Université de Lille 1
Master Informatique

Visualisations interactives de données

Auteurs : Arnaud Lemba et Théo Renou

Tuteurs : M. Boelaert et M. Hym

2018

Remerciements

Nous tenons à remercier M. Boelaert de nous avoir donnés l'opportunité de travailler sur ce projet et de nous avoir encadrés durant le déroulement de ce projet. Nous remercions aussi M. Hym pour nous avoir également encadrés durant ce projet.

Sommaire

Introduction

Actuellement en première année de Master Informatique, nous avons eu l'occasion de travailler sur le projet Visualisation interactives des données dans le cadre du module Projet individuel. Cela nous a permis de mettre en application nos connaissances et compétences acquises durant la formation, mais également d'en apprendre de nouvelles.

Nous allons vous présenter notre travail en deux parties. Il sera question dans un premier temps de l'existant du projet. Dans un second temps, nous vous présenterons les différents ajouts et modification que nous avons effectués.

Existant du projet

Logiciel R :

Un logiciel libre dédié aux statistiques et à la science des données, soutenu par la R Foundation for Statistical Camputing.

Dans le cadre ce projet, ce logiciel nous permet de produire des variables nécessaire pour l'affichage, d'inclure les différents éléments pour le fonctionnement de la page internet et de faire office de serveur.

Au cours de ce projet, nous avons eu à travailler via principalement deux composants : la bibliothèque D3.js et les éléments SVG.

D3 JS :

Une bibliothèque graphique JavaScript qui permet l'affichage

de données numériques sous une forme graphique et dynamique.

Il s'agit d'un outil important pour la conformation aux normes W3C

qui utilise les technologies courantes SVG, JavaScript et css pour

la visualisation de données.

SVG :

SVG permet la définition d'objets graphiques (génériques) qui peuvent utiliser pour une utilisation ultérieure.

Carte de Kohonen

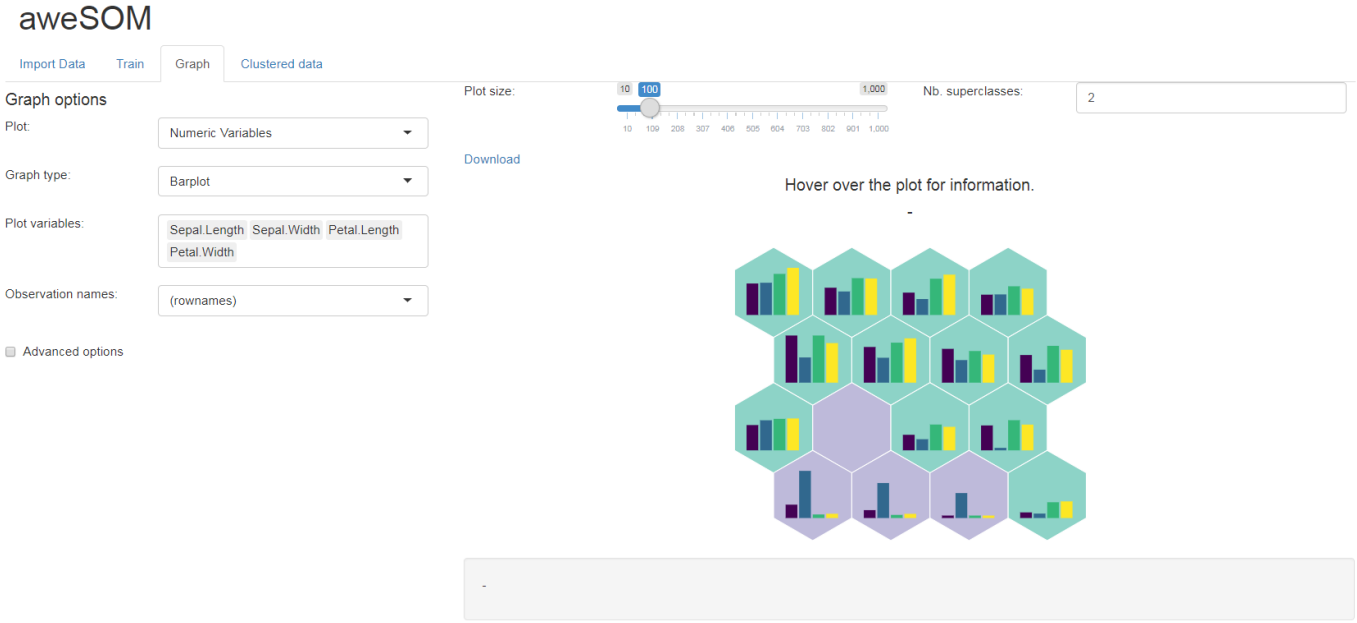
La carte de Kohonen est une topologie du réseau de Kohonen utilisée pour la classification de données.

