SPECTRALSEQUENCES

Hood Chatham
hood@mit.edu

Version 1.3.0
2022/2/20

The SPECTRALSEQUENCES package is a specialized tool built on top of PGF/Ti*k*Z 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 Ti*k*Z 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 Ti*k*Z commands. SPECTRALSEQUENCES also has a carefully designed error reporting system intended to ensure that it is as clear as possible what is going wrong.

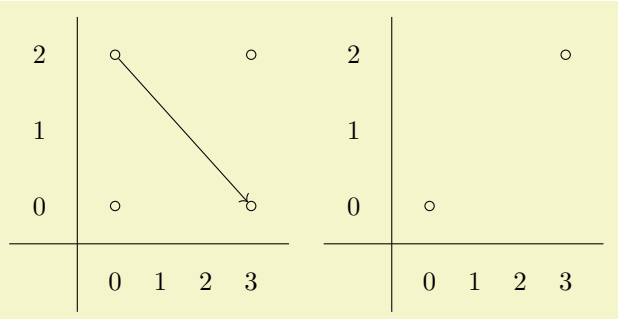
Many thanks to the authors of Ti*k*Z for producing such a wonderful package with such thorough documentation. I would have needed to spend a lot more time reading the Ti*k*Z code if the documentation weren't so excellent. I took ideas or code or both from tikzcd (part of the code for turning quotes into 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 LaTeX programming even though they probably found it dreadfully boring.

Contents

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/Ti*k*Z 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}[`⟨*options*⟩`]`
⟨*environment contents*⟩
`\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}[`⟨*options*⟩`]`
⟨*environment contents*⟩
`\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.

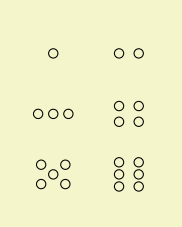
`\printpage[`⟨*options*⟩`]`

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[`⟨*options*⟩`](`⟨*x*⟩`,`⟨*y*⟩`)`

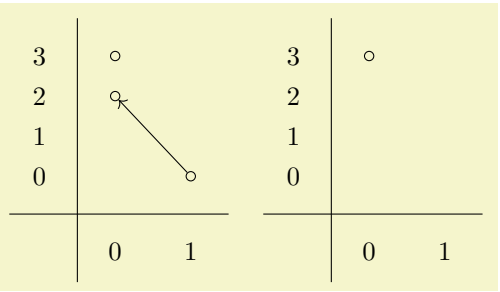
This places a class at `(x,y)` where `x` and `y` are integers. If multiple classes occur at the same position, SPECTRALSEQUENCES will automatically arrange them in a pre-specified pattern. This pattern may be altered using the`class pattern` option.



```
\begin{sseqpage}[ no axes, ymirror, yscale = 0.8 ]
\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{sseqpage}
```

The effect of the `\class` command is to print a Ti*k*Z node on a range of pages. Any option that would work for a Ti*k*Z `\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:

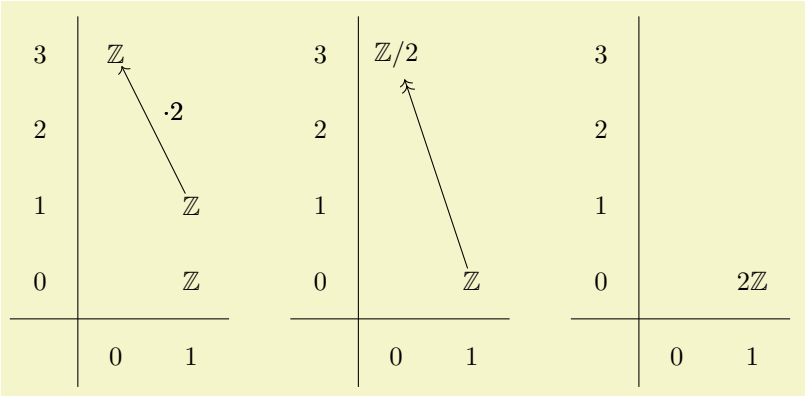



```
\begin{sseqdata}[ name = class example,
                  Adams grading,
                  yscale = 0.53 ]
\class(1,0)
\class(0,2)
\class(0,3)
\d2(1,0)
\end{sseqdata}
\printpage[ name = class example, page = 2 ]
\quad
\printpage[ name = class example, page = 3 ]
```

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.

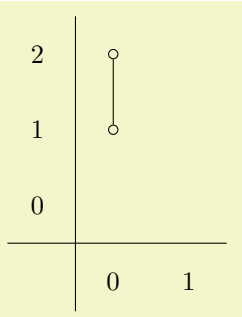


```
\begin{sseqdata}[name = replace class example, Adams grading, classes = {draw = none } ]
\class["\mathbb{Z}"](0,3)
\class["\mathbb{Z}"](1,1)
\class["\mathbb{Z}"](1,0)
\d["\cdot 2"]2(1,1)
\replacetarget["\mathbb{Z}/2"] %\replaceclass["\mathbb{Z}/2"](0,3)
\d[->]3(1,0)
\replacesource["2\mathbb{Z}"]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

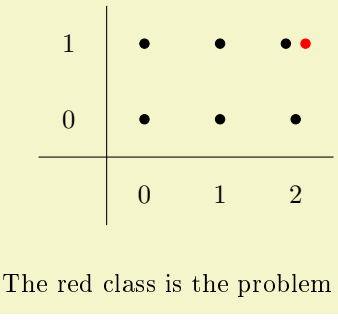



