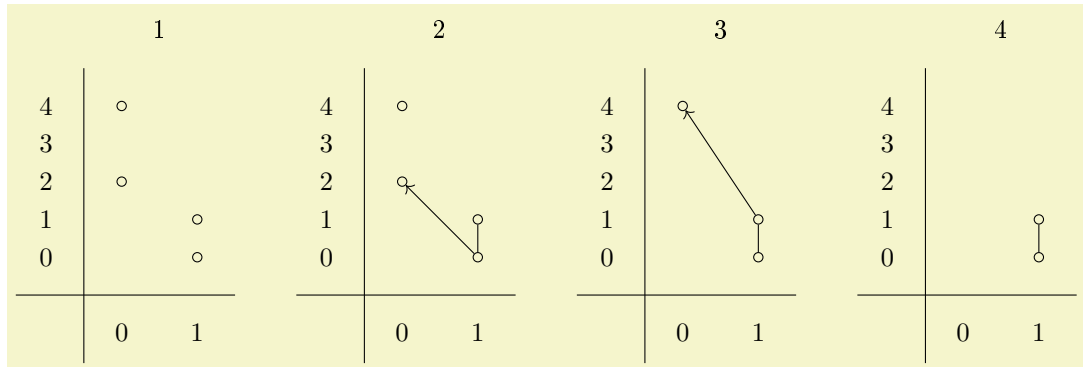
`page = $\langle page \rangle$ -- $\langle page max \rangle$`

This key is only for `\structline` and `\structlineoptions`. By default, the `\structline` command only adds a structure line starting on the page where the most recent generation of the source or target is born:



```
\begin{sseqdata}[ name = structpage example,
                  title = |page, yscale = 0.53 ]
\class(0,2)
\class(0,4)
\class(1,0)
\class(1,1)
\d2(1,0)(0,2) \replacesource
\d3(1,1)(0,4) \replacesource
\structline(1,0)(1,1)
\end{sseqdata}
\printpage[name = structpage example, page = 3]
\qquad
\printpage[name = structpage example, page = 4]
```

By specifying a page number, you can adjust which page the `\structline` starts on:



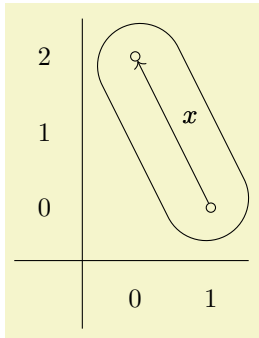
```
\begin{sseqdata}[ name = structpage example2, title = |page, yscale = 0.5 ]
\class(0,2)
\class(0,4)
\class(1,0)
\class(1,1)
\d2(1,0)(0,2) \replacesource
\d3(1,1)(0,4) \replacesource
\structline[page = 2](1,0)(1,1)
\end{sseqdata}
\printpage[ name = structpage example2, page = 1 ]
\qquad
\printpage[ name = structpage example2, page = 2 ]
\qquad
\printpage[ name = structpage example2, page = 3 ]
\qquad
\printpage[ name = structpage example2, page = 4 ]
```

Similarly, for `\structlineoptions` you can specify a minimum page on which to apply the options, or a range of pages.

4.4 Options for `\circleclass`

fit = *<coordinates or nodes>*

The `\circleclasses` command uses the `TikZ fitting library`. Sometimes it's desirable to make the resulting node fit extra things, for example a label. It doesn't necessarily end up looking great though.



```
\begin{sseqpage}[ Adams grading, axes gap = 0.7cm ]
\class(0,2)
\class(1,0)
% Fit in the label x and also a symmetric invisible label to maintain symmetry
\d["x"{name = x}, "x'"{name = x', opacity = 0}]2(1,0)
\circleclasses[fit = (x)(x'), rounded rectangle](1,0)(0,2)
\end{sseqpage}
```

rounded rectangle

You can put a shape as an option and it will change the shape of the node drawn by `\circleclasses`. Any shape will do, but I think that an `ellipse` or `rounded rectangle` are the only particularly appealing options.

`ellipse ratio = $\langle ratio \rangle$` (initially 1.2)

By default, the shape drawn by `\circleclasses` is a “newellipse” which is a custom defined shape that respects the option `ellipse ratio` which roughly controls how long and skinny versus short and fat the ellipse is. If you find that the ellipse is too long, try a larger value of this option, and conversely if it’s too fat try a smaller value. If no value is satisfactory, try out the `rounded rectangle` shape. (This is stolen from the following stack exchange answer: <https://tex.stackexchange.com/a/24621>.)

`class style`
`permanent cycle style`
`transient cycle style`
`this page class style`
`differential style`
`struct line style`

See the corresponding entry in the `TikZ` primitives section.

`page = $\langle page \rangle$ -- $\langle page max \rangle$`

By default, the ellipse will be drawn on the same set of pages that a structure line between the two classes would be drawn on. This specifies a range of pages for the ellipse to be drawn. Note that unlike with structure lines, you can instruct `\circleclasses` to draw the shape even on pages where one or both of the classes that it is fitting are dead.

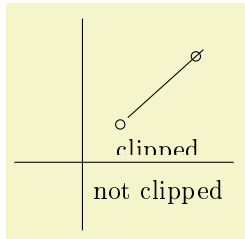
4.5 Options for `TikZ` primitives

background

This key instructs `SPECTRALSEQUENCES` to put the current `TikZ` primitive in the background. The way that the spectral sequence is printed is as follows:

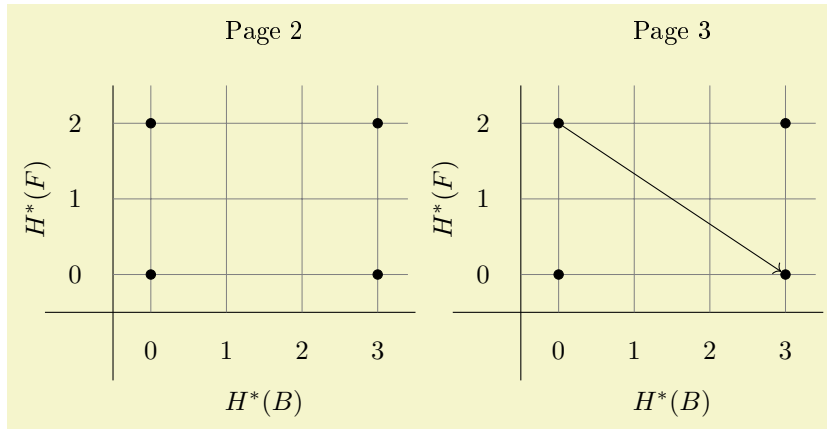
- The title, axes, axes ticks, and axes labels are printed (the appropriate steps are skipped when the `no title`, `no axes`, `no ticks`, or `no labels` keys are used or if no title or axes labels are provided).
- The `TikZ` background paths are printed.
- The clipping is inserted (unless the `no clip` key is used).
- All foreground elements (classes, differentials, structure lines, and normal `TikZ` paths) are printed.

In particular, this means that foreground `TikZ` paths can be clipped by the standard clipping, but background paths that are outside of the clipping expand the size of the `TikZ` picture.



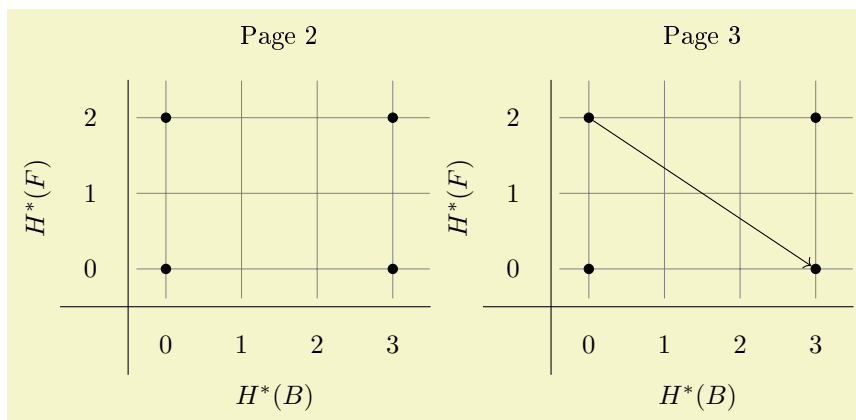
```
\begin{sseqpage}[ no ticks, yscale = 0.9, math nodes = false ]
\class(0,0)
\class(1,1)
\begin{scope}[background]
\draw(0.1,0.1)--(1.1,1.1);
\end{scope}
\node[background] at (0.5,-1) {not clipped};
\node at (0.5,-0.4) {clipped};
\end{sseqpage}
```

Here is an example where TikZ labels with the `background` key are used to add labels and a grid. Note that this styling is easier to make using the `title`, `x label`, `y label`, and `grid` options.



```
\begin{sseqdata}[ name = tikz background example, cohomological Serre grading, classes = fill ]
\begin{scope}[background]
\node at (\xmax/2,\ymax+1.2) {\textup{Page \page}};
\node at (\xmax/2,-1.7) {H*(B)};
\node[rotate = 90] at (-1.5,\ymax/2) {H*(F)};
\draw[step = 1cm, gray, very thin] (\xmin-0.5,\ymin-0.5) grid (\xmax+0.4,\ymax+0.5);
\end{scope}
\class(0,0)
\class(3,0)
\class(0,2)
\class(3,2)
\d3(0,2)
\end{sseqdata}
\printpage[name = tikz background example, page = 2]
\printpage[name = tikz background example, page = 3]
```

For this particular use case, it's probably better to use `title`, `x label`, and `y label`:



```

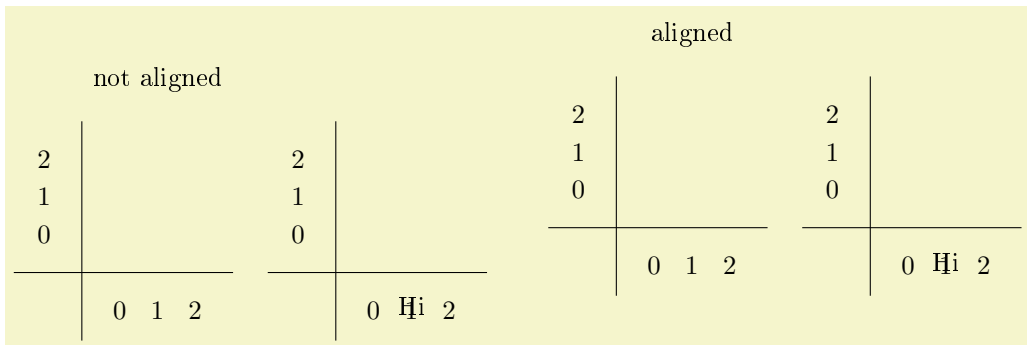
\begin{sseqdata}[ name = tikz background example2, cohomological Serre grading, classes = fill,
                  grid = go, title = { Page \page }, x label = {  $H^*(B)$  }, y label = {  $H^*(F)$  } ]
\class(0,0)
\class(3,0)
\class(0,2)
\class(3,2)
\d3(0,2)
\end{sseqdata}
\printpage[name = tikz background example2, page = 2]
\printpage[name = tikz background example2, page = 3]

```

But if you need more flexible labeling, you'll likely want to use tikz primitives with `background`. See `example_KF3.tex` for an instance where this key is useful.

One useful tip is that you can ensure consistent bounding boxes between different diagrams using

`\path[background] (smallest x, smallest y) -- (largest x, largest y);`



```

\begin{sseqdata}[ name = boundingboxes, x range = {0}{2}, y range = {0}{2}, scale = 0.5 ]
\end{sseqdata}
\printpage[ name = boundingboxes, title = not aligned ]
\quad
\printpage[ name = boundingboxes, x label = Hi ]
\quad
\begin{sseqpage}[ name = boundingboxes, keep changes, title = aligned ]
\path[background] (\xmin,\ymin-4) -- (\xmax,\ymax+2);
\end{sseqpage}
\quad
\printpage[ name = boundingboxes, x label = Hi, title = {} ]

```

`page constraint = <predicate>`
`page constraint or = <predicate>`

This places a constraint on the pages in which the TikZ primitive is printed. This predicate should look something like `(\page <= 4) && (\page >= 3)`. The predicate is anded together with any previous predicates, so that you can use this as an option for a `{scope}` and again for the individual TikZ primitive.

`\isalive(<coordinate>)`
`\isalive{(<coordinate 1>)...(<coordinate n>)}`

This command can only be used with `page constraint`. Saying

`page constraint = {\pars{\meta{x},\meta{y}\opt{,}\oarg{index}}}`

will print the TikZ primitive only on pages where the specified class is alive. Saying

`page constraint = {\isalive(\meta{coordinate 1})...(\meta{coordinate n})}`

is equivalent to

`page constraint = {\isalive\pararg{coordinate 1} &&...&& \isalive\pararg{coordinate n}}`

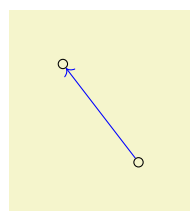
Writing

```
\draw[page constraint = {\isalive(1,0)(2,2)}](1,0)--(2,2);
```

is the same as `\structline(1,0)(2,2)`, except that you can't later use `\structlineoptions` on it (and it won't have the `struct lines` style applied).

```
c lass style
permanent cycle style
transient cycle style
this page class style
differential style
struct line style
```

These classes apply the styling of the corresponding element to your *TikZ* commands.



```
\begin{sseqpage}[ differentials = blue, yscale = 0.65, no axes ]
\class(0,2)
\class(1,0)
% This will be styled as if it were a differential
\draw[differential style] (1,0) -- (0,2);
\end{sseqpage}
```

See `\getdtarget` for a more natural example.

