Visualizing Cyclic Datasets: Circular Streamgraphs Using D3.js

Kevin Woodward
University of California, Santa Cruz
CMPS 161 – Data Visualization – Alex Pang
keawoodw@ucsc.edu

Abstract

This project is a foray into D3.js to tackle the problem of cyclic data being represented as streamgraphs in a circular format. While this problem isn't exactly unexplored territory, it is a simple concept that has many paths to a correct functionality.

The only absolute goal of this project is to create the aforementioned visualization structure and to learn D3.js along the way. Left open ended, the program document suggests creativity in development. In the spirit of this, a few more features have been added to make the program more user involved.

1 Introduction

The interest in cyclic or periodic time series data sets is relevant in many contexts. The versatility of this model becomes apparent when one realizes any data set over some consistent time is representable, and that trends in fluctuations become apparent much more quickly than simply viewing a spreadsheet or table.

The project development outline is more open ended as a result of the more ambiguous guidelines. This allows expansion in a variety of directions, including but not limited to a more intricate visualization method, expanding the flexibility of the types of data that can be represented by the model, etc. Any one of these expansions alongside the barebones requirements still results in a static result that is not interactive. For this reason, the expansion will be the addition of a simple GUI for adjustment of the resulting data.

The implementation will be created as a client side web application, using pure Javascript without jQuery or Node.js. While this may seem counter-intuitive, it serves as a challenge to test development understanding.

2.1 Methods - Gathering and Formatting Data

Prior to any actual code development the aim was to identify and standardize a few sets of cyclic data. These sets were initially chosen by the nature of their context rather than their actual values. In other words, because the type of the data implied its cyclicity rather than the data itself. The data chosen were: Average Monthly Temperatures at Nottingham, 1920-1939[1], Santa Cruz tide predictions in February 2017[2] (downloaded as .txt), and US Gas Prices by Month[3]. At this point, a standard had to be developed, as the goal of the project wasn't in robustness in parsing.

The standard set was relatively common among D3.js examples of a similar nature; 3 columns in a comma separated values file, key, value, time. The key represents a single time period of data of the context of the time cycle without any overlap. For example, if the data is monthly then a key would be for a year, and if daily then a key would be for a week. The value is self explanatory for most cases. It is the value that is 'stacked' in the streamgraph. The time is the identifier for a datum within a key. So for months, this would range from one to twelve, and for weeks one through seven.

[1]https://vincentarelbundock.github.io/Rdatasets/csv/datasets/nottem.csv

[2] https://tidesandcurrents.noaa.gov/noaatidepredictions/viewMonthlyPredictions.jsp?bmon=02&bday=01&byear=2017&timelength=monthly&timeZone=2&dataUnits=1&datum=MLLW&timeUnits=2&interval=highlow&format=Submit&Stationid=9413745

[3] https://vincentarelbundock.github.io/Rdatasets/csv/quantreg/gasprice.csv

Each data set had its own issues in formatting to the created standard. The tidal data contained three to four values a day, with multiple high and low readings. Averaging the values wouldn't be consistent as the days with less values would result in uneven ratios of high and low readings, so eventually only the first high reading daily was taken. The Nottem weather data only provided a time in decimal format (e.g. 1920.89836) as well as the value. This required some Excel manipulation to convert and extract the keys and times. The gas price data was similarly formatted to the weather data and required the same manipulations.

2.2 Methods - Initial D3.js Experimentation

The final product submitted went through multiple revisions and iterations. The first pass was a stacked bar chart to get a feel for the library and understand the framework. The largest issues faced here as simply understanding how D3.js integrated into JavaScript, through what functionality the result would be rendered in the webpage, and what is considered a good or bad practice with the library. Essentially most of this was a learning process but didn't contribute to the final product directly.

For this creation, some dummy CSV files were used to represent some ideal data set. The initial idea was to parse the data into some custom object in JavaScript prior to doing any manipulation or visualization. This proved to be more difficult than expected, and it lead to finding the d3.csv() function, which requests a CSV file to load, along with an accessor and callback. While standard in JavaScript, anonymous functions and callbacks haven't been something used personally very often. D3.js forces the use of both of these to have coherent code that is readable. The accessor functionality is handled in part by D3.js in the sense that it allows the user to access the CSV file as an object and its contents as if they were added to the prototype of that object. So, the accessor of the data was simply taking the relevant values (time and value).