Le réseau de Kohonen est un réseau de neurones artificiels orientés, constitués de 2 couches. Dans la couche d'entrée, les neurones correspondent aux variables décrivant les observations. La couche de sortie, elle, est le plus souvent organisée sous forme de grille (de carte) de neurones à 2 dimensions. Chaque neurone représente un groupe d'observations similaires.

Contexte

Le but de ce projet était de visualiser de données (ces données sont générées d'une manière aléatoire via le programme R ou importer via un fichier csv) d'une manière interactive sur les cartes auto- organisatrices de Kohonen. L'interactivité est faite via la bibliothèque D3.js. Les cartes interactives existantes, ses données étaient représentées sous forme d'un histogramme. Notre but était entre autres de transformer les histogrammes en nuage de points et d'établir l'interactivité sur ces points.

Capture d'écran de l'existant



Amélioration du projet existent

Nous avons au cours de ce projet effectuer différentes tâches afin d'améliorer le projet mis en place précédemment. Nous avons eu à mettre en place un nouveau type de graphique, mettre en place différentes interactivités possibles et d'optimiser le code créé précédemment.

Ajout d'un nouveau type de graphique

La première tâche nous ayant été confiée fut de rajouter un nouveau type de graphique au projet déjà existant. Ce nouveau graphique était un nuage de points. Pour cela, nous avons d'abord mis en place le nuage de points dans un fichier à part mais se basant sur le même format que le projet précédent. Afin de pouvoir afficher le nuage de points sur la page web, il a fallu effectuer plusieurs étapes.

La première étape a été de mettre en place les échelles des différents axes pour le nuage de points. Pour cela nous avons utilisé une fonction de la bibliothèque D3 permettant de créer une échelle en fournissant l'étendue des valeurs, c'est-à-dire la valeur maximum et minimum des valeurs x et y présentes dans l'ensemble des points affichés dans les nuages, et la taille souhaitée pour l'affichage des axes. Une fois ces échelles construites, il est nécessaire de créer les axes via la fonction axis() de D3.js permettant de créer un objet SVG (objet graphique) incorporant tous les éléments nécessaires à l'affichage dans la page HTML : la balise PATH permettant de dessiner le trait de l'axe et les balises affichant les valeurs à côté de l'axe. Afin de créer cet objet il faut, lors de l'appel de la fonction, fournir l'échelle correspondante afin de pouvoir afficher les bonnes valeurs pour l'axe et définir sa taille. Il est aussi possible de définir l'épaisseur souhaitée pour l'axe, la position de l'axe x ou y, et de définir le nombre maximal de valeurs à afficher sur l'axe.

```
var echelleY = d3.scale.linear()
                    .domain([nuageRange[1][0],nuageRange[1][1]])
                    .range([maxy,miny]);

var echelleX = d3.scale.linear()
                    .domain([nuageRange[0][0],nuageRange[0][1]])
                    .range([0,axe]);

var xAxe = d3.svg.axis()
                .scale(echelleX)
                .tickSize(0.5*(cellSize/100))
                .ticks(4)
                .orient("bottom");

var yAxe = d3.svg.axis()
              .scale(echelleY)
              .tickSize(0.5*(cellSize/100))
              .ticks(4)
              .orient("left");
```

Une fois les axes créés, il a fallu mettre en place pour chaque cellule une balise qui recevra les différents éléments nécessaires à l'affichage du graphique. Pour cela, on ajoute à la zone dédiée au SVG (balise SVG : zone comportant les cellules et les graphiques) autant de balises g (c'est-à-dire une balise permettant de regrouper différents objets tel que les axes et les points du nuage) qu'il y a de cellule présente. Pour chacune de ces balises, on lui définit ses dimensions afin de s'afficher correctement dans la cellule. On lui fournit aussi un ID afin de pouvoir plus facilement accéder aux balises de chaque cellule lors de l'ajout des axes et des points du nuage. En plus de cela il faut aussi effectuer une translation à ces balises afin de pouvoir faire en sorte que celles-ci soient bien affichées à leur cellule correspondante. Pour cela lors de la création des cellules, un tableau comportant le centre de chaque cellule (c'est-à-dire son positionnement x et y dans la balise SVG) est alimenté. On ajoute alors l'attribut transform (attribut permettant de manipuler des éléments HTML dans l'espace 2D ou 3D) avec comme valeur translate et les valeurs x et y de chaque cellule à chaque balise g.

```
for (var indice = 0; indice < nbRows*nbColumns; indice++) {
        svg.append("g")
         .attr("id", "ggraph" + (indice + 1) )
         .attr("width", (width) + "px")
         .attr("height",(height) + "px")
         .attr("transform", "translate(" + (coordinatesArrayX[indice]) + ',' +
(coordinatesArrayY[indice]) + ')' );
                }
```

Une fois ces zones dédiées aux graphiques créées, il a fallu y ajouter les axes et les points du nuage. Pour cela, on parcourt l'ensemble des balises g créés précédemment et on ajoute une nouvelle balise g qui contiendra les éléments nécessaires à l'affichage de l'axe. Une fois cette balise créée, l'ajout des éléments de l'axe se fait automatiquement, en utilisant la fonction call() de D3 et en lui fournissant en paramètre l'objet comportant les informations sur l'axe créée au début.