5 Miscellaneous Commands

5.1 Settings

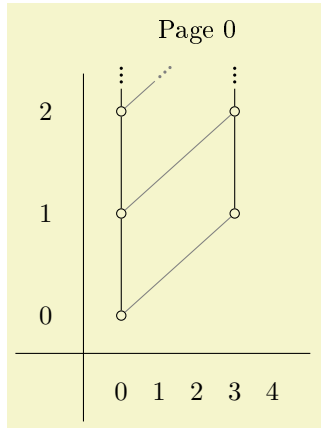
```
\sseqset{⟨keys⟩}
```

The `\sseqset` command is for adjusting the global options for all spectral sequences in the current scope, or for applying options to the rest of the current spectral sequence. For instance, if most of the spectral sequences in the current document are going to be Adams graded, you can say `\sseqset{Adams grading}` and all future spectral sequences in the current scope will have Adams grading (unless you specify a different grading explicitly). As another example, `\sseqset{no axes}` will suppress axes from spectral sequences in the current scope. Note that defaults only apply to new `{sseqdata}` environments or to unnamed `{sseqpage}` environments; they won't apply to existing spectral sequences.

You can also use `\sseqset` to create styles to be used in spectral sequences.

```
.global sseq style = ⟨keys⟩
.global sseq append style = ⟨keys⟩
.sseq style = ⟨keys⟩
.sseq append style = ⟨keys⟩
```

These handlers create reusable styles to be used in spectral sequences. If this style is a set of global options, then use the `.global sseq style` handler, whereas if it is supposed to be applied to individual features (classes, differentials, structure lines, circle classes, and *tikz* primitives) then use the `.sseq style` handler.



```
\sseqset{
  mysseq/.global sseq style = {
    Adams grading, title = Page |page,
    x range = {0}{4}, y range = {0}{2},
    xscale = 0.5, yscale = 1.35
  },
  htwostruct/.sseq style = { gray, thin }
}
\begin{sseqpage}[ mysseq ]
\class(0,0) \class(0,1) \class(0,2) \class(0,3)
\class(3,1) \class(3,2) \class(3,3)
\structline(0,0)(0,1) \structline(0,1)(0,2)
\structline(0,2)(0,3)
\structline(3,1)(3,2) \structline(3,2)(3,3)
\structline[htwostruct](0,0)(3,1)
\structline[htwostruct](0,1)(3,2)
\structline[htwostruct](0,2)(3,3)
\end{sseqpage}
```

`\SseqErrorToWarning`*(error-name)*

Turns the error with the given name into a warning. An error message will start by saying `spectralsequences error: "error-name"`. This is the name you need to put into this command.

```
\begin{quiet}
  environment contents
\end{quiet}
```

This environment quiets error messages that occur inside of it. `SPECTRALSEQUENCES` is pretty good at error recovery, and so most of commands will fail gracefully and do nothing if their preconditions aren't met. If there are any parsing errors in the body of the `{quiet}` environment, prepare to see low level internal error messages. You might also run into bugs in `SPECTRALSEQUENCES` – the error recovery code hasn't been that carefully tested. If you do get low level error messages, remember to comment out the `{quiet}` environment before trying to debug.

This is particularly useful for code reuse commands. Sometimes there is a source of long differentials that only applies to classes that haven't already supported shorter differentials. Sometimes there should be a structure line if a certain class exists, but it might not exist. In these cases, the `{quiet}` environment will help you out. See also `\DrawIfValidDifferential`, which is a variant of `\d` that behaves as if it were inside a `{quiet}` environment.

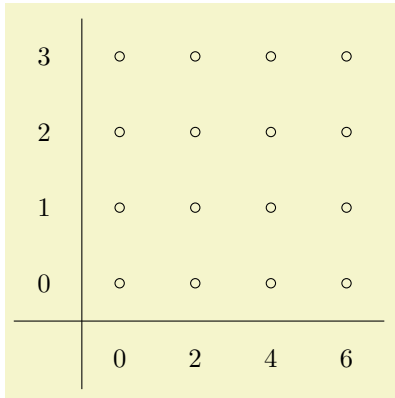
5.2 Code reuse commands

`\foreach`

This command is from `TikZ` and works in pretty much the same way in `SPECTRALSEQUENCES`, though with slightly better variants. The `\foreach` command is very flexible and has a lot of variants. The basic usage is `\foreach \x in {\meta{xmin},...,\meta{xmax}} \marg{loop body}` which will execute `\meta{loop body}` with `\x` set to each value between `<xmin>` and `<xmax>` inclusive. If you want a step greater than 1, try

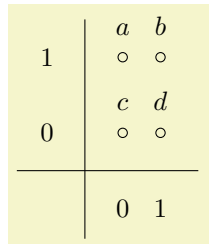
```
\foreach \x in {\meta{xmin},\meta{xmin}+\meta{xstep},...,\meta{xmax}}\marg{loop body}.
```

If you need to do multiple loops with a common body, you can just stack the `\foreach` commands:



```
\begin{sseqpage}[ xscale = 0.5, x tick step = 2 ]
\foreach \x in {0,2,...,6}
\foreach \y in {0,...,3}{
  \class(\x,\y)
}
\end{sseqpage}
```

You can also loop through tuples, for instance:



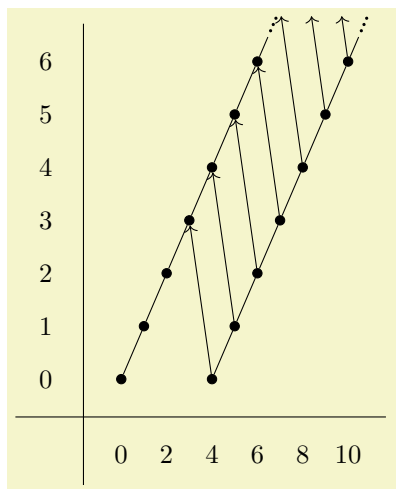
```
\begin{sseqpage}[ xscale = 0.5 ]
\foreach \x/\y/\label in {0/1/a,1/1/b,0/0/c,1/0/d}{
  \class["\label" above](\x,\y)
}
\end{sseqpage}
```

See the last example for `normalize monomial` for a better example of this usage.

There are tons of other things you can do with `\foreach`, though I haven't yet found need for them in combination with `SPECTRALSEQUENCES`. See the [TikZ manual](#) for more details.

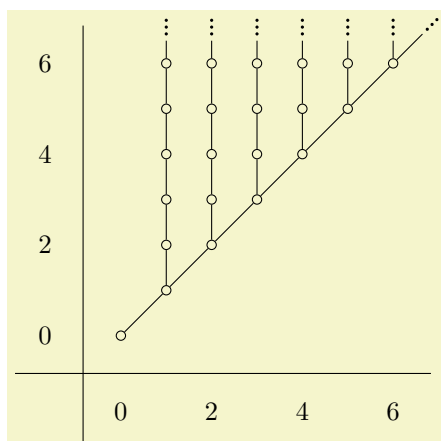
`\Do{<iterations>}{<loop body>}`
`\DoUntilOutOfBounds{<loop body>}`
`\DoUntilOutOfBoundsThenNMore{<extra iterations>}{<loop body>}`
`\iteration`

The one use case that `\foreach` doesn't cover all that well is if you want the loop to always repeat until the features you are drawing go off the page. This is what `\DoUntilOutOfBounds` and `\DoUntilOutOfBoundsThenNMore` are for. These help ensure that if you change the range of your chart, infinite families will automatically be drawn correctly without the need to adjust a bunch of loop bounds. The purpose of `\DoUntilOutOfBoundsThenNMore` is for towers that are receiving a differential. If your spectral sequence is Adams graded, and a tower is receiving a d_r differential from another tower, you should use `\DoUntilOutOfBoundsThenNMore{r}`:



```
\begin{sseqpage}[
  Adams grading, classes = fill,
  x range = {0}{10}, y range = {0}{6},
  x tick step = 2,
  xscale = 0.3, yscale = 0.7,
  run off differentials = {->}
]
\class(0,0)
\DoUntilOutOfBoundsThenNMore{3}{
  \class(\lastx+1,\lasty+1)
  \structline
}
\class(4,0)
\d3
\DoUntilOutOfBounds{
  \class(\lastx+1,\lasty+1)
  \structline
  \d3
}
\end{sseqpage}
```

You can also nest `\DoUntilOutOfBounds` reasonably:



```
\begin{sseqpage}[
  x range = {0}{6}, y range = {0}{6},
  tick step = 2,
  scale = 0.6
]
\class(0,0)
\DoUntilOutOfBounds{
  \class(\lastx+1,\lasty+1)
  \structline
  \DoUntilOutOfBounds{
    \class(\lastx,\lasty+1)
    \structline
  }
}
\end{sseqpage}
```

One important difference between `\foreach` and the `\Do` family of commands is that `\Do` has no effect on the stack. This is in order to ensure that they nest properly.

Note that if you are using these commands and you are planning to draw several pictures of the chart with restricted range, you need to specify a range for the `{sseqdata}` that contains all of the ranges of pages that you want to draw. If you then want to set a smaller default range, specify the smaller range the first time you use `{sseqpage}` or `\printpage` to draw the spectral sequence, and include the `keep changes` key.

The `\Do` command is less general than `\foreach`; the purpose is to provide a syntax for stack-based looping that is similar to `\DoUntilOutOfBounds` but with a fixed range. So `\Do{n}\margin{loop body}` repeats `<loop body>` `n` times. The assumption is that the loop body draws something relative to the position of the `\lastclass`.

If you need to know how many iterations one of these three commands has gone through, this is stored in the variable `\iteration`.

\NewSseqCommand`\<command>\{<argspec>\}\{<body>\}`

\DeclareSseqCommand`\<command>\{<argspec>\}\{<body>\}`

The `xparse` package provides these very powerful commands for defining macros. They are used internally to the `SPECTRALSEQUENCES` package to define `\class`, `\d`, etc. To help you create variants of these commands, I will record here the argument specifications for each of them. See the `xparse` manual for a better explanation and more information.

To make a command like `\class`, you can use the argument specification `0{r()}`. The argument type `0{<default>}` stands for a bracket delimited optional argument with default value `<default>`. In this case, we've specified the default to be empty. `r()` stands for a "required" argument delimited by `(` and `)`. In the command definition, access the optional argument with `#1` and the coordinate with `#2`.

`#1 = {key = value}; #2 = {x,y}`

`#1 = {}; #2 = {1,2,3}`

```
\DeclareDocumentCommand\demo{ 0{ } r() }
{
  \#1 = \textcolor{purple}{\{ \#1 \}};
  \#2 = \textcolor{purple}{\{ \#2 \}} }
\hbox{\demo[key = value](x,y)}
\bigskip
\hbox{\demo(1,2,3)}
```

If you want to separate out the coordinates into different arguments, you can use `0{u(u,u)}`. The argument type `u` stands for "until" and scans up until the next instance of the given character. So in this case, `#1` is of argument type `0` which is an option list, `#2` corresponds to the `u(` which is a throw-away argument, then `#3` corresponds to `u`, and contains the `x` coordinate, and `#4` corresponds to `u)` and contains the `y` coordinate. Note however that this will not match balanced parenthetical expressions.

```
#1 = {hi}; #3 = {x}; #4 = {y}

#1 = {}; #3 = {1}; #4 = {2}

#1 = {}; #3 = {(1+1)*2}; #4 = {2}

#1 = {}; #3 = {1}; #4 = {(1+1)*2}
```

```
\DeclareDocumentCommand\demo{ 0{} u( u, u) }
{ \#1 = \textcolor{purple}{\{\#1\}};
  \#3 = \textcolor{purple}{\{\#3\}};
  \#4 = \textcolor{purple}{\{\#4\}} }
\hbox{\demo[hi](x,y)}
\bigskip
\hbox{\demo(1,2)}
\bigskip
\hbox{\demo((1+1)*2,2)}
\bigskip
\hbox{\demo(1,(1+1)*2)} % uh-oh -- *2) is left off!
```

You can specify an optional argument delimited by parentheses using `d()`. Use the commands `\IfNoValueTF`, `\IfNoValueT`, and `\IfNoValueF` to test whether the user provided a value.

```
#1 = {hi}; #2 = {x,y}

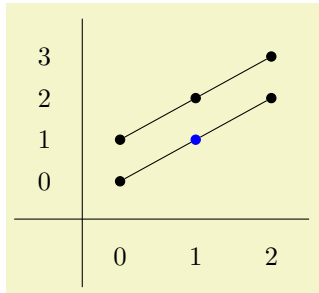
#1 = {options}; #2 = {no value}

#1 = {}; #2 = {1,2}

#1 = {}; #2 = {no value}
```

```
\DeclareDocumentCommand\demo{ 0{} d() } {
  \#1 = \textcolor{purple}{\{\#1\}};
  \#2 = \textcolor{purple}{\{
    \IfNoValueTF{#2}{no value}{#2}\}}
}
\hbox{\demo[hi](x,y)}
\bigskip
\hbox{\demo[options]}
\bigskip
\hbox{\demo(1,2)}
\bigskip
\hbox{\demo}
```

An example where this is actually useful:



```
\DeclareSseqCommand\etaclass{ 0{} d() } {
  \IfNoValueF{#2}{ \pushstack{#2} }
  \class[#1] ( \lastx+1, \lasty+1)
  \structline
}
\begin{sseqpage}[ classes = fill, yscale = 0.55 ]
\class(0,0)
\class(0,1)
\etaclass\etaclass
\etaclass[blue](0,0)\etaclass
\end{sseqpage}
```