In certain cases, the loading of CSV files would result in a nondescript D3.js error with exclusively internal callbacks. This was resolved by performing some intense StackOverflow research to find that excess space characters within the CSV file cause the d3.csv() function to recognize certain cells as strings, and throw errors when they are applied in situations that assume the value is of type Number. This was quickly resolved by performing a cast to Number in the data accessor.

At this point, an understanding of the basics of D3.js was acquired and a basic stacked bar chart could be displayed using dummy data acquired from a sample bar chart.

2.3 Methods - Creating A Circular Streamgraph

A majority of the development from a stacked bar chart to a circular streamgraph was smooth, but the roadblocks were very time consuming. The idea was continue with the 'completed' stacked bar chart and convert to the desired result. This caused more trouble than it was worth, as the method in which the SVGs, axes, and actual data were formatted ended up being different than what was started with. Alongside this, it was realized that D3.js V3 was being used, not the latest V4, due to my initial work was based off an existing example which used V3. Due to a very large changelist[4] and complete disregard for legacy compatibility, V3 was chosen to stay for the final product.

The initial change was to predefine certain groups of nested methods that were cluttering the already extremely long chains of calls. This meant moving out certain things such as value calculations and checks that could be determined prior to doing the D3.js CSV file request. Some examples of this are defining the stack layout and subsequent methods as a var to be called later, calculating and comparing radii, and setting the nesting scale. After this, the basis for how to create a circular area needed to be figured out. The model that was eventually settled on was using radial areas to represent the actual data of the streamgraph and to use offset and rotated axes along each time point to have the data be actually readable.

The radial area initially preset to some fixed inner and outer radii values, with the angle being based on the time values within the CSV file. The visual connections between the pieces were accomplished by interpolating in the "cardinal-closed" fashion, which when combined with rescaling of certain domains mentioned later, allowed the used datasets to be continuously connected. I must credit the Spline Editor[5] example for allowing me to understand the meaning behind the various types, as the D3.js Documentation[6] for

[4]https://github.com/d3/d3/blob/master/CHANGES.md

[5] https://bl.ocks.org/mbostock/4342190

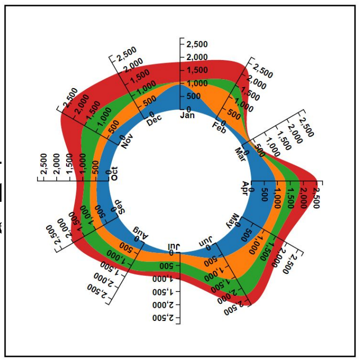
[6]

this as well as other things, is not very friendly. From here, drawing the circular values wasn't too much extra work. The main things that had to be done was nesting the data by keys, selecting all the layers, applying the path data to the layers, and filling them with a predetermined color scale (category10).

This worked fine with the dummy data I was using earlier, but caused issues with any of the other data I had acquired. D3.js would consistently throw a nondescript error. Because this only occurred on the data that was personally collected, it was concluded that it must have been related to that. After too much time it was discovered that each key in the nest hierarchy must contain the same number of values. This posed a problem, as all of the acquired data sets were not perfectly rounded off. For example, the weather data was missing a few months at the beginning of the first year and a few at the end of the last year. The simple solution to this was to trim the data sets so each cycle in the set had the correct number of values.

The axes were a bit more complicated, as D3.js does not have a native circular axis format. To accommodate the fact that the axes must fit within the 2*pi circumference range, the time scale of the angles for the data must be properly spaced. This is as simple as modifying the domain to be n divisions of size (2*pi)/n size. The range of the axes was set to these values, and SVG groups were appended to actually display the desired graphics and text. These groups were given multiple attributes, including a rotation transformation to properly align the angle of the axes to the actual points of data, a scale that was defined by the inner and outer radii to allow the axes to scale to the data properly, a "right" orientation so that the axis values were not parallel to the axis itself (rendering it unreadable), a variable number of axis ticks based on the ratio of the inner and outer radii evenly spaced between the minimum and maximum values, text that applies appropriate text to the base of each axis if it is months or weeks otherwise simply numbers (e.g. weeks corresponds to axes being Mon, Tue, Wed, etc.), and an offset from the inner radius to reduce overlap in most cases.