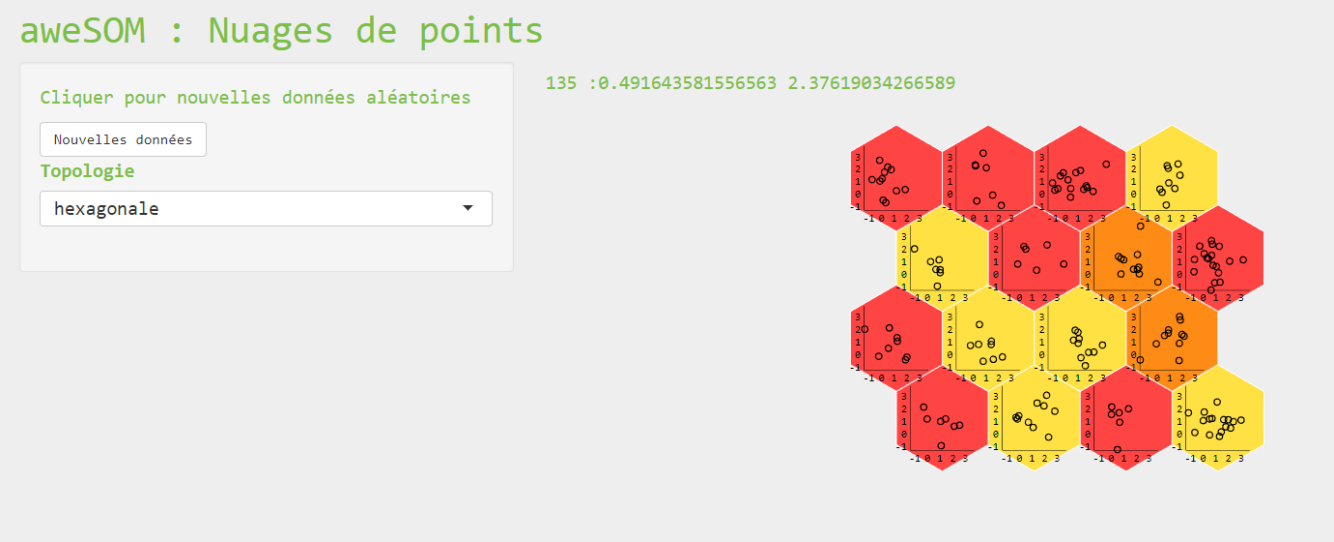
```
g.append("g")
  .attr("class","axis")
  .attr("font-size", cellSize/10 + "px")
  .attr("transform", "translate("+ 0 +","+ (axe + transpoX) +")" )
  .call(xAxe);
```

Une fois les axes mis en place, il a fallu ajouter les points du nuage. Pour cela, les données relatives aux points étaient stockées dans un objet composé d'une liste. Chaque élément de cette liste était composé de l'ensemble des points à ajouter à la cellule selon l'indice de l'élément. Chaque point possédait trois informations : son label, sa valeur x et sa valeur y. Afin de pouvoir ajouter les points à leur cellule attitrée, la méthode utilisée a été d'extraire l'élément comportant les points dans la liste. Une fois cet élément récupéré, il a fallu le parcourir afin de récupérer chaque point devant être ajouté. A chaque point prélevé, on récupère la balise g de la cellule correspondante et on y ajoute une nouvelle balise qui comportera les informations du point. Cette balise est une balise circle permettant de dessiner un cercle dans le graphique. Il a été nécessaire de fournir différents attributs à cette balise afin de pouvoir la placer correctement dans le graphique. Il a fallu fournir sa dimension r afin de définir sa taille. Et il a fallu aussi fournir sa position x et y mais en les modifiant via les échelles produites précédemment afin qu'elle ne sorte pas de la zone définie.