The `\structline` and `\changeclases` commands have argument specification `0{}r()r()`.

```
#1 = {hi}; #2 = {x,y}; #3 = {x',y'}

#1 = {}; #2 = {x,y,2}; #3 = {x',y',-1}
```

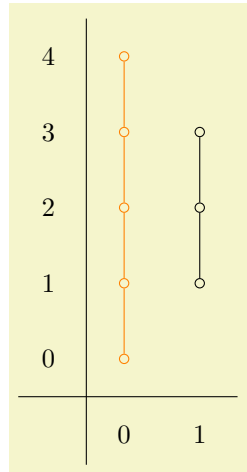
```
\DeclareDocumentCommand\demo{ 0{} r() r() }
{ \#1 = \textcolor{purple}{\{\#1\}};
  \#2 = \textcolor{purple}{\{\#2\}};
  \#3 = \textcolor{purple}{\{\#3\}} }
\hbox{\demo[hi](x,y)(x',y')}
\bigskip
\hbox{\demo(x,y,2)(x',y',-1)}
```

`\NewSseqGroup\<command>\<argspec>\<body>`
`\DeclareSseqGroup\<command>\<argspec>\<body>`

These are similar to `\NewSseqCommand` and `\DeclareSseqCommand` except that the commands defined take an optional square bracket delimited options list, followed by an optional parenthesis delimited `(x,y)` pair and wraps the body of the code in

```
\begin{scope}[xshift = \meta{xcoord}, yshift = \meta{ycoord}, options].
```

For instance:



```
\DeclareSseqGroup\tower {m} {
  \class(0,0)
  \foreach \n in {1,...,#1} {
    \class(0,\n)
    \structline(0,\n-1)(0,\n)
  }
}
\begin{sseqpage}
\tower[orange]{4}
\tower(1,1){2}
\end{sseqpage}
```

Here we define a command which takes an options list, a coordinate, and a single mandatory argument. The options are applied to all the classes and structlines, the coordinate shifts the origin, and the mandatory argument is the length of the tower.

\SseqCopyPage $\langle command \rangle \{ \langle spectral\ sequence\ name \rangle \} \{ \langle page \rangle \}$

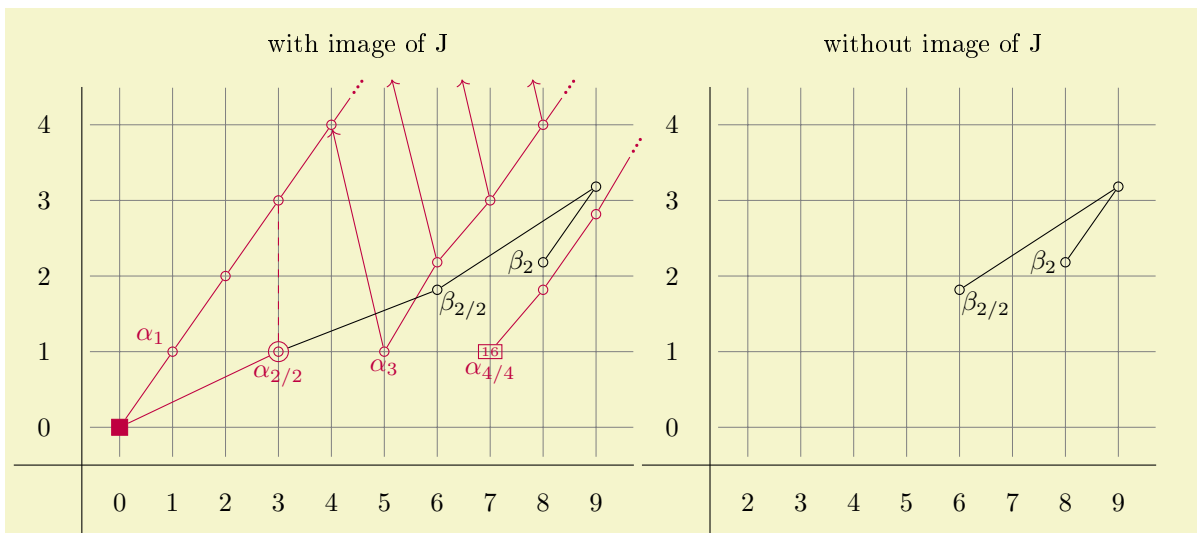
This defines $\langle command \rangle$ to print all of the classes and structlines on page \meta{page} of the spectral sequence named $\meta{spectral\ sequence\ name}$, throwing out differentials and tikz primitives. The resulting command has syntax as if you had used \DeclareSseqGroup with no extra arguments – that is, it takes an optional options list, which are passed to a scope, and an optional coordinate pair.

5.3 Families

SPECTRALSEQUENCES has a concept of “class families” that can be drawn or suppressed as a group, and that can have styling options applied to them as a group.

\SseqNewFamily $\{ \langle family\ name \rangle \}$

This makes a new family with name $\langle family\ name \rangle$. The effect of this is to define global options $\meta{family\ name}$ style which apply options to all classes in the family, $\draw \meta{family\ name}$ which causes the family to be drawn, and $\no \meta{family\ name}$ which suppresses the family (drawing is the default behavior of course). It also defines an option $\meta{family\ name}$ which puts a feature into the family – so it can be used as an option to any of the normal commands like \class , \d , \structline , etc and to a scope.



```

\SeqNewFamily{imJ}
\begin{sseqdata}[
  name=ANSS-S_2, Adams grading,
  class labels = {below}, label distance = -1pt,
  class placement transform = {rotate = 90, scale = 2},
  differentials = blue,
  x range = {0}{9}, y range = {0}{4},
  xscale = 0.7, grid = go
]
% Image of J classes
\begin{scope}[imJ]
\class[rectangle,fill,inner sep=3pt](0,0)
\DoUntilOutOfBoundsThenNMore{3}{\etaclass}
\classoptions["\alpha_1" above left](1,1)
\class["\alpha_{2/2}", circled = 2](3,1)
\structline(0,0) \structline[dashed](3,3)
\class["\alpha_3"](5,1) \d3\relax
\DoUntilOutOfBounds{\etaclass \d3 }
\class["\alpha_{4/4}", rectangle, "16" {font = \tiny, inside, inner sep = 1pt}](7,1)
\DoUntilOutOfBoundsThenNMore{3}{\etaclass}
\end{scope}

% Other classes
\class["\beta_{2/2}" {below right=-4pt}, insert = 1](6,2)
\structline(3,1)
\class["\beta_{2}" left](8,2)
\etaclass\structline(6,2,2) % This is a *nu
\end{sseqdata}

\printpage[ name = ANSS-S_2, title = with image of J, imJ style = purple]
\printpage[ name = ANSS-S_2, no imJ, x range={2}{9}, title = without image of J]

```

5.4 Utilities

\SeqParseInt $\langle\langle macro \rangle\rangle\langle\langle integer expression \rangle\rangle$

Stores the result of evaluating an integer expression into $\langle\langle macro \rangle\rangle$. An integer expression consists of +, -, *, /, parentheses, and macros that expand to more of the same. The exact rules regarding what is a valid expression are pretty much what you would expect. Note that juxtaposition is a syntax error, not multiplication, so 2(1+1) is invalid, you must say 2*(1+1).

\SeqIfEmptyTF $\langle\langle expression \rangle\rangle\langle\langle true code \rangle\rangle\langle\langle false code \rangle\rangle$

\SeqIfEmptyT $\langle\langle expression \rangle\rangle\langle\langle true code \rangle\rangle$

\SeqIfEmptyF $\langle\langle expression \rangle\rangle\langle\langle false code \rangle\rangle$

This tests if an expression is the empty expression. This is mainly useful for giving systematic labels to things.

\IfExistsTF $\langle\langle page \rangle\rangle\langle\langle coordinate \rangle\rangle\langle\langle true code \rangle\rangle\langle\langle false code \rangle\rangle$

\IfExistsT $\langle\langle page \rangle\rangle\langle\langle coordinate \rangle\rangle\langle\langle true code \rangle\rangle$

\IfExistsF $\langle\langle page \rangle\rangle\langle\langle coordinate \rangle\rangle\langle\langle false code \rangle\rangle$

Test whether a class of the given description exists. The description can be any valid coordinate, including a named coordinate. If the coordinate is not valid, this will return false.

\IfAliveTF $\langle\langle page \rangle\rangle\langle\langle coordinate \rangle\rangle\langle\langle true code \rangle\rangle\langle\langle false code \rangle\rangle$

\IfAliveT $\langle\langle page \rangle\rangle\langle\langle coordinate \rangle\rangle\langle\langle true code \rangle\rangle$

\IfAliveF $\langle\langle page \rangle\rangle\langle\langle coordinate \rangle\rangle\langle\langle false code \rangle\rangle$

Test whether a class is alive on the given page. If the class doesn't exist, this will return false.

\IfOutOfBoundsTF $\langle\langle coordinate \rangle\rangle\langle\langle true code \rangle\rangle\langle\langle false code \rangle\rangle$

\IfOutOfBoundsT $\langle\langle coordinate \rangle\rangle\langle\langle true code \rangle\rangle$

```

\IfOutOfBoundsF(<coordinate>){<false code>}
\IfInBoundsTF(<coordinate>){<true code>}{<false code>}
\IfInBoundsT(<coordinate>){<true code>}
\IfInBoundsF(<coordinate>){<false code>}

```

Test whether a class is in bounds or out of bounds. If no such class exists, returns false.

```

\IfValidDifferentialTF<page>(<coordinate>)(<target coordinate>){<true code>}{<false code>}
\IfValidDifferentialT<page>(<coordinate>)(<target coordinate>){<true code>}
\IfValidDifferentialF<page>(<coordinate>)(<target coordinate>){<false code>}
\DrawIfValidDifferentialTF[<options>]<page>(<coordinate>)(<target coordinate>){<true code>}{<false
code>}
\DrawIfValidDifferentialT[<options>]<page>(<coordinate>)(<target coordinate>){<true code>}
\DrawIfValidDifferentialF[<options>]<page>(<coordinate>)(<target coordinate>){<false code>}
\DrawIfValidDifferential[<options>]<page>(<coordinate>)(<target coordinate>){<false code>}

```

Test whether a differential is valid. There are many possible reasons for a differential to not be valid – the source or target class could not exist, the target could not be in the correct grading, etc. If any error would be generated if you supplied the same arguments to `\d`, this tests false. Otherwise, this tests true. The command `\DrawIfValidDifferential` draws the differential if it is valid, whereas `\DrawIfValidDifferentialT` draws the differential and executes the true code if it is valid, `\DrawIfValidDifferentialF` draws the differential or executes false code if it isn't valid, etc.

`\SseqNormalizeMonomial`

This command simplifies a monomial by combining like variables and removing bases that are raised to the 0th power, removing exponents of 1, removing 1's, and replacing the empty monomial with 1. The variables are sorted by first occurrence, use `\SseqNormalizeMonomialSetVariables` to set a different sort order. It outputs its result into `\result`. This command is specifically meant to be used as a value for class label handler or class name handler. See the example in the documentation for class label handler for a realistic example. The exponents must be integers or else it will misbehave.

$1, \quad x^2y, \quad x_2^4$	<pre> \SseqNormalizeMonomial{x^0y^0} \$\result\$, \quad \SseqNormalizeMonomial{x^3yx^{-1}z^0} \$\result\$, \quad \SseqNormalizeMonomial{1x_2^2x^2_2} \$\result\$ </pre>
------------------------------	---

`\SseqNormalizeMonomialSetVariables`

This command takes an undelimited list sets up `\SseqNormalizeMonomial` to sort the variables in the input in the specified order. This is useful for consistency, and particularly important for when you use it to name classes. For example:

$ab, \quad bx_1, \quad abx_1x_2x_3$	<pre> % We always want a first, then b, then x_1, then x_2, then x_3 \SseqNormalizeMonomialSetVariables{abx_1x_2x_3} \SseqNormalizeMonomial{ba} \$\result\$, \quad \SseqNormalizeMonomial{bx_1} \$\result\$, \quad \SseqNormalizeMonomial{abx_2x_3x_1} \$\result\$ </pre>
-------------------------------------	---

`\SseqAHSSNameHandler`

This command expects an argument of the form `\meta{monomial}[\meta{integer expression}]` and defines `\result` to be $m[n]$ where m is the result of applying `\SseqNormalizeMonomial` to the monomial, and n is the result of evaluation of the integer expression plus any `yshift` that is present. This is intended for use with in drawing Atiyah Hirzebruch spectral sequences. See `class name handler` for an example of the calculation of $KO^*\mathbb{RP}^8$.

5.5 Coordinate parsers and related

`\parsecoordinate``\<macro>(<coordinate>)`

This command parses the coordinate and puts the triple (x,y,n) into $\langle macro \rangle$. It also puts the components of the coordinate into macros, for instance if $\langle macro \rangle$ is $\langle coord \rangle$, then $\langle xcoord \rangle$ will contain the x coordinate, $\langle ycoord \rangle$ will contain the y coordinate, and $\langle ncoord \rangle$ will contain the index. The coordinate can be anything that would be valid to use in a differential or structure line; this is the macro that is used internally to handle coordinates.

