Mémoire

Auteur : Schillings, Carole
Promoteur(s) : Kasprzyk, Jean-Paul
Faculté : Faculté des Sciences
Diplôme : Master en sciences géographiques, orientation géomatique, à finalité spécialisée en geodata-expert
Année académique : 2020-2021
URI/URL : http://hdl.handle.net/2268.2/11759

Faculté des sciences
Département de géographie
Unité de géomatique

Développement d'un outil cartographique automatisant la symbolisation des entités géographiques 2D dans un environnement web

Mémoire réalisé par Carole SCHILLINGS

Pour l'obtention du diplôme :
Master en sciences géographiques, orientation géomatique,
à finalité spécialisée en géodata-expert

Année académique : 2020-2021

Promoteur : Jean-Paul Kasprzyk

Lecteurs : Roland Billen, Serge Schmitz

Remerciements

Tout d'abord, je souhaite remercier mon promoteur Monsieur Jean-Paul Kasprzyk pour ses nombreux conseils et son accompagnement lors de la réalisation de ce travail.

Je remercie également mes lecteurs, Monsieur Roland Billen et Monsieur Serge Schmitz pour l'intérêt qu'ils portent à ce mémoire.

Je souhaite adresser un tout grand merci à ma famille et particulièrement émes parents pour leur soutien et leurs nombreuses relectures.

Enfin, j'adresse un tout grand merci à mon compagnon Benjamin pour son soutien, son encouragement et son aide lors de mes années d'études.

Résumé

Le SIG web est devenu un support incontournable pour la diffusion d'informations géographiques. Cependant, la manipulation du JavaScript n'est pas aisée pour tout un chacun. C'est pourquoi l'objectif de ce travail est de concevoir un outil permettant d'automatiser la symbolisation des données sur une interface web avec une légende associée. L'utilisateur aura également l'opportunité, s'il le souhaite, de personnaliser une série de paramètres comme la taille, la couleur, la classification, etc.

Pour ce faire, une librairie JavaScript est implémentée avec une représentation automatique adéquate pour chaque combinaison niveau d'échelle/mode d'implantation. L'utilisateur non-initié doit uniquement fournir le fichier de données avec l'attribut choisi, la couche sur laquelle symboliser les entités et le niveau d'échelle de valeurs.

Pour la discrétisation des données quantitatives, un algorithme a été développé afin de déterminer la meilleure classification. Celui-ci se base sur le calcul des indices de T.A.I. et d'entropie.

Pour finir, les entités spatialement continues peuvent également être représentées et personnalisées en fournissant un fichier au format CovJSON.

Abstract

Web GIS has become an essential medium for the dissemination of geographic information. However, manipulating JavaScript is complex and consequently not easy for everyone. Thus, the objective of this work is to design a tool for an automatic data symbolization with an associated legend on a web interface. Additionaly, the user will have the opportunity to customize a series of parameters such as size, color, clasification, etc.

To do this, a JavaScript library is implemented with an appropriate automatic representation for each level of scale / layout combination. The unexperienced user must only provide the data file with the chosen attribute, the layer on which features will be symbolized and the scale level.

For the discretization of quantitative data, an algorithm has been developed to determine the best classification based on the calculation of T.A.I. and entropy.

Finally, spatially continuous features can as well be represented and customized by providing a file in CovJSON format.

Table des matières

I. Introduction générale 9
II. État de l'art 11
1. Web GIS et web mapping 11
2. Architecture côté serveur (WMS) versus côté client (JavaScript) 12
2.1. Architecture côté serveur 12
2.2. Architecture côté client 13
3. Sémiologie graphique 14
3.1. Variables rétiniennes 15
3.2. Perception des variables rétiniennes 15
3.3. Cartes à une composante 17
3.4. Série divergente 18
3.5. Cartes à deux composantes 19
4. Discrétisation d'une variable quantitative 19
4.1. Intervalles égaux 20
4.2. Progression arithmétique 20
4.3. Progression géométrique 21
4.4. Quantiles – classes d'égales fréquences 21
4.5. Écart-type 22
4.6. Moyennes emboitées 23
4.7. Jenks – Seuils naturels 23
4.8. Choix de classifications 24
5. Interactivité et animation 25
6. Formats d'échange de données spatiales 27
6.1. XML vs JSON 27
6.2. GML 28
6.3. GeoJSON, TopoJSON, CovJSON 29
7. Outils web GIS côté client 29
7.1. Outil de représentation cartographique 29
7.2. Outil de traitement cartographique 30
III. Hypothèses et méthodologie 31
IV. Développement 32

1. Phénomènes spatialement discrets 32
1.1. Nominal 33
1.2. Ordonné 35
1.3. Quantitatif 38
2. Phénomènes spatialement continus 41
V. Applications 43
1. Phénomènes spatialement discrets 43
1.1. Ponctuel nominal 43
1.2. Linéaire nominal 45
1.3. Zonal nominal 46
1.4. Ponctuel ordonné 47
1.5. Linéaire ordonné 48
1.6. Zonal ordonné 49
1.7. Ponctuel quantitatif 50
1.8. Linéaire quantitatif 51
1.9. Zonal quantitatif – classification 52
1.10. Zonal quantitatif – série divergente 53
1.11. Zonal quantitatif – point centré 54
2. Phénomènes spatialement continus 55
VI. Conclusions 56
VII. Bibliographie 58
VIII. Annexes 62

Table des figures

Figure 1 : Architecture 3-tiers 11

Figure 2 : Architecture côté serveur (Alesheikh et al., 2002 ; modifié) 12

Figure 3 : Architecture côté client (Alesheikh et al., 2002 ; modifié) 14

Figure 4 : Exemple de palettes de couleurs séquentielles (Colorbrewer2.org) 16

Figure 5 : Perception des trames 16

Figure 6 : Exemple de palettes divergentes (Colorbrewer2.org, modifié) 18

Figure 7 : Exemple de composition colorée 19

Figure 8 : Animations selon les variations possibles (Cauvin, 2008) 26

Figure 9 : Comparaison entre le format XML et JSON 28

Figure 10 : Diagramme de la fonction « nominal » 34

Figure 11 : Palettes Reds et YlOrBr 36

Figure 12 : Diagramme de la fonction « ordonné » 37

Figure 13 : Légende pour le ponctuel quantitatif 39

Figure 14 : Diagramme de la fonction « quantitatif » 41

Figure 15 : Diagramme de la fonction « continu » 42

Figure 16 : Application du ponctuel nominal 44

Figure 17 : Application du linéaire nominal 45

Figure 18 : Application du zonal nominal 46

Figure 19 : Application du ponctuel ordonné 47

Figure 20 : Application du linéaire ordonné 48

Figure 21 : Application du zonal ordonné 49

Figure 22 : Application du ponctuel quantitatif 50

Figure 23 : Application du linéaire quantitatif 51

Figure 24: Application du zonal quantitatif - classification 52

Figure 25 : Application du zonal quantitatif - série divergente 53

Figure 26 : Application du zonal quantitatif – point centré 54

Figure 27 : Application pour le continu 55

Table des tableaux

Tableau 1 : Variables rétiniennes 15

Tableau 2 : Carte à une composante (Donnay, 2013 ; modifié) 17

Tableau 3 : Avantages/inconvénients de la classification par intervalles égaux 20

Tableau 4 : Avantages/inconvénients de la classification par progression arithmétique 21

Tableau 5 : Avantages/inconvénients de la classification par progression géométrique 21

Tableau 6 : Avantages/inconvénients de la classification par quantiles 22

Tableau 7 : Avantages/inconvénients de la classification par écart-type 23

Tableau 8 : Avantages/inconvénients de la classification par moyennes emboitées 23

Tableau 9 : Avantages/inconvénients de la classification de Jenks 24

Tableau 10 : Variables graphiques 25

Tableau 11: Visualisation des phénomènes spatialement discrets 33

I. Introduction générale

Le SIG web est devenu un support incontournable pour la diffusion d'informations géographiques. Cependant, la manipulation du JavaScript n'est pas aisée pour tout un chacun. C'est pourquoi l'objectif de ce travail est de concevoir un outil permettant d'automatiser la symbolisation des données sur une interface web afin de réduire la charge de développement. L'utilisateur aura également l'opportunité, s'il le souhaite, de personnaliser une série de paramètres comme la taille, la couleur, la classification, etc. L'implémentation réalisée est un outil de cartographie web côté client. Le cœur du travail est constitué d'un développement JavaScript long et complexe autour de la symbolisation des entités géographiques.

Différentes symbolisations sont possibles pour des entités discrètes mais certaines ne sont pas compatibles avec le SIG web. Au contraire, la visualisation de données spatialement continues côté client est une problématique assez récente et peu de moyens existent pour les personnaliser. Dès lors, la question de recherche est la suivante : **comment automatiser la symbolisation des phénomènes géographiques dans un SIG web côté client ?**

Afin de répondre à cette question de recherche, ce travail est articulé en 5 grandes parties. La première synthétise les fondamentaux de la cartographie et définit les concepts du SIG web utilisés dans le développement. La deuxième partie comporte les hypothèses de recherche et les justifications de la méthodologie. La partie suivante contient les explications de l'implémentation réalisée. Cette dernière est ensuite appliquée pour chaque cas sur des données correspondantes dans la quatrième partie. Pour finir, la conclusion reprend les grandes étapes de ce travail, les limites de l'implémentation et les perspectives envisageables pour l'amélioration de celle-ci.

Notions de base

Selon Donnay (2013) : « Une donnée géographique est le résultat de l'observation d'un phénomène implicitement ou explicitement associé à une localisation à la surface de la Terre.» Les phénomènes géographiques sont soit spatialement discrets, c'est-à-dire qu'ils peuvent être généralisés comme une figure géométrique ou entité, soit continus, c'est-à-dire qu'ils ne sont pas limités dans le territoire analysé (Bertin, 1967). Il existe 3 modes d'implantation pour généraliser les phénomènes géographiques discrets :

- Implantation ponctuelle : représentation par un point, indépendante de la dimension de celui-ci ;
- Implantation linéaire : représentation par une ligne ayant une longueur mesurable mais indépendante de sa largeur ;
- Implantation zonale : représentation par un polygone ayant une superficie mesurable.

Les entités géographiques sont caractérisées par des attributs dont l'échelle de mesures peut être qualitative ou quantitative (Donnay, 2013). L'échelle qualitative reprend le niveau nominal, traduisant des modalités de différentes natures, et le niveau ordonné, traduisant une différence d'ordre ou une classification des données. L'échelle quantitative résulte d'une mesure ou d'un calcul et traduit une différence de quantités entre les valeurs.

II. État de l'art

1. Web GIS et web mapping

Un système d'informations géographiques (SIG ou GIS) est un système d'informations traitant des données géographiques. Il doit être capable d'acquérir des données, de les mémoriser, de les gérer et de les mettre à jour ainsi que de produire et de diffuser de l'information (Donnay & Pantazis, 1996). Les SIG sont apparus dans les années 60 et ont constamment évolué au fil des années, notamment grâce à l'invention du World Wide Web en 1990 qui permet le passage vers le SIG web (Fu, 2018). La première page web cartographique a été développée en 1993 par une compagnie américaine. Depuis lors, de nombreuses organisations ou utilisateurs exploitent des SIG web pour diffuser de l'information cartographique. L'utilisation de cet outil a de nombreux avantages : les informations partagées sont accessibles n'importe où, n'importe quand et par n'importe qui, tout appareil ayant un navigateur (desktop ou mobile) peut accéder aux données, l'utilisateur ne doit pas avoir de connaissances particulières pour visualiser les informations, etc. Cependant, l'utilisation du Web comme medium dépend de la connexion internet, du trafic ainsi que de la performance des clients et des serveurs (Kraak, 2004).

Le SIG web est construit selon un modèle d'architecture client/serveur 3-tiers. Celui-ci est un modèle qui divise l'application en 3 couches (Luqun et al., 2002) (Figure 1) :

- Un ou plusieurs postes clients : navigateur web, application mobile, etc. ;
- Un serveur distant, serveur web ;
- Un serveur de données.

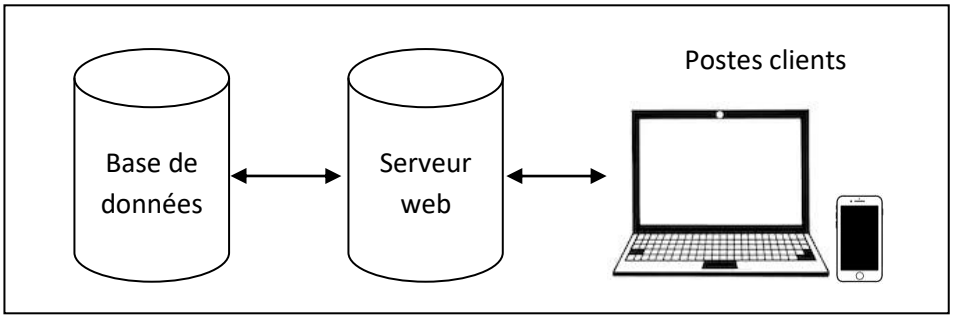


Figure 1 : Architecture 3-tiers

L'utilisation d'un modèle client/serveur permet d'optimiser les opérations étant donné qu'elles sont réalisées sur différentes machines ou encore de ne pas devoir stocker les données dans les postes clients (Kambalyal, 2010). L'utilisation de l'architecture 3-tiers, quant à elle, a pour avantage de ne requérir que de très peu de ressources clients puisqu'il ne fait que consulter l'information sur l'interface choisie et par conséquent des opérations complexes peuvent être réalisées sans problème. Cependant, le désavantage est qu'il est plus compliqué de créer et de maintenir un modèle de ce type en comparaison avec une architecture composée uniquement d'une base de données et de postes clients (2-tiers).

2. Architecture côté serveur (WMS) versus côté client (JavaScript)

Dans un SIG web, les traitements géographiques se divisent en deux catégories : le côté client et le côté serveur.

2.1. Architecture côté serveur

Dans cette architecture, l'utilisateur a uniquement accès à une interface communiquant avec un serveur et affichant les résultats (Alesheikh et al., 2002). Le processus de traitements est entièrement réalisé par le serveur. Celui-ci réalise un service web, il s'agit « d'un morceau de code qui s'exécute sur le serveur et peut effectuer une action en réponse à une demande client. » (Quinn, 2018). L'interaction entre l'interface client et le serveur est réalisée à l'aide de requêtes standardisées via le protocole HTTP (ou HTTPS). Celles-ci comportent le WMS (Web Map Service), affichant une image géo-référencée, le WMTS (Web Map Tiles Service), affichant des images géo-référencées tuilées, le WFS (Web Features Service), indiquant les attributs présents dans les données, etc.

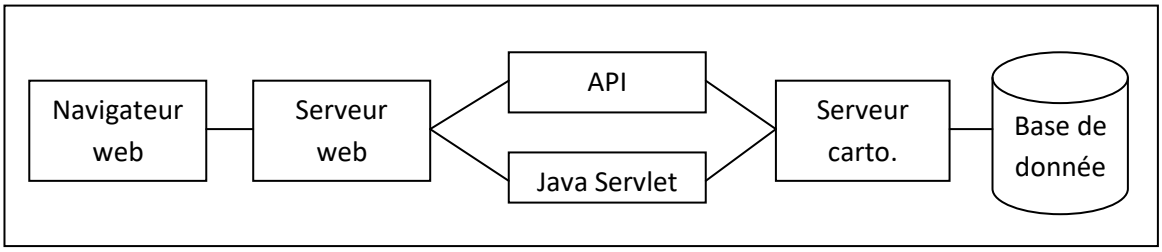


Figure 2 : Architecture côté serveur (Alesheikh et al., 2002 ; modifié)

Un intérêt de l'architecture côté serveur est qu'une connexion vers des API[1] (Application Programming Interface) ou vers des Java servlets[2] est possible, offrant des opportunités de traitements supplémentaires (Alesheikh et al., 2002). Par ailleurs, le serveur est généralement plus puissant que le client et une architecture côté serveur permet donc des traitements plus importants (Kulawiak et al., 2019). Pour les mêmes manipulations, le côté serveur obtient des résultats plus rapidement lorsqu'il s'agit d'opérations demandant une mémoire conséquente. Il est également plus rapide lors de traitements sur un volume très important de données.

Les avantages de l'architecture côté serveur sont les suivants (Kulawiak et al., 2019 ; Alesheikh et al., 2002) :

- Opérations plus rapides sur un large set de données et pour des traitements importants
- Contrôle et mise à jour centralisée
- Intégration facile dans des SIG-logiciels
- Requête standardisée

Les désavantages de l'architecture côté serveur sont les suivants :

- Temps de réponses plus long si un petit set de données
- Moins d'interactions possibles
- Échange avec le serveur à chaque opération

2.2. Architecture côté client

Dans l'architecture côté client, les données sont directement manipulées par le client, généralement un navigateur web (Figure 3). Ce dernier ne sait pas manier les données en html mais doit utiliser un code JavaScript ou un applet Java[3] pour étendre ses fonctionnalités (Alesheikh et al., 2002). L'utilisation de librairies JavaScript est la méthode la plus utilisée (voir point 7.1).

