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)

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 display
* colorspace to use for interpolation: lab, lch, hsv
* CorrectLightness (operator).
* Reverse order (operator)
* A sequential palette has some key colors. The interpolated colors cannot be selected, though they are tool-tipped, so they are created only when the displays are drawn. How many colors are displayed depends on nColors. Changing nColors redisplay. Setting nColors to any number less than the key colors shows only the key colors
* I want a display where the key colors show differently than the interpolated ones. Maybe a shape choice
* Writing out such a palette writes all the colors.
* New from Interpolated creates a new palette with all the colors in it.
* A diverging palette is defined by two sequential palettes and a center. Could we do this with two pointers?
* We assign state.colors to state.palette.eColors, for conciseness. The displayed colors are always state.colors. For the

Other notes

* For the formatting lights, need to scale the LAB plots to the data. Or some arbitrary amount.
* Transform range(a,b) to abSize (in pixels). Currently abSize/230. Need aRange and bRange, + and – extent. Tranform range of L\* to Lmin, Lmax (abSize-5-Math.round((abSize/105)\*lab[0]))). Would be nice to show min/max on the graph

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