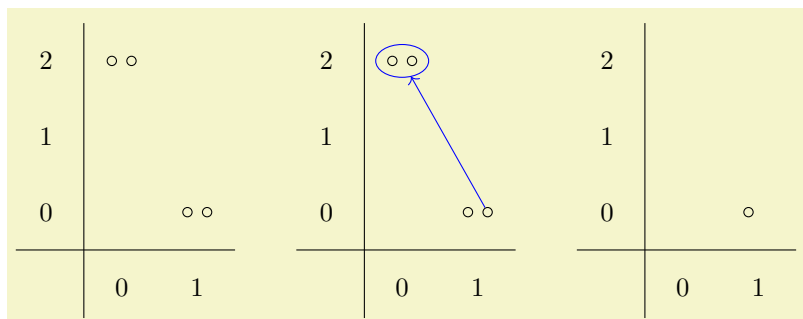
$\backslash getdtarget \langle macro \rangle \{ \langle page \rangle \} \{ \langle source coordinate \rangle \}$

Sets $\langle macro \rangle$ equal to the coordinates of the target position of a length $\langle page \rangle$ differential starting at $\langle source coordinate \rangle$. This helps to make commands that draw fancy differentials.

$\backslash parsedifferential \langle page \rangle (\langle differential coordinate \rangle) (\langle differential target \rangle)$

This has the same weird syntax of $\backslash d$, except that you are required to put braces around the page (if it has multiple digits) and you are required to provide at least one coordinate (you have to say $\backslash parsedifferential \{2\} (\backslash lastclass)$ for instance). This command is similar in effect to saying both $\backslash parsecoordinate \backslash source \backslash meta \{ source coordinate \}$ and $\backslash parsecoordinate \backslash target \backslash meta \{ target coordinate \}$ but it handles determination of the target coordinate for you.

For instance, consider the following example, suggested by Catherine Ray:



```
% 0{} U( r() ) is the arg-spec for \d, 0{} U( r() ) mm looks like \d with one extra mandatory arguments
\NewSseqCommand\twods{ 0{} U( r() ) d() m }{
  \parsedifferential{#2}{#3}{#4} % Store the target position in \target
  \nameclass{source}{\sourcecoord,\source} % give names to the three classes
  \nameclass{target1}{\targetcoord,\ntarget}
  \nameclass{target2}{\targetcoord,#5}
}
%
\circleclasses[ differential style, name path = circ,
  #1, page = #2-#2 ]
  (target1)(target2) % Circle the classes, use differential style
%
% record source and targets as hit.
\kill#2(source) \kill#2(target1) \kill#2(target2)
%
\path(target1)--(target2)
  coordinate[midway](midpt); % put a coordinate in the center of the two classes
\path[name path = lin] (source) -- (midpt); % save path from start to midpoint
%
% draw line in "differential style" from start to intersection point of circ and lin
\draw[ differential style, #1, page constraint= { \page == #2 },
  name intersections = { of = circ and lin } ]
  (source) -- (intersection-1);
}
\begin{sseqdata}[ name = cathe\$, Adams grading, differentials = { blue } ]
\class(0,2)\class(0,2)
\class(1,0)\class(1,0)
\twods2(1,0,-1,1){2}
\end{sseqdata}
\printpage[ name = cathe\$, page = 1 ]
\qqquad
\printpage[ name = cathe\$, page = 2 ]
\qqquad
\printpage[ name = cathe\$, page = 3 ]
```

$\backslash nameclass \{ \langle name \rangle \} \{ \langle coordinate \rangle \}$

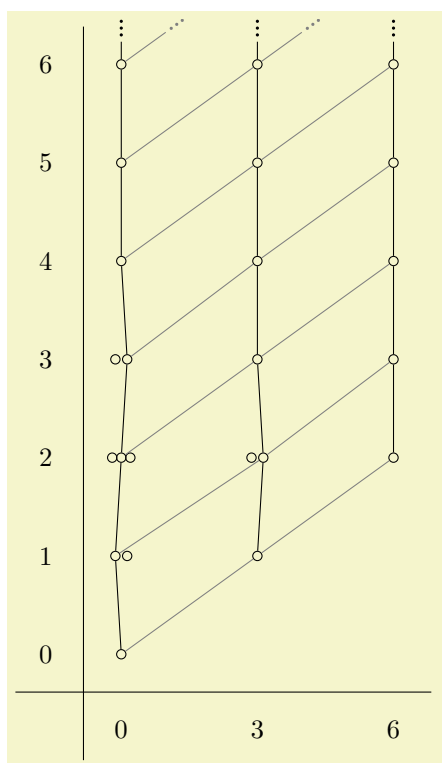
The `\nameclass` command gives a name to a class. It's similar to saying `\doptions[name = <name>]`, but faster. It's also similar to saying `\pushstack(<coordinate>)`. Giving temporary names to coordinates that you are going to use repeatedly makes the code easier to read and is faster (though this only matters in very large charts). See `\getdtarget` for an example.

`\tagclass{<tag>}{<coordinate>}`

The `\tagclass` command gives a tag to a class. It's similar to saying `\doptions[tag = <name>]`, but faster. See `example_tmfmayss.tex` for a use case for this.

`\gettag{<macro>}{<coordinate>}`

The `\gettag` command finds the most recent tag applied to the coordinate and stores it into `\cmdarg{macro}`. This is useful for connecting groups of nodes. For example, consider the following code, inspired by `example_tmfmayss.tex`.



```
\DeclareSseqCommand \tower { 0{} } {
  \begin{scope}[#1]
    \foreach\i in {1,...,7}{
      \class(\lastx,\lasty+1)
      \structline
    }
    \end{scope}
    \restorestack
}
\DeclareSseqCommand \htwtower { 0{} d() } {
  \IfNoValueF{#2}{
    \pushstack{#2}
  }
  \begin{scope}[#1]
    \gettag\thetag(\lastclass)
    \class(\lastx+3,\lasty+1)
    \structline[gray]
    \savestack
    \foreach\i in {1,...,7}{
      \class(\lastx,\lasty+1)
      \structline
      \structline[gray](\lastx-3,\lasty-1,\thetag)
    }
    \restorestack
  \end{scope}
}
\begin{sseqpage}[ y range = {0}{6}, x tick step = 3,
  xscale=0.6, yscale=1.3 ]
\class(0,2) \class(0,3) \class(3,2)
\class[tag = h_0^i](0,0)
\tower[tag = h_0^i]
\class(0,2) \class(0,1)

\htwtower[tag = h_2 h_0^i](0,0)
\htwtower[tag = h_2 h_0^i]
\end{sseqpage}
```

5.6 The class stack

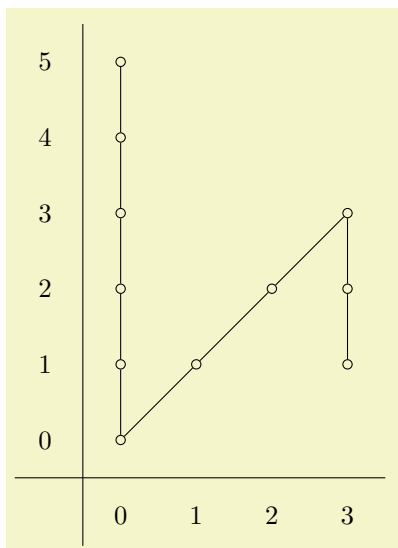
The class stack is a linked list of the classes in the order that they are produced that `SPECTRALSEQUENCES` maintains. Whenever you use the `\class` or `\replaceclass` command, the class you added is pushed onto the stack.

The following commands are used to access the stack:

`\lastx<n>`
`\lasty<n>`

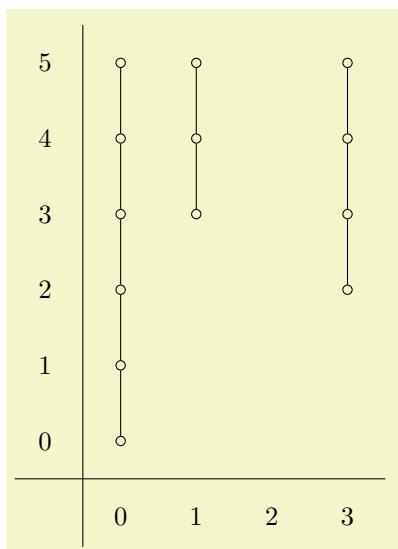
`\lastclass⟨n⟩`

The commands `\lastx` and `\lasty` evaluate to the x and y position, respectively, of the n th class on the stack. If $n = 0$ you can leave it off. The command `\lastclass` evaluates to the coordinate of the most recent class on the stack. This is useful for writing turtle-style code:



```
\DeclareSseqCommand \etaclass {}{
  \class(\lastx+1,\lasty+1)
  \structline
}
\DeclareSseqCommand \divtwoclass {}{
  \class(\lastx,\lasty-1)
  \structline
}
\begin{sseqpage}
\class(0,0)
\savestack
\foreach \y in { 1,...,5 }{
  \class(0,\y)
  \structline
}
\restorestack
\etaclass\etaclass\etaclass
\divtwoclass\divtwoclass
\end{sseqpage}
```

You can use `\lastx` and `\lasty` in other contexts than in the body of a `SPECTRALSEQUENCES` command, most notably inside `\SseqParseInt` (they also work correctly when used inside of `\pgfmathparse` and its siblings). For instance, consider the following tower command:



```
\DeclareSseqCommand \tower { } {
  \savestack
  \SseqParseInt\numclasses{\ymax-\lasty0}
  \foreach \n in {1,...,\numclasses}{
    \class(\lastx,\lasty+1)
    \structline
  }
  \restorestack
}
\begin{sseqpage}[y range = {0}{5}]
\class(0,0)\tower
\class(1,3)\tower
\class(3,2)\tower
\end{sseqpage}
```

`\pushstack(⟨coordinate⟩)`

This adds a class to the top of the stack. The coordinate is specified using the same syntax as a coordinate for `\structline` or `\replaceclass`.

`\savestack`

`\restorestack`

This saves and reverts the stack. Saves nest. Most frequently, you will want to use these at the start and end of a command.

6 Styles

The SPECTRALSEQUENCES package has a large number of styles which control the appearance of specific components (e.g., classes, differentials, or structure lines) of a spectral sequence. Each style has two corresponding keys: `classes` and `change classes`. Saying `classes = {\langle keys \rangle}` adds the keys to the list of options used to style every future class, whereas `change classes = {\langle keys \rangle}` only makes sense in a `{sseqpage}` environment, and temporarily overwrites the list of options. Note that `change classes` only applies to classes that existed before the current page, and that even with the `keep changes` option, the `change classes` options are local to the current page. Compare:

change new classes	change old classes

```
\begin{sseqdata}[ name = style example ]
\class(0,0)\class(1,1)
\end{sseqdata}
\begin{sseqpage}[ name = style example,
                  classes = { fill, blue },
                  title = change new classes ]
\class(0,1)\class(1,0)
\end{sseqpage}
\quad
\begin{sseqpage}[ name = style example,
                  change classes = { fill, blue },
                  title = change old classes ]
\class(0,1)\class(1,0)
\end{sseqpage}
```

You can modify these styles outside of a spectral sequence or inside it using `\sseqset`, you can modify them as options to the `{sseqdata}` and `{sseqpage}` environments, or you can modify them as arguments to the `{scope}` environment.

In cases where the same drawing feature is affected by multiple of these styles, the more specific style takes precedence. For instance, for a class that is the source or target of a differential on the current page, the precedence order from lowest to highest goes: `sseq style`, `class style`, `transient cycle style`, `this page cycle style`, and then any options from scopes in the order they appear, and any local options (the options that come right with the class, e.g., `\class[local options](x,y)`). If you don't want the options to your scopes to override more specific styles, use `sseq`:

everything is orange	only structure line is orange

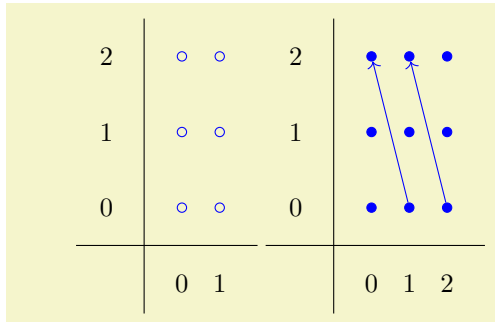
```
\begin{sseqpage}[ classes = { blue, fill },
                  title style = { align = center, text width = 2.4cm },
                  title = { everything is orange } ]
\begin{scope}[orange]
\class(0,0) \class(0,1)
\structline(0,0)(0,1)
\end{scope}
\end{sseqpage}

\begin{sseqpage}[ classes = { blue, fill },
                  title style = { align = center, text width = 2.4cm },
                  title = { only structure line is orange } ]
\begin{scope}[ sseq = orange ]
\class(0,0) \class(0,1)
\structline(0,0)(0,1)
\end{scope}
\end{sseqpage}
```

Throughout, “class” and “cycle” are synonyms.