[1] **API** : « Ensemble de fonctions qui facilitent, via un langage de programmation, l'accès aux services d'une application. » (JDW, 2019)
[2] **Servlet** : « Ensemble de routines standards, accessibles et documentées, qui sont destinées à faciliter au programmeur le développement d'applications. » (OQLF, 2002)
[3] **Applet** : « Petit programme destiné à effectuer une tâche très précise, et conçu pour s'exécuter à l'intérieur d'une autre application. » (OQLF, 2009)

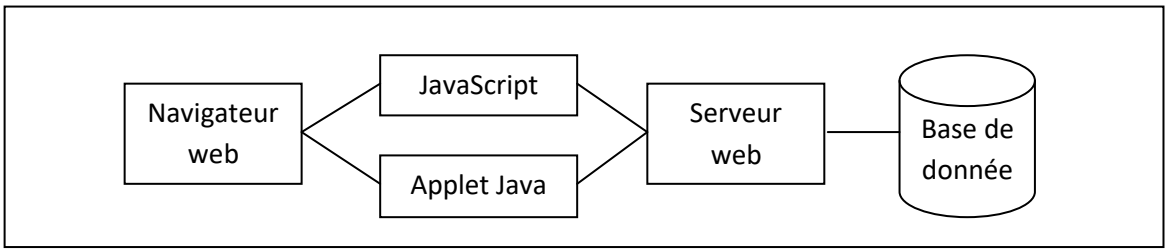


Figure 3 : Architecture côté client (Alesheikh et al., 2002 ; modifié)

Pour des données de petite taille ou pour des petites manipulations, l'architecture côté client obtient des résultats beaucoup plus rapidement que le côté serveur (Kulawiak et al., 2019). En effet, une interaction entre le navigateur web et le serveur ne s'avère pas nécessaire à chaque opération.

Les avantages de l'architecture côté client sont (Kulawiak et al., 2019 ; Alesheikh et al., 2002) :

- Temps de réponses court pour des petites manipulations ou pour un petit set de données
- Larges possibilités de traitements grâce aux plug-in des librairies
- Plus d'interactions et animations possibles
- Meilleure qualité pour l'affichage des données (non limité à la qualité de l'image PNG/JPEG/etc.)

Les inconvénients de l'architecture côté client sont :

- Non optimal pour un large set de données ou pour des traitements importants
- Pas de standard
- Clients pas toujours compatibles

3. Sémiologie graphique

« La sémiologie graphique est l'ensemble des techniques et méthodes visant à adapter un mode de représentation graphique à l'information représentée en fonction de codes esthétiques et de conventions. » (Géoconfluences, 2017)

3.1. Variables rétiniennes

Bertin (1967) définit 8 variables graphiques dans un plan :

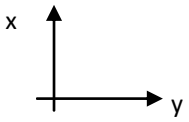
La position sur l'axe des x	
La position sur l'axe des y	
La taille ou superficie du symbole	
La valeur de blanc à noir (l'intensité)	
La couleur (la teinte)	
Le grain[4]	
L'orientation	
La forme	

Tableau 1 : Variables rétiniennes

En cartographie, les deux premières variables (Tableau 1), la position sur l'axe des x et la position sur l'axe des y, sont fixes sur le territoire et ne peuvent donc pas varier (Donnay, 2013). Les autres (taille, valeur, couleur, grain, orientation et forme) sont appelées variables rétiniennes. À noter également que la variation des variables n'est pas infinie : elle est limitée par la capacité de l'œil humain à distinguer les différentes variations mais aussi par l'emplacement des éléments sur le territoire (risque de chevauchements).

3.2. Perception des variables rétiniennes

Les variables rétiniennes ne présentent pas toutes la même perception visuelle. Selon Bertin (1967), la forme, l'orientation, la couleur (la teinte) et le grain ont une perception nominale tandis que la valeur et la taille ont une perception ordonnée. La taille peut aussi être perçue comme niveau quantitatif. Cependant, une perception de

[4] « Le grain correspond à la densité de symboles par unité de surface. » (Donnay, 2013)

taille quantitative est uniquement possible avec des symboles simples et une proportionnalité (le rapport entre deux quantités est égal au rapport de la superficie des symboles correspondants).

La couleur (la teinte et l'intensité) est également utilisée pour une échelle qualitative ordonnée au travers respectivement d'une palette de couleurs et des niveaux de gris. Brewer et Harrower (2003) ont créé un outil permettant de choisir une palette de couleurs adéquates en fonction du nombre de classes et de la nature des données (Figure 4). Bien que la couleur n'ait pas une perception quantitative, les phénomènes spatialement continus peuvent être représentés à l'aide d'une palette de couleurs ou des niveaux de gris.

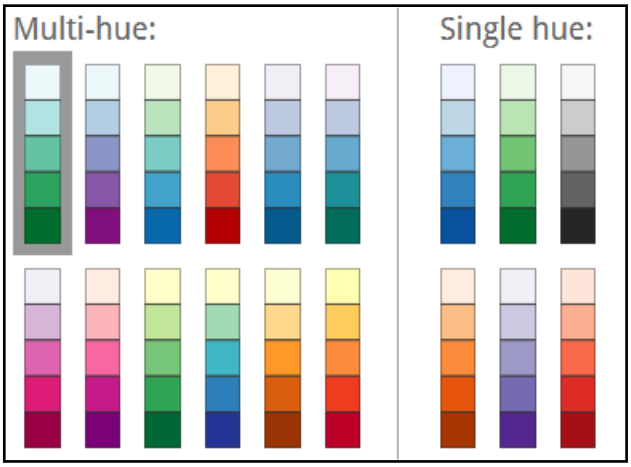


Figure 4 : Exemple de palettes de couleurs séquentielles (Colorbrewer2.org)

L'utilisation de trame (Figure 5) avec une variation de formes donne une perception nominale tandis qu'une variation d'épaisseurs ou d'espacements donne une perception ordonnée (Donnay, 2013).

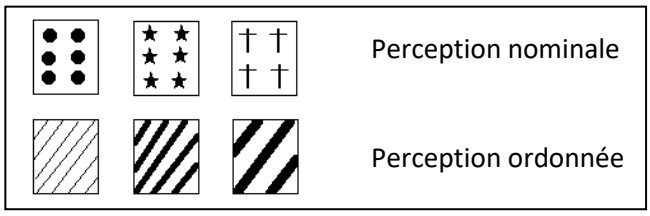


Figure 5 : Perception des trames

3.3. Cartes à une composante

Le mode d'implantation influence également le choix de la variable. Toujours d'après Bertin (1967), une variation de couleurs (teintes) est perceptible dans les trois modes d'implantation tandis que l'utilisation du grain en mode ponctuel est limitée par la taille du symbole. Une variation de valeurs ou d'orientations peut difficilement être mise en place en mode linéaire. Pour finir, la taille, la forme et l'orientation ne s'appliquent pas à une implantation zonale mais peuvent être appliquées à des symboles positionnés sur la géométrie.

La combinaison des perceptions au niveau de l'échelle et du mode d'implantation donne un ou plusieurs choix possibles de représentations adéquates (Tableau 2).

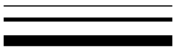
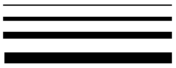
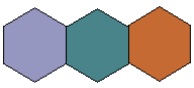
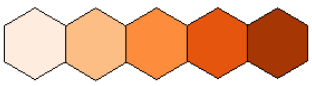
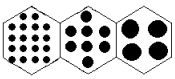
	Nominal	Ordonné	Quantitatif
Ponctuel	Forme  Teinte 	Taille 	Taille proportionnelle 
Linéaire	Forme  Teinte 	Taille 	Taille proportionnelle 
Zonal	Teinte 	Palette de couleurs 	Symbole de taille proportionnelle 
Continu			Palette de couleur 

Tableau 2 : Carte à une composante (Donnay, 2013 ; modifié)

La cartographie d'un attribut quantitatif en implantation zonale est très limitée et peu efficace (Donnay, 2013). Deux solutions sont alors possibles : une transformation de la géométrie ou une transformation de l'échelle de mesures. La première consiste à réaliser une carte à symboles centrés, c'est-à-dire que le centre de gravité de chaque polygone est calculé et un cercle de taille proportionnelle à la valeur de l'attribut y est implémenté. Dans ce cas, il y a une perte de la dimension zonale. Cette transformation est principalement utilisée lorsque l'attribut n'a pas de composante zonale. La seconde possibilité est de discrétiser les valeurs (voir point 4) pour réaliser une carte d'un attribut ordonné en implantation zonale. À l'inverse, cette transformation est utilisée lorsque l'attribut a une composante zonale (par exemple, la densité de population) car l'utilisation de cette visualisation pour un attribut qui n'en a pas peut fausser la perception du lecteur.

3.4. Série divergente

Lorsque les données sont articulées autour d'un point de rupture, la série de données est dite divergente et une palette du même type doit être utilisée (Harrower & Brewer, 2003). Ces palettes sont toujours multi-teintes. Lorsque les données sont composées d'un pivot ou d'une classe centrale (par exemple : la valeur zéro, des valeurs positives et des valeurs négatives), ce pivot est représenté par la couleur jaune ou gris tandis que les autres classes sont construites à l'aide de deux palettes séquentielles de part et d'autre (Figure 6, à gauche) (Donnay, 2013). Dans le cas contraire (par exemple : des ordres de grandeur de température : très chaud, chaud, froid, très froid), la palette est également construite à l'aide de deux palettes séquentielles mais le point de rupture n'est pas représenté (Figure 6, à droite).

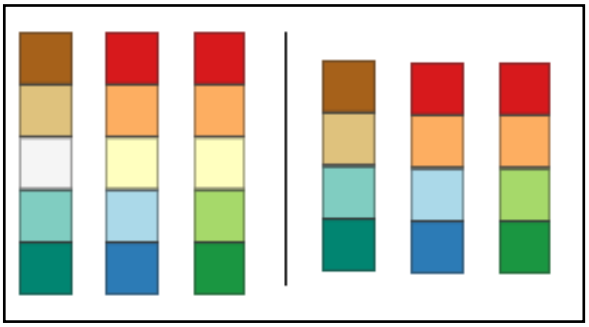


Figure 6 : Exemple de palettes divergentes
(Colorbrewer2.org, modifié)

3.5. Cartes à deux composantes

Lorsque deux composantes doivent être représentées sur la même entité géographique, les possibilités sont plus limitées (Donnay, 2013). La meilleure représentation serait de faire une transformation (rapport, différence, etc.) pour n'avoir qu'une seule composante pour les deux attributs.

En implantation ponctuelle et linéaire:

- Le nominal est représenté par une variation de teintes pour l'un et par une variation de formes pour l'autre.
- L'ordonné est représenté par une variation de tailles pour l'un et par une variation d'intensités pour l'autre.
- Le quantitatif est représenté par des tailles proportionnelles pour les deux attributs.

En implantation zonale :

- Une des composantes est représentée en zonale et l'autre composante en ponctuelle.
- L'échelle ordonné/ordonné est représentée par une composition colorée.
- La représentation quantitatif/quantitatif est peu efficace.

Figure 7 : Exemple de composition colorée

4. Discrétisation d'une variable quantitative

Comme indiqué au point 2, le choix de variables rétiniennes pour les attributs dont l'échelle est quantitative est très limité. Pour remédier à cela, une discrétisation ou classification de la variable quantitative peut être envisagée, permettant de passer d'une échelle quantitative à une échelle qualitative ordonnée. Différentes classifications sont possibles et sont détaillées ci-dessous (Cauvin et al., 1987).

4.1. Intervalles égaux

Comme son nom l'indique, cette méthode se base sur des intervalles égaux donc sur des classes de même étendues. Soit k le nombre de classes, le calcul de l'étendue se fait comme suit :

$$e = \frac{\text{Max} - \text{min}}{k} \quad (1)$$

Les limites de classes sont ensuite définies en ajoutant k fois la valeur de e :

Classe 1 : [min ; min+e [
Classe 2 :[min+e ; min+2e [
Classe 3 : [min+2e ; min+3e [
…
Classe k : [min+(k-1)e ; Max]

Avantages	**Inconvénients**
Exécution facile	Pas de comparaison possible entre variables
Effectifs égaux dans chaque classe (si distribution uniforme)	Possibilité de classes vides (si distribution dissymétrique)

Tableau 3 : Avantages/inconvénients de la classification par intervalles égaux

4.2. Progression arithmétique

L'étendue des classes augmente progressivement selon une progression arithmétique. Si on définit la raison R de la progression : l'étendue de la première classe sera R, celle de la seconde sera 2R, l'étendue de la classe k sera kR. Comme l'étendue de la variable est égale à la somme des étendues de chaque classe, on peut en déduire que la raison est

$$R = \frac{Max - min}{\sum_{j=1}^{k} j} \quad (2)$$

Les limites de classes sont les valeurs de limites précédentes plus k fois la raison :

Classe 1 : [min ; min+R [
Classe 2 :[min+R ; (min+R)+2R [
Classe 3 : [min+3R ; (min+3R)+3R [
…
Classe k : [min+ (1+2+…+(k-1).R) ; Max]

Avantages	Inconvénients
Étale les petites valeurs	Possibilité de classes vides
Classes plus détaillées dans la partie inférieure que dans la partie supérieure de la distribution	

Tableau 4 : Avantages/inconvénients de la classification par progression arithmétique

4.3. Progression géométrique

L'étendue des classes augmente progressivement selon une progression géométrique, c'est-à-dire avec une multiplication de la raison plutôt qu'une addition en comparaison avec la progression arithmétique. Comme le maximum vaut le minimum multiplié par la raison autant de fois qu'il y a de classes, on peut déduire que

$$R^k = \frac{Max}{min} \Leftrightarrow \log(R) = \frac{\log(Max) - \log(\min)}{k} \quad (3)$$

Les limites des classes sont définies comme R^k fois le minimum :

Classe 1 : [min ; minR [
Classe 2 :[minR ; minR² [
Classe 3 : [minR² ; minR³ [
…
Classe k : [$minR^{k-1}$; Max]

Avantages	Inconvénients
Étale les petites valeurs	Impossibilité d'avoir des valeurs ≤ 0
Efficace pour une distribution asymétrique (vers la gauche)	Possibilité de classes vides

Tableau 5 : Avantages/inconvénients de la classification par progression géométrique

4.4. Quantiles – classes d'égales fréquences

Cette classification retient un nombre de valeurs égales dans chaque classe. Soit N le nombre de valeurs totales, le nombre de valeurs par classe :

$$n = \frac{N}{k} \quad (4)$$

La valeur de n doit être arrondie. Les limites des classes sont définies en comptant le nombre de valeurs par classe :

Classe 1 : [min ; x(n) [
Classe 2 :[x(n) ; x(2n) [
Classe 3 : [x(2n) ; x(3n) [
…
Classe k : [x(k-1.n) ; Max]

Avantages	**Inconvénients**
Pas de classe vide	Peut regrouper des valeurs fort éloignées
Répartition équilibrée	Pose des problèmes lorsque beaucoup d'entités ont la même valeur

Tableau 6 : Avantages/inconvénients de la classification par quantiles

4.5. Écart-type

La classification par écart-type convient pour une distribution uniforme et une distribution normale. Par conséquent, il est recommandé de réaliser une transformation au préalable. La première étape est de calculer la moyenne ($\bar{x}$) et l'écart-type de la distribution (s).

Ensuite, deux choix de limite de classes sont possibles :

- Nombre pair de classes, par exemple pour 4 classes :

Classe 1 : [min ; $\bar{x} - s$ [
Classe 2 :[$\bar{x} - s$; $\bar{x}$ [
Classe 3 : [$\bar{x}$; $\bar{x} + s$ [
Classe 4 : [$\bar{x} + s$; Max]

- Nombre impair de classes, par exemple pour 5 classes :

Classe 1 : [min ; $\bar{x} - 1{,}5.\, s$ [
Classe 2 :[$\bar{x} - 1{,}5.\, s$; $\bar{x} - 0{,}5.\, s$ [
Classe 3 : [$\bar{x} - 0{,}5.\, s$; $\bar{x} + 0{,}5.\, s$ [
Classe 4 : [$\bar{x} + 0{,}5.\, s$; $\bar{x} - 1{,}5.\, s$
Classe 5 : [$\bar{x} + 1{,}5.\, s$; Max]

Avantages	Inconvénients
Permet les comparaisons entre variables	Risque de classe centrale très peuplée
	Limite difficile à comprendre pour un public non averti

Tableau 7 : Avantages/inconvénients de la classification par écart-type

4.6. Moyennes emboitées

Le principe de cette méthode est de considérer la moyenne arithmétique comme séparation de deux classes puis de calculer la moyenne arithmétique de chaque sous-ensemble pour trouver deux nouvelles limites. Par conséquent, le nombre de classes est obligatoirement une puissance de deux. Par exemple pour 4 classes, les limites sont :

Classe 1 : [min ; m1 [
Classe 2 :[m1 ; $\bar{x}$ [
Classe 3 : [$\bar{x}$; m2 [
Classe 4 : [m2 ; Max]

Avec m1 et m2 respectivement les moyennes entre [min ; $\bar{x}$ [et [$\bar{x}$; Max].