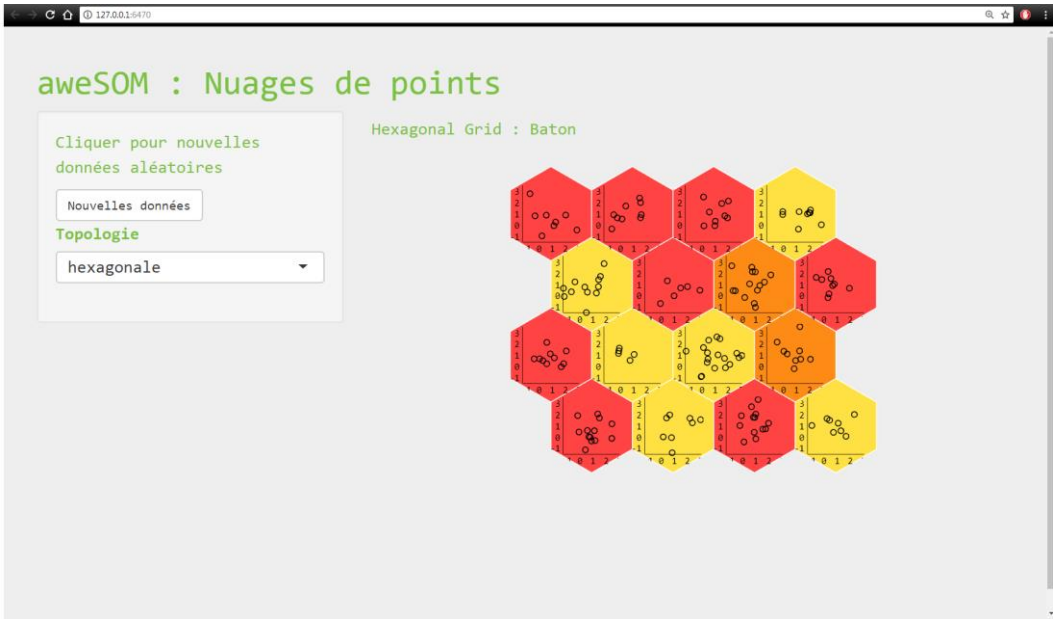
```
for (var indice=0; indice<nbRows*nbColumns; indice++){
    var dataCell=nuageData[indice+1];

        for(var j=0; j<dataCell.x.length; j++){
          var g=d3.select("#ggraph" + (indice+1));
          g.append("circle")
            .attr("cx",function(){
               return (echelleX(dataCell.x[j]));})
            .attr("cy",function(){
               return (echelleY(dataCell.y[j]));})
```

```
            .attr("r",function(){
               return 3*(cellSize/100);})
                 .style("stroke", "black")
                 .style("fill","none");}}
```

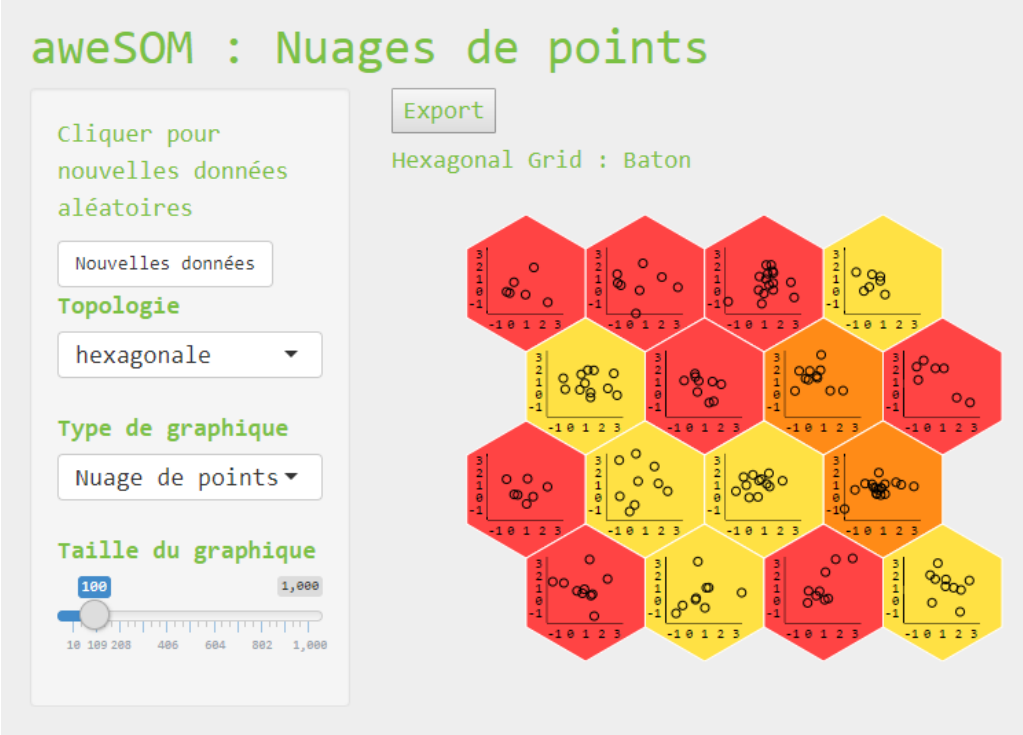
Une fois ces différentes étapes effectuées, on obtient un graphique de type nuage de points dans chaque cellule. Pourtant ces étapes ne furent pas suffisantes. Le projet initial avait comme possibilité le choix de forme des cellules : rectangulaire ou hexagonale, hors lorsque le format choisi était hexagonal, la zone d'affichage étant d'une forme carrée les valeurs des axes empiétaient sur les cellules voisines.



