Visual Analytics
Practical 3

Michael Weisz

February 20 – Hilary Term 2017

Summary

This report summarises my work on the third practical *Parallel Coordinates Plots* as part of the *Visual Analytics* course at Hilary Term 2017 at Oxford University.

I have attempted levels 1 - 3 with all the required functionality such as the reordering of axes, color-coding as a tool for grouping related data points, and brushing. The subtleties of each level including the relevant excerpts of the source code are described in detail in the corresponding sections.

For the visualisations, I used the *D3.js*[1] charting library. In addition, I made use of the following existing examples and external source code

- Parallel Coordinate Plot in D3 repository [2]
- Parallel Coordinate Plot by Jason Davies [3]
- Line Chart by Mike Bostock [4]
- ColorLuminance Javascript Function [5]

Level 1

This level asked for an adaptation of an existing parallel coordinate plot in order to visualise the data provided in the file *WorldExconomy2015.xlsx*.

The resulting plot contains seven axes for each of the 19 data points and is shown in figure *1*. It supports the reordering of axes and a brushing functionality.

[1] `https://d3js.org`
[2] `http://mbostock.github.io/d3/talk/20111116/iris-parallel.html`
[3] `https://bl.ocks.org/jasondavies/1341281`
[4] `https://bl.ocks.org/mbostock/3883245`
[5] `https://www.sitepoint.com/javascript-generate-lighter-darker-color/`

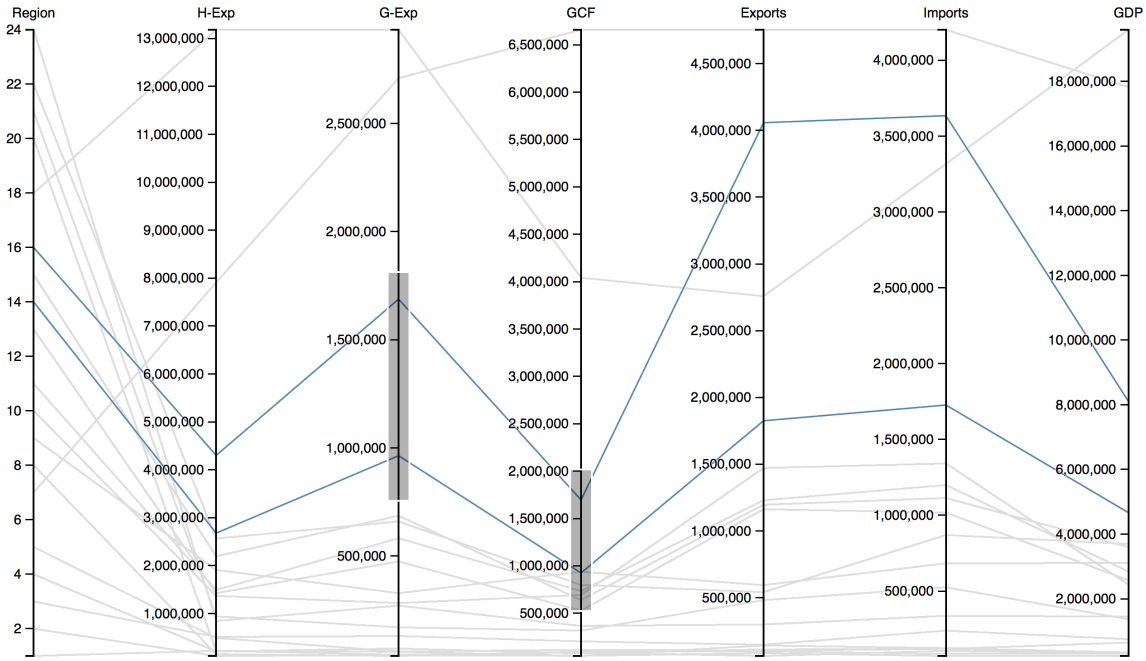


Figure 1: Parallel coordinate plot for level 1 with brushing

Level 2

The second level asked to plot multiple datasets simultaneously and to group them by colour in the graph based on their *region* dimension.

In addition, the dataset contained categorical data for which I had to adapt the creation of the axes and the implementation of the brushing functionality.

