Notes on creating gradients.

Chroma.js

mix(c1,c2,d,cSpace) returns a color d along the line from c1 to c2 in the specified color space.

Brewer.<name> Returns the color brewer scale associate with that name

(has breaks and k-means clusters, but only in 1D)

F=scale(cArray) //creates a scale function. F(d) returns the color at d.

F = scale(cArray).domain(dArray) //assigns distances in dArray to cArray colors

F = scale(cArray).domain(dArray).mode(cSpace) //’lab’, ‘lch’

F = scale(‘brewerName’)

Scale().correctLightness() Redistribute the lightness to be uniform (I suspect)

B= Bezier(cArray) //does a Bezier interpolation, looks like it risks doing too much smoothing

Cubehelix() //a twist through color space,

.limits(data,’k’,5) But I think this is 1D data only.

Model

Create interpolation points using the palette tool. Use the various scale operators to generate more colors. Need:

* nSteps: read/write text box. This shows the number of points to generate
* colorspace to use for interpolation: lab, lch
* CorrectLightness (operator).
* Reverse order (operator)
* A sequential palette has some key colors and some interpolated colors. They are in two separate arrays in the palette. It is optional to show the key colors
  + Ideally, we would distinguish and only allow editing of the original colors.
  + Cases:
    - We want to assert it is a sequential, but display and edit the key colors like any other palette
    - We want to generate a sequence, view and edit it only
    - We want to generate a sequence, view it and its seeds, edit the seeds, regenerate
* Save Interpolated creates a new palette with all the colors in it.
* We assign state.colors to state.palette.eColors, for conciseness. The displayed colors are always state.colors. For the overlay, we have a challenge with the current plotting routines. Either we combine them in state.colors, or expand them to add the overlay
* A diverging palette is defined by two sequential palettes and a center. Could we do this with two pointers?
* Read reads only seeds, uses the type in the dropdown as the default, overwrites it when reading XML
* Write needs to save the interpolated values (the actual palette), but we also need a way to read/write the seeds. Could tie this to what we see? Let’s try a radio: Input, Interpolated, Both. If both on write, create two separate palettes.

Scaling and filtering

* Map range[a\* b\*] to x,y.
* Given aMin, aMax, bMin, bMax, pWidth, pHeight.
  + Find AtoW, BtoH. (Max-Min)/wSize
  + From aMin and bMin, compute orig in a,b, then convert to pixels using function above
* Created display controls for this as a global field for now, want to add controls soon.
* Filtering should filter the data based on ranges of L,C and H.
* Fit to data check box is any easy way to implement some of the most critical cases right away (formatting lights, tinted grays, etc.)

Display

We have three display procs

* AllPalettes: Let’s assume this will show the input colors only for now
* displayColorspace: The LAB plots.
  + Input only: Works by default
  + Generated only: needs to use a different array. Would just setting state.colors to the gColors work?
  + Both: This is the tricky one. Ideally, we have a way to distinguish the two. Need to first plot the gColors, then overlay the input
  + Selection: index into state.colors. If we let state.colors be gColors, then select/deselect might work.
* Reset needs to clear the gColors