```
\begin{sseqdata}[name=replacestructlines]
\class(0,1)
\class(0,2)
\structline
\class(1,0)
\d2(1,0)(0,2)
\replacetarget\replacestructlines
\end{sseqdata}
\printpage[name=replacestructlines, page=3]
```

`\classoptions[`⟨*options*⟩`](`⟨*x*⟩`,`⟨*y*⟩`,`⟨*n*⟩`)`
`\classoptions[`⟨*options*⟩`](`⟨*classname*⟩`)`
`\classoptions[`⟨*options*⟩`]`

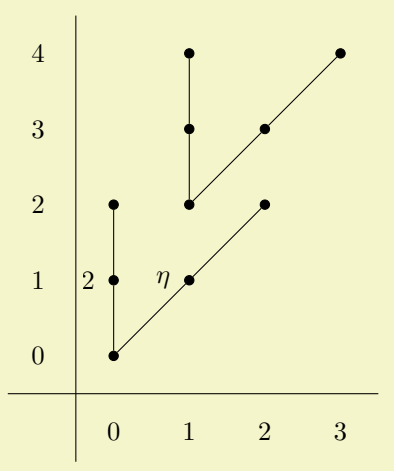
This adds options to an existing class. This can be used in a `{sseqpage}` environment to modify the appearance of a class for just one drawing of the spectral sequence, for instance to highlight it for discussion purposes.

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`:



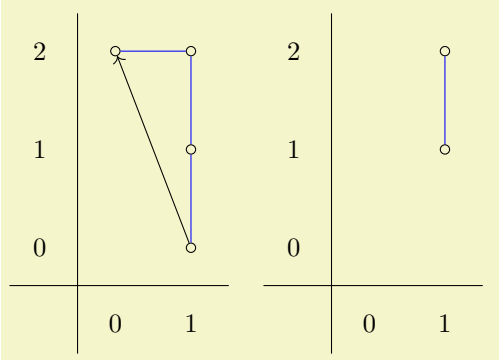

```
\NewSseqGroup\mygroup {} {
    \class(0,0)
    \class(0,1)
    \class(0,2)
    \class(1,1)
    \class(2,2)
    \structline(0,0)(0,1)
    \structline(0,1)(0,2)
    \structline(0,0)(1,1)
    \structline(1,1)(2,2)
}
\begin{sseqpage}[ classes = fill, class labels = { left = 0.3em } ]
\mygroup(0,0)
\mygroup(1,2)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

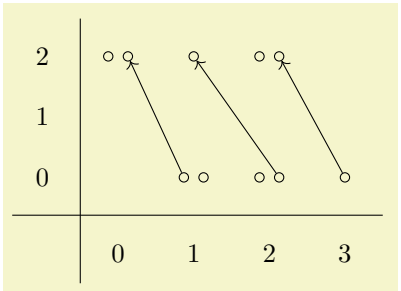
If there are multiple nodes in the source or target, you may specify which one the differential should go to using an index or tag for ⟨*source n*⟩ or ⟨*target n*⟩. It is also possible to provide the name of the source coordinate and an optional target, or to separately provide the source and target coordinate, either as names or as `(`⟨*x*⟩`,`⟨*y*⟩`,`⟨*n*⟩`)`. 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 ]
```

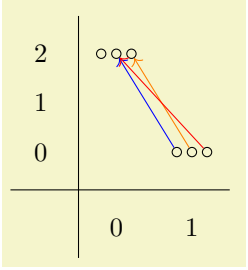
If there are multiple nodes in the source or target coordinate, then there is a funny syntax for indicating which one should be the source and target:

```
\d\meta{page}\pars{\meta{x},\meta{y}\opt{,\sourcen,\targetn}}
```




```
\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*⟩]

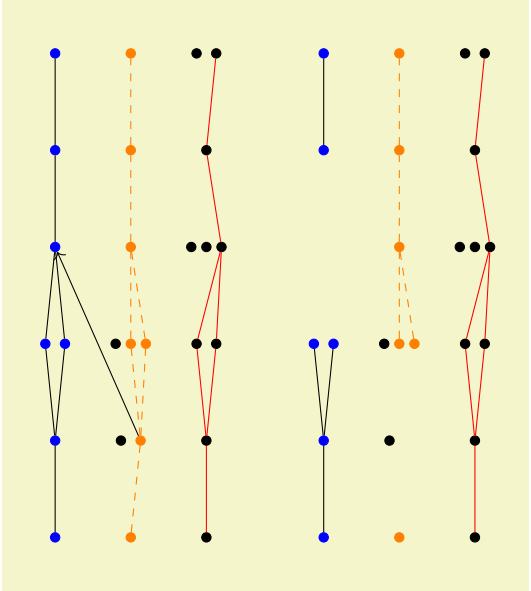
This command sets the indicated coordinate to die on the indicated page, but does not establish a target for the differential. This is useful if you want to draw your own differential using tikz (see `\getdtarget`) or if you are not drawing the class on the other side of the differential for clutter 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 change it. Also, the structline will disppear the first time after this the source or target has a differential, but this can be changed with the `\replacestructlines` command.