Color-Coding Listing *1* shows how the lines are drawn in different colours depending on their *region* attribute.

```
1  var color;
2
3  d3.csv("../data/population_merged.csv", function(error, myData) {
4    // ...
5
6    // Extract colours for each region
7    color = d3.scale.ordinal()
8    .domain(myData.map( function (d) { return d['Region']; }))
9    .range(d3.scale.category10().range());
10
11   // ...
12 });
```

Listing 1: Color-coding depending on region attribute

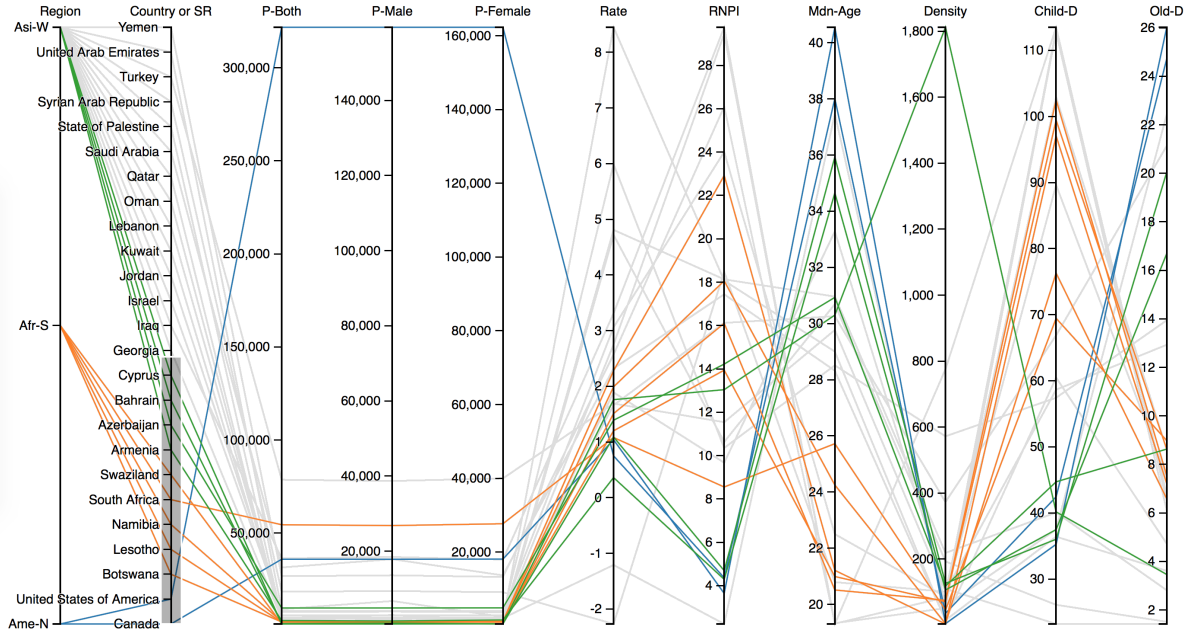


Figure 2: Parallel coordinate plot for level 2 with color-coding

Ordinal Axes The following excerpt of the source code shows how I implemented the creation of the different axes depending on their data type.

```
1  d3.csv("../data/population_merged.csv", function(error, myData) {
2    // ...
3
4    // Extract the list of dimensions and create a scale for each.
5    x.domain(dimensions = d3.keys(myData[0]).filter(function(d) {
6      if (d == 'Region' || d == 'Country or SR') { // Categorical
           Data
7        y[d] = d3.scale.ordinal()
8        .domain(myData.map(function(p) { return p[d]; }))
9        .rangePoints([height, 0]);
10     } else { // Numerical Data
11       y[d] = d3.scale.linear()
12       .domain(d3.extent(myData, function(p) { return +p[d]; }))
13       .range([height, 0]);
14     }
15
16     return true;
17   }));
18   // ...
19 });
```

