Tuto1: Développer des visualisation de données avec D3.js

Nicolas Médoc Luxembourg Inst. of Science and Technology

Outline

1. Objectifs du tutoriel

2. Introduction à D3.js

3. Sélection et liaison des données

Objectifs du tutoriel

- **Développer des visualisations de données avec D3.js** : liaison des données aux éléments DOM et contrôle des variables visuelles
- **Développer des interactions avec D3.js** : Update pattern et transitions animées
- **Vues multiples coordonnées** : avec D3.js et des composants React
- **Travail à rendre** : construction de deux visualisations synchronisées avec D3.js et React (date à définir)

Qu'est-ce que D3.js ?

`https://d3js.org/`

- **Une bibliothèque open source pour la visualisation de données** basée sur les standards du Web (HTML, CSS, SVG, Canvas, JavaScript).
- Une **boîte à outils de bas niveau** composée de modules permettant de préparer et transformer les données, de dessiner des graphiques pilotés par les données et d'interagir avec eux.
- **Visualisations dynamiques, interactives et animées** : grâce aux mécanismes de liaison et de jointure des données, D3.js facilite l'interactivité avec des transitions animées en réponse aux actions de l'utilisateur ou à des événements externes.
- **Ce n'est pas une bibliothèque de graphiques prédéfinis** : mais un ensemble de primitives de bas niveau permettant de créer/adapter finement les visualisations/layouts à des besoins spécifiques (pour des visualisations rapides et simples, des bibliothèques avec graphiques intégrés sont préférables : Observable Plot en JavaScript ou Matplotlib en Python).

Premier exemple : grille de Hermann

- Dépôt Git pour le premier tutoriel : git clone https://github.com/nicolasmedoc/Tuto1-D3js
- Utiliser un IDE JavaScript (par ex. `https://code.visualstudio.com/`)
- Approche déclarative par appels de fonctions chaînées
- Pour **ajouter des éléments DOM** : .append("g")
- Pour **mettre à jour leurs attributs** : .attr("class","matSvgG")
- Pour **supprimer des éléments DOM** : .remove()

Voir ci-dessous et dans index.html comment initialiser le <svg> et ajouter un groupe <g> qui contiendra les éléments de la visualisation.

```
// 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 + ")")
;
```

Grille de Hermann : ajout de l'arrière-plan

Dans renderMatrix(), ajouter l'élément Rect pour définir la couleur de fond, recouvrant la taille de la grille (calculée en fonction de cellSize et cellMargin).

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

Grille de Hermann : sélection et liaison des données

- Appeler **selection.selectAll(selector)** pour rechercher dans la structure DOM tous les éléments enfants correspondant au sélecteur (classe, identifiant, élément DOM).
- Appeler **selection.select(selector)** pour retourner le premier élément enfant correspondant.
- Appeler **selection.data(array, idGetter)** pour lier un tableau de données à la sélection.
- Appeler **enter()** après data() pour sélectionner les nouvelles données non encore associées à des éléments DOM.
- Appeler **exit()** pour sélectionner les éléments DOM associés à des données qui n'existent plus.

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

Grille de Hermann : ajout d'éléments avec enter()

Dans la fonction renderMatrix(), créer des goupes <g> qui contiendront les éléments graphiques des cellules. Stocker la sélection dans 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)+")"
    })
;
```

Vérifier la structure DOM avec 32 éléments.

Grille de Hermann : ajout d'éléments enfants (rect/circle)

```
// 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")
;
```

Vérifier la structure DOM avec le rectangle ajouté dans chaque élément .

Grille de Hermann : atténuation de l'effet avec des coins arrondis

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

Essayer différents rayons de coins arrondis (10, 20, 30).

Grille scintillante

1) Changer la couleur de fond en gris
2) Ajouter un cercle blanc dans le coin de chaque cellule

Variables pré-attentives : couleur

Les éléments de la grille ont un attribut value qui contient des données quantitatives. Ils ont tous une valeur 1, sauf un élément à une position aléatoire qui dont la valeur est 2. Utiliser la couleur noire pour value = 1 et rouge pour value = 2. Est-ce un bon encodage visuel ? Quels sont les problèmes ?

Variables pré-attentives : longueur

Créer une barre au lieu d'un carré et associer la valeur à une distance du domaine [0–2] vers l'espace visuel [0–cellSize].

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

Générer une grille avec une seule ligne pour comparer les tailles.

Variables pré-attentives : position

Remplacer le rectangle par un cercle dont la position est associée à la valeur de la cellule. Générer une grille avec une seule ligne pour comparer les positions.

Variables pré-attentives : taille

Utiliser un rectangle dont la taille est associée à la valeur de la cellule.
Générer une grille avec plusieurs lignes.