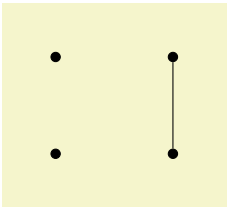
```
\DeclareSseqGroup\tower {} {
    \class(0,0)
    \foreach \y in {1,...,5} {
        \class(0,\y)
        \structline
    }
    \class(0,2)
    \structline(0,1,-1)
    \structline(0,3,-1)
}
\begin{sseqdata}[ name = structline example,
                  classes = { circle, fill },
                  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*⟩)

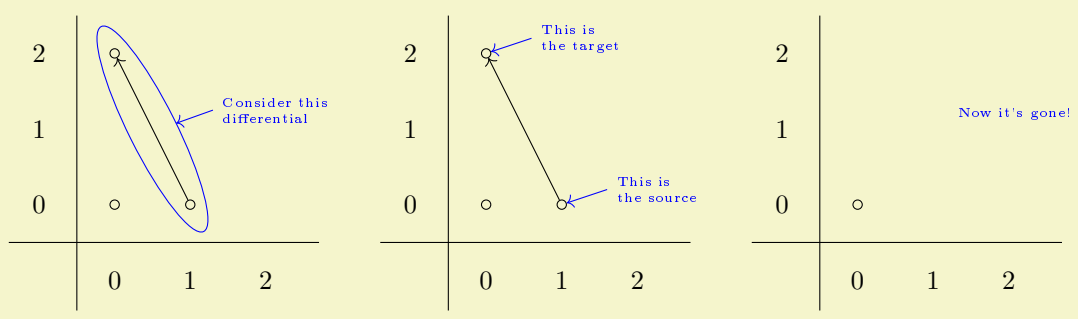
This command adds options to an existing extension. Its syntax is identical to \extension.

\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} \qquad
%
\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} \qquad
%
\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}
```

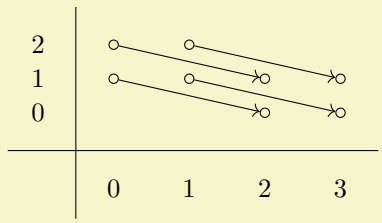
4 Options for the main commands

4.1 Universal options

The following options work with all of the drawing commands in this package, including `\class`, `\d`, and `\structline`, `\extension`, their friends `\replaceclass`, `\classoptions`, `\doptions`, `\structlineoptions`, `\extensionoptionsand` `\replacestructlines`, as well as with Ti*k*Z primitives.

`xshift` = ⟨*integer*⟩
`yshift` = ⟨*integer*⟩

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




```
\begin{sseqpage}[ cohomological Serre grading, yscale = 0.45 ]
\foreach \x in {0,1} \foreach \y in {0,1} {
    \begin{scope}[ xshift = \x, yshift = \y ]
    \class(2,0)
    \class(0,1)
    \d2(0,1)
    \end{scope}
}
\end{sseqpage}
```

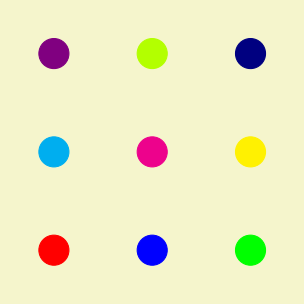
This code segment is very useful so SPECTRALSEQUENCES has the command `\NewSseqGroup` which to make code like this more convenient. The following code produces the same output as above:

```
\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.

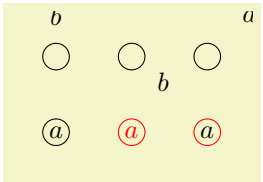


```
\begin{sseqpage}[ classes = {fill,inner sep = 0.4em},
                  no axes, scale = 1.3 ]
\class[red](0,0)
\class[blue](1,0)
\class[green](2,0)
\class[cyan](0,1)
\class[magenta](1,1)
\class[yellow](2,1)
\class[blue!50!red](0,2) % a 50-50 blend of blue and red
\class[green!30!yellow](1,2) % 30%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

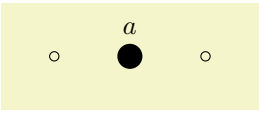
For a class, the ⟨*text*⟩ is placed in the position `inside` the node by default – in effect, the ⟨*text*⟩ becomes the label text of the node (so saying `\class["label text"](0,0)` causes a similar effect to saying`\node at (0,0) {label text};`). There are other position options such as`left`, `above 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

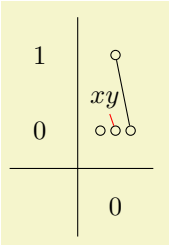
to the `above` position instead of going in the `\node` text by default. Also, this won't respect the various label style options like `labels`, etc.




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

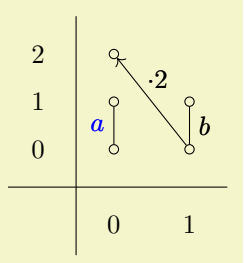
`pin` = ⟨*style*⟩

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}
```

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.



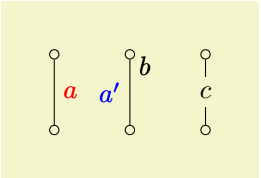