Avantages	Inconvénients
Pas de classe vide	L'amplitude des classes est très variable
Pas d'importance aux valeurs extrêmes	Obligatoirement puissance de deux classes

Tableau 8 : Avantages/inconvénients de la classification par moyennes emboitées

4.7. Jenks – Seuils naturels

Cette classification est basée sur la loi probabiliste de Fisher. Il s'agit d'un algorithme itératif qui vise à minimiser la variance intra-classes et à maximiser la variance inter-classes. L'utilisateur définit un nombre de classes k et calcule :

$$GVF = SCE_B/SCE_W \quad (5)$$

$$SCE_B = \sum_{j=1}^{k} (\overline{x_j} - \bar{x})^2 \quad (6)$$

$$SCE_W = \sum_{j=1}^{k} \sum_{i=min_j}^{max_j} (x_i - \bar{x_j})^2 \quad (7)$$

Il faut ensuite déplacer une entité de la classe ayant la plus grande variance vers la classe voisine la plus proche puis recalculer GVF jusqu'à ce qu'il ne soit plus améliorable.

Avantages	Inconvénients
Minimise l'erreur de généralisation	Pas de comparaison entre variables
Maximise l'homogénéité des classes	La durée de calcul peut être importante

Tableau 9 : Avantages/inconvénients de la classification de Jenks

Il existe également des classifications sur base graphique mais leur utilisation est peu recommandée et leur rigueur est discutable étant donné que les limites de classes sont définies visuellement par l'auteur.

4.8. Choix de classifications

Différents indices permettent de comparer les classifications et d'identifier la plus appropriée selon les données. La méthode la plus connue est proposée par Jenks : il s'agit de l'indice TAI (Tabular Accuracy Index) qui « mesure l'écart entre les valeurs originales et les valeurs correspondantes transformées par l'utilisation des classes » (Jenks & Caspall, 1971). La méthode de seuils naturels obtient généralement le meilleur résultat pour ce critère. Plus cet indice est proche de 1, meilleure est la discrétisation. La formule de cet indice :

$$TAI = 1 - \frac{\sum_j \sum_i |Z_{ij} - m_j|}{\sum_i |Z_i - M|} \quad (8)$$

Avec Z_{ij} : les valeurs Z_i de la classe j, m_j : la moyenne de la classe j, Z_i : chaque valeur des données, M : la moyenne générale.

Le second indice intéressant mesure la perte d'informations de la classification : il s'agit de l'entropie (Cauvin et al., 1987). Tout d'abord, l'entropie maximale doit être calculée avec k le nombre de classes :

$$H_{max} = \frac{\log k}{\log 2} \quad (9)$$

Ensuite, l'entropie de la classification est calculée grâce à la formule :

$$H = \sum_j \left(pj \frac{\log p_j}{\log 2} \right) \quad (10)$$

Avec p_j : la probabilité d'apparition dans la classe, c'est-à-dire le rapport entre nombre d'éléments dans la classe j et le nombre total de valeurs.

L'indice est ensuite trouvé par :

$$R = 1 - \frac{H}{H_{max}} \quad (11)$$

La classification est meilleure lorsque l'indice se rapproche de 0. La méthode d'égales fréquences obtient le meilleur résultat.

5. Interactivité et animation

Les variables rétiniennes sont utilisées depuis longtemps sur cartes papiers puis sur cartes statiques. Avec l'évolution de la technologie, la cartographie a vu naitre de nouveaux outils de représentation comme l'interactivité et de nombreuses définitions de celle-ci existent suivant le domaine. Selon Jensen (1998) : « L'interactivité est la mesure de la capacité potentielle d'un médium à laisser l'utilisateur exercer une influence sur le contenu et/ou la forme de communication médiate ». Cette définition met en avant le contrôle qu'a l'utilisateur sur les informations qu'il souhaite voir. En effet, la cartographie interactive est de plus en plus présente dans notre quotidien et la population a pris l'habitude de pouvoir se déplacer, zoomer, cliquer sur une carte afin d'obtenir plus d'informations. Par conséquent, de nouvelles variables graphiques ont commencé à être utilisées avec le développement de la 3D et de l'interactivité. Elles peuvent également être utilisées pour représenter de l'information géographique (Semmo et al. 2015) :

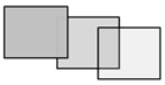
Le matériau	
La texture	
La transparence	
L'animation	

Tableau 10 : Variables graphiques

Dans le cadre d'un SIG web, l'animation est une variable intéressante. Selon Vasiliev (1997), « une carte animée est une carte qui fournit une image mouvante de l'information géographique à travers le temps ». Différents aspects de la carte peuvent changer : l'espace, le temps ou l'attribut (Cauvin et al., 2008). Des combinaisons de variations sont possibles et donnent lieu à différentes animations (Figure 8).

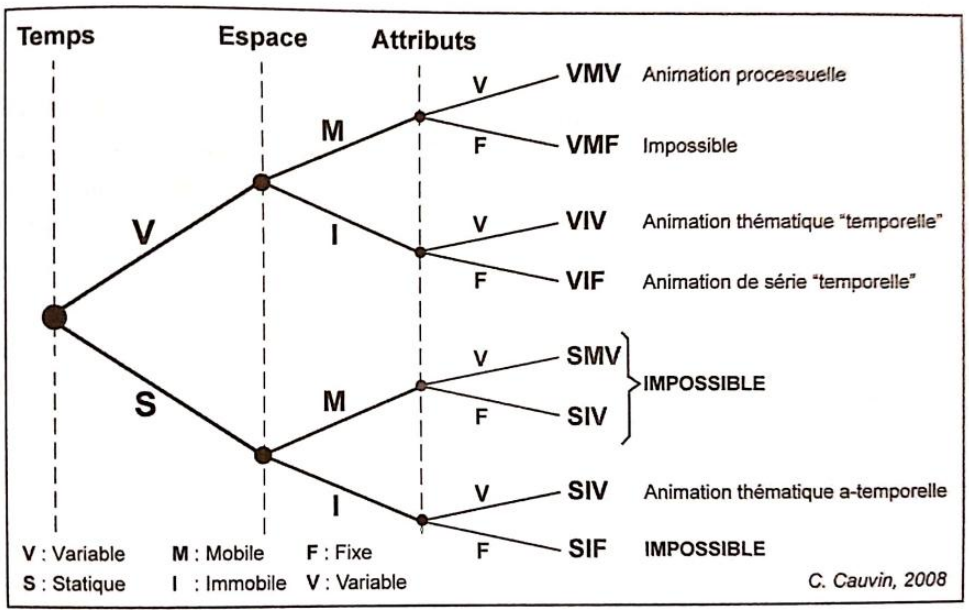


Figure 8 : Animations selon les variations possibles (Cauvin, 2008)

Les plus utilisées sont l'animation thématique temporelle (temps variable, espace immobile et attribut variable) et l'animation de série temporelle (temps variable, espace immobile et attribut fixe). Elles donnent à la carte une illusion de mouvement comme une série de cartes qui se succèdent. Une fréquence de changements doit être définie afin de ne plus distinguer les différentes images statiques mais elle dépend du volume de données et de la puissance du medium qui diffuse l'animation (Harrower & Fabrikant, 2008). Le temps d'exécution d'une animation est théoriquement infini. Cependant, plus celui-ci augmente, plus l'utilisateur aura des difficultés à se souvenir des informations présentées. Par conséquent, les animations temporelles ont généralement une durée de quelques minutes maximum. Une légende temporelle sous forme d'horloge, de texte ou d'une barre de défilement est utilisée pour indiquer à l'utilisateur l'avancée dans le temps. Cette échelle peut également varier pour se concentrer sur une période en particulier. L'interactivité peut également y être

associée pour choisir une date/heure, ralentir/accélérer le défilement, mettre en « pause » l'animation pour analyser un élément, etc.

L'apparition des SIG web n'a pas uniquement modifié l'interaction possible entre un utilisateur et une carte, elle a également modifié le rôle même de la cartographie (Kraak, 2004). En effet, une carte n'est plus uniquement une abstraction de la réalité géographique, elle est également très utilisée à des fins de recherches dans une base de données comme critère de localisation ou comme prévisualisation de résultats d'une requête spatialisée.

6. Formats d'échange de données spatiales

Différents formats de données spatiales sont utilisés pour l'échange des données sur internet. Les principaux sont basés sur le format JSON (JavaScript Object Notation) et le XML (Extensive Markup Language). Ils font partie des métaformats, c'est-à-dire qu'ils contiennent une syntaxe standardisée mais n'ont pas de balises ou de clés prédéfinies (W3C, 2015).

6.1. XML vs JSON

Le XML tout comme le JSON sont des formats hiérarchisés, ils sont conçus pour stocker et échanger des données facilement sous forme de texte auto-descriptif (lisible et compréhensible par l'homme) (W3schools, 2021a, 2021b, 2021c). Le XML comporte des balises semblables au HTML mais celles-ci ne sont pas prédéfinies. Le JSON quant à lui est défini selon la syntaxe JavaScript (mais est indépendant du langage) (Figure 9).

XML

```
<étudiants>
    <étudiant>
        <prénom>Martin</prénom> <nom>Dupont</nom>
    </étudiant>
    <étudiant>
        <prénom>Carine</prénom> <nom>Dubois</nom>
    </étudiant>
    <étudiant>
        <prénom>Robert</prénom> <nom>Durant</nom>
    </étudiant>
</étudiants>
```

JSON

```
{"étudiants":[
  { "prénom":"Martin", "nom":"Dupont" },
  { "prénom":"Carine", "nom":"Dubois" },
  { "prénom":"Robert", "nom":"Durant" }
]}
```

Figure 9 : Comparaison entre le format XML et JSON

Le format JSON, plus récent, est devenu une alternative au XML. Ses avantages sont nombreux : syntaxe plus simple, les langages orienté-objet peuvent le traduire plus facilement et plus rapidement (Šimec & Magličić, 2014). Cependant, le format XML est plus adapté pour transférer des documents contenant différents types de données. Les deux formats sont donc complémentaires. Dans le cadre d'un SIG web et d'une communication entre serveur et navigateur web, le JSON est donc le format qui convient le mieux.

6.2. GML

Le GML (Geography Markup Language) est une application du langage XML pour les entités géographiques (OGC, 2021). Il comporte une première partie avec les métadonnées (système de projection, géométrie, etc.) et une seconde contenant les données.

6.3. GeoJSON, TopoJSON, CovJSON

Différents formats dérivés du JSON sont apparus ces dernières années, les plus connus sont les 3 suivants :

GeoJSON (Geographic JavaScript Object Notation) est un format d'encodage de données géographiques (IETF, 2016). Il contient à la fois les attributs et les géométries des entités.

Le TopoJSON (Topology JavaScript Object Notation) est une extension du GeoJSON qui conserve la topologie des données (Bostock, 2017). Dans ce format, les géométries sont représentées à l'aide d'arcs : un ensemble d'arcs peut représenter à la fois un polygone mais également des polylignes. Il est donc plus compact et moins volumineux cependant moins aisé à manipuler.

Le CovJSON (Coverage JavaScript Object Notation) est un format d'encodage de données de « couverture » (Blower et al., 2017), c'est-à-dire des données rasters (sous forme de grille), des séries temporelles (plusieurs données à différentes périodes pour un même point), des profils verticaux (plusieurs données à différentes altitudes pour le même point), etc. Il a été développé pour dépasser les limites du GeoJSON qui ne permet pas l'utilisation d'une géométrie sous forme de grille ou de coordonnées temporelles.

7. Outils web GIS côté client

7.1. Outil de représentation cartographique

Plusieurs librairies JavaScript sont disponibles pour représenter des entités géographiques. Les principales sont *Leaflet, OpenLayers* et *Mapbox*. Ces librairies sont assez complètes et mises à jour très régulièrement. Il existe également des librairies JavaScript développées pour une utilisation précise de représentations comme *Covjson-reader*.

Les 3 librairies de représentation cartographique ont énormément de choses en commun (Khitrin, 2017). Cependant, *Leaflet* a la particularité d'être séparée en différents modules. Il en découle une réduction de la taille de la librairie et une augmentation de la rapidité de rafraichissement des pages. *OpenLayers* possède

nativement la plupart des fonctionnalités tandis que *Leaflet* utilise des plugins. *Mapbox* et *OpenLayers* supporte le WebGL[5] tandis que *Leaflet* non (Zunino et al., 2020). Un autre avantage de la librairie *Mapbox* est qu'elle permet l'utilisation de fond de carte vectoriel. Pour des données rasters, *OpenLayers* montre de meilleures performances tandis que pour des données vecteurs, *Mapbox* obtient des résultats probants. Cependant, la diversité des plugins annexes développés pour *Leaflet* fait de celle-ci la plus complète pour une utilisation vectorielle actuellement. Cette librairie, combinée à ses nombreux plugins, offre énormément de possibilités de symbolisation et sera utilisée pour la partie développement de ce mémoire.

La librairie *Covjson-reader* permet de lire le format de fichier CovJSON et de le traduire en Object JavaScript. Le plugin *Leaflet-coverage* permet de visualiser dans *Leaflet* les données CovJSON à l'aide de la librairie *Covjson-reader*.

7.2. Outil de traitement cartographique

Turf.js est une librairie de traitement cartographique et d'analyse spatiale. Elle offre de nombreuses possibilités de traitement des données comme le passage d'une géométrie à une autre, la recherche du centre de gravité, l'intersection entre deux géométries, le calcul d'un Buffer, etc.

Geostats et *Classybrew* sont des librairies de classifications statistiques. La première est plus générale tandis que la seconde fournit une représentation facilitée des données géographiques. L'intérêt de *Geostats* est qu'elle contient un grand nombre de méthodes permettant de récupérer le minimum, le maximum, le nombre d'individus, la variance etc. pour chaque classe. *Classybrew* permet de choisir parmi plusieurs algorithmes de classification et d'automatiquement créer l'association couleur/valeur récupérable par une méthode.

[5] « WebGL est une spécification d'affichage 3D pour les navigateurs web. Elle permet d'utiliser le standard OpenGL depuis le code JavaScript d'une page web » (Techno-sciences.net, 2021).

III. Hypothèses et méthodologie

L'objectif de ce mémoire est de réaliser un outil cartographique permettant la symbolisation automatique des entités géographiques 2D dans un SIG web. Pour ce faire, une librairie JavaScript est développée contenant une représentation automatique pour chaque niveau d'échelle/mode d'implantation possible.

Pour les phénomènes spatialement discrets, l'hypothèse est de combiner plusieurs librairies JavaScript déjà existantes. De nombreuses librairies permettant la symbolisation et la classification statistique ont été développées afin de faciliter la cartographie d'entités discrètes en SIG web. Ces librairies utilisent des fichiers GeoJSON en entrée.

Concernant les phénomènes spatialement continus, la méthode la plus utilisée pour les visualiser est de passer par un WMS (Web Map Service). Il existe très peu de moyens pour faire de la cartographie client avec des données continues. Celle-ci présente pourtant de nombreux avantages décrits au point II.2. Comme démontré précédemment, l'utilisation d'un GeoJSON n'est pas possible pour des données continues. Il est donc nécessaire d'employer un fichier CovJSON avec les deux librairies permettant sa lecture et son utilisation dans Leaflet (*Leaflet-coverage* et *Covjson-reader*).

IV. Développement

Comme expliqué dans la partie hypothèses et méthodologie, l'objectif est de développer une librairie JavaScript contenant une représentation automatique pour chaque niveau d'échelle/mode d'implantations possibles. Plus exactement, la librairie permet automatiquement d'afficher les données, de créer une légende correspondante et d'interroger la couche avec l'apparition, au clic, d'un popup contenant les informations présentes dans la table d'attribut.

La liste complète des librairies utilisées est disponible en Annexe 1.

1. Phénomènes spatialement discrets

Une fonction est implémentée par niveau d'échelle (nominal, ordonné, quantitatif) pour les phénomènes spatialement discrets. Le developpeur doit être en mesure de savoir dans quelle catégorie se situent ses données afin de pouvoir utiliser la librairie. Le mode d'implantation (ponctuel, linéaire et zonal) est déterminé automatiquement par une condition sur le premier attribut du fichier GeoJSON. Par exemple pour le ponctuel :

```
if (data.features[0].geometry["type"]=="Point"
||data.features[0].geometry["type"]=="MultiPoint")
```

Une condition pour le type « Line/MultiLine » et « Polygon/MultiPolygon » est également créée de manière similaire.

La symbolisation choisie pour chaque combinaison d'échelle/mode d'implantation est présentée dans le tableau ci-dessous (Tableau 11).

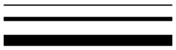
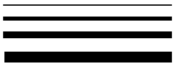
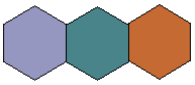
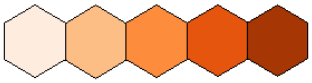
	Nominal	Ordonné	Quantitatif
Ponctuel	Teinte 	Taille 	Taille proportionnelle 
Linéaire	Teinte 	Taille 	Taille proportionnelle 
Zonal	Teinte 	Palette de couleurs 	Discrétisation ou cercles de taille proportionnelle

Tableau 11: Visualisation des phénomènes spatialement discrets

1.1. Nominal

Pour le niveau d'échelle nominal, une variation de teintes est utilisée car une variation de formes n'est pas évidente à définir en web mapping (nombre de formes assez limité). La fonction implémentée est la suivante (Annexe 2 pour le ponctuel, Annexe 3 pour le linéaire, Annexe 4 pour le zonal):

```
function nominal(map, file, attribute, colorPalette="mpn65", size="1")
```

Les paramètres obligatoires sont : *map* le nom de la couche sur laquelle la symbolisation est réalisée, *file* le nom du fichier GeoJSON contenant les données et *attribute* l'attribut sur lequel porte la symbolisation. Les paramètres optionnels que l'utilisateur a la possibilité de définir sont : *colorPalette* la palette de couleurs (la palette « mpn65 » est définie par défaut) et *size* la taille du rayon pour le ponctuel ou l'épaisseur de la ligne pour le linéaire (fixée à 1). Lorsque ces derniers paramètres ne sont pas indiqués, les options par défaut sont appliquées. Si l'utilisateur souhaite laisser la valeur par défaut pour un paramètre optionnel mais indiquer une valeur pour le suivant, *undefined* doit remplacer l'élément manquant.