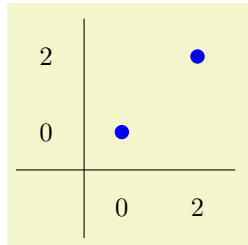
```
sseqs = {\langle keys \rangle}
change sseqs = {\langle keys \rangle}
sseq = {\langle keys \rangle}
change sseq = {\langle keys \rangle}
```

This passes options to all features in all future spectral sequences in the current scope. Note that for many global options you can set a default directly by saying `\sseqset{key = \marg{value}}` and this is in some cases preferable.



```
% Applies to both of the following sseqs:
\sseqset{ sseqs = { blue, scale = 0.5 } }
\begin{sseqpage}
\foreach \x in {0,1}
\foreach \y in {0,1,2} {
  \class(\x,\y)
}
\end{sseqpage}
\begin{sseqpage}[ Adams grading, classes = fill ]
\foreach \x in {0,1,2}
\foreach \y in {0,1,2} {
  \class(\x,\y)
}
\d2(1,0)
\d2(2,0)
\end{sseqpage}
```

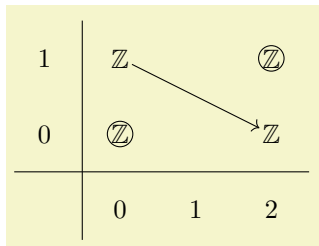
```
classes = {\langle keys \rangle}
cycles = {\langle keys \rangle}
change classes = {\langle keys \rangle}
change cycles = {\langle keys \rangle}
```



```
\begin{sseqpage}[ classes = { blue, fill, minimum width = 0.5em },
                  scale = 0.5, x tick step = 2, y tick step = 2 ]
\class(0,0)
\class(2,2)
\end{sseqpage}
```

```
permanent classes = {\langle keys \rangle}
permanent cycles = {\langle keys \rangle}
change permanent classes = {\langle keys \rangle}
change permanent cycles = {\langle keys \rangle}
```

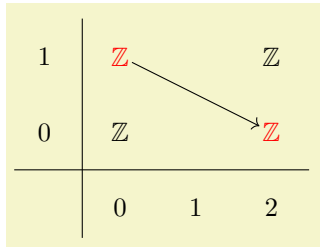
These options change the appearance of all permanent cycles (e.g., those classes which never support or are hit by a differential). For instance, we can circle the permanent cycles automatically. In the following example, note that because `permanent cycles` is more specific than `classes`, the `permanent cycles = {draw}` command takes precedence over the `classes = {draw = none}` command and the permanent cycle nodes are drawn.



```
\begin{sseqpage}[ cohomological Serre grading,
                  classes = { draw = none },
                  permanent cycles = {draw} ]
\foreach \x in {0,2} \foreach \y in {0,1} {
  \class["\mathbb{Z}"](\x,\y)
}
\d2(0,1)
\end{sseqpage}
```

```
transient classes = {\langle keys \rangle}
transient cycles = {\langle keys \rangle}
change transient classes = {\langle keys \rangle}
change transient cycles = {\langle keys \rangle}
```

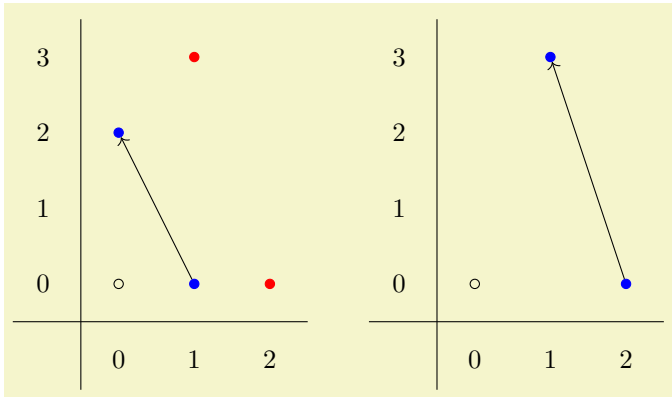
These options change the appearance of all transient cycles (e.g., those classes which eventually support or are hit by a differential). Again, this takes precedence over the `classes` option.



```
\begin{sseqpage}[ cohomological Serre grading,
                  classes = { draw = none },
                  transient cycles = red ]
\foreach \x in {0,2} \foreach \y in {0,1} {
  \class["\mathbb{Z}"](\x,\y)
}
\d2(0,1)
\end{sseqpage}
```

```
this page classes = {\langle keys \rangle}
this page cycles = {\langle keys \rangle}
change this page classes = {\langle keys \rangle}
change this page cycles = {\langle keys \rangle}
```

These options change the appearance of all cycles which support or are hit by a differential on this page. Any class that is hit on the current page is also a transient cycle, and so `this page classes` takes precedence over `transient cycles`.



```
\begin{sseqdata}[ name = this page cycles example, Adams grading,
                  transient cycles = { red, fill }, this page cycles = { blue } ]
\class(0,0)
\class(0,2) \class(1,0)
\class(1,3) \class(2,0)
\d2(1,0) \d3(2,0)
\end{sseqdata}
\printpage[ name = this page cycles example, page = 2 ] \quad
\printpage[ name = this page cycles example, page = 3 ]
```

```
edges = {\langle keys \rangle}
differentials = {\langle keys \rangle}
struct lines = {\langle keys \rangle}
change edges = {\langle keys \rangle}
change differentials = {\langle keys \rangle}
change struct lines = {\langle keys \rangle}
```

The `edges` key applies to both differentials and structure lines. The `differentials` and `struct lines` keys both take precedence over `edges`.

```
this page struct lines = {\langle keys \rangle}
change this page struct lines = {\langle keys \rangle}
```

This style applies to structure lines whose source or target is hit on the current page. It takes precedence over `struct lines`.

```
tikz primitives = {\langle keys \rangle}
change tikz primitives = {\langle keys \rangle}
```

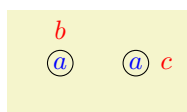
Applies to all TikZ primitives.

```
labels = {\langle keys \rangle}
change labels = {\langle keys \rangle}
```

This style applies to labels on classes, differentials, and structure lines. All the more specific label styles take precedence over it.

```
class labels = {\langle keys \rangle}
inner class labels = {\langle keys \rangle}
outer class labels = {\langle keys \rangle}
change class labels = {\langle keys \rangle}
change inner class labels = {\langle keys \rangle}
change outer class labels = {\langle keys \rangle}
```

Inner class labels specifically applies to class labels that are inside the node, outer class labels specifically applies to ones outside it:



```
\begin{sseqpage}[ no axes, classes = { inner sep = 1pt },
  label distance = 2pt,
  outer class labels = { red },
  inner class labels = { blue } ]
\class["a", "b" above](0,0)
\class["a", "c" right](1,0)
\end{sseqpage}
```

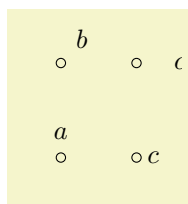
```
edge labels = {\langle keys \rangle}
differential labels = {\langle keys \rangle}
struct line labels = {\langle keys \rangle}
change edge labels = {\langle keys \rangle}
change differential labels = {\langle keys \rangle}
change struct line labels = {\langle keys \rangle}
```

6.1 Style-like options

The options are not styles, but can be modified in the same set of places (namely, anywhere):

```
label distance = \langle dimension \rangle
```

This sets the default distance from a class to an outer label. There are also variants like above label distance corresponding to above, below, left, right, above left, above right, below left, and below right.

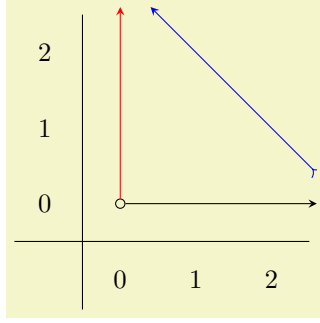


```
\begin{sseqpage}[ label distance = 0.3em,
  right label distance = 0em,
  no axes, yscale = 1.25 ]
\class["a" above](0,0)
\class["b" above right](0,1)
\class["c" right](1,0)
\class["c" {right = 1em}](1,1)
\end{sseqpage}
```

```
run off = \langle start tip \rangle - \langle end tip \rangle
run off struct lines = \langle start tip \rangle - \langle end tip \rangle (initially ...-...)
run off differentials = \langle start tip \rangle - \langle end tip \rangle (initially ...-...)
```

Change the default behavior of run off edges for either all edges, just structure lines, or just differentials respectively. Local arrowhead options override this.

If an edge runs off the edge of the clipping, SPECTRALSEQUENCES automatically add an arrowhead to indicate that the edge continues. This option controls which arrow head is added if the start or end of an edge runs off the page.



```
\begin{sseqpage}[ x range = {0}{2}, y range = {0}{2},
draw orphan edges, run off = >-stealth ]
\class(0,0)
\class(3,0) \class(0,3)
\structline(0,0)(3,0)
\structline[red](0,0)(0,3)
\structline[blue](3,0)(0,3)
\end{sseqpage}
```

`class label handler = \langle function \rangle`

The value of `class label handler` is a function that is applied to all labels before displaying them. It should put its output into `\result`. This is intended to help with code reusability. Because these handlers may crash or have annoying side-effects on some input, you may want to toggle the value of this command on and off. To turn this off for the rest of the current spectral sequence you can say `\sseqset{class label handler = {}}`. You can also use the `class label handler` key in a `{scope}`.

The main function SPECTRALSEQUENCES provides for use here is `\SseqNormalizeMonomial{#1}`. This makes it convenient to translate expressions with polynomial labels. You can write your own handlers if your T_EX programming skills are sufficient. Let me know if there are any other functions that you want here, and if you implement them yourself, please send me your implementation. Here is an example of a function that evaluates an arithmetic expression:

1		②①
0	②	
	0	1

```
\begin{sseqpage}[
class label handler = { \SseqParseInt\result{#1} }
]
\class["1+1"](0,0)
\class["1+2*(1+3*(4-1))"](1,1)
\end{sseqpage}
```

Here's an example using `\SseqNormalizeMonomial`:

3	α^3	$\alpha^3 y$	$\alpha^3 y^2$	$\alpha^3 y^3$
2	α^2	$y \alpha^2$	$\alpha^2 y^2$	$y^3 \alpha^2$
1	α	αy	αy^2	αy^3
0	1	y	y^2	y^3
	0	1	2	3

```
\NewSseqGroup \test {m} {
\class["1#1"](0,0)
\class["\alpha#1"](0,1)
\class["y#1"](1,0)
\class["\alpha y#1"](1,1)
}
\begin{sseqpage}[
class label handler = { \SseqNormalizeMonomial{#1} },
classes = { draw = none }, class labels = { font = \small } ]
\test{}
\test[red](2,0){y^2}
\test[orange](2,2){\alpha^2 y^2}
\test[blue](0,2){\alpha^2}
\end{sseqpage}
```

Here is another example which demonstrates a useful idiom for drawing Serre spectral sequences. For a more complete example, see `example_KF3n.tex`. Note the use of braces in `{Sq1\iota2}`. Without braces, `\SseqNormalizeMonomial` will simplify `Sq1xSq2x` into `S2q3x2`, which is obviously undesirable, so the correct way to input this is `{Sq1x}{Sq2x}`. Unfortunately, `\foreach` strips a pair of braces from its arguments, so you need to put two pairs of braces.

4	x^2	$x^2\iota_2$	$x^2Sq^1\iota_2$	$x^2\iota_2^2$	
3	αx	$\alpha x\iota_2$	$\alpha xSq^1\iota_2$	$\alpha x\iota_2^2$	
2	x	$x\iota_2$	$xSq^1\iota_2$	$x\iota_2^2$	
1	α	$\alpha\iota_2$	$\alpha Sq^1\iota_2$	$\alpha\iota_2^2$	
0	1	ι_2	$Sq^1\iota_2$	ι_2^2	
	0	1	2	3	4

```
\begin{sseqpage}[
  xscale = 1.4,
  classes = { draw = none },
  class label handler = {\SseqNormalizeMonomial{#1}} ]

\foreach \x/\xlabel in
{ 0/1, 2/\iota_2, 3/{Sq^1\iota_2}, 4/\iota_2^2 }
\foreach \y/\ylabel in
{ 0/1, 1/\alpha, 2/x, 3/\alpha x, 4/x^2 }
{
  \class["\ylabel\xlabel"] (\x,\y)
}
\end{sseqpage}
```

class name handler = $\langle function \rangle$

The value of **class name handler** is a function that is applied to all names before using them. It should put its output into `\result`. The main functions intended for use here are `\SseqAHSSNameHandler` and `\SseqNormalizeMonomial`, though you can make your own. This is applied both when you name the class, as in `\class[name=a](\meta{coordinate})` and when you refer to a class by name, as in `\structline(a)(b)`. One advantage if you are using `\SseqNormalizeMonomial` is that you can ensure that `xy` and `yx` refer to the same class:

1	$\circ xy^2$
0	$\circ xy \circ$
	0 1

```
\SseqNormalizeMonomialSetVariables{xy}
\begin{sseqpage}[ class name handler = \SseqNormalizeMonomial,
  classes = {show name = {right,pin}} ]
\class[name = xy](0,0)
\class[name = yxy](0,1)
\class(1,0)
\structline(yx)(yyx)
\end{sseqpage}
```

The next page is an example drawing the Atiyah Hirzebruch spectral sequence computing $KO^*\mathbb{R}P^8$.

class name prefix = $\langle prefix \rangle$

class name postfix = $\langle postfix \rangle$

These keys add respectively a prefix or a postfix to all names of all classes in the scope. This is applied both when you name the class, as in `\class[name=a](\meta{coordinate})` and when you refer to a class by name, as in `\structline(a)(b)`. The prefix and postfix are added to the command before the **class name handler** is applied, so if you say `name=\meta{the name}` then the name actually given to the class is the value of `\result` after saying