Afin de pallier à ce problème, une première solution proposée a été de rogner la forme de la zone d'affichage. Pour cela, il a fallu mettre en place un attribut spécial dans la feuille de style pour la zone. Cet attribut est nommé « clip-path », il permet de choisir la forme souhaitée pour un élément. Afin de pouvoir faire en sorte que la forme soit hexagonale, il a été nécessaire de définir que la forme devait être un polygone et fournir chaque point du polygone en fournissant sa localisation en x et y en pourcentage de la forme initiale de la zone.

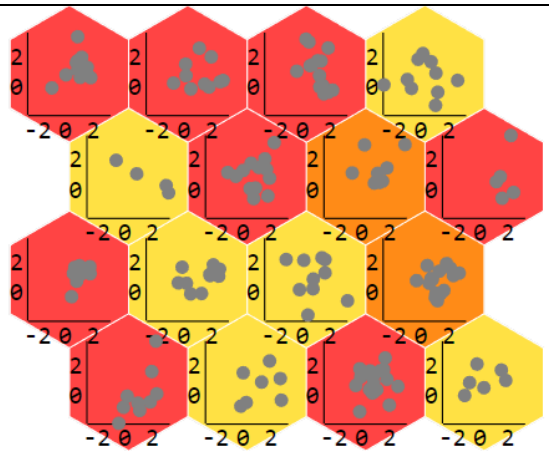
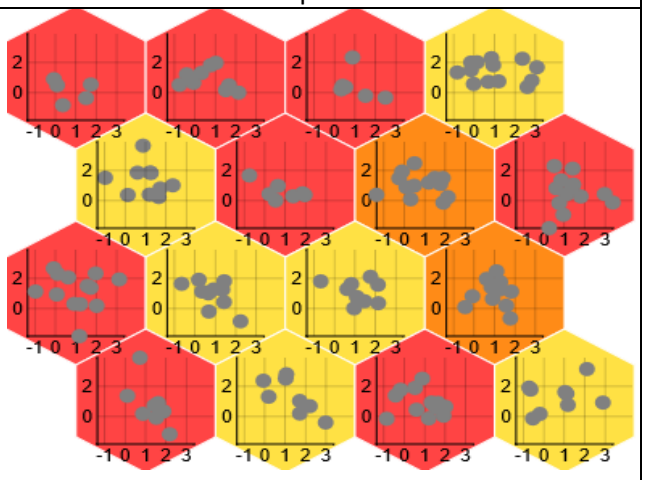


Cette solution n'ayant pas été sélectionnée par l'auteur du projet, il a fallu trouver une autre solution. Une autre possibilité a été de procéder à des ajustements aux étapes effectuées lors de la création du nuage de points. Dans un premier temps, il a fallu réduire la zone dédiée au graphique. Pour cela il a fallu diminuer les attributs permettant de définir les dimensions de la zone, mais il a aussi fallu redéfinir les dimensions des échelles permettant la construction des axes et le placement des points. Si les échelles n'avaient pas été redéfinies, les zones dédiées aux graphiques étaient automatiquement réajustées selon les éléments qu'elles possédaient. Une fois cette étape effectuée, étant donné que les zones dédiées aux graphiques étaient plus petites, il a été nécessaire de repositionner correctement ces zones dans leur cellule en effectuant de léger décalage horizontal et vertical dans l'attribut transform. La dernière modification effectuée a été de réduire la taille de police des valeurs des axes. Une fois ces étapes faites, on obtient le résultat souhaité c'est-à-dire un nuage de points dans chaque cellule.



Affichages de tous les axes et grilles par rapport à la carte

Le but était d'afficher les axes et grilles sur chacune des hexagones de la carte.

Avant	Après
	

Méthodes employées :

d3.select() :

Les sélections permettent une transformation puissante du modèle d'objet documentaire (DOM) en fonction des données: définition d'attributs, de styles, de propriétés, de contenu HTML ou textuel, etc. En utilisant les sélections d'entrée et de sortie de la jointure de données, vous pouvez également ajouter ou supprimer des éléments pour correspondre aux données.

Les méthodes de sélection renvoient généralement la sélection en cours, ou une nouvelle sélection, permettant l'application concise de plusieurs opérations sur une sélection donnée via le chaînage des méthodes.

append("g") :

C'est une méthode qui permet d'ajouter un élément 'g' au SVG.
L'élément g est utilisé pour regrouper les formes SVG.