Listing 2: Linear and Ordinal Axes

Brushing The following excerpt of the source code shows how I implemented the brushing functionality which is handled differently depending on whether we have categorical or numerical data.

```
function brush() {
  var actives = dimensions.filter(function(p) { return !y[p].brush.
      empty(); }),
  extents = actives.map(function(p) { return y[p].brush.extent();
      });
  foreground.style("display", function(d) {
  return actives.every(function(p, i) {
    var selection = d[p];

    // Make Brushing work for categorial axis
    if (p == 'Region' ||  p  == 'Country or SR') {
      selection = y[p](d[p]);
    }

    return extents[i][0] <= selection && selection <= extents[i
        ][1];
  }) ? null : "none";
});
```

Listing 3: Brushing for Linear and Ordinal Axis

Level 3

The third and optional level asked for a way to show the temporal relationship of the *world economy* dataset. I decided to implement this by offering a separate line chart which displays an aggregation (in this case the average) of an axis over the years when clicking on that axis in the parallel coordinate plot.

In addition, data points which have a common *country* attribute are drawn in the same colour but with different saturations depending on their *year* attribute (more recent is darker).

The following excerpt shows my implementation of a function called *createLineChart* which displays a line chart for the given dimension name.

```
function showLineChart(dimensionName) {
  var m = [80, 80, 80, 80]; // margins
  var w = 1000 - m[1] - m[3]; // width
  var h = 400 - m[0] - m[2]; // height

  var years = [1995, 2000, 2005, 2010, 2015];


  var dataTime = aggregateData(csvData, dimensionName);

  // Extract maximum Y value to determine y-axis height
  var maxYValue = 0;
  for (var i = 0; i < dataTime.length; i++) {
    maxYValue = Math.max(maxYValue, dataTime[i].avg);
  }

```

```
17
18    // Clear previous graph
19    document.getElementById("graph").innerHTML = "";
20    document.getElementById("subgraph-title").innerHTML = "Average: "
          + dimensionName;
21
22
23    // Create Axes
24    var xTime = d3.scale.ordinal().domain(dataTime.map(function(d) {
          return d.year; })).rangeRoundBands([0, w]);
25    var yTime = d3.scale.linear().domain([0, maxYValue]).range([h,
          0]);
26
27
28    // Create line
29    var line = d3.svg.line()
30    .x(function(d,i) {
31      return xTime(d.year) + m[0];
32    })
33    .y(function(d) {
34      return yTime(d.avg);
35    })
36
37    // Add an SVG element with the desired dimensions and margin.
38    var graph = d3.select("#graph").append("svg:svg")
39    .attr("width", w + m[1] + m[3])
40    .attr("height", h + m[0] + m[2])
41    .append("svg:g")
42    .attr("transform", "translate(" + m[3] + "," + m[0] + ")");
43
44    // create y-axis
45    var xAxis = d3.svg.axis().scale(xTime).ticks(5).orient("bottom");
46
47
48    // Add the x-axis.
49    graph.append("svg:g")
50    .attr("class", "x axis")
51    .attr("transform", "translate(0," + h + ")")
52    .call(xAxis);
53
54    // Create y-axis
55    var yAxis = d3.svg.axis().scale(yTime).ticks(4).orient("left");
56
57    // Add the y-axis to the left
58    graph.append("svg:g")
59    .attr("class", "y axis")
60    .attr("transform", "translate(-25,0)")
61    .call(yAxis);
62
63    // Add the line
64    graph.append("svg:path").attr("d", line(dataTime)).attr('fill', '
          none').attr('stroke', 'blue');
65  }
```