```
\begin{sseqpage}[ Adams grading, yscale = 0.63 ]
\class(0,0)
\class(0,1)
\class(0,2)
\structline["a"' blue](0,0)(0,1)
\class(1,0)
\class(1,1)
\structline["b"](1,0)(1,1)
\d[ "\cdot 2" { pos = 0.7, yshift = -5pt } ] 2 (1,0)
\end{sseqpage}
```

You can use the style options `labels`, `edge labels`, `differential labels`, `struct line labels`, and `extension 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 Ti*k*Z 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 coolor 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

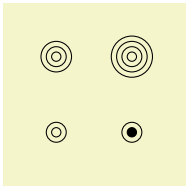
Because the main job of the `\class` command is to print a Ti*k*Z `\node` on the appropriate pages of the spectral sequence, most options that would work for a Ti*k*Z node also work for the commands `\class`, `\replaceclass`, and `\classoptions`. Here are a few that you might care about:

A Ti*k*Z shape

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

`circlen` = ⟨*n*⟩

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.

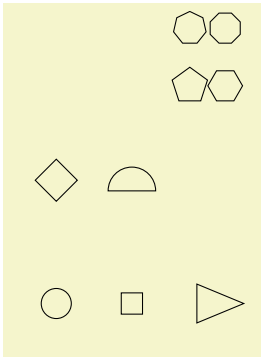


```
\begin{sseqpage}[ no axes ]
\class[circlen = 2](0,0)
\class[circlen = 2,fill](1,0)
\class[circlen = 3](0,1)
\class[circlen = 4](1,1)
\end{sseqpage}
```

`newellipse`
`ellipse ratio` = ⟨*ratio*⟩

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 Ti*k*Z shapes in the shapes library, which you can load using the command `\usetikzlibrary{shapes}`. The following are some examples:

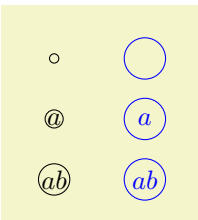


```
\begin{sseqpage}[ no axes, classes = { inner sep = 0.4em },
                  class placement transform = { scale = 1.8 },
                  yscale = 1.63 ]
\class(0,0)
\class[isosceles triangle](2,0)
\class[rectangle](1,0)
\class[diamond](0,1)
\class[semicircle](1,1)
\class[regular polygon, regular 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 Ti*k*Z manual for more information.

`minimum width` = ⟨*dimension*⟩
`minimum height` = ⟨*dimension*⟩
`minimum size` = ⟨*dimension*⟩
`inner sep` = ⟨*dimension*⟩
`outer sep` = ⟨*dimension*⟩

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:

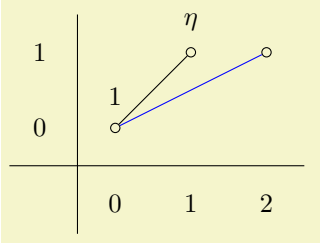



```
\begin{sseqdata}[ name = minimum width example, no axes, yscale = 0.8 ]
\class["ab"](0,0)
\class["a"](0,1)
\class(0,2)
\end{sseqdata}
\printpage[ name = minimum width example ]
\printpage[ name = minimum width example,
  change classes = { blue, minimum width = width("ab") + 0.5em } ]
```

`name` = ⟨*node name*⟩

The `\class` command makes a Ti*k*Z node on appropriate pages. You can refer to this node using Ti*k*Z 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`.

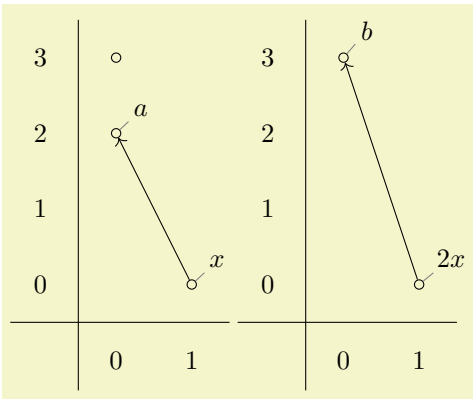
Named classes are immune to coordinate transformations. For example, in the following code, `xshift` does not apply to the nodes specified by `(id)` and `(eta)` but does apply to the coordinate specified by `(1,1)`:




```
\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` = ⟨*label options*⟩

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.



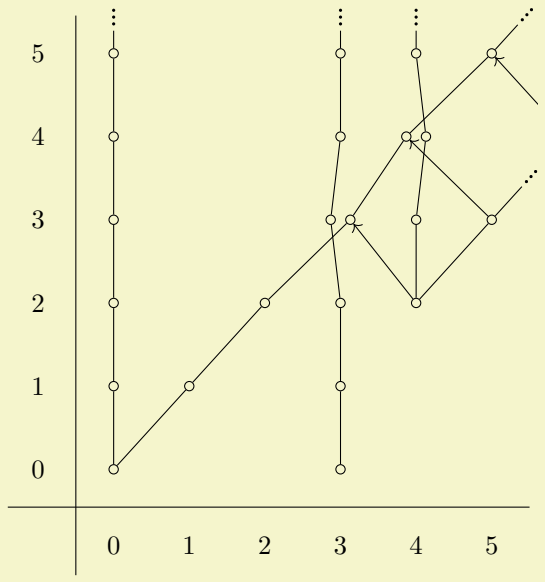

```
\begin{sseqdata}[
    name = show name example,
    this page classes = { show name = { above right, pin } }
]
\class[ name = a ](0,2)
\class[ name = b ](0,3)
\class[ name = x ](1,0)
\d2(x)(a)
\replacesource[name=2x]
\d3(x)(b)
\end{sseqdata}

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

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

`tag` = ⟨*tag*⟩

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:

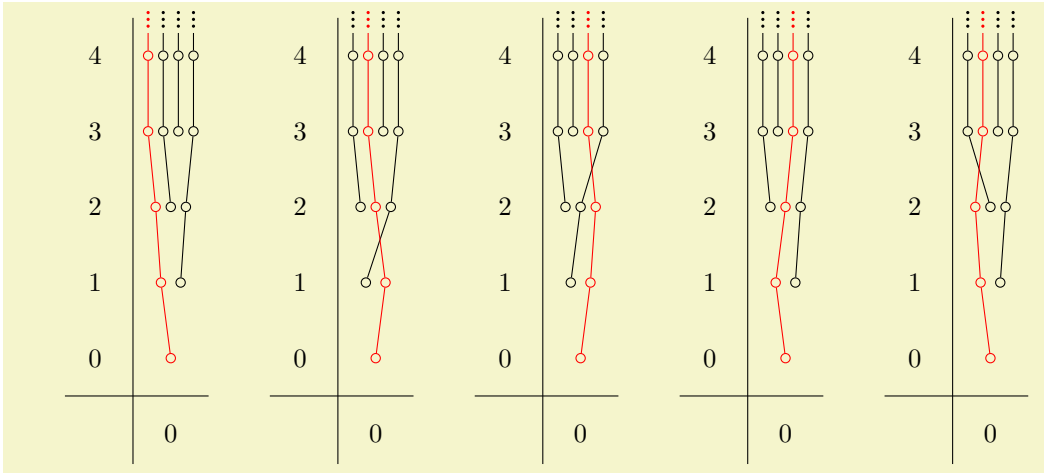



```
\DeclareSseqGroup\tower {} {
    \class(0,0)
    \foreach \i in {1,...,11} {
        \class(0,\i)
        \structline(0,\i-1,-1)(0,\i,-1)
    }
}
\NewSseqGroup\hvee {} {
    \tower(0,0)
    \foreach \i in {1,...,11} {
        \class(\i,\i)
        \structline(\i-1,\i-1,-1)(\i,\i,-1)
    }
}
\begin{sseqpage}[ 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` = ⟨*integer*⟩

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:

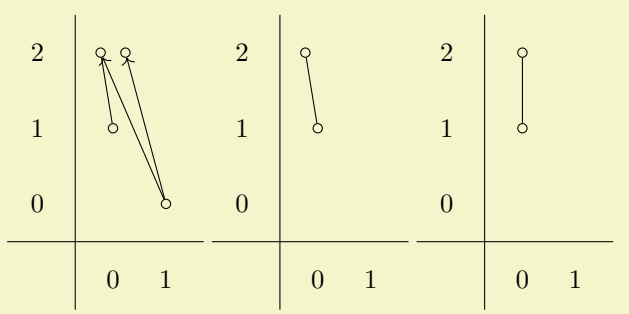


```
\DeclareSseqGroup \tower {} {
    \class(0,0)
    \DoUntilOutOfBounds {
        \class(\lastx,\lasty+1)
        \structline
    }
}

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

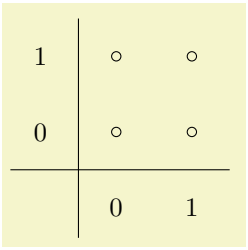
`tooltip` = ⟨*text*⟩

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:

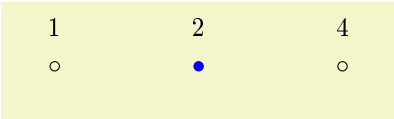



```
\begin{sseqpage}[classes = {tooltip = {(\xcoord,\ycoord)}}]
\class(0,0)
\class(0,1)
\class(1,0)
\class(1,1)
\end{sseqpage}
```

There's another example at the beginning of the section on the class stack.

page = ⟨*page*⟩--⟨*page max*⟩
generation = ⟨*generation*⟩--⟨*generation max*⟩

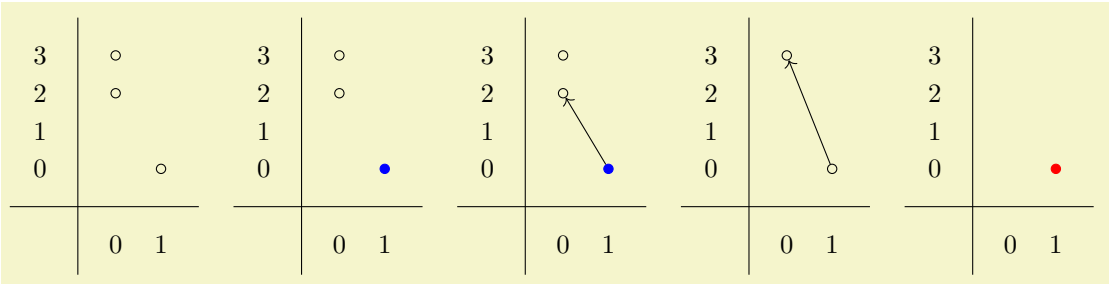
These options only work in \classoptions. The page option gives a range of pages for which the options apply to. If only one page is specified, it is the minimum page and the option applies to all larger pages.




```
\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 ] \qquad
\printpage[ name = page_example, page = 2 ] \qquad
\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.



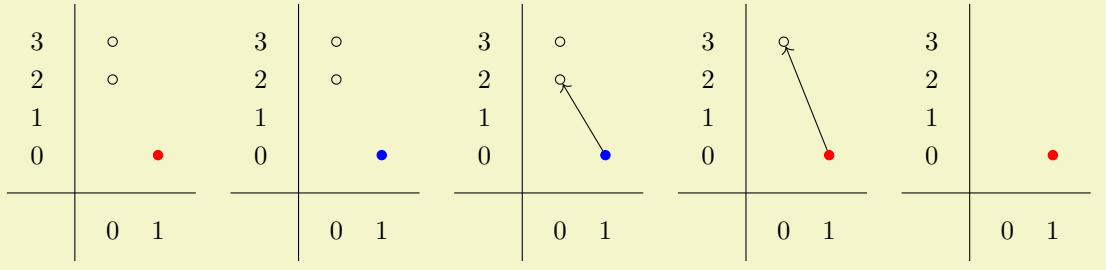
```
\begin{sseqdata}[ name = page_example2, Adams grading, xscale = 0.6, yscale = 0.5 ]
\class(0,2)\class(1,0)
\d2(1,0)
\replacesource
\class(0,3)
\d3(1,0)
\replacesource
\classoptions[fill, red](1,0) % (a) applies to most recent (last) generation.
\end{sseqdata}