Cette méthode a été employée à l'objet issu de d3.select().

.attr("class", "x axis") :

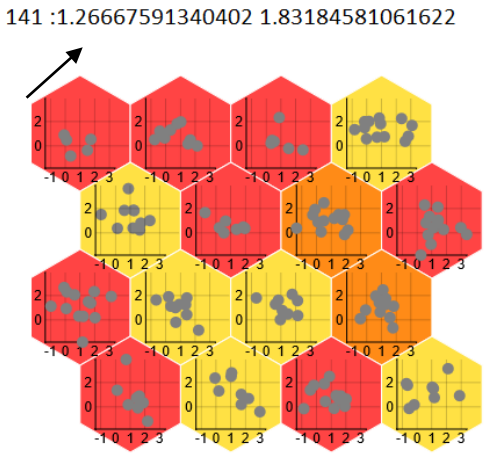
Cette méthode permet d'ajouter l'attribut class dans l'élément
ajouté via la méthode append ci haut.
.call(xAxe) :

Cette méthode appliquée à l'objet issu
De append() permet d'appeler l'objet xAxe du programme
qui permet de dessiner les axes.
.attr("transform", "translate("+ 0 +","+ (width/2 +10) +")") :

Cette méthode appliquée à l'objet issu de append() permet de bien positionner les axes sur le graphique.

Affichage des infos sur les points

L'objectif de cette interactivité sur le nuage de points a été d'afficher la valeur de points pointé via la souris.



Afin de pouvoir obtenir ce résultat, deux modifications ont été apportée au projet. La première fut lors de l'ajout des points à leur nuage. Cette modification a été de rajouter un attribut d'interactivité HTML à chaque cercle créé. Cet attribut est « onmouseover » qui permet lorsque le curseur de la souris survole un des cercles d'effectuer une action souhaité. Dans notre cas cette action a été de faire appel à une fonction permettant l'affichage des données du point. Lors de l'appel à la fonction, les données du point, c'est-à-dire son label, sa valeur x et y, sont également placé en paramètre de la fonction.

```
.attr("onmouseover","affichelabelnuage(" + dataCell.label[j] + ',' + dataCell.x[j] + ',' + dataCell.y[j] + ");")
```

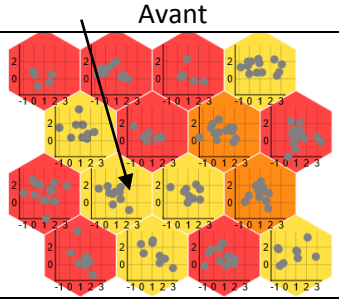
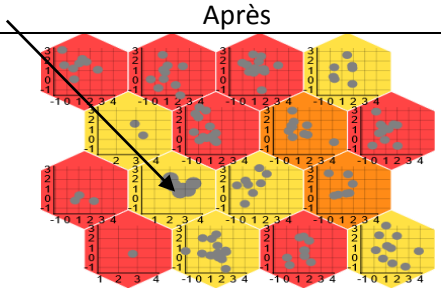
La deuxième modification a été de mettre en place cette fonction. La fonction comporte deux étapes, la première qui construit une chaine de caractère avec les informations placé en paramètre. Puis la deuxième a pour but d'afficher cette chaine dans la zone lui étant dédié. Pour cela la fonction recherche dans la page, l'élément ayant comme ID «grid-ref » et remplace ce que comportait cet élément par la chaine de caractère construite.

```
function affichelabelnuage(label,x,y){
                var ch=label + ' :' + x + ' '+y;
                document.getElementById("grid-ref").innerHTML=ch;
}
```

Zoom de certains points du graphique

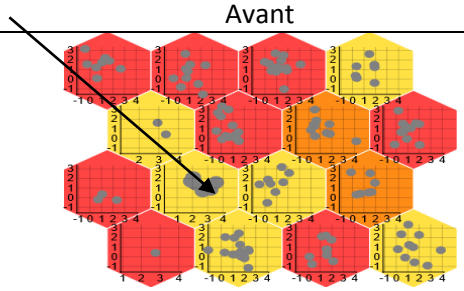
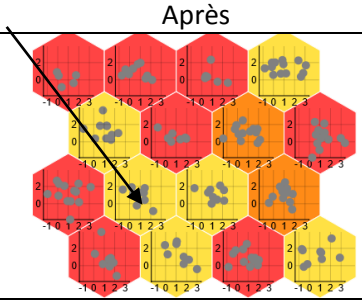
Le but serait que lors du passage de la souris sur certains points avec la molette de la souris de l'avant que les points qui sont proches de curseur de la souris à une certaine distance fixée soient gardés pour le zoom et que d'autres points soient masqués. Lors du zoom, l'axe des abscisses est décalé en fonction de l'abscisse le plus petit de points gardés pour le zoom.