Le principe de cette fonction est de parcourir les valeurs présentes dans la colonne de l'attribut choisi et de stocker les valeurs distinctes dans un tableau. Une couleur de la

palette choisie est ensuite associée à chacune des entrées du tableau. Comme il s'agit du nominal, la palette doit contenir une teinte différente pour chaque valeur. La librairie *palette.js* est utilisée pour générer les couleurs. Cette librairie contient les palettes « Paul Tol » et « ColorBrewer », différents choix sont donc possibles pour du qualitatif nominal comme par exemple « mpn65 », « qualitative », « cb-Paired », etc. Cependant, il faut un nombre suffisant de couleurs dans la palette pour correspondre au nombre de catégories ; si ce n'est pas le cas, une alerte indique à l'utilisateur que le nombre de valeurs est trop important pour la palette choisie. Lors de la création de la géométrie, celle-ci est colorisée en récupérant la couleur correspondante à la valeur de l'attribut.

Comme précisé précédemment, la taille des rayons de cercles, en ponctuel, ainsi que l'épaisseur des lignes, en linéaire, peuvent être choisies par l'utilisateur. Si ce dernier ne les définit pas, les valeurs par défaut sont appliquées.

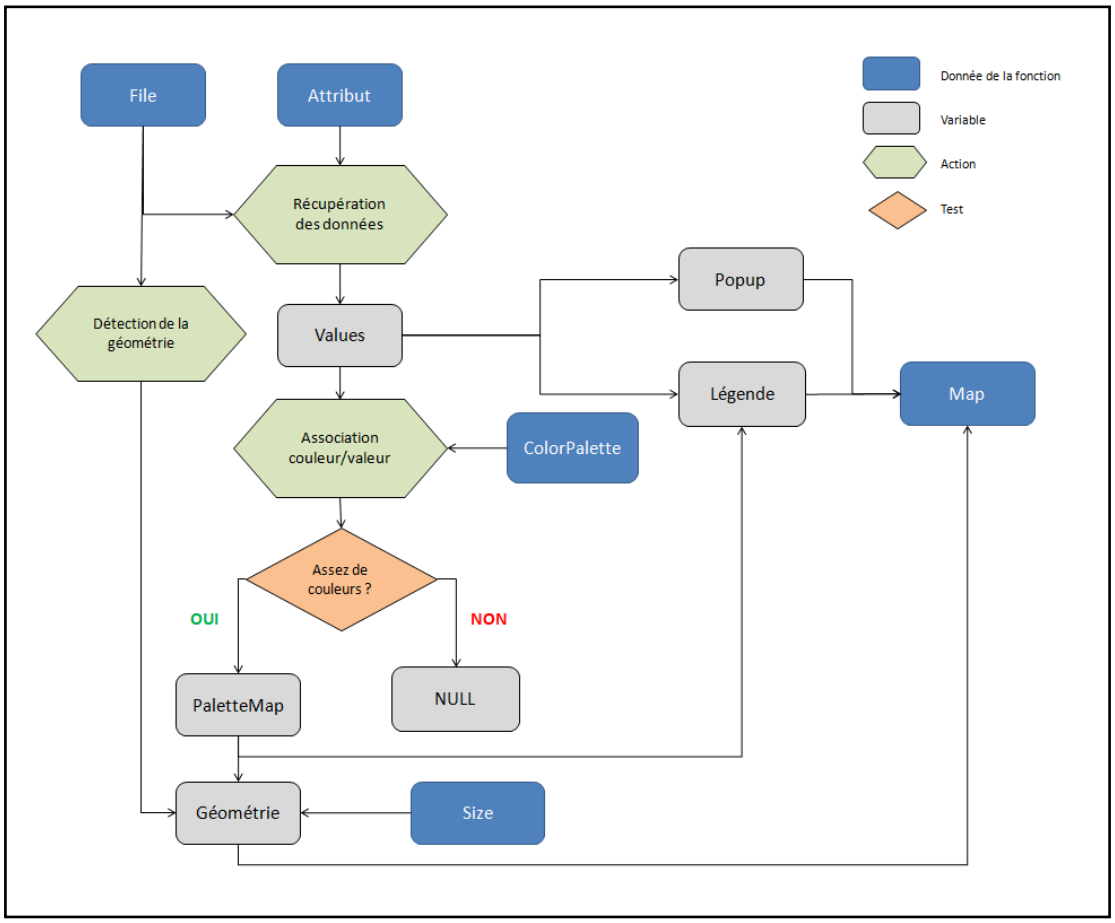


Figure 10 : Diagramme de la fonction « nominal »

1.2. Ordonné

Le ponctuel et le linéaire ordonné sont représentés par une variation de tailles. Le zonal est quant à lui matérialisé par une variation de teintes. Dans les 3 cas, la première étape consiste à stocker les différentes catégories dans un tableau. Il est ensuite nécessaire de vérifier si ces dernières sont des nombres, si c'est le cas, le tableau est trié par ordre croissant. Dans le cas contraire, une alerte indique à l'utilisateur qu'il doit définir un ordre. En effet, le paramètre *order* peut contenir l'ordre dans lequel les catégories doivent être placées. Plusieurs tests vérifient au préalable si le nombre de catégories indiquées par l'utilisateur correspond à celui des données et si celles-ci sont présentes dans le tableau.

L'étape suivante consiste à fournir une valeur de taille (pour le ponctuel et le linéaire) ou une couleur (pour le zonal) à chaque catégorie. Pour obtenir une perception ordonnée, il est nécessaire d'avoir une équidistance graphique entre chaque palier.

Dans le cas du ponctuel, représenté par des points, le rayon correspond à la formule :

$$R = \sqrt{{r_{min}}^2 * 2^i} \quad (12)$$

Avec R : le rayon de la classe, r_{min} : le rayon de la première classe et i : le numéro de classe allant de 0 à n. Le rayon minimum est soit défini par l'utilisateur lors de l'appel à la fonction, soit la valeur par défaut est appliquée. Pour le linéaire, l'équation utilisée est :

$$E = e_{min} * 2^i \quad (13)$$

Avec E : l'épaisseur de la classe, e_{min} : l'épaisseur de la première classe et i : le numéro de classe allant de 0 à n.

La fonction implémentée pour l'ordonné est la suivante (Annexe 5 pour le ponctuel, Annexe 6 pour le linéaire, Annexe 7 pour le zonal):

```
function ordonned (map,file, attribute, order=null, color=null,
first_size=1, diverging=false)
```

À nouveau les 3 paramètres obligatoires (*map, file et attribute*) sont présents. Un paramètre optionnel permet de définir la couleur ou la palette de couleurs. Si celui-ci

n'est pas défini, il prendra la valeur par défaut. Dans le cas du ponctuel et de l'ordonné, il s'agit de « #FF0000 », c'est-à-dire la couleur rouge. Pour le zonal, la palette par défaut est « cb-Reds » (Figure 11). Comme expliqué ci-dessus, l'élément *first_size* permet à l'utilisateur de définir la taille représentant la première classe. Enfin, *order* contient l'ordre des catégories. Si l'utilisateur ne définit pas d'ordre mais que les données l'exigent, un message d'alerte apparait.

Le fonctionnement du zonal ordonné est le même que pour le zonal nominal, à l'exception de la palette de couleurs qui doit être séquentielle, c'est-à-dire que les couleurs sont de mêmes teintes ou de teintes proches mais l'intensité change. A nouveau, la librairie *palette.js* est utilisée avec par exemple « cb-Reds » qui contient plusieurs intensités de rouge ou encore « cb-YlOrBr » contenant des teintes allant du jaune au brun en passant par l'orange (Figure 11).

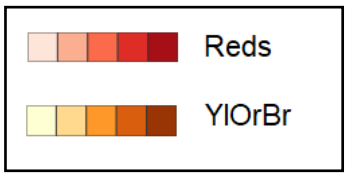


Figure 11 : Palettes Reds et YlOrBr

Le dernier argument de la fonction, *diverging*, indique s'il s'agit d'une série divergente (par défaut, définit « false »). Dans ce cas, les 3 implantations sont symbolisées par une variation de teintes. À nouveau, la librairie *palette.js* est utilisée avec une palette divergente (par exemple : «cb- BrBG » ou « cb-RdYlGn », « cb-RdBu » est appliquée par défaut). Les mêmes étapes que pour le nominal sont effectuées avec cette palette divergente.

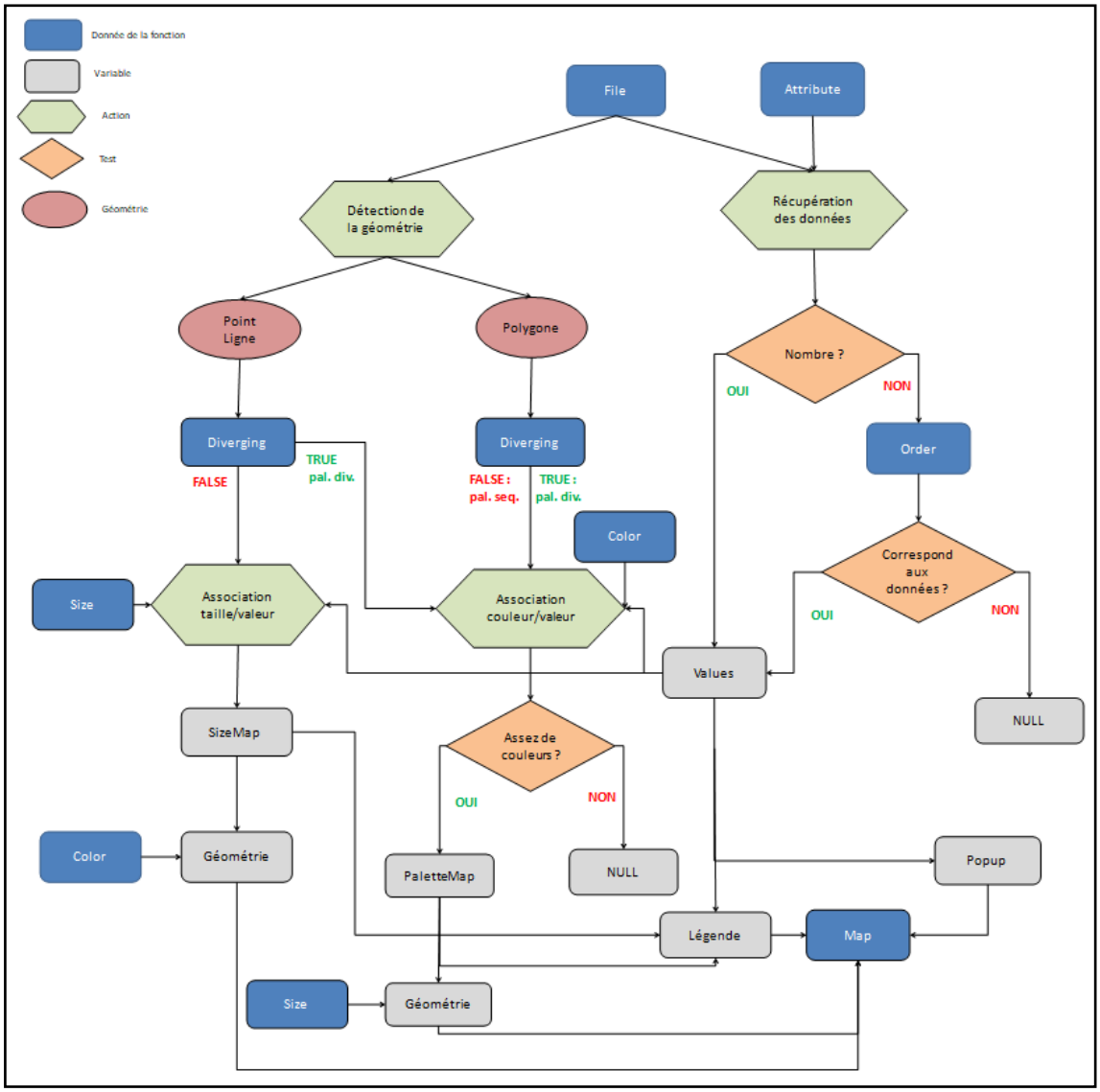


Figure 12 : Diagramme de la fonction « ordonné »

1.3. Quantitatif

Le quantitatif est représenté par une variation de tailles proportionnelles pour le ponctuel et le linéaire. Les données sont tout d'abord stockées dans un tableau en vérifiant qu'il s'agit de nombres. Ce tableau est ensuite trié par ordre croissant et les valeurs minimum, maximum et moyenne sont repérées. La formule permettant d'obtenir le rayon des cercles pour le ponctuel est la suivante :

$$s = \frac{\Delta S}{\Delta V}(v - v_{min}) + s_{min}$$

$$\pi R^2 = \frac{\pi R^2_{max} - \pi R^2{}_{min}}{v_{max} - v_{min}}(v - v_{min}) + \pi R^2{}_{min}$$

$$R = \sqrt{\frac{R^2{}_{max} - R^2{}_{min}}{v_{max} - v_{min}}(v - v_{min}) + R^2_{min}} \quad (14)$$

Avec R : le rayon du cercle obtenu, v : la valeur de la donnée, v_{min} : la valeur minimale dans les données et $R_{min/max}$: le rayon minimum/maximum défini par l'utilisateur.

Tandis que l'équation pour allouer une épaisseur de ligne proportionnelle en linéaire est donnée par :

$$e = \frac{\Delta e}{\Delta v}(v - v_{min}) + e_{min}$$

$$= \frac{e_{max} - e_{min}}{v_{max} - v_{min}}(v - v_{min}) + e_{min} \quad (15)$$

Avec e : l'épaisseur de ligne obtenue, v : la valeur de la donnée, $e_{min/max}$: l'épaisseur minimale/maximale définie par l'utilisateur et v_{min} : la valeur minimale des données.

La fonction est la suivante (Annexe 8 pour le ponctuel, Annexe 9 pour le linéaire) :

```
function quantitatif(map,file, attribute, color=null, first_size=1,
last_size=10, classification=true, classif=null, numClasses=5,
polygonColor="#ffffff", diverging=false)
```

Les paramètres à définir sont semblables à ceux de l'ordonné : les 3 premiers obligatoires (*map, file, attribute*) et les suivants optionnels avec une valeur par défaut pour la première/dernière taille de rayon/d'épaisseur et pour la couleur des éléments.

Une légende particulière est implémentée pour le ponctuel afin que les 3 cercles correspondant à la valeur minimum, maximum et moyenne du tableau s'emboitent (Figure 13).

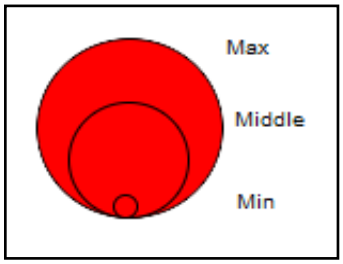


Figure 13 : Légende pour le ponctuel quantitatif

Comme expliqué dans l'état de l'art, deux choix sont possibles pour le zonal quantitatif. L'argument *classification* (définit « true » par défaut) permet de choisir l'option de symbolisation pour ce mode d'implantation.

Lorsque les données ont une composante zonale, une discrétisation des données est réalisée via les classifications présentées au point 4 de l'état de l'art. Dans ce cas, les polygones sont colorisés à l'aide d'une palette de couleurs séquentielle en fonction de la classe à laquelle ils appartiennent (Annexe 10). Pour ce faire, la librairie *Classybrew.js* est utilisée. Cette dernière permet de classer les données en fonction de la classification, la palette de couleurs et le nombre de classes choisis par l'utilisateur. Ceux-ci sont des paramètres optionnels et ont une valeur par défaut appliquée lorsque l'utilisateur ne définit pas les arguments.