\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`

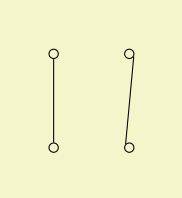
These commands represent the `x` and `y` coordinate of the current class when used in class options. The only use I have for them is in the `tooltip` option, but maybe there is some other purpose for them.

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 Ti*k*Z options that you could apply to a Ti*k*Z "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 = ⟨*anchor*⟩
target anchor = ⟨*anchor*⟩

Because you can't use the normal Ti*k*Z mechanism for specifying the source and target anchors, SPECTRALSEQUENCES has these two keys for `\d`, `\structline`, and `\extension`:



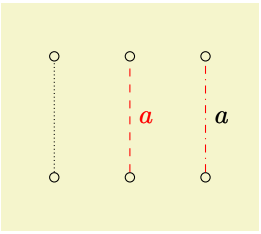
```
\begin{sseqpage}[ no axes, yscale = 1.24 ]
\foreach \x in {0,1} \foreach \y in {0,1} {
    \class(\x,\y)
}
\structline(0,0)(0,1)
\structline[ source anchor = north west, target anchor = -30 ](1,0)(1,1)
\end{sseqpage}
```

shorten > = ⟨*distance*⟩
shorten < = ⟨*distance*⟩

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

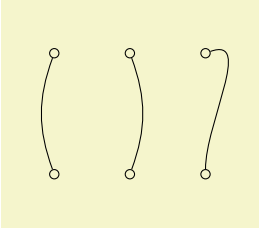
Dash patterns:

See the Ti*k*Z manual for a complete explanation of the dash pattern related options. Some examples:




```
\begin{sseqpage}[ no axes, yscale = 1.6 ]
\foreach \x in {0,1,2} \foreach \y in {0,1} {
    \class(\x,\y)
}
\structline[densely dotted](0,0)(0,1)
\structline[dashed, red, "a"](1,0)(1,1)
\structline[dash dot, red, "a" black](2,0)(2,1)
\end{sseqpage}
```

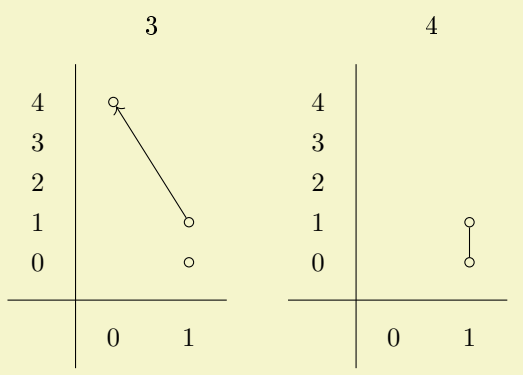
bend left = ⟨*angle*⟩
bend right = ⟨*angle*⟩
in = ⟨*anchor*⟩
out = ⟨*anchor*⟩



```
\begin{sseqpage}[no axes,yscale = 1.6]
\foreach \x in {0,1,2} \foreach \y in {0,1} {
    \class(\x,\y)
}
\structline[bend left = 20](0,0)(0,1)
\structline[bend right = 20](1,0)(1,1)
\structline[in = 20, out = north](2,0)(2,1)
\end{sseqpage}
```

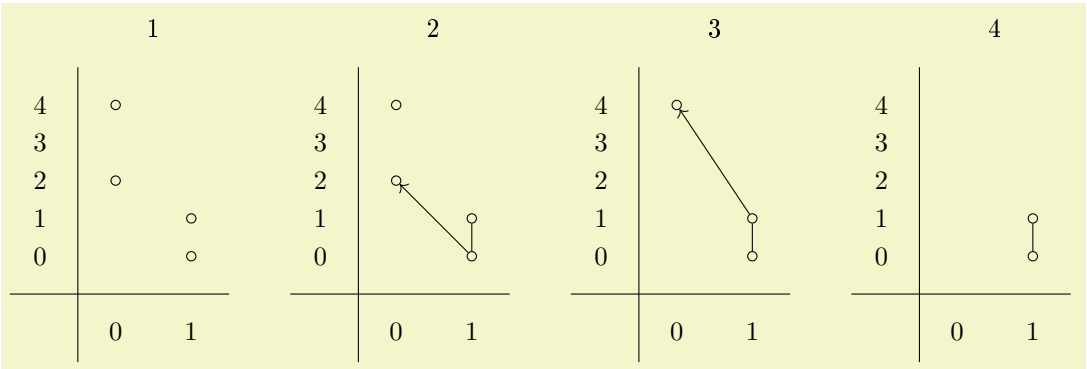
page = ⟨*page*⟩--⟨*page max*⟩

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:



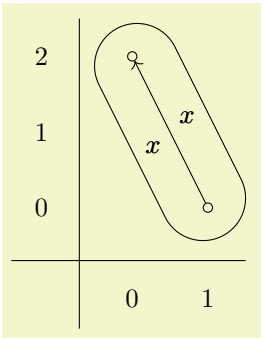

```
\begin{sseqdata}[ name = structpage example2, title = \page, yscale = 0.5 ]
\class(0,2)
\class(0,4)
\class(1,0)
\class(1,1)
\d2(1,0)(0,2) \replacesource
\d3(1,1)(0,4) \replacesource
\structline[page = 2](1,0)(1,1)
\end{sseqdata}
\printpage[ name = structpage example2, page = 1 ]
\qquad
\printpage[ 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` = ⟨*ratio*⟩ (initially `1.2`)

By default, the shape drawn by `\circleclasses` is a "newelipse" which is a custom defined shape that respects the option `elipse 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`
`extension style`

See the corresponding entry in the Ti*k*Z primitives section.

`page` = ⟨*page*⟩--⟨*page max*⟩

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.

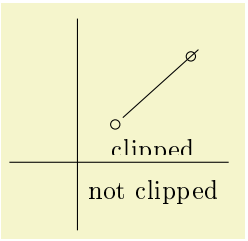
4.5 Options for Ti*k*Z primitives

`background`

This key instructs SPECTRALSEQUENCES to put the current Ti*k*Z primitive in the background. The way that the spectral sequence is printed is as follows:

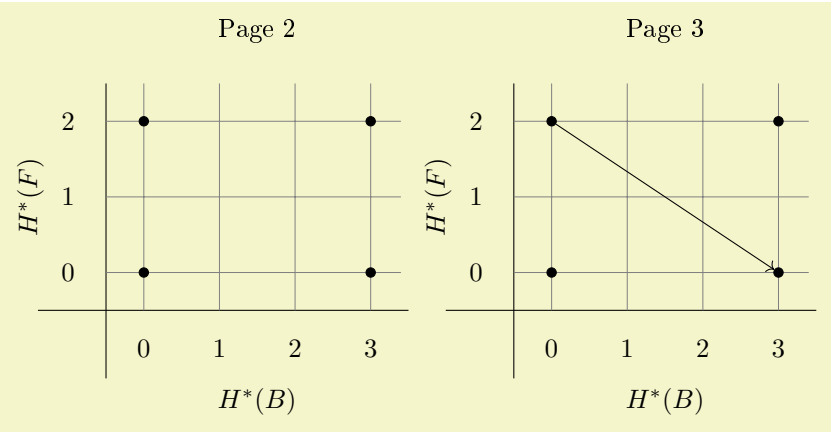
- The title, axes, axes ticks, and axes labels are printed (the appropriate steps are skipped when the `no title`, `no axes`, `no ticks`, or `no labels` keys are used or if no title or axes labels are provided).
- The Ti*k*Z background paths are printed.
- The clipping is inserted (unless the `no clip`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

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



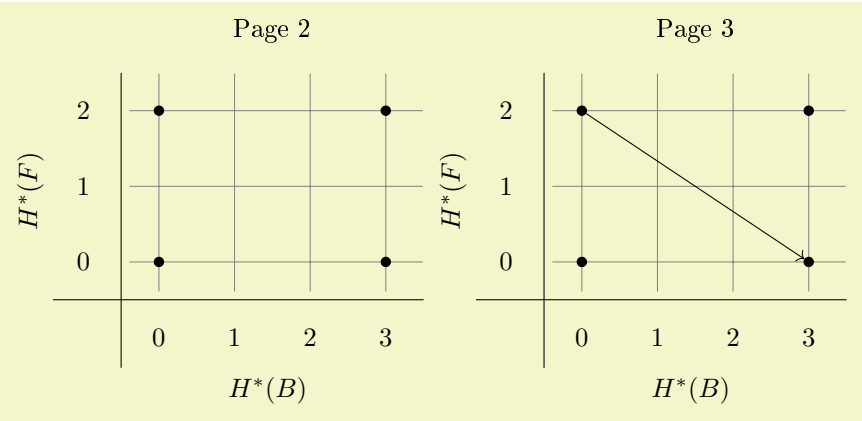

```
\begin{sseqpage}[ no ticks, yscale = 0.9, math nodes = false ]
\class(0,0)
\class(1,1)
\begin{scope}[background]
\draw(0.1,0.1)--(1.1,1.1);
\end{scope}
\node[background] at (0.5,-1) {not clipped};
\node at (0.5,-0.4) {clipped};
\end{sseqpage}
```

Here is an example where Ti*k*Z 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`:

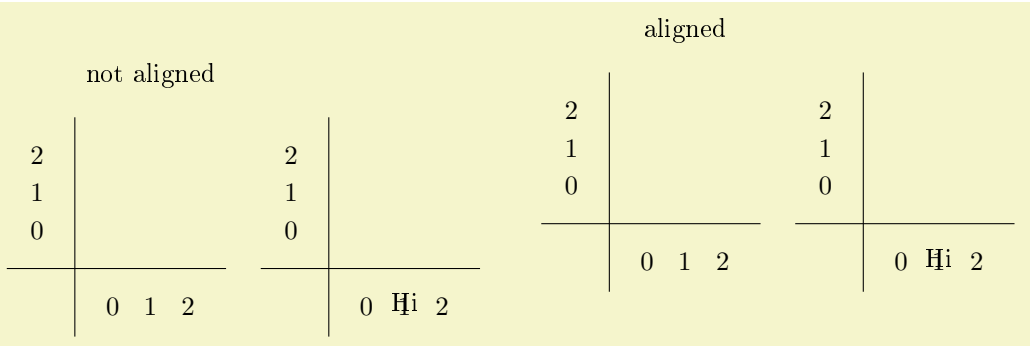


