The Spectral Sequence Chart File Format

Hood Chatham

February 15, 2020

This is a JSON format. First I will describe the main concepts from a practical perspective, describing they are for.

1. A *node*. This describes a glyph, the way the glyph should be colored, and the way it should be filled. Nodes do not correspond to any specific type of mathematical data, though if the E_2-page consists of objects in a certain abelian category, the nodes MAY correspond to the set of indecomposable objects in that category.

2. A *class*. The classes indicate the mathematical E_2-page of the spectral sequence (or the E_n page for some different starting value n). How exactly the classes indicate the mathematical E_2 page is up to the discretion of the person or program that makes the chart. In principle, classes correspond to indecomposable summands of the E_2-page. If the E_2 page is a vector space, then perhaps every class corresponds to a basis element. On the other hand, a single class might indicate the entire (nonzero) vector space, and you might use different glyphs to distinguish the dimension.

   A class has (x, y) coordinates, a sequence of "page transitions" on which the display of the class changes (generally because it interacts with a mathematical differential, though it need not interact with a display differential). For each page range, the class has an associated node describing the way the glyph should look in that page range. The class may also have names, labels, tooltips, positioning settings, cross references against a linear algebra "back end" data set, etc.

3. A *chart differential*. A chart differential is a line that is drawn between a pair of classes on some particular page. Chart differentials indicate roughly what the mathematical differentials in the spectral sequence do. They only roughly indicate the mathematical differentials are matrices and it is not in general possible to represent a matrix with a set of lines. Frequently, the chart differentials only indicate the rank of the mathematical differential.

4. A *structline* is a line that is drawn between a pair of classes on some particular range of pages. Structlines generally indicates some sort of extra mathematical structure on the E_2 page of a spectral sequence, usually multiplicative structure but not necessarily. As with differentials, the structlines only very coarsely approximate the mathematical structure they indicate. It is also possible that the structlines just serve as visual guides to make it easier to group classes together and see patterns and do not have any mathematical meaning. A structline has a source class, a target class, an optional page range.

5. A *chart extension* is a line that is drawn between a pair of classes on page ∞. Chart extensions generally indicate mathematical extensions either in the additive or multiplicative structure of the spectral sequence. As with all of the other things, the connection between chart extensions and mathematical extensions is up to the discretion of the person or program producing the charts.

Our goal is to be able to display spectral sequence charts in different target formats. Our targets are HTML5 canvas, svg, and tikz. Part of the challenge here is that the features and behavior of these different output formats are quite different. HTML5 canvas has interactivity, good tooltip support, scaling, panning, event handlers, etc. However, it needs to have good draw speed so it can update in real time.

TikZ has much better support for arrowheads and fine tuned drawing. However it is not interactive and scale has to be fixed when the code is generated. Even if we can automatically turn a HTML5 view into a similar looking LaTeX view, it's a challenge to try to determine whether the converted output will look good. Getting good output will probably require manual adjustment, which could be done like the way spectralsequences charts are adjusted now. The amount of time and annoyance this takes depends on getting a reasonably accurate picture out of Canvas though – with Canvas we can manually adjust on the fly with immediate feedback whereas LaTeX requires running the code to extract the latex, running the latex compiler (slow) and then inspecting the pdf output.

Because of the differences between Canvas, SVG, and tikz, there will necessarily be a decent number of Canvas or tikz specific options.

Now I will describe the data structures. TODO: Some of the following objects have names that may clash with preexisting key words in certain languages (`ChartClass` is the biggest offender). (1) Is this a problem for any of them? (2) If it is a problem, what is a better name. Currently I've added the prefix `Chart` to anything that has a prior math meaning to distinguish from that math meaning. Perhaps the same could be done to distinguish from programming meanings? Maybe add `Chart` as a prefix to everything uniformly?

TODO: What should we stipulate for invalid values? Requiring crashes in as many situations as possible probably is the best way of maintaining future flexibility, because then no existing files will have invalid values that lead to unspecified behavior. I think that we should allow programs to do two things (1) NEVER inspect a field or (2) check the field invariants and crash if they are invalid on start. I should probably start by making a validator that checks the full specification. Then programs could ensure they are compliant with the crashing conditions by running the validator on parsing a file before doing other work and before writing a JSON output file.

Also, are do file formats usually contain requirements of programs that manipulate them, say regarding how they behave when the invariants of the file format aren't satisfied?

1. This isn't a type or anything, but `infinity` is going to be a specific large positive `int`. I have used 10000 at various points, but I am not completely in love with that choice. It's hard to turn infinity into a number. Maybe something like "14514179" would stand out clearly in the files. A better option?

2. `Color`. This is a `string`. Question: how does one specify a color in a way that behaves consistently between HTML/svg and tikz? Special value `"None"`.

3. `ArrowSpecifier` This is a `string`. TODO: What sort of string? We should probably just use the tikz setup. Indicates the styling desired of the tip and tail of an arrow.

4. A `ChartShape`. This should be a function that takes a stroke color and a fill color, a scale factor, a center, and possible additional parameters specific to the glyph and draws an appropriate picture on the page. Question: How do we define a glyph so that it is (a) Textually represented (b) Flexible (c) Works in the various possible output formats. Probably we just have to specify representations in each of the targets we want to support. We should define a shape class for HTML5 canvas/javascript output format so that we can instantiate the javascript shape class from the JSON shape object. In tikz a shape should turn into a tikz node. Shapes come with a name and are referred to by name. Until we figure out the details of the shape format, we can offer a collection of built in shapes.