Le paramètre *classif* est défini à « null » si aucune classification ne lui est assignée. Dans ce cas, le T.A.I. et l'entropie sont calculés pour chaque mode de classifications présent dans la librairie *Classybrew.js* (Jenks, intervalles égaux et quantiles). La fonction *best_classification* (Annexe 12) prend en entrée le tableau de valeurs *series,* les limites de classes *breaks* et le numéro de classes *numClasses.* Tout d'abord, un tableau contenant les valeurs entre deux limites de classes est créé. La valeur absolue entre chaque élément *values[i]* de la classe et sa moyenne *classes_j.mean()* est calculée puis sommée *sum_i*. Pour ce faire, la librairie *geostat.js* est utilisée afin de faciliter le calcul des moyennes. Les valeurs obtenues pour chaque classe *sum_i* sont ensuite sommées *sum_j*.

```
for(var i=0; i<values.length;i++){
      sum_i += Math.abs(values[i]-classes_j.mean());}
sum_j += sum_i;
```

Les valeurs absolues de la différence entre chaque élément *serie[i]* et la moyenne générale *classes.mean()* sont également sommées *sum*. Enfin, le T.A.I. est calculé par 1 diminué du rapport entre la somme *sum_j* et la somme *sum.*

```
for(var i=0; i<series.length;i++){
      sum += Math.abs(series[i]-classes.mean());}
var TAI = 1 - (sum_j/sum);
```

Pour l'entropie, le *H_max* est calculé par le rapport entre le logarithme du nombre de classes sur le logarithme de 2. La variable *pj* est le rapport entre le nombre d'éléments présents dans la classe *classes_j.pop()* et le nombre total d'éléments *classes.pop().* Cette variable est ensuite multipliée par le rapport entre son logarithme et le logarithme de 2 puis sommée *H*. L'entropie est calculée par 1 diminué du rapport entre l'opposé de *H* et le *H-max*.

```
var H_max = Math.log(numClasses)/Math.log(2);
for(var j=1;j<breaks.length;j++){
      var pj = classes_j.pop()/classes.pop();
      H += pj * (Math.log(pj)/Math.log(2));}
var entropy = 1 - (-H/H_max);
```

Si *diverging* est « true » la palette de couleurs par défaut est « RdBu », sinon « Reds ». Le fonctionnement des palettes divergentes est le suivant : les valeurs sont réparties en fonction de leurs nombres. S'il s'agit d'un nombre impair, les valeurs possèdent une classe centrale sinon les valeurs sont réparties de part et d'autre de la limite. Par exemple pour la palette « RdBu », avec 4 valeurs : les deux premières seront en bleu et les deux suivantes seront en rouge ; avec 5 valeurs : les deux premières seront en bleu, la classe centrale sera en blanc et les deux dernières seront en rouge. Il n'est pas possible de visualiser des classes non symétriques.

La deuxième possibilité, lorsque les données n'ont pas de composante zonale, est de rechercher le centre de gravité du polygone et d'y placer un cercle de taille

proportionnelle à la valeur (Annexe 11). Une première couche est ajoutée à la carte avec les polygones colorisés par le paramètre *polygonColor* puis une seconde couche contient les cercles positionnés au centre de gravité de chaque polygone. Le fonctionnement des cercles proportionnels est le même que pour le ponctuel quantitatif et la légende a également le même format.

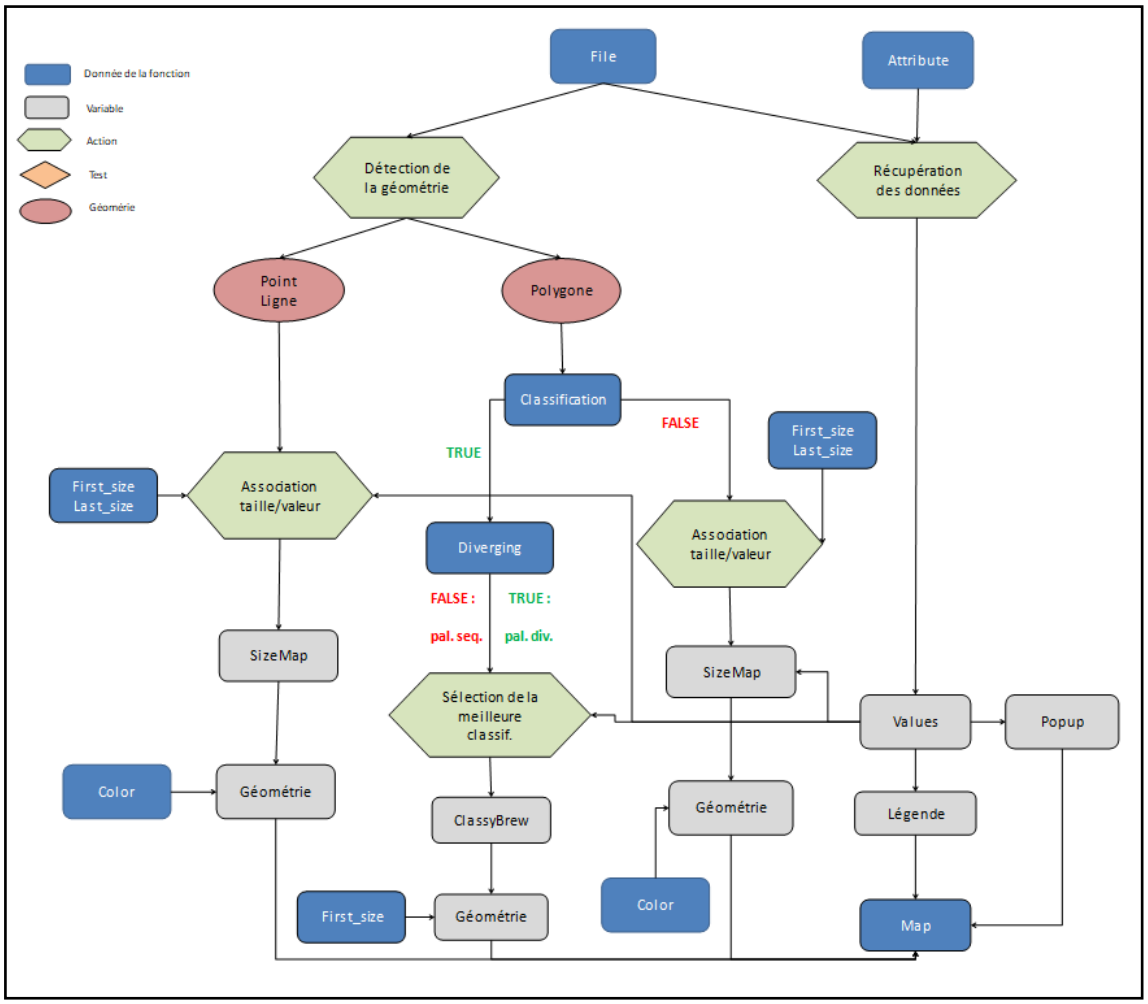


Figure 14 : Diagramme de la fonction « quantitatif »

2. Phénomènes spatialement continus

Comme précisé précédemment, l'utilisation du GeoJSON n'est pas adéquate dans le cas de phénomènes spatialement continus : un fichier CovJSON doit être utilisé. La fonction permettant la symbolisation automatique des données continues se présente sous la même forme que précédemment (Annexe 13) :

```
function continu(map, data, attribute, colorPalette)
```

Les 3 arguments principaux (*map, data, attribute*) sont toujours présents mais un paramètre supplémentaire est défini. Comme les données sont quantitatives, une palette de couleurs linéaire est utilisée, le paramètre *colorPalette* doit soit contenir un tableau contenant les deux couleurs de début et de fin de la palette (par exemple, ['#FFFFFF', '#FF0000'] pour obtenir une palette allant du blanc au rouge), soit *undefined* ou *null* qui appliquera la palette de couleurs par défaut. La symbolisation des phénomènes spatialement continus exploite la librairie *Leaflet-coverage*. Celle-ci visualise automatiquement les données CovJSON à travers la librairie *covjson-reader* mais elle crée également un popup contenant les attributs, et la légende qui correspond à la carte.

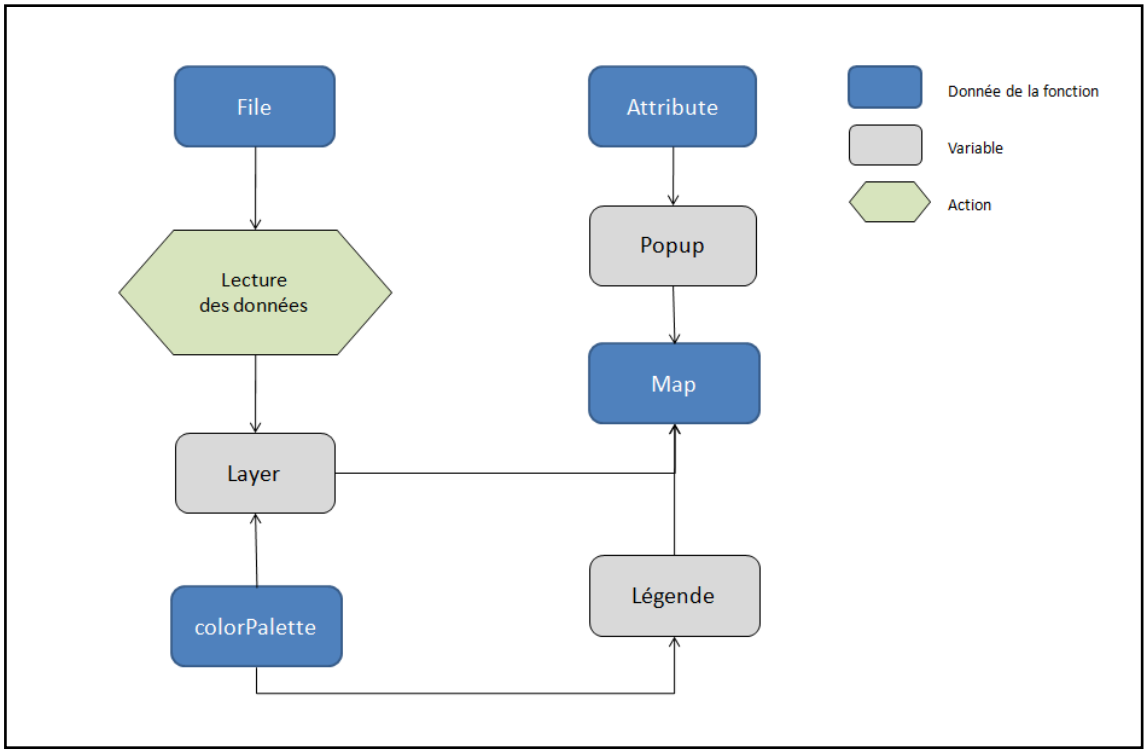


Figure 15 : Diagramme de la fonction « continu »

V. Applications

Une application est présentée pour chaque combinaison mode d'implantation/niveau d'échelle de valeurs. Chacune exposera un exemple de formulation minimale que requiert la fonction ainsi qu'un exemple de formulation maximale contenant tous les arguments personnalisables par l'utilisateur. A noter également que dans les applications suivantes, l'argument *file* contient un chemin relatif vers les fichiers GeoJSON ou TopoJSON en local, celui-ci peut être remplacé par une variable contenant les données récupérées sur un serveur web.

1. Phénomènes spatialement discrets

1.1. Ponctuel nominal

Fonction : `nominal(map, file, attribute, colorPalette="mpn65", size="1")`

Donnée : Localisation des commerces par domaine dans l'arrondissement de Charleroi (SEGEFA, 2019)

Formulation minimale : *nominal (map, "data/commerce.geojson", "domaine")*

Formulation maximale : *nominal (map, "data/commerce.geojson", "domaine", "qualitative", 5)*

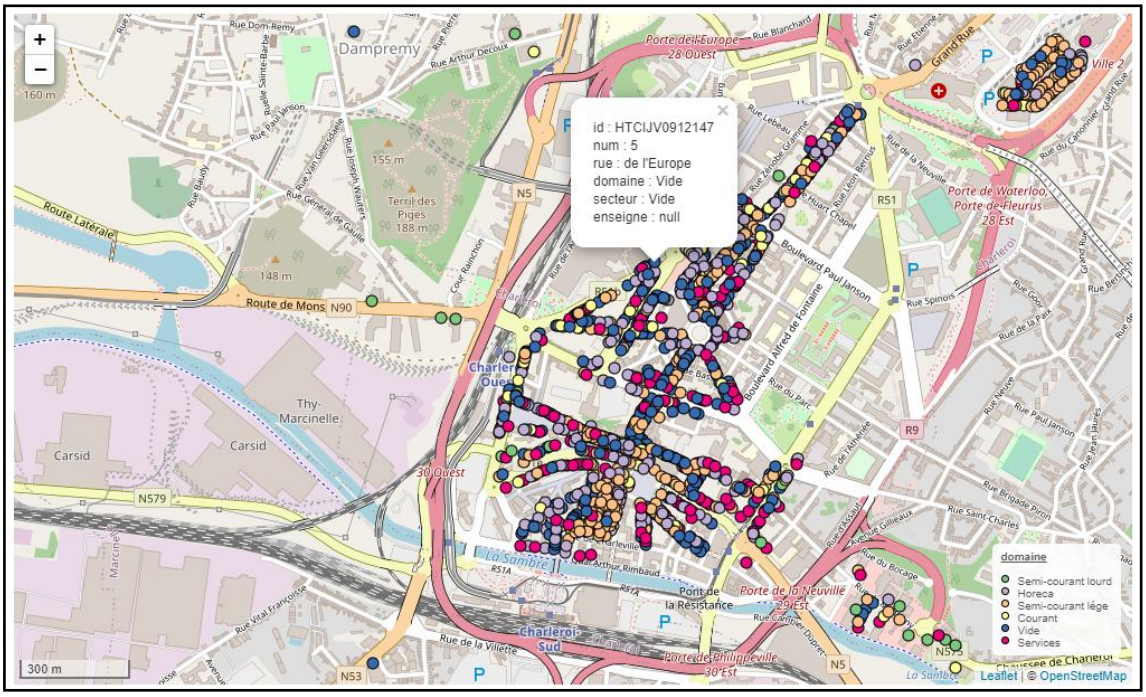


Figure 16 : Application du ponctuel nominal

1.2. Linéaire nominal

Fonction : `nominal(map, file, attribute, colorPalette="mpn65", size="1")`

Donnée : Localisation des principaux cours d'eau de Belgique (Atlas de Belgique, 2021)

Formulation minimale : *nominal (map, "data/riviere.geojson", "NAME")*

Formulation maximale : *nominal (map, "data/riviere.geojson", "NAME", "qualitative", 3)*

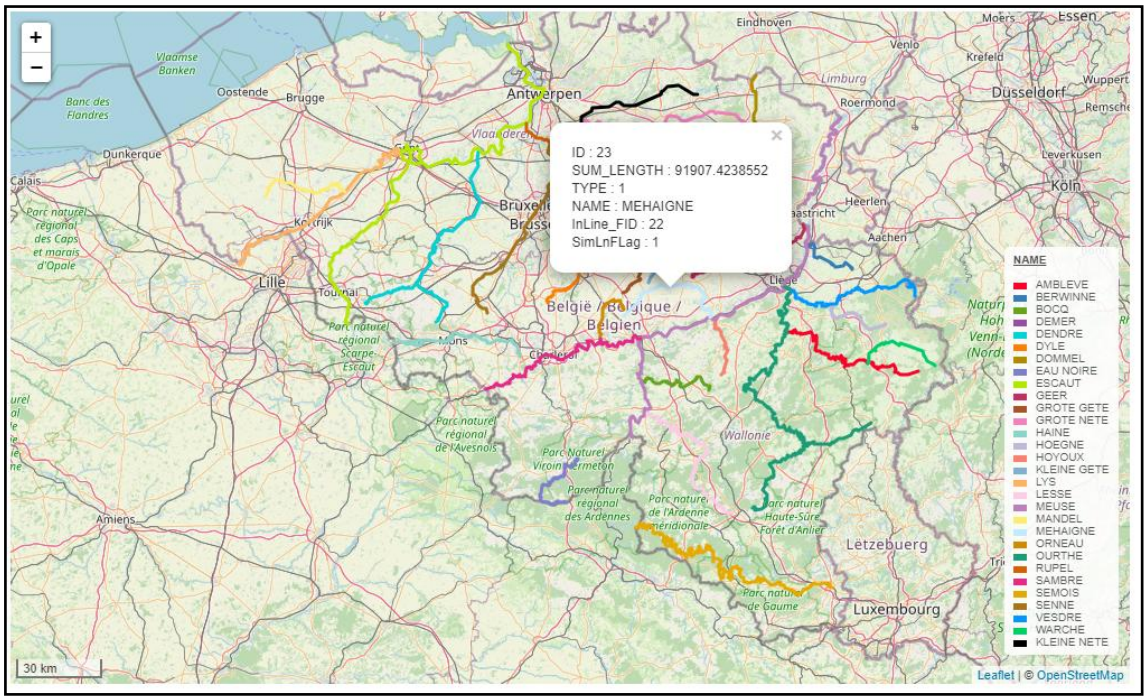


Figure 17 : Application du linéaire nominal

1.3. Zonal nominal

Fonction : `nominal(map, file, attribute, colorPalette="mpn65", size="1")`

Donnée : Localisation des provinces belges (Atlas de Belgique, 2021)

Formulation minimale : *nominal (map,"data/province.geojson","Name")*

Formulation maximale : *nominal (map,"data/province.geojson","Name","qualitative")*

