Tuto1: Developing data visualizations with D3.js

Nicolas Médoc Luxembourg Inst. of Science and Technology

Outline

1. Tutorials objectives

2. Introduction to D3.js

3. Selection and data binding

Tutorial objectives

- **Developing data visualization with D3.js**: data binding and controlling visual variables
- **Developing interactions with D3.js**: pattern update and animated transitions
- **Coordinated multiple views**: with D3.js and React components
- **Assignment**: building two synchronized visualizations with D3js and React (04/11/2025)

What is D3.js?

`https://d3js.org/`

- **An open source library for data visualization** built on Web standards (HTML, CSS, SVG, Canvas, javascript).
- A **low-level toolbox** with modules to prepare and transform the data, to draw data-driven graphics, and to interact with them.
- **Dynamic, interactive and animated visualizations:** thanks to the data binding and join mechanisms, D3.js facilitates interactivity with animated transitions in response to user inputs or external events.
- **Not a high-level chart library:** but a collection of low-level primitives to tailor the visualizations to fit specific requirements (for quick and basic visualizations, libraries with built-in charts are prefect: Observable Plot in javascript or MathPlotLib in python).

First example: Hermann grid

- git repository for the first tutorial git clone https://github.com/nicolasmedoc/Tuto1-D3js
- use a Javascript IDE (e.g. `https://code.visualstudio.com/`))
- declarative approach by calling a chain of methods
- to **append DOM elements**: .append("g")
- to **update their attributes**: .attr("class","matSvgG")
- to **remove DOM elements**: .remove()

See below and in index.html how to initialize the <svg> and add a group <g> that will contain the visualization elements.

```
// append the svg element to the viewContainer div, the size includes the margin
matSvg = viewContainer.append("svg")
    .attr("width", width + margin.left + margin.right)
    .attr("height", height + margin.top + margin.bottom)
;
// append the g element containing the shape, shifted by the margin
// all attributes including transfom methods applies to all its children
matSvgG = matSvg.append("g")
    .attr("class","matSvgG")
    .attr("transform", "translate(" + margin.left + "," + margin.top + ")");
;
```

Hermann grid: add the background

In renderMatrix() add the <rect> element covering the size of the grid based on cellSize and cellMargin.

```
matSvgG.append("rect")
    .attr("class","backgroundrect")
    .attr("width",(cellSize+cellMargin)*nbCols + cellMargin)
    .attr("height",(cellSize+cellMargin)*nbRows + cellMargin)
    .attr("fill","black")
;
```

Hermann grid: selection and data binding

- Call **selection.selectAll(selector)** to search in the DOM structure all the child elements corresponding to the criteria given in selector (a class, an id, a DOM element).
- Call **selection.select(selector)** return the first child element in the DOM structure.
- Call **selection.data(array, idGetter)** to bind an array of data items to the selection. It returns a selection of DOM elements bound to any item in the new array that match with an item in the previous data binding (comparison with the attribute given by the idGetter function, or by default based on object references).
- Call **enter()** chained after data() to select all new data items that were not bound previously (they are not yet associated to elements in the DOM). At the first call, it returns a selection of items with placeholder elements (they don't exist yet), to then append new elements in the DOM.
- Call **exit()** chained after data() or after enter() to select old DOM elements related to the data items that don't exist anymore in the new array. You can decide to remove the DOM element or update any visual variable.

See illustrations at `https://bost.ocks.org/mike/selection/#enter-update-exit`.

Hermann grid: Using selection, data binding and enter() to add new elements

In the function renderMatrix(), create <g> elements and store the selection in cellG

```
const cellG = matSvgG.selectAll(".cellG")
    // selectAll returns all elements in the group matSvgG with the class .cellG (empty the first time)
    // bind the data with genData and indexes
    .data(genData,(cellData)=>cellData.index)
    .enter()
    // enter() returns all data items to add:
    // if they doesn'exist in the select but exist in the new array
    // create groups <g> to add square in each cell of the grid
    .append("g")
    .attr("class","cellG")
    // apply translate transformation to define the right position of each cell
    .attr("transform",(cellData)=>{
        return "translate("+(cellData.colPos*(cellSize+cellMargin) + cellMargin)+","+
                           (cellData.rowPos*(cellSize+cellMargin) + cellMargin)+")"
    })
;
```

Check the DOM structure with 64 <g> elements

Hermann grid: Append child elements (rect/circle) to each <g>

```
// render rect as child of each element "g"
cellG.append("rect")
    .attr("class","CellRect")
    .attr("width",cellSize-1)
    .attr("height",cellSize-1)
    .attr("fill","white")
;
```

Check the DOM structure with the added rect in each <g> element

Hermann grid: attenuation of effect with rounded corners

```
// render rect as child of each element "g"
cellG.append("rect")
...
    .attr("rx","10") // try 20, 30
;
```

Scintillating grid

1) Change the background color in "gray"
2) Add a white circle in the corner of each cell

```
cellG.append("circle")
     .attr("class", "circleCorner")
     .attr("cx", -cellMargin/2)
     .attr("cy", -cellMargin/2)
     .attr("r", cellMargin)
     .attr("fill","white")
;
```

Pre-attentive variables: color

All data items have the value attribute = 1 excepted one at a random position with a value = 2. Use black color for value=1 and red color for value = 2.

```
// render rect as child of each element "g"
cellG.append("rect")
...
    .attr("fill",(cell)=>{
        if (cell.value===2)
            return "red";
        else{
            return "black";
        }
    })
;
```

What are the problems of this visual encoding ?

Pre-attentive variables: length

create a bar instead of a square and map the value to the distance from the domain [0 -2] to the visual space [0-cellSize]

```
// render rect as child of each element "g"
cellG.append("rect")
...
        .attr("width",2)
        .attr("height",(cell)=>{
            return cell.value * cellSize /2
        })
;
```

Generate a grid with one row to compare sizes

Pre-attentive variables: position

Change the the rect by a circle with the position mapped to the cell value
Generate a grid with one row to compare position

Pre-attentive variables: size

Use a rect with the size mapped to the cell value
Generate a grid with multiple rows