`\classnamehandler{\meta{name prefix}\meta{the name}\meta{name postfix}}`

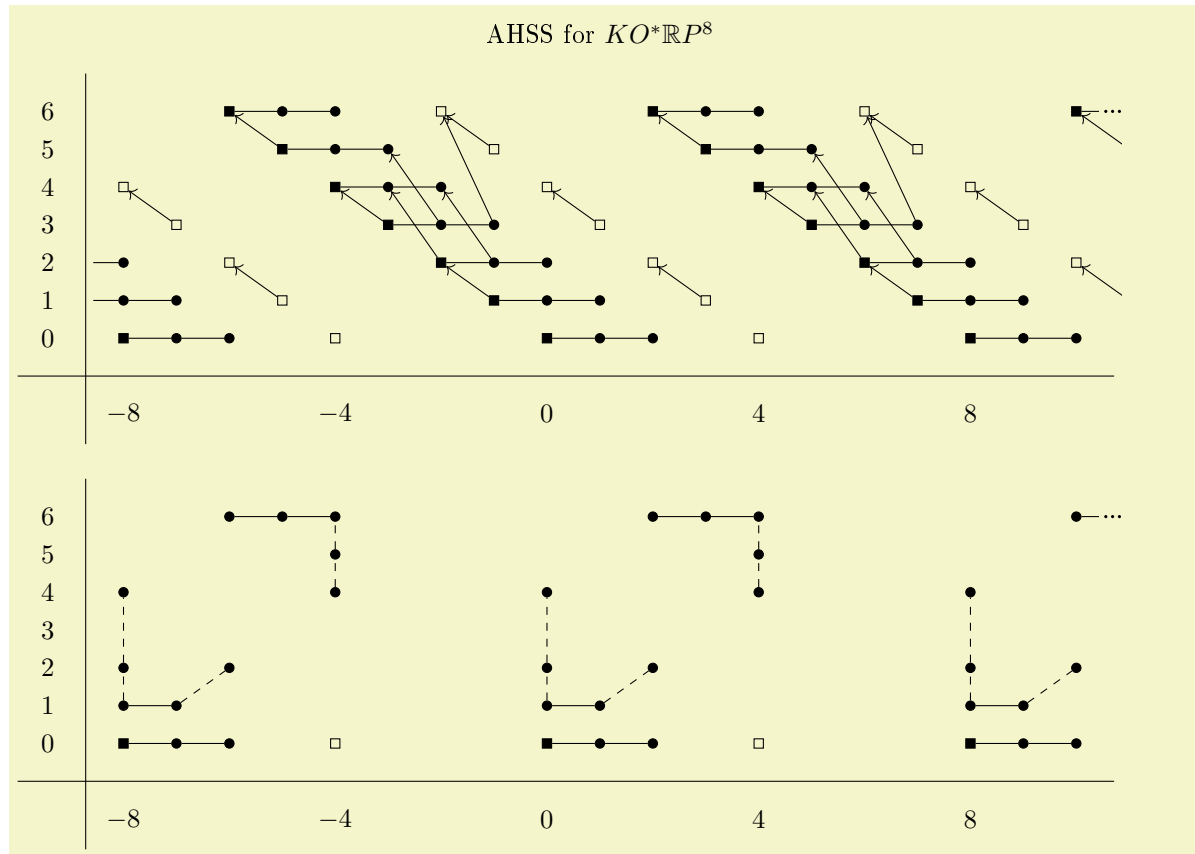
7 Global Options

These options can only be set at the beginning of a `{sseqdata}` or `{sseqpage}` environment. When it makes sense, you can also set a default value using `\sseqset`. Generally, these options either modify the plot style or the logic for the spectral sequence.

name = $\langle sseq name \rangle$

This option must be used with the `{sseqdata}` environment where it indicates the name of the spectral sequence, which will be used with the `{sseqpage}` environment or `\printpage` command to draw the

An AHSS using class name handler = \SseqAHSSNameHandler :



```
\NewSseqCommand \KOstar {m}{
  \begin{scope}[ xshift = -#1, xshift = 8*|b, yshift = #1 ]
    \class[rectangle, name = b^{\b}[0]](0,0)
    \class[name = b^{\b} e[0]](1,0) \structline
    \class[name = b^{\b} e^2[0]](2,0) \structline
    \class[rectangle, fill = none, name = b^{\b} u[0]](4,0)
  \end{scope}
}
\NewSseqCommand \KOpcell {m}{
  \d1(b^{\b}[#1]) \replacetarget
  \d1(b^{\b} u[#1]) \replacetarget
  \structline(b^{\b}[#1+1])(b^{\b} e[#1+1])
  \structline[page = 2, dashed](b^{\b} e[#1])(b^{\b} e^2[#1+1])
  \structline[page = 2, dashed](b^{\b} e^2[#1])(b^{\b} u[#1+1])
}
\NewSseqCommand \KOetacell {m}{
  \DrawIfValidDifferential2(b^{\b}[#1])
  \d2(b^{\b} e[#1])
  \DrawIfValidDifferential3(b^{\b} e^2[#1])
}
\begin{sseqdata}[ name = KO^*\mathbb{R}P^8, Adams grading, class name handler = \SseqAHSSNameHandler,
  classes = fill, x range = {-8}{10}, xscale = 0.7, yscale = 0.5, x tick step = 4 ]
\foreach \b in {-2,...,2}{
  \foreach \n in {0,...,6}{\KOstar{\n}}
  \foreach \n in {1,3,...,5}{\KOpcell{\n}}
  \foreach \n in {2,3}{\KOetacell{\n}}
  \structline[dashed](b^{\b}e^2[2])(b^{\b}u[4])
  \structline[dashed](b^{\b}[4])(b^{\b}e[5])
}
\end{sseqdata}
\printpage[ name = KO^*\mathbb{R}P^8, title = AHSS for $KO^*\mathbb{R}P^8$ ] \vskip6pt
\printpage[ name = KO^*\mathbb{R}P^8, page = 6 ]
```

spectral sequence. The name used in a `{sseqdata}` environment must be new unless the environment is used with the `update existing` key in which case the `{sseqdata}` environment will add to the existing spectral sequence. It is optional when used with `{sseqpage}`, and if included the name given must be the name of an existing spectral sequence.

`page = <page number>–<page max> (initially 0)`

This key is for `{sseqpage}` and `\printpage`. It specifies which page of the spectral sequence is to be printed. On page r , all `\classes` that are not hit by differentials on pages less than r will be printed, as well as all `\structlines` whose source and target classes are both printed on page r , and all differentials of length exactly r . The special value `page = 0` prints all classes, differentials, and structure lines but no extensions. The special value `page=\infty` will draw the infinity page with extensions.

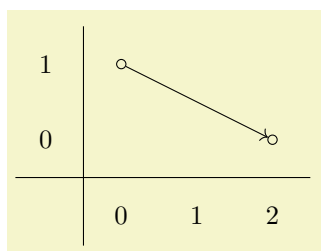
If you use this key with a range, e.g., `page=2--5`, then all differentials in that range of lengths will be drawn, so in the example differentials of length greater than or equal to 2 and less than or equal to 5 are drawn. The larger number only changes which differentials are drawn, classes and structlines are drawn as if the page was the small endpoint of the range. If you use 0 for the `<page max>`, then all differentials longer than `<page number>` will be drawn, e.g., `page=5---0` draws all differentials of length at least 5, but no extensions. Using `page=5---\infty` will draw all differentials of length at least 5 and also all extensions whose endpoint classes are in the final generation on page 5.

`degree = {\langle x degree \rangle}{\langle y degree \rangle}`
`cohomological Serre grading`
`homological Serre grading`
`Adams grading`

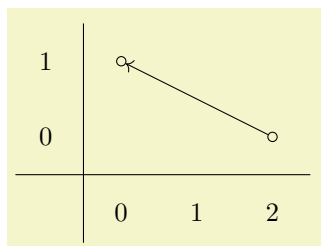
Specifies the degree of differentials. The `<x degree>` and `<y degree>` should both be mathematical expressions in one variable `#1` that evaluate to integers on any input. They specify the x and y displacement of a page `#1` differential. In practice, they will be linear expressions with `#1` coefficient 1, -1, or 0.

The `degree` option must be given before placing any differentials. It can be specified at the beginning of the `{sseqdata}` environment, at the beginning of the `{sseqpage}` environment if it is being used as a standalone page, or as a default by saying `\sseqset{degree = \marg{x degree}\marg{y degree}}` or `\sseqset{Adams grading}` outside of the `{sseqdata}` and `SPECTRALSEQUENCES` environments.

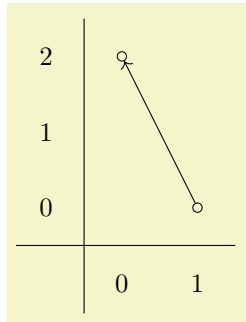
You can make a named grading convention by saying `\sseqset{mygrading/.sseq grading = {\langle x degree \rangle}{\langle y degree \rangle}}`. Then later passing `my grading` to a spectral sequence is equivalent to saying `degree = \marg{x degree}\marg{y degree}`. The following grading conventions exist by default:



```
% equivalent to degree = {\#1}{1-#1}:
\begin{sseqpage}[ cohomological Serre grading ]
\class(0,1)
\class(2,0)
\d2(0,1)
\end{sseqpage}
```



```
% equivalent to degree = {-#1}{#1-1}:
\begin{sseqpage}[ homological Serre grading ]
\class(0,1)
\class(2,0)
\d2(2,0)
\end{sseqpage}
```



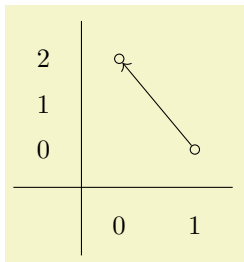
```
% equivalent to degree = {-1}{#1}:
\begin{sseqpage}[ Adams grading ]
\class(0,2)
\class(1,0)
\d2(1,0)
\end{sseqpage}
```

strict degree

lax degree

If the degree is strict, then L^AT_EX will throw an error if you try to specify a differential that doesn't have the proper grading. The degree is strict by default.

```
\begin{sseqdata}[ name = laxdegree, Adams grading ]
\class(0,2)
\class(1,0)
\d3(1,0)(0,2) % Error: differential does not respect grading.
               % Target should be in position (0,3) but instead it is (0,2)...
\end{sseqdata}
```



```
\begin{sseqdata}[ name = laxdegree, Adams grading, lax degree, yscale = 0.6 ]
\class(0,2)
\class(1,0)
\d3(1,0)(0,2) % No error because degree checking is off
\end{sseqdata}
\printpage[ name = laxdegree, page = 3 ]
```

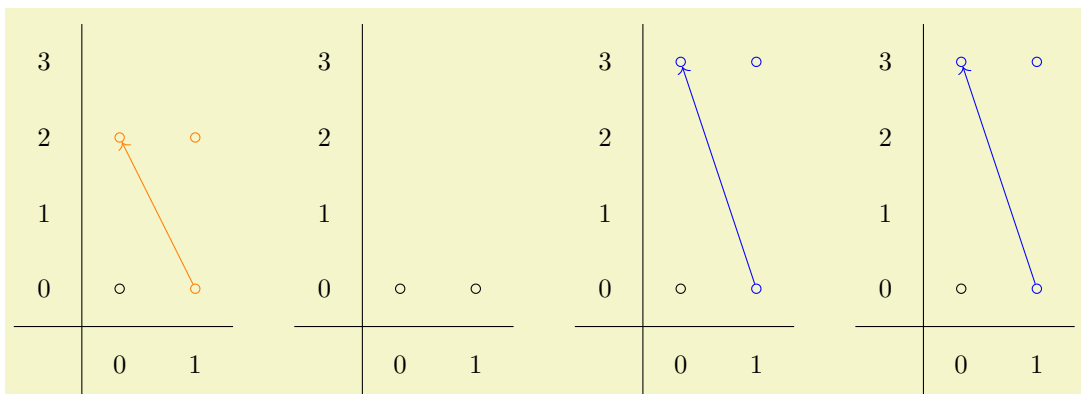
update existing

This key is only for the `{sseqdata}` environment. It specifies that the current `{sseqdata}` environment is adding data to an existing spectral sequence. If you don't pass this key, then giving a `{sseqdata}` environment the same `name` as a different `{sseqdata}` environment will cause an error. This is intended to help you avoid accidentally reusing the same name.

keep changes = *<boolean>*

(default true)(initially false)

This option is only for the `{sseqpage}` environment, and only works when a `name` is provided. This option specifies that all of the commands in the current `{sseqpage}` environment should be carried forward to future pages of the same named spectral sequence. For example:



```

\begin{sseqdata}[ name = keep changes example, Adams grading, y range = {0}{3} ]
\class(0,0)
\class(1,0)
\end{sseqdata}

\begin{sseqpage}[ name = keep changes example, sseq = orange ]
\class(0,2)
\class(1,2)
\classoptions[orange](1,0)
\d2(1,0)
\end{sseqpage} \quad

\printpage[ name = keep changes example, page = 2 ] \quad

\begin{sseqpage}[ name = keep changes example, sseq = blue, keep changes ]
\class(0,3)
\class(1,3)
\classoptions[blue](1,0)
\d3(1,0)
\end{sseqpage} \quad

\printpage[ name = keep changes example, page = 3 ]

```

Note that the orange classes and differential do not persist because the `keep changes` option is not set in the first `{sseqpage}` environment, but the blue classes and differential do, since the `keep changes` option is set in the second `{sseqpage}` environment.

no differentials
draw differentials

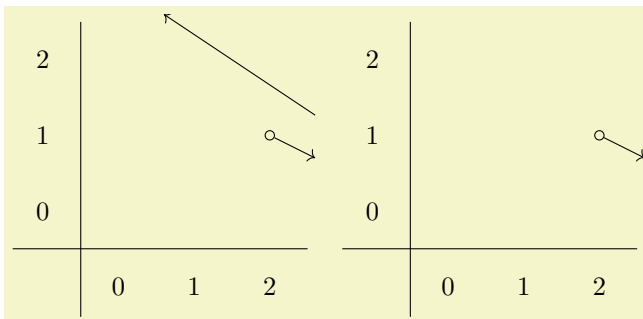
The option `no differentials` suppresses all of the differentials on the current page, whereas `draw differentials` causes the page appropriate differentials to be drawn. This is useful for explaining how the computation of a spectral sequence goes, or if you want to display one of the edges of the spectral sequence, like in `example_KF3n.tex`.

no struct lines
draw struct lines

The option `no struct lines` suppresses all of the structure lines on the current page, whereas the option `draw struct lines` causes the page appropriate structure lines to be drawn.

no orphan edges
draw orphan edges

An edge is an “orphan” if both its source and target lie off the page. By default these are drawn, but with the option `no orphan edges` they are not. If the option `no orphan edges` has been set, `draw orphan edges` undoes it.