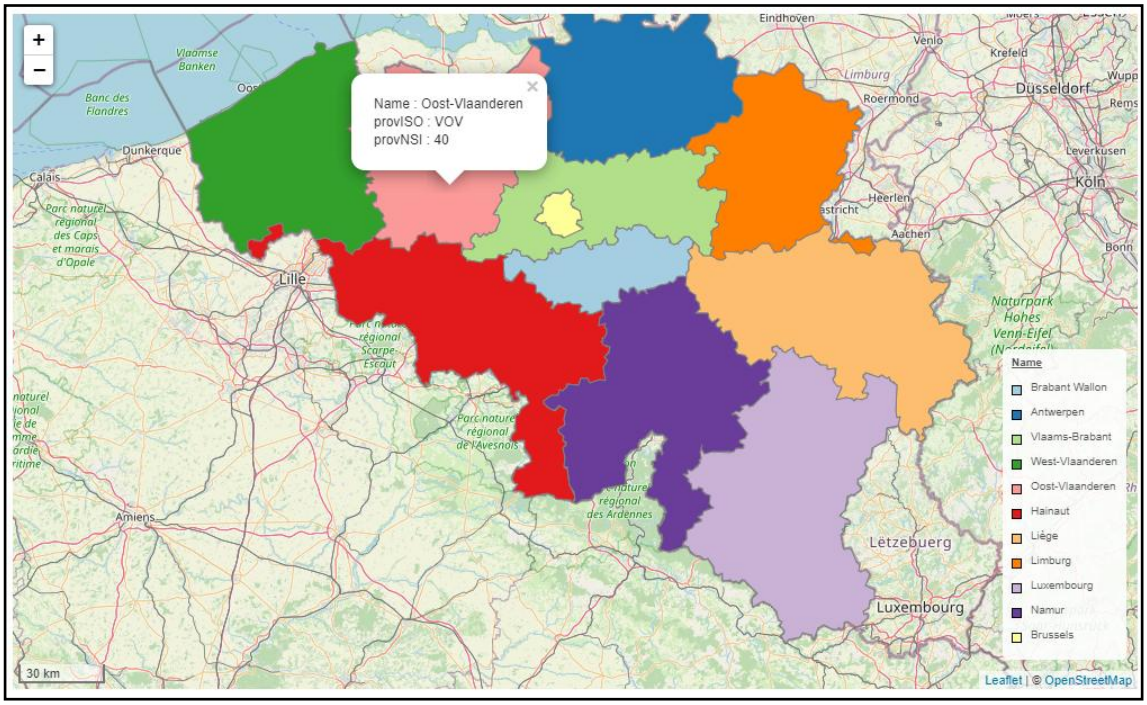


Figure 18 : Application du zonal nominal

1.4. Ponctuel ordonné

Fonction : `ordonned (map, file, attribute, order=null, color=null, first_size=1, diverging=false)`

Donnée : Modernisme (0=pas moderne, 5=très moderne) des commerces de Fléron (PLURIS, 2019)

Formulation minimale : *ordonned(map, "data/commerce_ordo.geojson", "Modernisme")*

Formulation maximale : *ordonned(map, "data/commerce_ordo.geojson", "Modernisme", [0,1,2,3,4,5], "#FF0000", 1, false)*

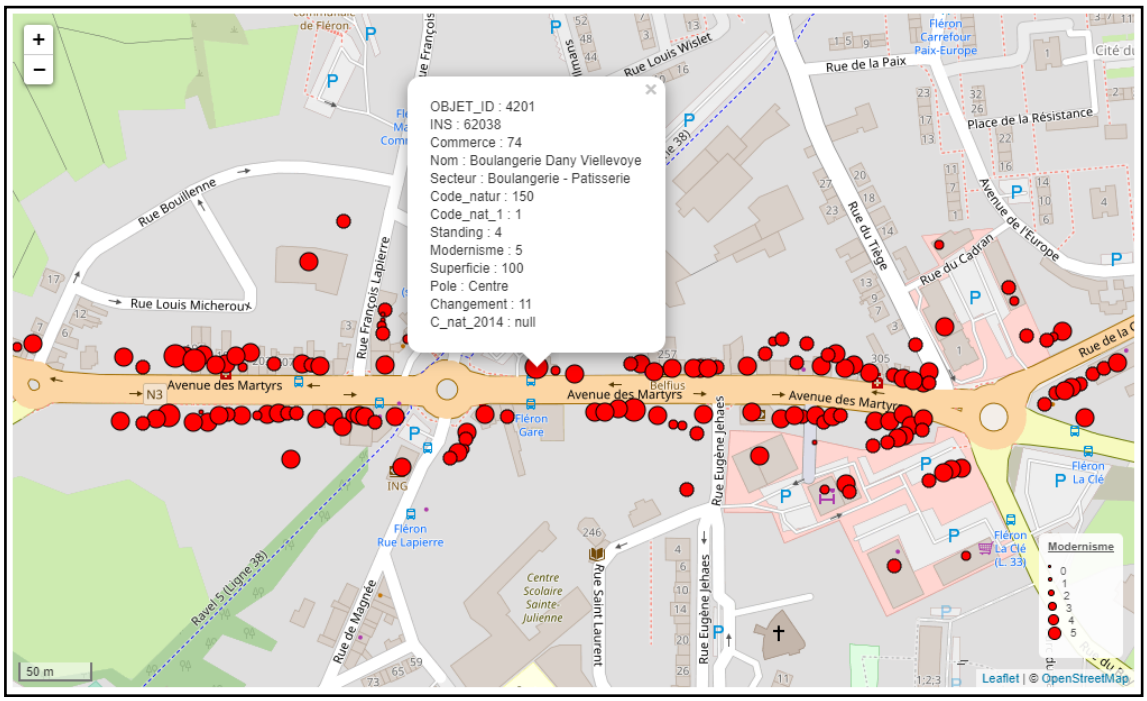


Figure 19 : Application du ponctuel ordonné

1.5. Linéaire ordonné

Fonction : `ordonned (map, file, attribute, order=null, color=null, first_size=1, diverging=false)`

Donnée : Type des principaux cours d'eau de Belgique (Atlas de Belgique, 2021)

Formulation minimale : *ordonned (map, "data/riviere.geojson", "TYPE")*

Formulation maximale : *ordonned (map, "data/riviere.geojson", "TYPE", undefined, "#FF0000", 2, false)*

Figure 20 : Application du linéaire ordonné

1.6. Zonal ordonné

Fonction : `ordonned (map, file, attribute, order=null, color=null, first_size=1, diverging=false)`

Donnée : Aptitude (0=idéal, 5=pas apte) à la culture des sols de la commune de Soumagne (PLURIS, 2019)

Formulation minimale : *ordonned(map,"data/sol_culture.geojson","Culture")*

Formulation maximale : *ordonned(map,"data/sol_culture.geojson","Culture", undefined, cb-Reds, null, false)*

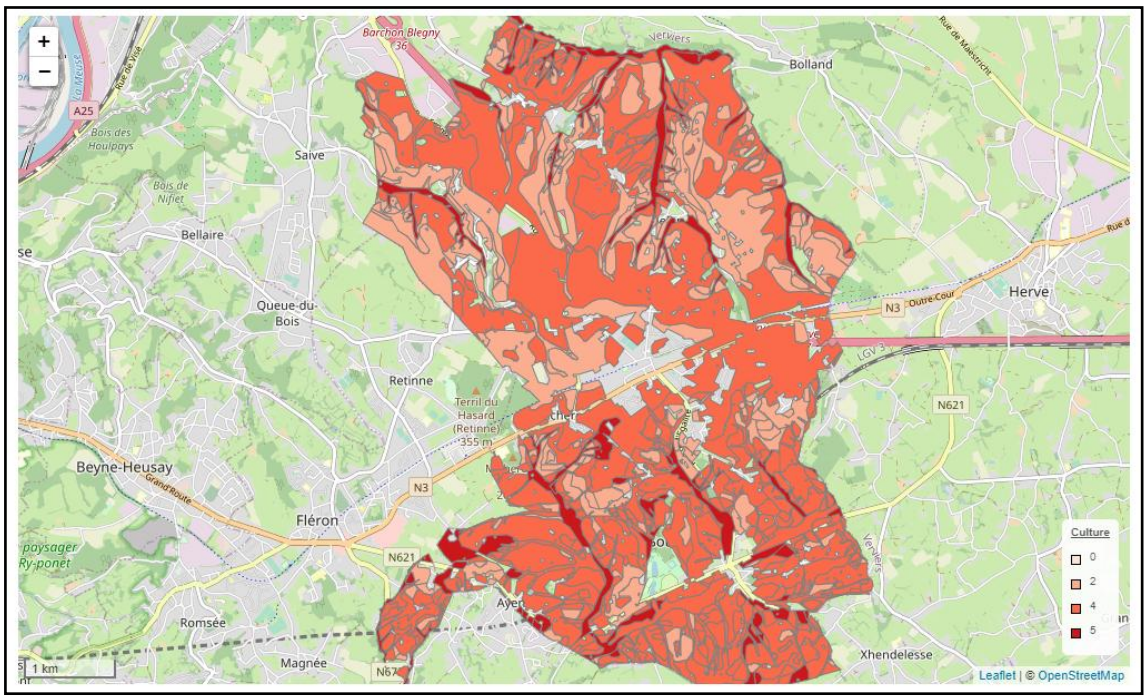


Figure 21 : Application du zonal ordonné

1.7. Ponctuel quantitatif

Fonction : `quantitatif(map, file, attribute, color=null, first_size=1, last_size=10, classification=true, classif=null, numClasses=5, polygonColor="#ffffff", diverging=false)`

Donnée : Capacité des stations Velib dans la région parisienne (Paris Data, 2021)

Formulation minimale : *quantitatif (map,"data/velib.geojson","capacity")*

Formulation maximale : *quantitatif (map,"data/velib.geojson","capacity", "#FF0000", 5, 20)*

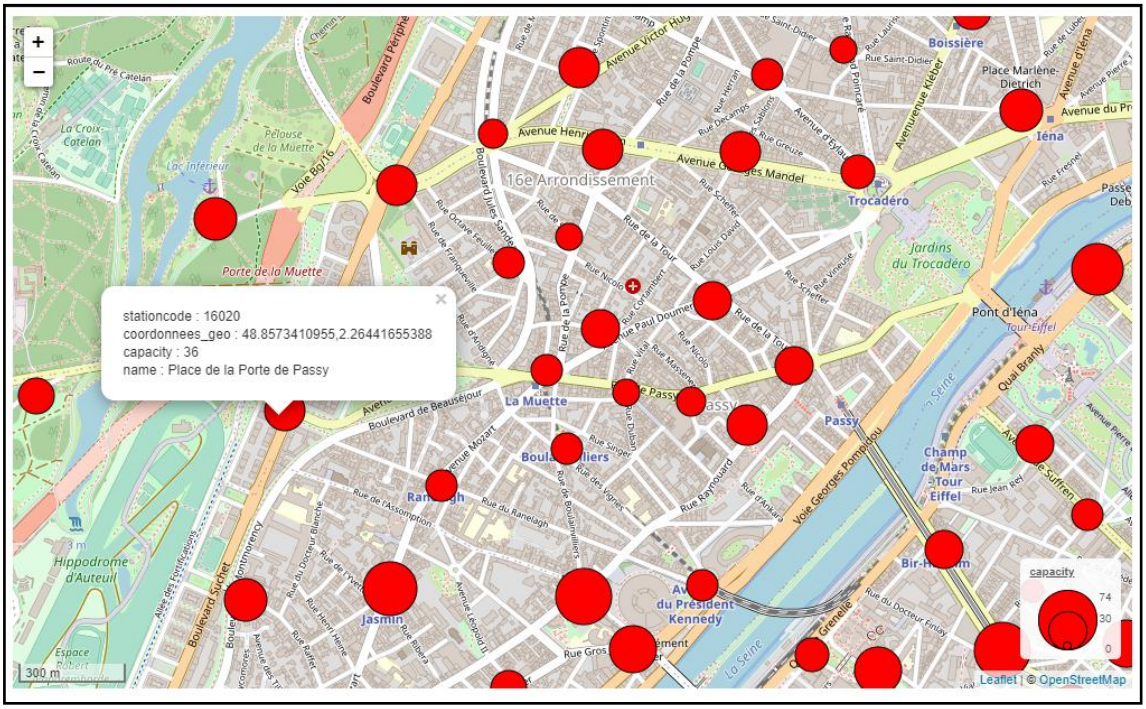


Figure 22 : Application du ponctuel quantitatif

1.8. Linéaire quantitatif

Fonction : `quantitatif(map, file, attribute, color=null, first_size=1, last_size=10, classification=true, classif=null, numClasses=5, polygonColor="#ffffff", diverging=false)`

Donnée : Longueur des principaux cours d'eau de Belgique (Atlas de Belgique, 2021)

Formulation minimale : *quantitatif (map, "data/riviere.geojson", "SUM_LENGTH")*

Formulation maximale : *quantitatif (map, "data/riviere.geojson", "SUM_LENGTH", "#FF0000", 1, 10)*

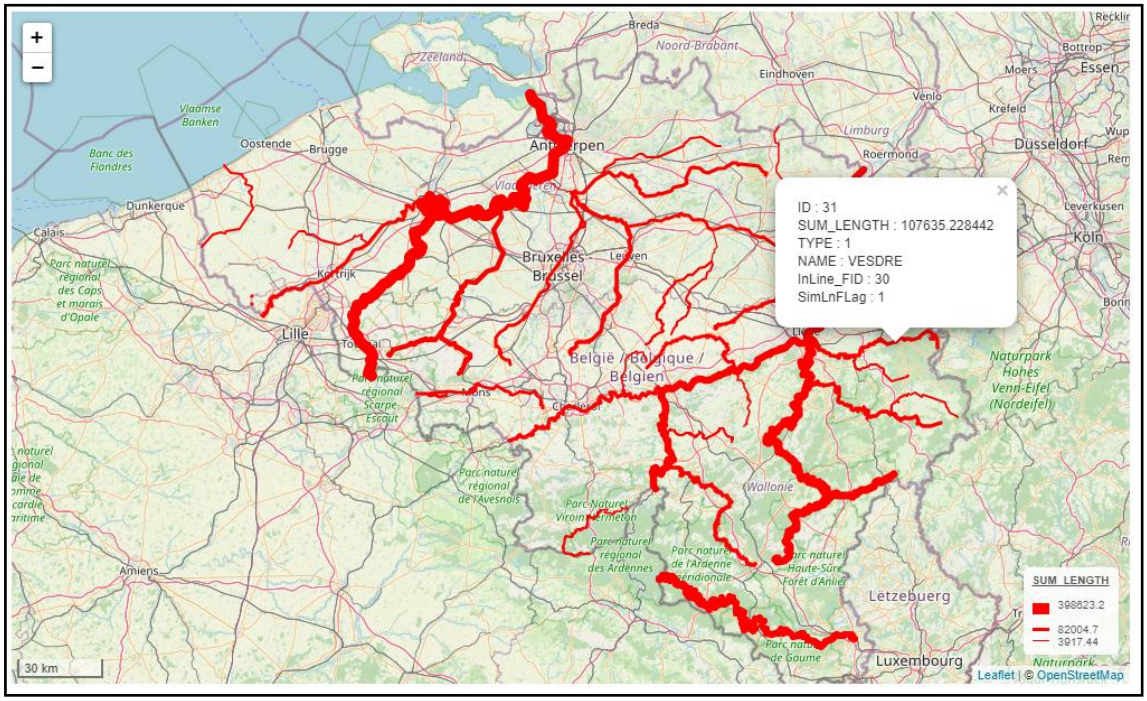


Figure 23 : Application du linéaire quantitatif

1.9. Zonal quantitatif – classification

Fonction : `quantitatif(map, file, attribute, color=null, first_size=1, last_size=10, classification=true, classif=null, numClasses=5, polygonColor="#ffffff", diverging=false)`

Donnée : Densité de population (habitation/km²) par commune belges en 2015 (Statbel, 2015)

Formulation minimale : *quantitatif(map,"data/densite.geojson","DENSITE")*

Formulation maximale : *quantitatif(map,"data/densite.geojson","DENSITE", "Reds", null, null, true,"equal-interval", 5, null, false)*

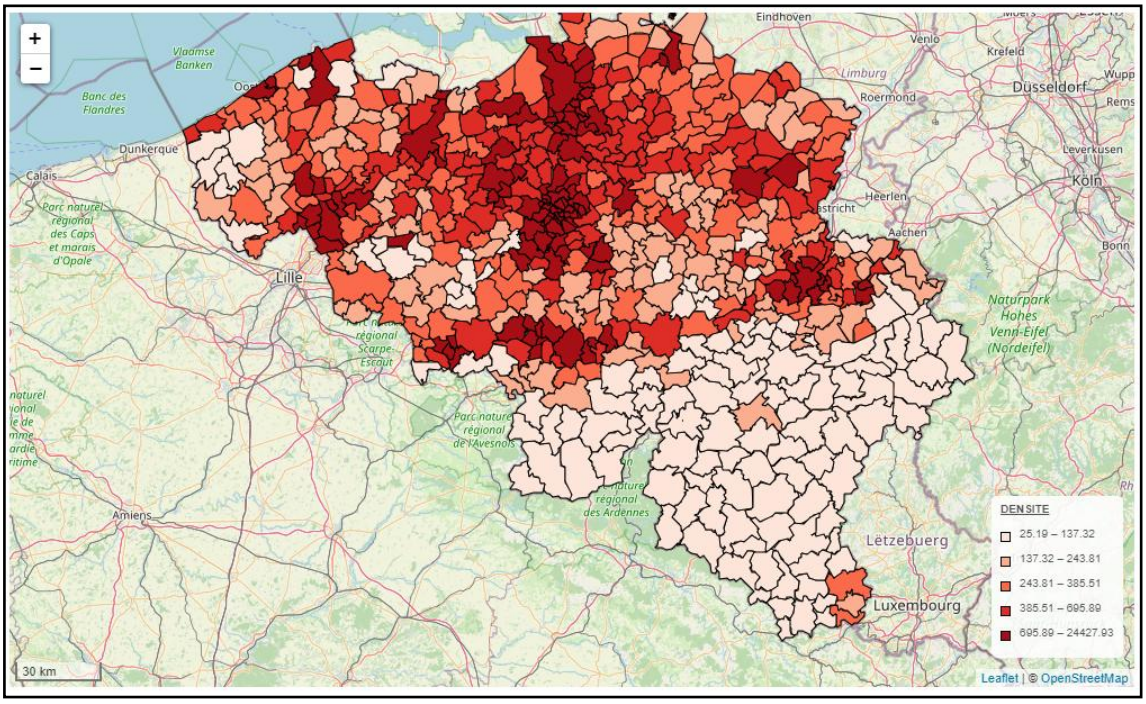


Figure 24: Application du zonal quantitatif - classification

1.10. Zonal quantitatif – série divergente

Fonction : `quantitatif(map, file, attribute, color=null, first_size=1, last_size=10, classification=true, classif=null, numClasses=5, polygonColor="#ffffff", diverging=false)`