Following is an image illustrating how certain attributes contributed to my final visualization:



With a visual example, it is much easier to understand the

—

eference/blob/mast

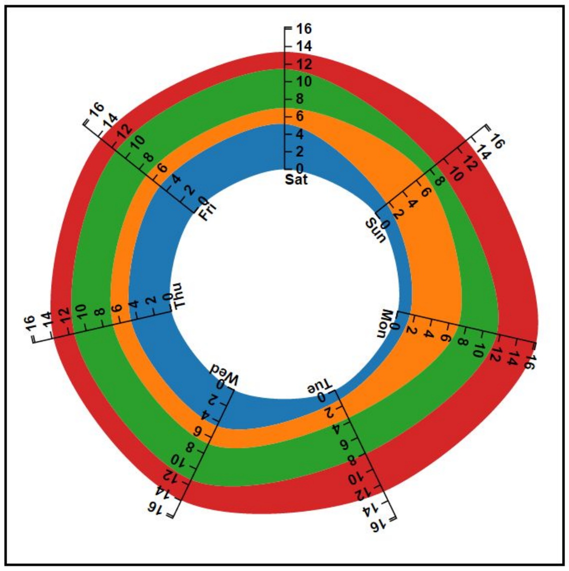
ate

contribution of each attribute, such as the month text, number of ticks, tick spacing, etc.

2.4 Methods - Visualizing The Chosen Data Sets

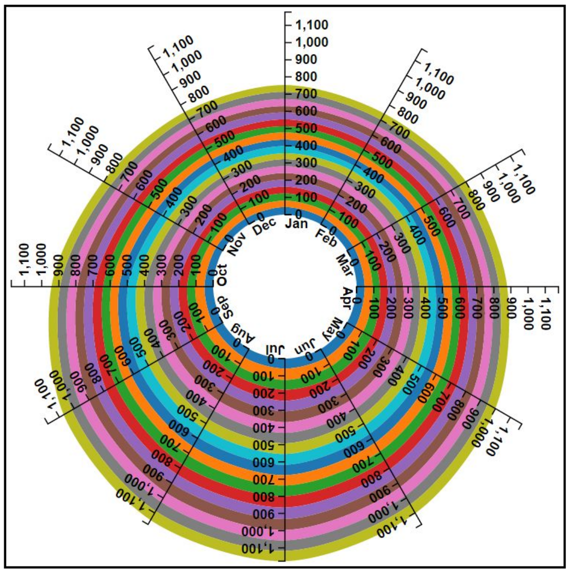
With the infrastructure to create circular streamgraphs in place, visualizing the data mentioned in section 2.1 was very simple. Due to the fact that the data chosen was only thought to be cyclic instead of confirmed to by cyclic, some of the results are relatively bland. Here are three images depicting the three data sets chosen:

Weekly tidal data: It is apparent that the values fluctuate in a cyclic pattern which generally agrees with the known trend. It is important to keep in mind that this data is a set of predictions and not actual readings, which may account for any oddities.



Monthly gas prices: This is probably the least interesting set shown. The original logic was that gas prices would fluctuate seasonally, as more gas is consumed in the colder months. While this seems to remain true, the fluctuation is almost negligible. There is barely a few hundred dollars of fluctuation over the course of entire set of twelve years.

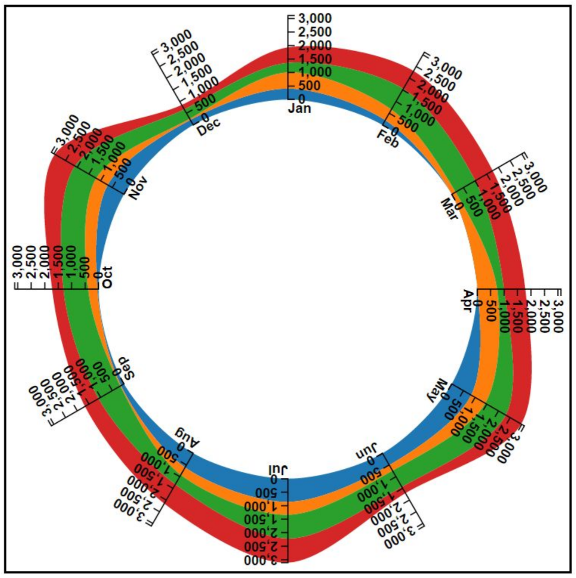
Nottem monthly weather data: It seemed like the gas price concept was a derivative of some actual fluctuation. Because of this, the idea of weather data was simple and useful. In this set the trend occurred, but with much more apparent results. It is obvious the colored area is an ellipse/egg-like shape, indicating cooler months in the winter season. While this is obvious, it was a good test to determine how well data could be interpreted from the created visual model.