5. A `ChartNode` is an `object` with the following fields:

   - `shape`: a `string` the name of a `ChartShape`.
   - `fill` (optional): A `Color` the fill color. Default: `"none"`.
   - `stroke` (optional): the stroke color. Default: `"black"`.
   - `scale` (optional): the scale. Will be applied on top of scaling from other sources. Default: 1.
   - `Opacity` (optional). A `float` between 0 and 1. The opacity. Default: 1. (TODO: Should floats outside of the range be clipped? Crash? Undefined?)

6. A `ChartClass` is an `object` with the following fields:

   The `ChartClass` local coordinate system (TODO: where does this go?) Each `ChartClass` has a local coordinate system where $(0, 0)$ is the center of the `ChartClass`. This is achieved from the `global_coordinate_transform` by (1) translating so that the center of the `ChartClass` is at $(0, 0)$ then (2) scaling so that TODO how much do we scale by?.

   - `x`: An `int`. The x coordinate in the chart.
   - `y`: An `int`. The y coordinate in the chart.
   - `xoffset` (optional): a `float`. Default: 0. An offset from where the class would normally be centered.
   - `yoffset` (optional): a `float`. Default: 0. An offset from where the class would normally be centered. (We need to ensure that these offsets are interpreted as consistently as possible between output formats so that if the HTML5 version of the chart looks good, the tikz version of the chart will look good). `transition_pages` : a `list` of `int [int]` of pages where the class transitions. Invariant: strictly increasing.
   - `nodes` : a `list` of `int` or `none [int | none]` indexing into the list of `ChartNode`s. Invariant: `len(nodes) == len(transition_pages+1)`. Invariant: Negative or out of range node indexes are INVALID. These nodes control the class display on each page range $0 \leq x <$ `transition_pages[0]`, `transition_pages[1]` $\leq x <$ `transition_pages[2]`, ..., `transition_pages[len - 1]` $\leq x <$ `infinity`. If an entry is `null`, then no class will be displayed in that range.
   - `tooltip` (optional): a `string`. Should support latex. TODO: define new line symbols. (Default: none).
   - `labels` (optional): TODO: Define how these are specified. Main use case is tikz target, but we want the HTML5 target. (Default: none)

7. An `ChartEdge` is an `object`. All edges will be of the subtype `ChartDifferential`, `ChartStructline`, or `ChartExtension`.

   The `ChartEdge` local coordinate system (TODO: where does this go?) Each `ChartEdge` has a local coordinate system where $(0,0)$ is the center of the `source` of the `ChartEdge` and $(1,0)$ is the center of the `target` of the edge. This is achieved from the `global_coordinate_transform` by (1) translating so that the center of the `source` of the `ChartEdge` is at $(0,0)$ then (2) scaling so that the edge has unit length and (3) rotating around the origin so that the target of the edge is at $(1,0)$.

   - `source_class` : An `int`, an index of a into the list of `ChartClass`es. Invariant: Negative or out of range nodes are INVALID.
   - `target_class` : An `int`, an index of a into the list of `ChartClass`es. Invariant: Negative or out of range nodes are INVALID.
   - `color` (optional): A `Color`. Default: depends on global fields of the parent `SpectralSequenceChart`.
   - `opacity` (optional): A `float` between 0 and 1. The opacity of the edge.
   - `bend` (optional) : A `float`. Makes a bezier curve from the source to target, the bigger the number the more the arrow should curve rightward (facing from source to target). Negative is a leftward bend. TODO: Define this in terms of Bezier control points. Need to look up what Tikz does. (No default.)
   - `control_points` (optional): An `object`. Two possibilities depending on whether you want a cubic or quadratic Bezier curve:

     `{x : <float>, y:<float>} | {x1 : <float>, y1 : <float>, x2 : <float>, y2 : <float>}`.

     These points are interpreted in the local coordinate system of the edge. (No default.)
   - `arrow_type`: An `ArrowSpecifier`. Controls the arrow tip and the arrow tail. (Default specified by Global options.)

8. A `ChartDifferential` is an `ChartEdge` with the following additional fields:

   - `page` : An `int`. The page on which the differential occurs. Invariant: $0 <$ `page` $<$ `infinity` where `infinity` is the special integer defined above.

9. A `ChartStructline` is an `ChartEdge` with the following additional fields:

   - `max_page` (Optional): An `int`. The maximum page on which the `ChartStructline` may be drawn. Invariant: $0 <$ `page` $\leq$ `infinity` where `infinity` is the special integer defined above. (Default: `infinity`).
   - `min_page` (Optional): An `int`. The minimum page on which the `ChartStructline` may be drawn.

10. A `ChartExtension` is an `ChartEdge` with no additional fields.

11. A `SpectralSequenceChart` is an `object` with the following fields:

    - Range fields: `x/y min/max`: Maximum range `start_view_x/y min/max`: Starting range, only for interactive viewers. `tikz_x/y min/max` Tikz range, fallback to `x/y min/max` if it is missing.

- `shape_list` : The `list` of `ChartShape`s.
- `node_list` : The `list` of `ChartNode`s.
- `class_list` : The `list` of `ChartClass`es.
- `differential_list` : The `list` of `ChartDifferential`es.
- `structline_list` : The `list` of `ChartStructline`s.
- Handler fields for HTML5 Canvas? TODO: Does this belong here? If so, how should they be set up? (Javascript string function declaration? Viewer dependent?)
- `grid_type` : A `string`. TODO: what are the possible options? Does Canvas support it or only tikz? Maybe they should have separate fields?
- Various other tikz global options. (Maybe we can use a "`\printpage`" style command and insert these global tikz controls there?)