Donnée : Evolution de la population entre 2011 et 2019 dans la commune de Liège et ses communes limitrophes par secteur statistique (PLURIS, 2019)

Formulation minimale : *quantitatif(map, "data/evolution_pop.geojson", "Evol_popul", undefined, null, null, true, undefined, undefined, null, true)*

Formulation maximale :

quantitatif(map,"data/evolution_pop.geojson","Evol_popul","RdBu", null, null, true, "quantile", 4, null ,true)

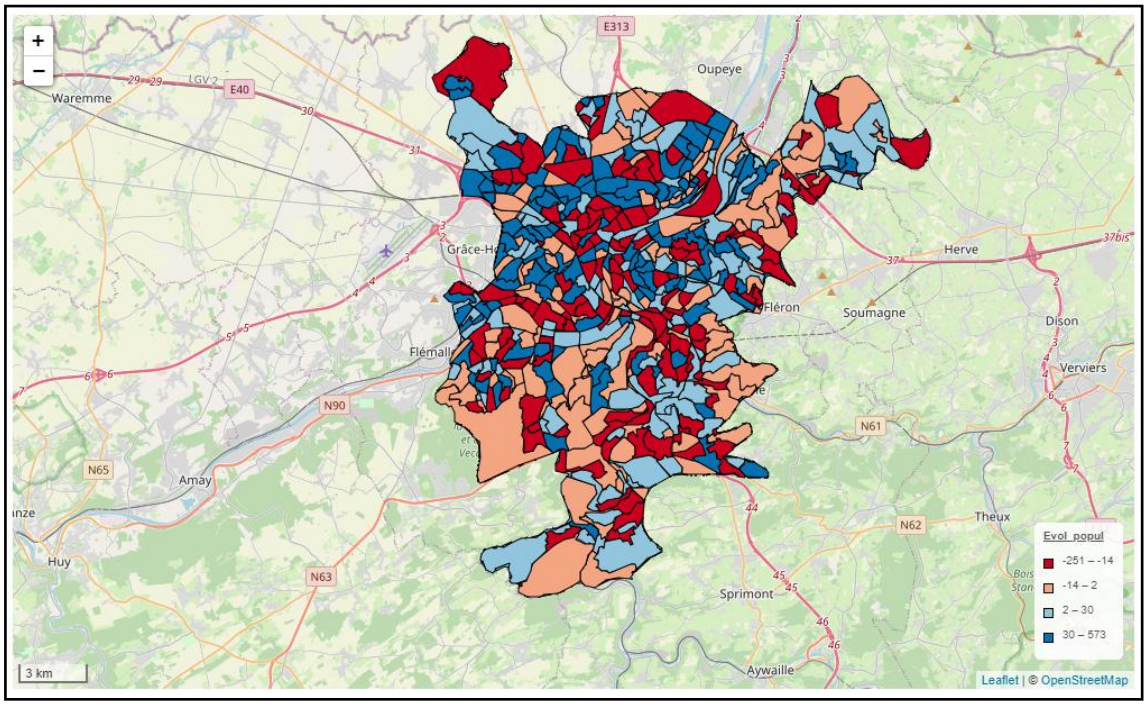


Figure 25 : Application du zonal quantitatif - série divergente

1.11. Zonal quantitatif - point centré

Fonction : `quantitatif(map, file, attribute, color=null, first_size=1, last_size=10, classification=true, classif=null, numClasses=5, polygonColor="#ffffff", diverging=false)`

Donnée : Poste de salarié par commune dans la province de Liège en 2019 (WalStat, 2019)

Formulation minimale : *quantitatif(map, "data/emploi_liege.geojson", "emploi", undefined, undefined, undefined, false)*

Formulation maximale : *quantitatif(map, "data/emploi_liege.geojson", "emploi", undefined, 5, 20, false, null, null, "#FDFFD5" ,false)*

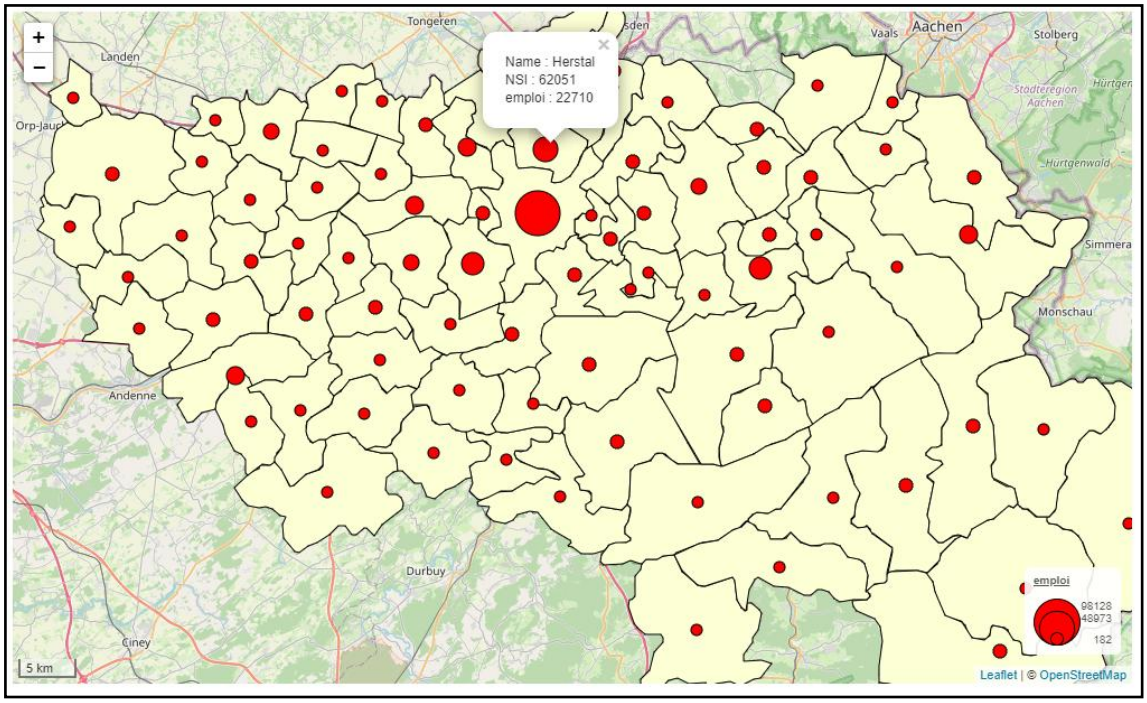


Figure 26 : Application du zonal quantitatif – point centré

2. Phénomènes spatialement continus

Fonction : `continu(map, file, attribute, colorPalette=null)`

Donnée : Température moyenne de l'air en juin 2013 en Allemagne (Riechert & Blower, 2016)

Formulation minimale : *continu (map, 'data/temperature.covjson', 'temperature')*

Formulation maximale : *continu(mymap, 'data/temperature.covjson', 'temperature', ['#FFFFFF', '#FF0000'])*

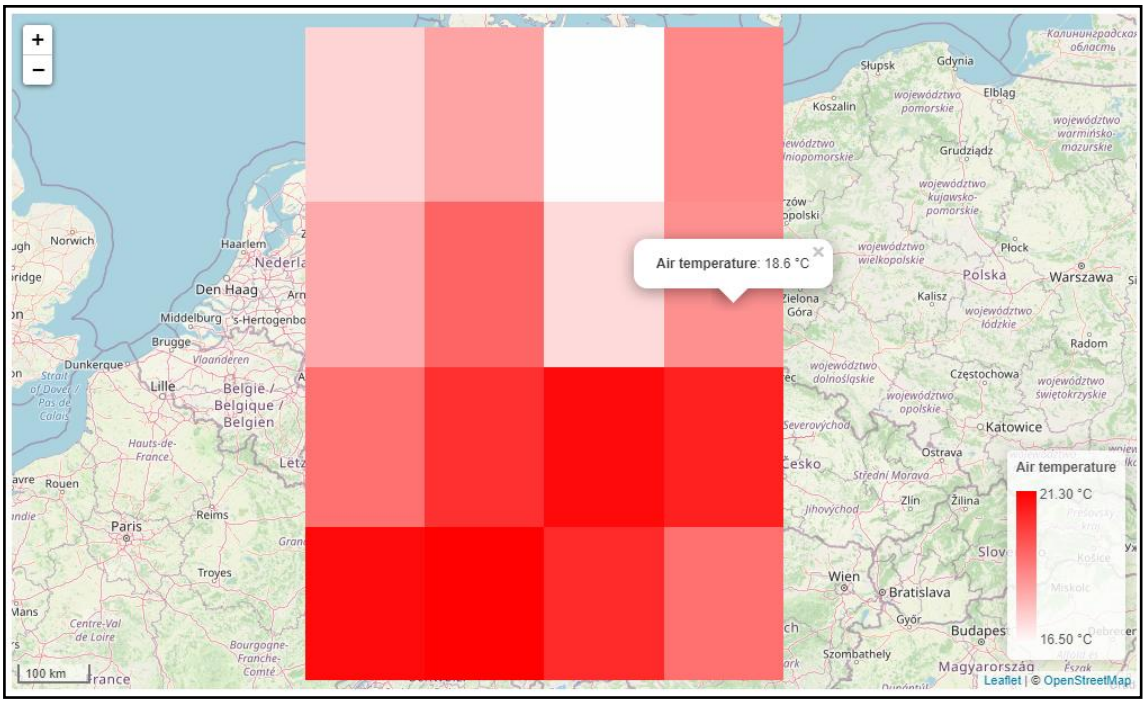


Figure 27 : Application pour le continu

VI. Conclusions

L'objectif de ce travail était de développer un outil permettant la symbolisation automatique des entités dans un SIG web avec comme question de recherche : **comment automatiser la symbolisation des phénomènes géographiques dans un SIG web côté client ?**

En effet, l'état de l'art a permis de justifier une cartographie côté client plutôt que côté serveur par ses nombreux avantages. Il a également mis en évidence les différentes possibilités de représentations en cartographie classique. Une représentation adéquate pour chaque combinaison niveau d'échelle/mode d'implantation a ensuite été choisie pour l'implémentation. La seule contrainte technique pour un utilisateur non-initié à la cartographie est de connaitre le niveau d'échelle des données qu'il souhaite symboliser. En effet, cette contrainte représente une des limitations de cet outil et une perspective intéressante serait de développer un algorithme qui détecterait automatiquement s'il s'agit de données nominales, ordonnées, quantitatives ou continues.

Un algorithme a été développé afin de déterminer la meilleure classification pour les données. Celui-ci se base sur le calcul des indices de T.A.I. et d'entropie. Il serait également intéressant que ce choix soit positionné par rapport à l'histogramme de répartition des classes. Une autre limitation de cette implémentation est l'utilisation de la librairie *Classybrew* qui ne propose que 3 classifications parmi celles présentées.

L'état de l'art a mis en évidence les différentes possibilités de formats de données et démontré une utilisation plus adéquate du format CovJSON pour des données continues. Ce dernier est un format très récent ; il deviendra plus répandu et de nouveaux outils de manipulation et de personnalisation seront probablement développés dans le futur.

L'implémentation réalisée dans ce travail permet de représenter et de personnaliser les données spatialement continues sous forme de grille. Cependant, d'autres types de données peuvent être contenus dans le format CovJSON comme par exemple les séries

temporelles, les profils verticaux, etc. Une évolution de l'outil permettrait de représenter et de pouvoir personnaliser également ces types de données au format CovJSON.

L'importance de l'interaction et des animations temporelles a été démontrée dans l'état de l'art mais elles n'ont pas été intégrées dans l'implémentation. La possibilité de manipuler une composante temporelle serait une perspective intéressante dans la continuité de ce travail.

Cet outil permet de visualiser des séries divergentes. Cependant, il serait utile de pouvoir également définir une valeur pivot autour de laquelle les valeurs s'articuleraient.

L'intérêt majeur de cette librairie est qu'elle réduit considérablement le nombre de lignes de code pour la représentation des données. En effet, une seule ligne est nécessaire contre plusieurs dizaines pour une symbolisation « manuelle ». Il s'agit donc d'un gain temporel et financier assez conséquent pour les développeurs.

Il n'existe aucune librairie connue à ce jour qui permette à la fois de symboliser automatiquement les données en détectant leur géométrie mais également d'afficher des popups et une légende correspondant au niveau d'échelle. En effet, la création de légendes pour les variables avec cercles proportionnels (présentée à la Figure 13) est particulièrement longue et complexe pour que les symboles se superposent parfaitement. Par conséquent, une création automatique permet de diminuer considérablement la complexité et la longueur de l'implémentation avec un résultat « cartographiquement correct » pour ce type de variables.

L'outil développé offre des moyens de symbolisation pour chaque type de données géographiques en un temps record dans une interface web. À l'image des SIG-logiciels tels que QGIS ou ArcMap, une personnalisation complète est également mise à disposition.

VII. Bibliographie

Agafonkin, V. (2020). *Leaflet*. Github. https://github.com/Leaflet/Leaflet. Consulté le 14 novembre 2020.

Alesheikh, A. A., Helali, H. & Behroz, H. A. (2002). Web GIS : technologies and its applications. *In ISPRS Commission IV : geospatial theory, processing and applications*, 9-12 juillet 2002, Ottawa, Canada. The international archives of the photogrammetry, remote sensing and spatial information sciences, XXXIV(4), 9 p.

Atlas de Belgique (2021). *Données cartographiques*. Atlas de Belgique. https://www.atlas-belgique.be/index.php/fr/ressources/donnees-cartographiques/. Consulté le 22 janvier 2021.

Autolib Velib Métropole (2021). *Vélib' - Localisation et caractéristique des stations*. Paris – Data. https://opendata.paris.fr/explore/dataset/velib-emplacement-des-stations/information/. Consulté le 22 janvier 2021.

Bertin, J., Barbut, M., Bonin, S., & Arbellot, G. (1967). *Sémiologie graphique : les diagrammes - les réseaux - les cartes.* Paris : Gauthier-Villars. 431 p.

Blower, J., Riechert, M. & Roberts, B. (2017). *Overview of the CoverageJSON format.* World wide web consortium. https://www.w3.org/TR/covjson-overview/. Consulté le 2 juin 2021.

Bostock, M. (2017). *TopoJSON*. Github. https://github.com/topojson/topojson. Consulté le 1 juin 2021.

Cauvin, C., Escobar, F. & Serradj, A. (2008). *Cartographie thématique 5 : des voies nouvelles à explorer.* Paris : Lavoisier, 320 p.

Cauvin, C., Reymond, H. & Serradj, A. (1987). *Discrétisation et représentation cartographique*. Montpellier : GIP Reclus, 116 p.

Donnay, J.-P. (2013). *Guide de rédaction des cartes thématiques. Méthodes et consignes.* Liège : Université de Liège. 194 p.

Donnay, J.-P. & Pantazis, D. N. (1996). L'ambiguïté des systèmes d'information géographique en géographie appliquée. *BSGLg*, 32 (1), 345-351.

Fu, P. (2018). Web GIS introduction. *In Getting to know web GIS*, 3e éd. Redlands : Esri Press, 1-39.

Géoconfluences (2017). *Sémiologie graphique*. Géoconfluences. http://geoconfluences.ens-lyon.fr/glossaire/semiologie-graphique. Consulté le 28 septembre 2020.

Georget, S. & Curl, D. (2020). *Geostats*. Github. https://github.com/simogeo/geostats. Consulté le 22 avril 2021.

Google INC (2014). *Palette.js*. Github. https://github.com/google/palette.js/. Consulté le 22 janvier 2021.

Harrower, M. & Brewer, C. (2003). ColorBrewer.org: an online tool for selecting colour schemes for maps.*The cartographic journal*, 40(1). 27-37. https://doi.org/10.1179/000870403235002042.

Harrower, M. & Fabrikant, S. I. (2008). The role of map animation in geographic visualization. *In* Dodge , M., McDerby, M. & Turner, M., *Geographic visualization: concepts, tools and applications*. Chichester : John Wiley & Sons Ltd, 49-65.

Herlocker, M. (2016). *Turf*. Github. https://github.com/Turfjs/turf. Consulté le 27 mars 2021.

IETF (2016). *The GeoJSON format*. Internet Engineering Task Force. https://datatracker.ietf.org/doc/html/rfc7946. Consulté le 1 juin 2021.

JDN (2019). *Interface de programmation : API ou Application Programming Interface*. Journal du net. https://www.journaldunet.fr/web-tech/dictionnaire-du-webmastering/1203559-api-application-programming-interface-definition-traduction/. Consulté le 21 mai 2021.