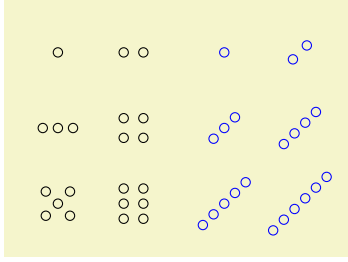
```

\begin{sseqdata}[
  name = orphan edges example,
  cohomological Serre grading,
  x range = {0}{2}, y range = {0}{2} ]
\class(0,3) \class(3,1)
\d3(0,3)
\class(2,1) \class(4,0)
\d2(2,1)
\end{sseqdata}
\printpage[ name = orphan edges example ]
\quad
\printpage[ name = orphan edges example,
  no orphan edges ]

```

class pattern = \langle class pattern name \rangle (initially standard)

This key specifies the arrangement of multiple classes at the same coordinate. The default value is `standard`.



```
\begin{sseqdata}[ name = class pattern example, no axes, ymirror ]
\class(0,0)
\class(1,0) \class(1,0)
\class(0,1) \class(0,1) \class(0,1)
\class(1,1) \class(1,1) \class(1,1) \class(1,1)
\class(0,2) \class(0,2) \class(0,2) \class(0,2) \class(0,2)
\class(1,2) \class(1,2) \class(1,2) \class(1,2) \class(1,2) \class(1,2)
\end{sseqdata}

\printpage[ name = class pattern example, class pattern = standard ]
\printpage[ name = class pattern example, change classes = blue,
class pattern = linear, class placement transform = { rotate = 45 } ]
```

You can add new class patterns using `\SseqNewClassPattern`:

`\SseqNewClassPattern{<class pattern name>}{<offsets>}`

Creates a new class pattern. For example, the `linear` class pattern is created using the command:

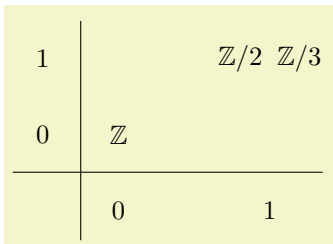
```
\SseqNewClassPattern{linear}{
(0,0);
(-0.13,0)(0.13,0);
(-0.2,0)(0,0)(0.2,0);
(-0.3,0)(-0.1,0)(0.1,0)(0.3,0);
(-0.4,0)(-0.2,0)(0,0)(0.2,0)(0.4,0);
(-0.5,0)(-0.3,0)(-0.1,0)(0.1,0)(0.3,0)(0.5,0);
}
```

For instance the third row indicates that if there are three classes at the position (x,y) they should be printed at $(x-0.2,y)$, (x,y) , and $(x+0.2,y)$. You can give as many rows as you like; SPECTRALSEQUENCES will throw an error if there are more classes in any position than the maximum number that your class pattern can handle – for instance, the `linear` class pattern can handle up to six classes based on this definition.

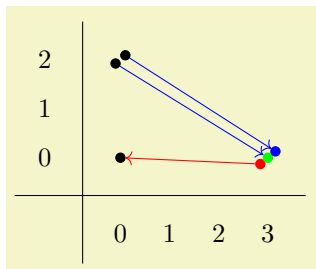
`class placement transform = {<transform keys>}`

`add class placement transform = {<transform keys>}`

The option `class placement transform` allows the user to specify a TikZ coordinate transform to adjust the relative position of multiple nodes in the same (x,y) position. The `class placement transform` key overrides the previous value of transformations, the `add class placement transform` just adds the new transformation to the end of the list. This coordinate transform can only involve rotation and scaling, no translation. Specifying a scaling factor helps if the nodes are too large and overlap. In some cases a rotation makes it easier to see which class is the target of a differential.



```
\begin{sseqpage}[ class placement transform = { xscale = 1.5 },
class pattern = linear,
classes = { draw = none },
xscale = 2, x axis extend end = 0.7cm ]
\class["\mathbb{Z}"](0,0)
\class["\mathbb{Z}/2"](1,1)
\class["\mathbb{Z}/3"](1,1)
\end{sseqpage}
```



```
\begin{sseqpage}[ class placement transform = { rotate = 40 },
cohomological Serre grading, scale = 0.65,
classes = fill, differentials = blue ]
\class(0,0)
\class(0,2)\class(0,2)
\class[red](3,0)\class[green](3,0)\class[blue](3,0)

\d3(0,2,1,2)
\d3(0,2,-1,-1)
\draw[->,red](3,0,1)--(0,0);
\end{sseqpage}
```

With multiple large class labels, the best option is to arrange the classes vertically:

2	$\iota_2 P^1 \iota_3$	$\iota_2 \beta P^1 \iota_3$
	$\iota_2 \iota_3 \beta \iota_3$	$\iota_2 P^1 \beta \iota_3$
1		$\iota_2 (\beta \iota_3)^2$
0	$P^1 \iota_3$	$\beta P^1 \iota_3$
	$\iota_3 \beta \iota_3$	$P^1 \beta \iota_3$
		$(\beta \iota_3)^2$
	7	8

```

\begin{sseqpage}[ classes = {draw = none }, xscale = 2, yscale=1.55,
  class pattern = linear,
  class placement transform = { scale = 1.5, rotate = 90 },
  right clip padding = 20pt, top clip padding = 20pt,
  x axis gap = 30pt, y axis gap = 20pt ]
\class["\iota_3\beta\iota_3"](7,0)
\class["P^1\iota_3"](7,0)

\class["\iota_2\iota_3\beta\iota_3"](7,2)
\class["\iota_2P^1\iota_3"](7,2)

\class["(\beta\iota_3)^2"](8,0)
\class["P^1\beta\iota_3"](8,0)
\class["\beta P^1\iota_3"](8,0)

\class["\iota_2(\beta\iota_3)^2"](8,2)
\class["\iota_2P^1\beta\iota_3"](8,2)
\class["\iota_2\beta P^1\iota_3"](8,2)
\end{sseqpage}

```

`math nodes = $\langle boolean \rangle$` (default true)(initially true)

This key instructs SPECTRALSEQUENCES to put all labels in math mode automatically.

7.1 Global coordinate transformations

Of the normal TikZ coordinate transformations, only the following can be applied to a SPECTRALSEQUENCES chart:

`scale = $\langle factor \rangle$`
`xscale = $\langle factor \rangle$`
`yscale = $\langle factor \rangle$`
`xmirror`
`ymirror`

Scale the chart by $\langle factor \rangle$. Under normal circumstances, you can tell TikZ to mirror a chart by saying, for instance, `xscale = -1`, but SPECTRALSEQUENCES needs to be aware that the chart has been mirrored in order to draw the axes correctly. Thus, if you want to mirror a spectral sequence, use the `xmirror` and `ymirror` options as appropriate.

`rotate = $\langle angle \rangle$`

It probably won't look great if you pick an angle that isn't a multiple of 90 degrees.

7.2 Plot options and axes style

`x range = { $\langle x min \rangle$ } { $\langle x max \rangle$ }`
`y range = { $\langle y min \rangle$ } { $\langle y max \rangle$ }`

These options set the x range (respectively y range) to be a specific interval. By default, if no range is specified then the range is chosen to fit all the classes. If an x range is specified but no y range, then the y range is chosen to fit all the classes that lie inside the specified x range, and vice versa. The values must be integers – if you want to extend the x axis a noninteger amount, try using `x axis start extend` or `x axis end extend`.

`range check standard (default)`
`range check sideways`
`range check off`

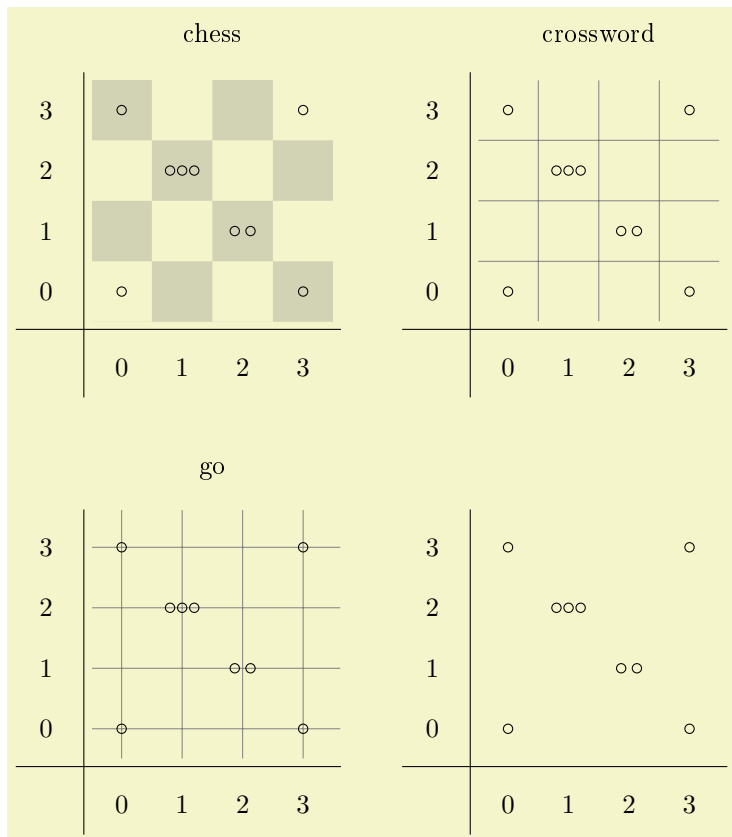
These options control the range checking behavior.

By default SPECTRALSEQUENCES will try to compute whether the chart fits onto the page (without causing overfull boxes). The computation isn't perfectly accurate and it assumes that the entire page is available for the chart to use up, so for instance if the chart is indented because of `\textindent` or not at the top of the page, this might give overly large values. Also, the values computed aren't perfectly accurate.

If the range check is not desirable, you can use `range check off`. If the diagram is being placed onto the page rotated 90 degrees, then use `range check sideways`. Inside of a `{sideways}` environment from the `rotating` package, this is the default.

`grid = <grid type>`
`grid color = <color>`
`grid step = <positive integer>`
`x grid step = <positive integer>`
`y grid step = <positive integer>`

Makes SPECTRALSEQUENCES draw a grid. The grid types and a significant part of the code that produces them were stolen from the `sseq` package.



```
\begin{sseqdata}[ name = grid example, scale = 0.8 ]
\class(0,0)
\class(3,0)
\class(2,1)\class(2,1)
\class(1,2)\class(1,2)\class(1,2)
\class(0,3)
\class(3,3)
\end{sseqdata}
\hbox{
\printpage[ name = grid example, grid = chess, title=chess ]
\qqquad
\printpage[ name = grid example, grid = crossword, title=crossword ]
}
\vskip20pt
\hbox{
\printpage[ name = grid example, grid = go, title=go ]
\qqquad
\printpage[ name = grid example, grid = none ]
}
```

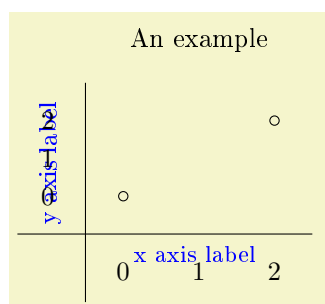
It is possible to make your own grid type by defining the command `\sseq@grid@yourgridname` to draw a grid.

It is a known problem that acrobat reader displays grids very dark and thick. This is caused by the enhance thin lines “feature” of Acrobat reader. You can turn it off globally by going into the acrobat reader preferences window and unchecking the “Enhance thin lines” checkbox in the “Rendering” section. Unfortunately, there is no way to instruct Acrobat Reader to not apply this feature to a particular document. This has apparently been annoying graphic designers for over a decade.

The grid color option changes the color of the grid; the default value is **gray**. The grid step keys change the grid step.

```
title = <text>
title style = <keys>
x label = <text>
y label = <text>
x label style = <keys>
y label style = <keys>
label style = <keys>
```

This make chart labels.



```
\begin{sseqpage}[ title = { An example }, yscale = 0.5,
  x label = { x axis label },
  y label = { y axis label },
  label style = { blue, font = \small },
  x label style = { yshift = 5pt },
]
\class(0,0)
\class(2,2)
\end{sseqpage}
```

Note that if you make multiple versions of the same chart and some of the charts have labels and others don't, they might not align the way you want. An easy solution to this is to use `\path[background] (min x, min y) -- (max x, max y);` where the coordinates are below and to the left, respectively above and to the right, of everything else drawn in any picture. This makes the bounding boxes for all of the pictures the same size, so that they line up even if the exact collection of things drawn changes. See the example at the end of the **background** key for an illustration of this.

