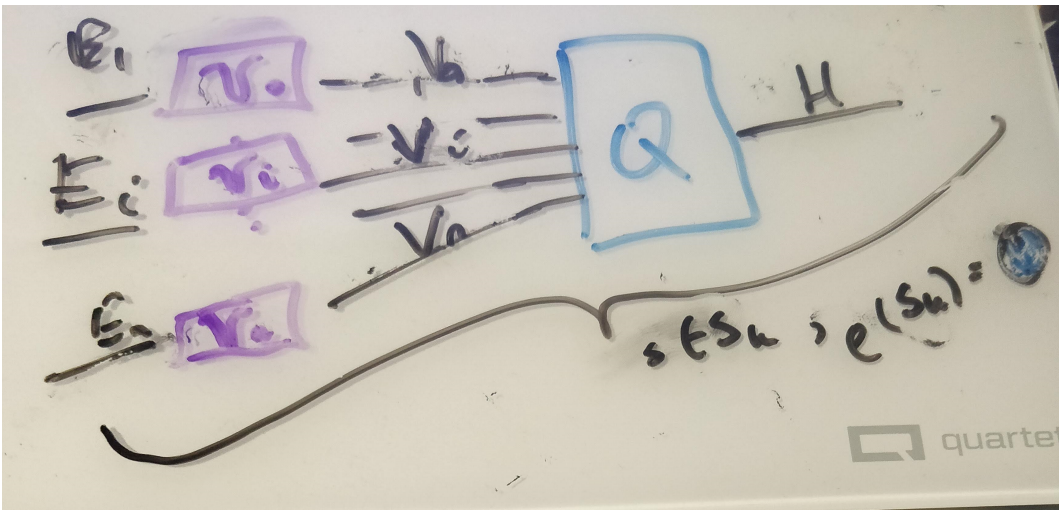

1 Matplottoy

We build on top of the existing Matplotlib architecture [1] so that we can initially focus on the data to graphic transformations and rely on Matplotlib for the other graphical elements of the visualization and the rendering. For a primitive graph such as a bar, scatter, line, our path is: python

E = Section() V = 'parameter': ('field': transformer)

fig, ax = plt.subplots() artist = Q(E, V) ax.add$_artist(artist)$

1.1 Data: $\mathcal{E}$

We propose a semantic markup of the fiber and connectivity that we will later use to check the constraints of the ν and Q. python class FiberBundle: def $_init_{(self,K,F):"""K:'tables':[]F:'fieldname':(type,monoidaction)"""}$

def is$_section(self, section)$: """$checksifasectionisfromagivenfiberbundle : arevaluesinF, arekeysinK"""pass$

We add a monoid field to the schema like structure of the fiber since the monoid actions are the constraints on the ν and Q. We implement K using the triangulization scheme described in section **??**. This means we expect data to be provided as tables where the name encodes the connectivity:

vertex - disconnected points

edge - 1D continuity along the edge

face - 2D continuity along the face

Nested continuity is encoded in the fiber. For example, one way to encode a movie is a 1D timeseries of Frame objects, where each Frame is a nm continuous array.

1.1.1 Sections

As described in section **??**, the data values are encode as the section of the fiber bundle. We implement this as as a wrapper class around a data container object of the form: python class Section: FB = FiberBundle(K, F)

def $_{init_{(self,*args):passdefview(self,simplex="vertex"):convertdataintoatomiccolumnorderreturnfieldname:data}}$

The view function is the closest analog to τ, as it returns a a data container type object. We specify field based top level indexing so that we can pair the fields with transformer ν objects.

1.2 Encoding: $\mathcal{V}$

We implement ν as encoder objects with an optional '$_{init_{.methodthatcanbeusedtospecifytherangeoftargetvisualvariables.}}$

python class Encoder: def $_{init_{(self,args):passdefvalidate(self,monoid):checkiftransformsupportsmonoidactionpassdefconvert(self,value):con}}$

The validate method specifies the monoids for which this a valid ν and checks against the specification in the fiber. The convert method converts a value from data space to an internalized normal form as described in table **??**.

1.3 Graphic: $\mathcal{H}$

We inherent from the current Matplotlib artists for our Q, which is responsible for constructing a curried fully specified internal representation of the graphic. python class Q(BaseClass): required = optional = def $_{init_{(self,data,*args,kwargs):checkthatrequiredandoptionalvisualparametersarepassedincheckthatth}}$

While equation **??** specifies that the values of μ can be generated before they are input to Q, we curry the opertaions of obtaining the data and performing the transforms to the draw method because that allows us to more easily propogate updates to the data to the screen.

1.4 Case Study: Iris(Penguins)