La flèche sur le graphique indique l'hexagone actif pour le zoom.

Avant	Après
	

Dezoom

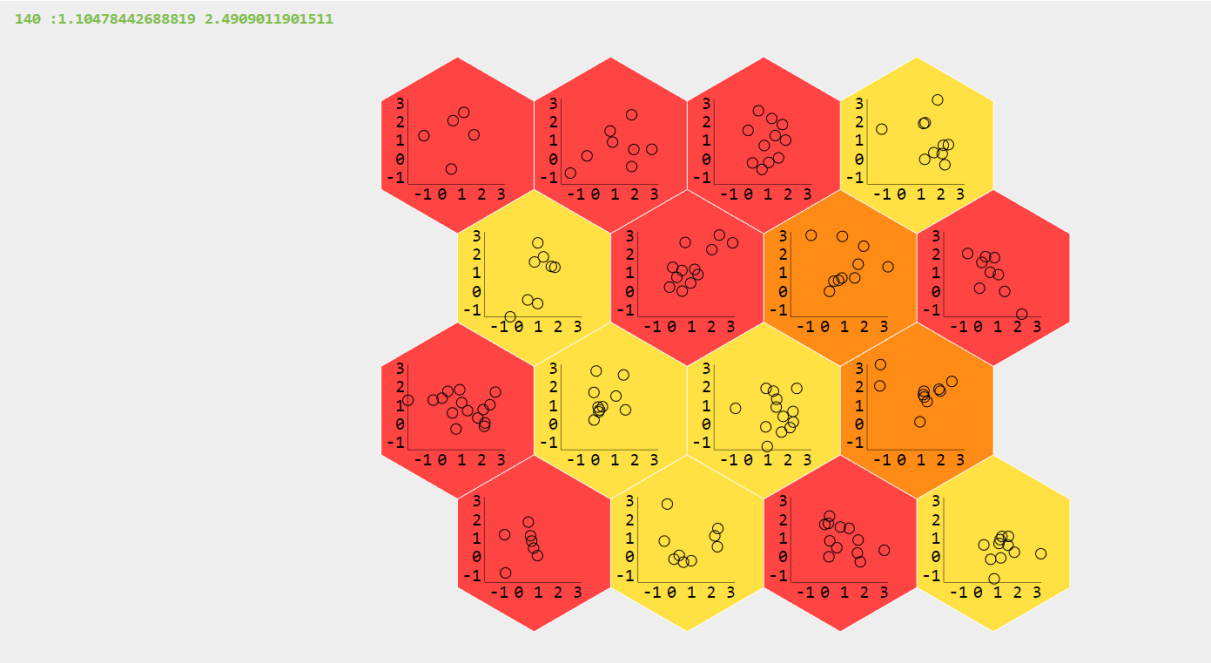
Après le zoom, le dezoom permet que les points masqués précédemment soient réaffichés.

Avant	Après
	

Exportation des graphiques

Une des interactivités demandée au cours de ce projet, fut d'avoir la possibilité d'exporter les graphiques en format HTML tout en gardant les fonctionnalités d'interactivité précédente. La méthode utilisée fut d'exporter ces graphiques via un URL de données, c'est-à-dire de transmettre les différents éléments nécessaire à l'affichage au sein même de l'URL. Afin de pouvoir effectuer cette exportation, un bouton a été ajouté à la page HTML, ce bouton permet d'activer la fonction s'occupant de l'exportation. La fonction d'exportation crée alors une nouvelle balise quelconque qui servira à stocker l'URL contenant les informations à exporter. Une fois cette balise créée, on stocke la zone dédiée au SVG grâce à son ID, défini lors de sa création, dans une variable. Les fonctions permettant l'interactivité sont également stockées dans une autre variable afin de pourvoir garder l'interactivité du graphique. Le CSS (zone de style) et la zone permettant l'affichage des points sont aussi récupérés et stockés. Une fois ces différents éléments enregistrés, elles sont combinées afin de ne faire qu'une seule et unique chaine de caractère qui servira à l'exportation. Ensuite, on définit alors que la nouvelle balise créée au début de la fonction aura comme fonctionnalité d'effectuer un téléchargement et on lui fournit le nom du fichier à télécharger. L'étape suivante consiste à définir la référence de la balise, pour cela on construit un URL en utilisant la méthode

« data:text/html », qui permet de définir que le document sera un document HTML, et en ajoutant la chaine de caractère produite précédemment. On simule alors un clic utilisateur sur la balise afin de lancer le téléchargement du fichier. On obtient alors l'affichage des graphiques dans un fichier HTML à part ne nécessitant plus l'utilisation du logiciel R et ayant une taille de fichier largement inférieur à celui d'origine.