```
\begin{sseqdata}[ name = tikz background example2, cohomological Serre grading, classes = fill,
                  grid = go, title = { Page \page }, x label = { $H^*(B)$ }, y label = { $H^*(F)$ } ]
\class(0,0)
\class(3,0)
\class(0,2)
\class(3,2)
\d3(0,2)
\end{sseqdata}
\printpage[name = tikz background 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 = boundingboxex, x range = {0}{2}, y range = {0}{2}, scale = 0.5 ]
\end{sseqdata}
\printpage[ name = boundingboxex, title = not aligned ]
\quad
\printpage[ name = boundingboxex, x label = Hi ]
\qquad
\begin{sseqpage}[ name = boundingboxex, keep changes, title = aligned ]
\path[background] (\xmin,\ymin-4) -- (\xmax,\ymax+2);
\end{sseqpage}
\quad
\printpage[ name = boundingboxex, x label = Hi, title = {} ]
```

`page constraint` = ⟨*predicate*⟩
`page constraint or` = ⟨*predicate*⟩

This places a constraint on the pages in which the Ti*k*Z 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 Ti*k*Z 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 Ti*k*Z 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}}
```

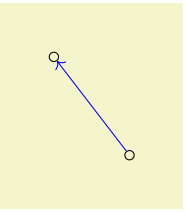
Writing

```
\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).

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

These classes apply the styling of the corresponding element to your Ti*k*Z 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

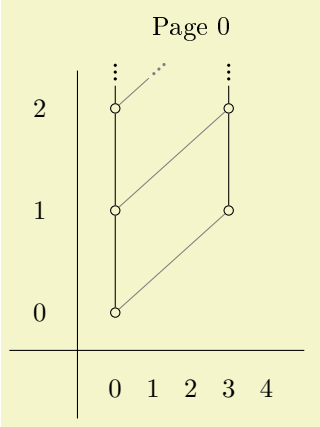
`\sseqset{`⟨*keys*⟩`}`

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.

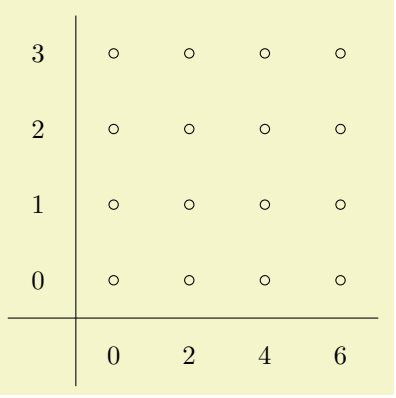
5.2 Code reuse commands

\foreach

This command is from Ti*k*Z 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

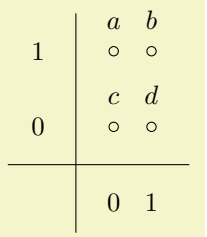
`\foreach \x in {\meta{xmin},\meta{xmin}+\meta{xstep},...,\meta{xmax}}\marg{loop body}`.

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:



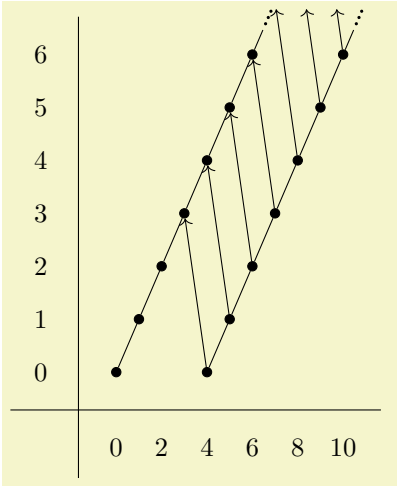

```
\begin{sseqpage}[ xscale = 0.5 ]
\foreach \x/\y/\label in {0/1/a,1/1/b,0/0/c,1/0/d}{
    \class["\label" above](\x,\y)
}
\end{sseqpage}
```

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.

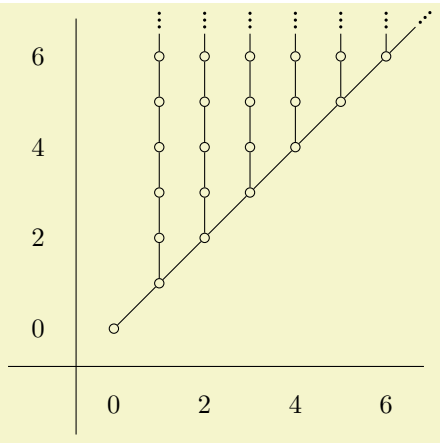
`\Do`{⟨*iterations*⟩}⟨*loop body*⟩
`\DoUntilOutOfBounds`⟨*loop body*⟩
`\DoUntilOutOfBoundsThenNMore`{⟨*extra iterations*⟩}⟨*loop body*⟩
`\iteration`

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}\marg{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 `O{}r()`. The argument type `O{`⟨*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{ O{} 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 `O{}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 `O` 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.