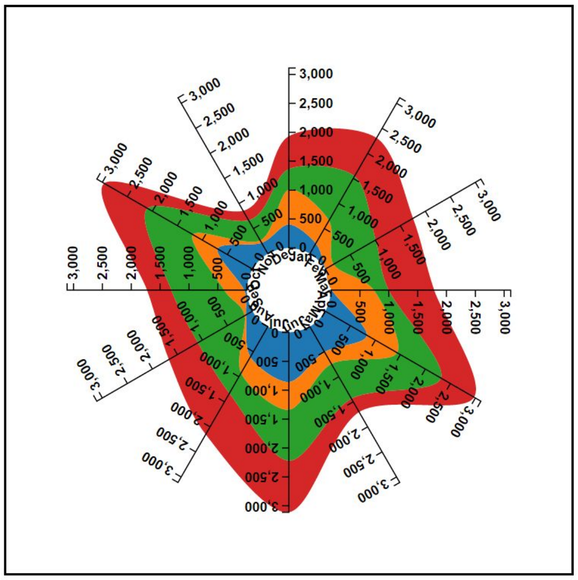


While these outputs are informative, some flexibility in visualization and some actual interesting visuals were desired.

3 Additional Features

The things I added are simple, yet make the whole process much nicer. Sliders were added for the inner radius, outer radius, and the SVG container size. Along with this, a random data button was added to showcase the range in data display the program can do. Allowing the inner and outer radius to change results in visualizations that more accurately depict the goal of what the data is trying to show. For example, here are two extreme cases:





These are two visualizations of the exact same random data. Depending on what the intention of the user is, they may or may not want to showcase certain qualities about the data. This is especially important with data sets that have subtle fluctuations, such as the weather or gas price data sets mentioned above. Having a user element to the program captures interest as well.

The random data button allows for more interesting streamgraphs than the ones resulting from real data. This was added essentially to show that this program is capable more than the output of the data shown previously. The random factor of this data in fact only applies to the value field. It bases itself on a monthly data set with 4 cycles, and a separate accessor is passed to the d3.csv() function which randomizes the fields from 0 to 1000.

Lastly, the SVG dimension resize slider is simply to create as small or large of an image as desired.

4 Limitations

There are a few limitations that aren't strictly issues, but that should be resolved in an optimal setting.

CORS errors. Due to the nature of web security, Google Chrome doesn't allow file loading from alternate sources. This is the reasoning behind the buttons for datasets, rather than an input for uploading. While uploads wouldn't work at all, having the files stored in the same directory of the project did as long as the file is not being loaded locally. This is again the reasoning for the "Nothing showing up?" prompt on the program.html page. The only way to get around this was to host the project on my UCSC page[7], which the local file redirects to, as well as the report page.

Narrowness of the input data. Ideally this program should be able to take in CSV files that don't have to strictly adhere to the arbitrary standard set. Realistically this is just more code for parsing, and doesn't actually directly improve the visualization.

Redrawing. I could not find a way to effectively traverse and update the SVG groups after recalculating due to a change in the radii or SVG dimensions. The suboptimal fix applied was to delete all children from the svg DOM element and redo the whole calculations. While this isn't optimal, it doesn't result in a performance decrease. However, it is the cause of the flashing effect that is visible on changing a slider.

Continuity with other data. As mentioned before, the data sets used had to be trimmed so the same number of values were in every key for nesting purposes. Even if this weren't an issue, it would be a messy problem to display a missing data segment using the current method.

5 Conclusion

The design and creation of this program did allow the interpretation of data sets and their cyclicity. Prior to this, it would have been difficult for me to interpret the data represented in the project and to determine if it was periodic in some way. While it was useful, there are little to few advantages over a regular streamgraph, aside from visual sugar.

D3.js is a valuable tool to be familiar with, and in many ways it seems like that knowledge is the main benefit of this project.

There are obvious improvements to be made given enough time, but most of them were not in the scope of this project. Overall this project was a success and fulfilled the requirements while adding a small set of extra features.

[7] https://people.ucsc.edu/~keawoodw/