Une fois le téléchargement effectué on supprime la balise afin de ne pas garder d'élément superficiel dans la page HTLM de départ.

Pourtant cette fonctionnalité n'a pas fonctionné sur certains navigateurs internet (exemple : Firefox). En effet les normes sur l'encodage de ces URLS de données n'étant pas identiques entre les différents navigateurs cela a engendré un refus de compréhension du fichier chez certains navigateurs. Il a alors fallu ajouter une étape à l'exportation qui a eu pour but d'encoder la chaine de caractères servant de données pour l'URL. Pour cela la chaine de caractère est d'abord fournit à la fonction encodeURIComponent() qui permet de remplacer chaque exemplaire de certains caractères par des séquences d'octets correspondantes. Cette fonction renvoie alors une nouvelle chaine de caractères encodée qui servira alors à la création de l'URL.

```
function download(){
	var a = document.body.appendChild(
		document.createElement("a"));
	a.setAttribute("display","none");
	var css ="<style type=\"text\/css\"> " +
document.getElementById("css").innerHTML + "</style>";
	var header="<header><h4 id=\"grid-ref\"></h4></header>";
	var fonct= "<script>" + document.getElementById("funct").innerHTML +
"<\/script>";
```

```
var svg=document.getElementById("svggraph").outerHTML;
var fichier=encodeURIComponent(css+header+fonct+svg);
a.download = "export.html";
a.href = "data:text/html," + fichier ;
a.click();
a.parentNode.removeChild(a);}
```

Optimisation

Une fois l'ajout du nouveau type de graphique et la mise en place de l'interactivité, une dernière étape nous a été demandée. Nous devions ajouter ce graphique et ces nouvelles fonctionnalités au projet existant. Hors le projet contenait déjà un certain nombre de type de graphiques différents. Chaque graphique possédait une fonction par type de forme de cellule, il nous a donc été demandé d'optimiser le code existant ainsi que celui du nuage de points de façon à ce qu'il n'y est qu'une fonction pour chaque type de graphique et que la construction des cellules et des graphiques se fassent dans des fonctions séparées.

La première étape a donc été d'appliquer ces changements au fichier ne contenant que le nuage de points. La construction du graphique selon la forme hexagonale ou carrée des cellules étant très similaire, il a alors été facile de fusionner les deux fonctions en une seule. La différence entre ces deux fonctions résidait principalement dans le positionnement de la zone du graphique dans la cellule et dans les différents décalages a effectué afin que le graphique soit centré et sans débordement dans les cellules voisines. Les fonctions ayant des variables identiques mais des valeurs différentes selon la forme de la cellule, il a donc été nécessaire d'ajouter des paramètres à la fonction permettant la création du graphique.

Une fois cela effectué, il a fallu extérioriser la création des variables nécessaires pour la création des graphiques de la fonction de mise en place des cellules. Certaines variables, n'ayant aucun impact sur la mise en place de cellules ont pu être sorti directement de la fonction et introduit juste avant l'appel à la fonction permettant la création du nuage. Mais dans le cas de la cellule de forme hexagonale, certaines variables étaient nécessaires à la mise en place des cellules et lors du placement des nuages dans les cellules. Il a donc fallu pour cela définir au début du programme que ces variables étaient publiques c'est-à-dire pouvant être utilisées à n'importe quel endroit du programme. Une fois cette factorisation de code effectué pour le nuage de points, les étapes suivantes furent d'effectuer la même factorisation pour les autres types de graphiques. Hors lors de la programmation du projet effectuée par nos prédécesseurs, la mise en place des graphiques s'est présentée dans un premiers temps sous la forme de cellule carrée, puis dans un deuxième temps sous la forme hexagonale. Hors le traitement de la mise en place sur forme carrée ne correspondant pas à celle de la forme hexagonale, ils ont codé cette mise en place de façon complètement différente. Il a donc été nécessaire de reprogrammer l'ensemble des différents types de graphiques en se basant sur la même méthode que celle pour le nuage de points.

Conclusion

Nous avons pu au cours de ce projet effectuer différentes taches tels que l'ajout d'un nouveau type de graphique, la mise en place d'interactivité et une optimisation de code pour un projet mis en place par des étudiants des années précédentes.

Ce projet nous a été très enrichissant car il nous a permis de découvrir la bibliothèque D3.js qui nous étaient inconnue avant ce projet. De plus cela nous a permis d'approfondir nos connaissances dans la programmation en JavaScript via l'utilisation de cette bibliothèque et l'utilisation des éléments SVG.

Bibliographie

D3.js : https://d3js.org

GitHub Projet aweSOM : https://github.com/jboelaert/aweSOM

Logiciel R : https://www.r-project.org/

Github PJI: https://github.com/Arnaudlille/PJI_Arnaud_Theo