Listing 4: createLineChart Function

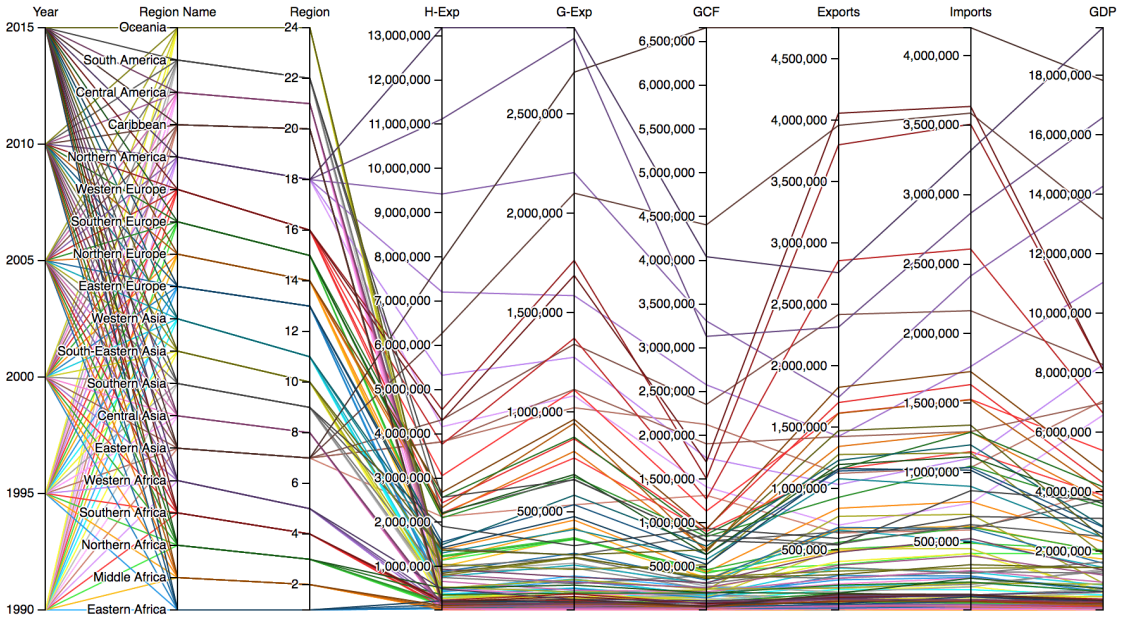
In order to aggregate the values for a given dimension and averaging over the different years I created the function *aggregateData* whose implementation

is shown below.

```
function aggregateData(data, dimName) {
  var years = ["1990", "1995", "2000", "2005", "2010", "2015"];

  // Initialse averages
  var averages = [0, 0, 0, 0, 0, 0];

  // Compute average for each year
  for (var y in years) {
    year = years[y]
    var count = 0;
    for (var i = 0; i < data.length; i++) {
      if (data[i]['Year'] == year) {
        averages[y] += parseFloat(data[i][dimName]);
      count++;
      }
    }
    averages[y] = averages[y] / count;
  }

  // Transform data into a list of objects
  var output = [];
  for (var i = 0; i < years.length; i++) {
    output.push({"year": years[i], "avg": averages[i]});
  }

  return output;
}
```

Listing 5: aggregateData function

Figure *3* shows the final plot after clicking on the *GDP* axis. It can clearly be seen how the average *GDP* has increased over the years between 1950 and 2015.



Average: GDP

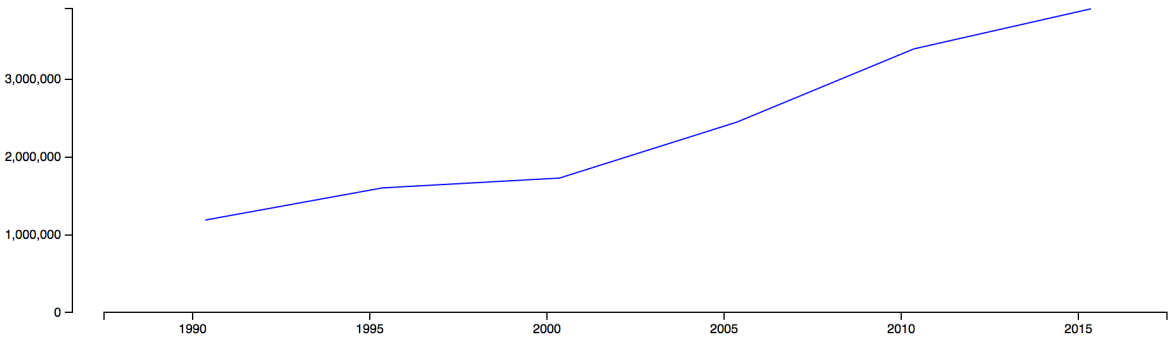


Figure 3: Parallel coordinate plot for level 3 with additional time series averaging the values of the axis that has been clicked on over the years