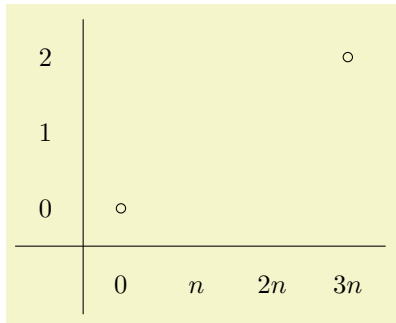
```
no title
draw title
no x label
no y label
no labels
draw x label
draw y label
draw labels
```

Suppress or unsuppress the title, x label, y label, or both x and y labels, respectively.

```
no x ticks
no y ticks
no ticks
draw x ticks
draw y ticks
draw ticks
```

Suppress axes ticks (the numbers next to the axes). Only matters if axes are drawn. You can make your own ticks using *TikZ* inside a `{scope}` environment with the **background** key. For instance,

you might want to label the axes as $0, n, 2n, \dots$. You can achieve this as follows: (you can also use `x tick handler`).



```
\begin{sseqpage}[ no x ticks, x range = {0}{3} ]
\begin{scope}[ background ]
  \node at (0,\ymin - 1) {0};
  % \vphantom is fragile so we have to throw in an extra \protect
  \node at (1,\ymin - 1) {\protect\vphantom{2}n};
  \foreach \n in {2,..., \xmax}{
    \node at (\n,\ymin - 1) {\n n};
  }
\end{scope}
\class(0,0)
\class(3,2)
\end{sseqpage}
```

`x tick step` = $\langle \text{positive integer} \rangle$ (initially 1)
`y tick step` = $\langle \text{positive integer} \rangle$ (initially 1)
`tick step` = $\langle \text{positive integer} \rangle$ (initially 1)

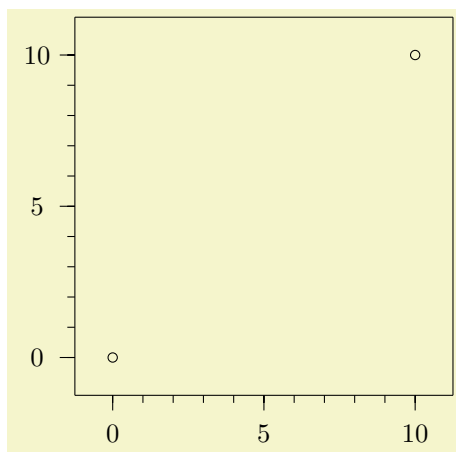
Sets the interval between labels.

`x tick offset` = $\langle \text{integer} \rangle$ (initially 0)
`y tick offset` = $\langle \text{integer} \rangle$ (initially 0)
`tick offset` = $\langle \text{integer} \rangle$ (initially 0)

Sets the label offset – by default the ticks will always be the set of numbers that are 0 mod $\langle \text{tick step} \rangle$. Change it so that the ticks are the set of numbers that are $\langle \text{tick offset} \rangle$ mod $\langle \text{tick step} \rangle$.

`x major tick step` = $\langle \text{nonnegative integer} \rangle$ (initially 0)
`y major tick step` = $\langle \text{nonnegative integer} \rangle$ (initially 0)
`major tick step` = $\langle \text{nonnegative integer} \rangle$
`x minor tick step` = $\langle \text{nonnegative integer} \rangle$ (initially 0)
`y minor tick step` = $\langle \text{nonnegative integer} \rangle$ (initially 0)
`minor tick step` = $\langle \text{nonnegative integer} \rangle$

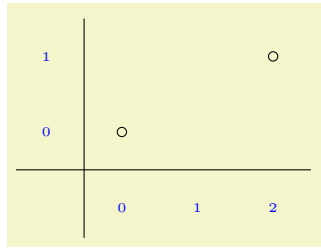
If these are nonzero, they control the placement of tick marks on the axes. The value 0 prevents tick marks from being drawn. Uses `x tick offset` as an offset.



```
\begin{sseqpage}[
  tick step = 5,
  major tick step = 5,
  minor tick step = 1,
  scale = 0.4,
  axes type = frame
]
\class(0,0)
\class(10,10)
\end{sseqpage}
```

`x tick style` = $\{ \langle \text{keys} \rangle \}$
`y tick style` = $\{ \langle \text{keys} \rangle \}$
`tick style` = $\{ \langle \text{keys} \rangle \}$

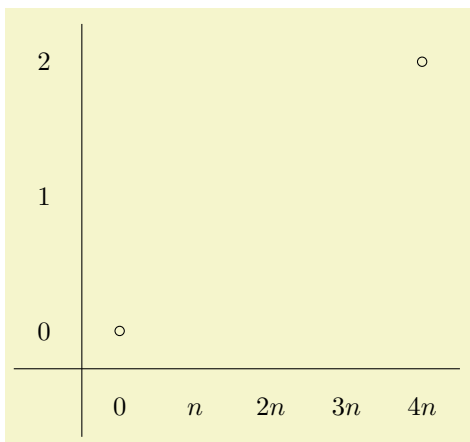
Change the tick style:



```
\begin{sseqpage}[ tick style = { blue, font = \tiny } ]
\class(0,0) \class(2,1)
\end{sseqpage}
```

`x tick handler` = $\langle \text{function} \rangle$ (initially #1)
`y tick handler` = $\langle \text{function} \rangle$ (initially #1)
`tick handler` = $\langle \text{function} \rangle$ (initially #1)

The value for `x tick handler` should be a function that takes in the current `x` value and outputs the appropriate tick. Correspondingly with `y tick handler`. The `tick handler` key sets both.

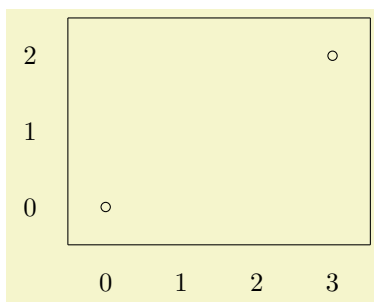


```
\begin{sseqpage}[ x range = {0}{4}, yscale = 1.78,
x tick handler = {
  \ifnum#1 = 0\relax
    0
  \else
    \ifnum#1 = 1\relax
      % \vphantom is fragile so we \protect it
      \protect\vphantom{2}n
    \else
      #1n
    \fi
  \fi
}
]
\class(0,0)
\class(4,2)
\end{sseqpage}
```

7.3 Layout

`x axis type` = $\langle \text{type} \rangle$ (initially border)
`y axis type` = $\langle \text{type} \rangle$ (initially border)
`axes type` = $\langle \text{type} \rangle$ (initially border)
`no x axis`
`no y axis`
`no axes`

The $\langle \text{type} \rangle$ is either `border`, `center`, `frame`, or `none`. `no axes` is a shorthand for `axes type=none`. The `border` type is the default and puts the axes on the bottom and left of the picture. The `center` type by default places the axes to pass through (0,0); this can be modified using the `x axis origin` and `y axis origin` keys. See `example_KRAHSS.tex` and `example_KUHPSS.tex` for examples where this is used. The `frame` type draws a frame around the entire chart:



```
\begin{sseqpage}[ axes type = frame ]
\class(0,0)
\class(3,2)
\end{sseqpage}
```

`x axis origin` = $\langle x \text{ value} \rangle$ (initially 0)

`y axis origin = $\langle y \text{ value} \rangle$` (initially 0)

If you use `axes type = center`, these keys change the position of the axes. Otherwise, they are ignored.

`x axis gap = $\langle dimension \rangle$` (initially 0.5cm)

`y axis gap = $\langle dimension \rangle$` (initially 0.5cm)

`axes gap = $\langle dimension \rangle$` (initially 0.5cm)

`x tick gap = $\langle dimension \rangle$` (initially 0.5cm)

`y tick gap = $\langle dimension \rangle$` (initially 0.5cm)

`x axis extend start = $\langle dimension \rangle$` (initially 0.5cm)

`y axis extend start = $\langle dimension \rangle$` (initially 0.5cm)

`x axis extend end = $\langle dimension \rangle$` (initially 0.5cm)

`y axis extend end = $\langle dimension \rangle$` (initially 0.5cm)

`x axis tail = $\langle dimension \rangle$` (initially 0.9cm)

`y axis tail = $\langle dimension \rangle$` (initially 0.9cm)

`x axis clip padding = $\langle dimension \rangle$` (initially 0.1cm)

`y axis clip padding = $\langle dimension \rangle$` (initially 0.1cm)

`right clip padding = $\langle dimension \rangle$` (initially 0.1cm)

`left clip padding = $\langle dimension \rangle$` (initially 0.1cm)

`top clip padding = $\langle dimension \rangle$` (initially 0.1cm)

`bottom clip padding = $\langle dimension \rangle$` (initially 0.1cm)

`custom clip = $\langle clip \text{ path} \rangle$`

Give a custom clipping. The clipping specified must be in the form of a valid TikZ path, for instance `\clip (0,0) rectangle (10,10);`. This clipping is also applied to any grid and is used to draw ellipses on appropriate differentials or structure lines that go out of bounds and to determine whether a differential or structure line is an “orphan”. It is not applied to any background elements, which is important because these are often used for axes labels and such that should lie outside of the clipping region. Weird things can happen with out of range edges if you provide an oddly shaped path.

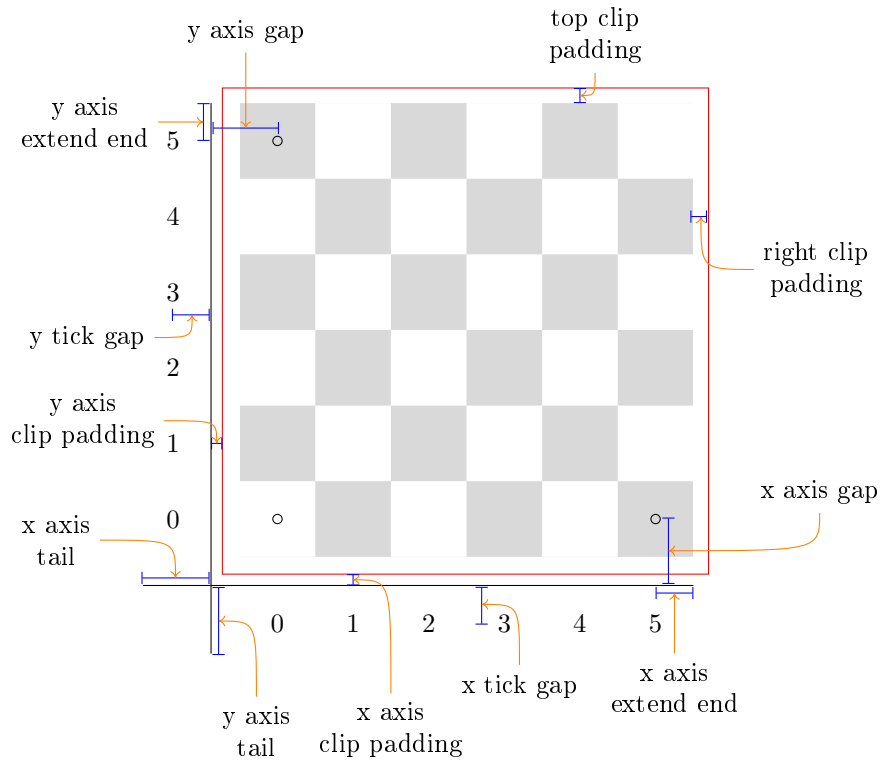
`clip = $\langle boolean \rangle$` (default true)(initially true)

If this is false the spectral sequence chart won’t be clipped. I’m not really sure why you would want that, but there might be some use case. Setting this to be false is not fully supported, and it’s possible that weird things will happen with some of the edges that go out of range.

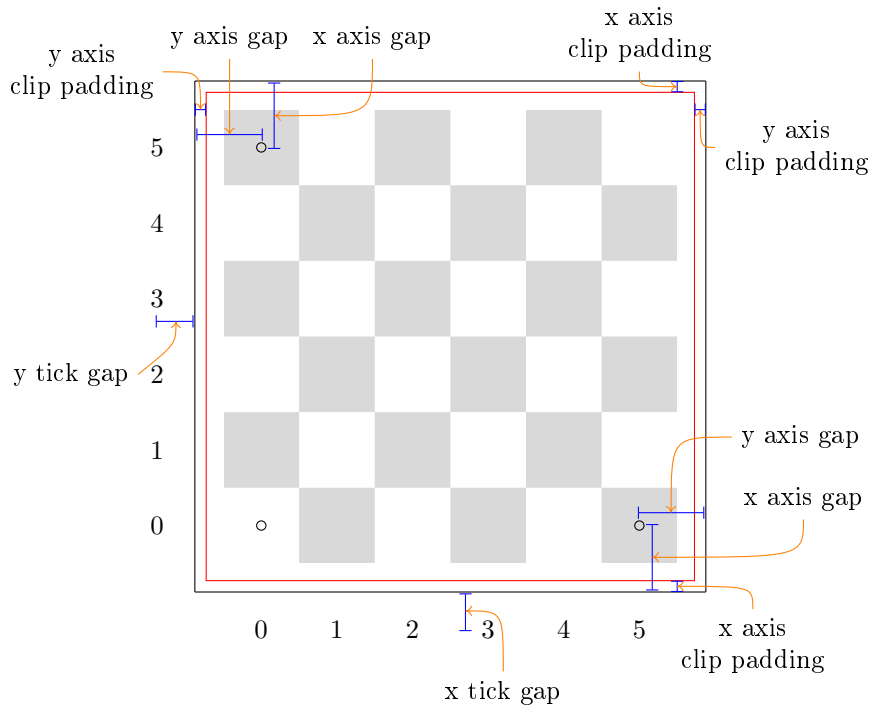
`rotate labels = $\langle boolean \rangle$` (default true)(initially false)

If you use `rotate = 90` but also want the labels rotated (so that the whole chart is sideways) use this key.

Axis type border layout



Axes type frame layout



Axes type center layout