Jenks, G., & Caspall, F. (1971). Error on choroplethic maps: definition, measurement, reduction. *Annals of The Association of American Geographers*, 61(2), 217-244.

Jensen, J. (1998). Interactivity: tracking a new concept in media and communication studies. *Nordicom Review*, 12, 185-204.

Kambalyal, C. (2010). *3-Tier Architecture*. Sushil Consultants Inc. https://channukambalyal.tripod.com/NTierArchitecture.pdf. Consulté le 4 octobre 2020.

Khitrin, M. O. (2017). Comparison of JavaScript Libraries for Web-Cartography. *Bulletin of the South Ural State University - Computer technologies, automatic Control & tadioelectronics*, 17, 147-152. https://doi.org/10.14529/ctcr170317.

Kraak, M.-J. (2004). The role of the map in a Web-GIS environment. *Journal of geographical systems*, 6, 83-93. https://doi.org/10.1007/s10109-004-0127-2.

Kulawiak, M., Dawidowicz, A. & Pacholczyk, M. (2019). Analysis of server-side and client-side Web-GIS data processing methods on the example of JTS and JSTS using open data from OSM and geoportal. *Computers & Geosciences*, 129, 26-37. https://doi.org/10.1016/j.cageo.2019.04.011.

Luqun, Li., Jian, Li. & Yu, Tian. (2002). The study on web gis architecture based on jnlp. *In ISPRS Commission IV : geospatial theory, processing and applications*, 9-12 juillet 2002, Ottawa, Canada. The international archives of the photogrammetry, remote sensing and spatial information sciences, XXXIV(4), 6 p.

OGC (2021). *Geography Markup Language*. Open Geospatial Consortium. https://www.ogc.org/standards/gml. Consulté le 1 juin 2021.

OpenJS foundation (2020). *Jquery*. Github. https://github.com/jquery/jquery. Consulté le 14 novembre 2020.

OQLF (2002). *Fiche terminologique : servlet Java*. Office québécois de la langue française. http://gdt.oqlf.gouv.qc.ca/ficheOqlf.aspx?Id_Fiche=8386532. Consulté le 21 mai 2021.

OQLF (2009). *Fiche terminologique : applet*. Office québécois de la langue française. http://gdt.oqlf.gouv.qc.ca/ficheOqlf.aspx?Id_Fiche=8375292. Consulté le 21 mai 2021.

Quinn, S. (2018). Web SIG. *In* Wilson, J. P. (ed), *The geographic information science & technology body of knowledge*. https://doi.org/10.22224/gistbok/2018.1.11.

Riechert, M. (2016a). *Covjson-reader*. Github. https://github.com/Reading-eScience-Centre/covjson-reader. Consulté le 1 avril 2021.

Riechert, M. (2016b). *Covutils*. Github. https://github.com/Reading-eScience-Centre/covutils. Consulté le 1 avril 2021.

Riechert, M. (2016c). *Leaflet-coverage*. Github. https://github.com/Reading-eScience-Centre/leaflet-coverage. Consulté le 1 avril 2021.

Riechert, M. & Blower, J. (2016). *The CoverageJSON format specification*. CoverageJSON. https://covjson.org/spec/. Consulté le 28 mars 2021.

Semmo, A., Trapp, M., Jobst, M. & Döllner, J. (2015). Cartography-oriented design of 3D geospatial information visualization - Overview and techniques. *The cartographic journal*, 52(2), 95-106, https://doi.org/10.1080/00087041.2015.1119462.

Statbel (2015). *Structure de la population*. Statbel. https://statbel.fgov.be/fr/themes/population/structure-de-la-population#figures. Consulté le 21 novembre 2016.

Šimec, A. & Magličić, M. (2014). Comparison of JSON and XML data formats. *In Central european conference on information and intelligent systems*, 17-19 septembre 2014, Varaždin, Croatie. 4 p.

Tanner, J. (2015). *Classybrew*. Github. https://github.com/tannerjt/classybrew. Consulté le 22 janvier 2021.

Techno-science.net (2021). *WebGL - Définition et Explications*. Techno-science.net. https://www.techno-science.net/glossaire-definition/WebGL.html. Consulté le 2 juin 2021.

Vasiliev, I. (1997). Mapping time. *Cartographica: the international journal for geographic information and geovisualization*, 34(2), 1-51. https://doi.org/10.3138/D357-234G-2M62-4373.

W3C (2015). *Meta formats.* World wide web consortium. https://www.w3.org/standards/webarch/metaformats. Consulté le 22 mai 2021.

W3schools (2021a). *Introduction to XML.* https://www.w3schools.com/xml/xml_whatis.asp. World Wide Web schools. Consulté le 22 mai 2021.

W3schools (2021b). *JSON - Introduction.* https://www.w3schools.com/js/js_json_intro.asp. World Wide Web schools. Consulté le 22 mai 2021.

W3schools (2021c). *JSON vs XML.* https://www.w3schools.com/js/js_json_xml.asp. World wide web schools. Consulté le 22 mai 2021.

Walstat (2019). *Postes de travail salarié dans l'économie sociale selon le lieu du siège principal*. IWEPS. https://walstat.iweps.be/walstat-catalogue.php?niveau_agre=C&theme_id=18&indicateur_id=841002&sel_niveau_catalogue=T&ordre=0. Consulté le 3 juin 2021.

Zunino, A., Velázquez, G. & Celemín, J., Mateos, C., Hirsch, M.& Rodriguez Alvarez, J. (2020). Evaluating the performance of three popular web mapping libraries: a case study using Argentina's life quality index. *ISPRS International Journal of Geo-Information*, 9(10), 20 p. https://doi.org/10.3390/ijgi9100563.

VIII. Annexes

Annexe 1 : Liste des librairies utilisées 63

Annexe 2 : Ponctuel nominal 64

Annexe 3 : Linéaire nominal 65

Annexe 4 : Polygonal nominal 66

Annexe 5: Ponctuel ordonné 67

Annexe 6: Linéaire ordonné 70

Annexe 7 : Polygonal ordonné 73

Annexe 8: Nominal quantitatif 75

Annexe 9: Linéaire quantitatif 77

Annexe 10: Polygonal quantitatif – classification 78

Annexe 11: Polygonal quantitatif – point centré 80

Annexe 12: Fonction best_classification 82

Annexe 13: Continu 83

Annexe 1 : Liste des librairies utilisées

- Leaflet.js (Agafonkin, 2020)
- Jquery.min.js (OpenJS foundation, 2020)
- Palette.js (Google INC, 2014)
- Classybrew.min.js (Tanner, 2015)
- Turf.min.js (Herlocker, 2016)
- Covjson-reader.src.js (Riechert, 2016a)
- Covutils-lite.src.js (Riechert, 2016b)
- Leaflet-coverage.src.js (Riechert, 2016c)
- Geostats.min.js (Georget & Curl, 2020)

Annexe 2 : Ponctuel nominal

```
function nominal(map,file,attribute,colorPalette="mpn65", size="1")
{
$.getJSON(file,function(data)
{
      //Point or MultiPoint geometry
    if (data.features[0].geometry["type"]=="Point"
    ||data.features[0].geometry["type"]=="MultiPoint")
      {
                var values =[];
                for (var i = 0; i < data.features.length; i++){
                if (data.features[i].properties[attribute] == null)
                        continue;
                if (!values.includes(data.features[i].properties[attribute]))
          {values.push(data.features[i].properties[attribute]);}}

          var paletteMap = new Map();
          var pal = palette(colorPalette,values.length);
          if(!pal){alert("Too many values for this palette");}
                for(var i = 0; i < pal.length; i++) {
                        paletteMap.set(values[i],pal[i]);}

                L.geoJson(data,{
                pointToLayer: function (feature,latlng){
                var marker = L.circleMarker (latlng,{radius :size,
                fillColor:'#'+ paletteMap.get(feature.properties[attribute]),
                color:"#000000", weight: 1,fillOpacity:1});

                var prop = Object.keys(feature.properties);
                var popup = "<p>";
                for(var i=0; i< prop.length;i++)
                        {popup += prop[i] + " : " + feature.properties[prop[i]]
                        + (prop[i+1] ? "<br>" : "</p>")};
                marker.bindPopup(popup) ;
                return marker}
                }).addTo(map);

                var legend = L.control({position: 'bottomright'});
                legend.onAdd = function (map) {
                var div = L.DomUtil.create('div', 'point');
                div.innerHTML = '<b><u>'+ attribute  +'</b></u></br></br>';
                for (var i = 0; i < pal.length; i++) {
                        div.innerHTML +=
                        <i style="background: #'+ pal[i] +'; width :'+ size
                        +'px; height :'+ size+'px"></i>' +values[i];
                        for(var j=0; j<(size/5);j++){
                                div.innerHTML +='<br>';} }
                return div;};
                legend.addTo(map);
      }
```

Annexe 3 : Linéaire nominal

```
1 //LineString or MultiLineString geometry
 2     if (data.features[0].geometry["type"]=="LineString" ||
       data.features[0].geometry["type"]=="MultiLineString")
 3       {
 4                 var values =[];
 5                 for (var i = 0; i < data.features.length; i++){
 6                 if (data.features[i].properties[attribute] == null) continue;
 7                 if  (!values.includes(data.features[i].properties[attribute]))
 8                           {values.push(data.features[i].properties[attribute]);}}
 9
10           var paletteMap = new Map();
11           var pal = palette(colorPalette,values.length);
12           if(!pal){alert("Too many values for this palette");}
13                 for(var i = 0; i < pal.length; i++) {
14                           paletteMap.set(values[i],pal[i]);}
15
16                 L.geoJson(data,{
17
18                 onEachFeature : function(feature, layer){
19                                     var prop = Object.keys(feature.properties);
20                                     var popup = "<p>";
21                                     for(var i=0; i< prop.length;i++)
22                                           {popup += prop[i] + " : " +
                                           feature.properties[prop[i]]
23                                           + (prop[i+1] ? "<br>" : "</p>")};
24                                     layer.bindPopup(popup);},
25
26                 style: function(feature) {
27                 return { color:"#"+paletteMap.get(feature.properties[attribute]),
28                           weight : size, opacity :1};}
30                 }).addTo(map);
31
32                 var legend = L.control({position: 'bottomright'});
33                 legend.onAdd = function (map) {
34                 var div = L.DomUtil.create('div', 'line');
35                 div.innerHTML = '<b><u>'+ attribute  +'</b></u></br></br>';
36
37                 for (var i = 0; i < pal.length; i++) {
38                           div.innerHTML +=
39                           '<i style="background: #'+ pal[i] +'; width: '
40                           +size*4+'px; height: '+size*2+'px"></i>' +values[i];
41                 for(var j=0; j<(size/4);j++){
42                                               div.innerHTML +='<br>';} }
43                 return div;};
44                 legend.addTo(map);
45       }
```

Annexe 4 : Polygonal nominal

```
//Polygon or MultiPolygon geometry
    if(data.features[0].geometry["type"]=="Polygon" ||
    data.features[0].geometry["type"]=="MultiPolygon")
      {
                var values =[];
                for (var i = 0; i < data.features.length; i++){
                if (data.features[i].properties[attribute] == null) continue;
                if (!values.includes(data.features[i].properties[attribute]))
                        {values.push(data.features[i].properties[attribute]);}}

          var paletteMap = new Map();
          var pal = palette(colorPalette,values.length);
          if(!pal){alert("Too many values for this palette");}
                for(var i = 0; i < pal.length; i++) {
                        paletteMap.set(values[i],pal[i]);}

                L.geoJson(data,{
                        onEachFeature : function(feature, layer){
                        var prop = Object.keys(feature.properties);
                        var popup = "<p>";
                        for(var i=0; i< prop.length;i++)
                        {popup += prop[i] + " : " + feature.properties[prop[i]]
                        + (prop[i+1] ? "<br>" : "</p>")};
                        layer.bindPopup(popup) ;},

                        style: function(feature) {
                        return {color:"grey", weight : 1, fillColor : "#"
                        +paletteMap.get(feature.properties[attribute]),
                        fillOpacity : 1 };}
                }).addTo(map);

                var legend = L.control({position: 'bottomright'});
                legend.onAdd = function (map) {
                var div = L.DomUtil.create('div', 'polygon');
                div.innerHTML = '<b><u>'+ attribute  +'</b></u></br></br>';

                for (var i = 0; i < pal.length; i++) {
                        div.innerHTML +=
                                '<i style="background: #'+ pal[i] +'"></i>'
                                +values[i]+ '<br><br>';}
                return div;};
                legend.addTo(map);
      }
```

Annexe 5: Ponctuel ordonné

```
function ordonned (map,file,attribute,order=null, color=null, first_size=1, diverging=false)
{
$.getJSON(file,function(data)
{
      //Point or MultiPoint geometry
    if (data.features[0].geometry["type"]=="Point"
    ||data.features[0].geometry["type"]=="MultiPoint")
      {
                if (color == null){
                        if (diverging==true)
                                color="cb-RdBu";
                        else
                                color="#FF0000";}

                var values =[];
                for (var i = 0; i < data.features.length; i++){
                if (data.features[i].properties[attribute] == null) continue;
                if (!values.includes(data.features[i].properties[attribute]))
                {values.push(data.features[i].properties[attribute]);}}

      if (order != null){
                if (order.length==values.length){
                        for (var i=0; i<order.length;i++)
                                if (!values.includes(order[i])){
                                alert(order[i] + " is not in the data.");
                                        return 0;}
                        values=order;}
                else if (order.length < values.length){
                   alert("Categories are missing, there is "+values.length+"
                   values in data.");
                        return 0;}
                else
                   {alert("Too much categories, there is "+values.length+"
                   values in data.");
                        return 0;}}
      else
                for (var i=0; i<values.length;i++){
                        if(typeof(values[i]) != "number"){
                                alert("You have to define your order.");
                                return 0;}
                        else {
                                const byValue = (a,b) => a - b;
                                values.sort(byValue);}}
```

```
    // Diverging symbolisation
    if (diverging==true){
        var paletteMap = new Map();
        var pal = palette(color,values.length);
        if(!pal){alert("Too many values for this palette");}
             for(var i = 0; i < pal.length; i++) {
                      paletteMap.set(values[i],pal[i]);}

             L.geoJson(data,{
             pointToLayer: function (feature,latlng){
             var marker = L.circleMarker (latlng,{radius :first_size,
             fillColor: '#'+paletteMap.get(feature.properties[attribute]),
                      color:"#000000", weight: 1,fillOpacity:1});

             var prop = Object.keys(feature.properties);
             var popup = "<p>";
             for(var i=0; i< prop.length;i++)
             {popup += prop[i] + " : " + feature.properties[prop[i]]
                      + (prop[i+1] ? "<br>" : "</p>")};
             marker.bindPopup(popup) ;
             return marker}
             }).addTo(map);

             var legend = L.control({position: 'bottomright'});
             legend.onAdd = function (map) {
             var div = L.DomUtil.create('div', 'point');
             div.innerHTML = '<b><u>'+ attribute  +'</b></u></br></br>';
             for (var i = 0; i < pal.length; i++) {
                      div.innerHTML +=
                       '<i style="background: #'+ pal[i] +'; width :'+
                       first_size
                      +'px; height :'+ first_size+'px"></i>' +values[i];
                      for(var j=0; j<(first_size/5);j++){
                               div.innerHTML +='<br>';} }
             return div;};
             legend.addTo(map);}
```

```
78                  // Sequential symbolisation
 79                 else{
 80                 var sizeMap = new Map();
 81                 for(var i = 0; i < values.length; i++) {
 82
        sizeMap.set(values[i],Math.sqrt(Math.pow(first_size,2)*Math.pow(2,i)));}
 83
 84                 L.geoJson(data,{
 85                 pointToLayer: function (feature,latlng){
 86                 var marker = L.circleMarker (latlng,{radius
                    :sizeMap.get(feature.properties[attribute]),
 87                 fillColor:color,color:"#000000", weight: 1,fillOpacity:1});
 88                 var prop = Object.keys(feature.properties);
 89                 var popup = "<p>";
 90                 for(var i=0; i< prop.length;i++)
 91                    {popup += prop[i] + " : " + feature.properties[prop[i]]+
                    (prop[i+1] ? "<br>" : "</p>")};
 92                 marker.bindPopup(popup) ;
 93                 return marker}
 94                 }).addTo(map);
 95
 96                 var legend = L.control({position: 'bottomright'});
 97                 legend.onAdd = function (map) {
 98                 var div = L.DomUtil.create('div', 'point');
 99                 div.innerHTML = '<b><u>'+ attribute  +'</b></u></br></br>';
100                 for (var i = 0; i< values.length; i++) {
101                          var radius = sizeMap.get(values[i]);
102                          div.innerHTML +=
103                           ('<i style="background :'+ color +'; width: '+
                           radius+'px; height: '
104                                   + radius + 'px"></i>' + values[i]);
105                          for(var j=0; j<((radius+1)/10);j++){
106                                   div.innerHTML +='<br>';}
107                          ;}
108                 return div;};
109                 legend.addTo(map);}
110       }
```

Annexe 6: Linéaire ordonné

```
1 //LineString or MultiLineString geometry
 2     if (data.features[0].geometry["type"]=="LineString" ||
       data.features[0].geometry["type"]=="MultiLineString")
 3       {
 4                 if (color == null){
 5                         if (diverging==true)
 6                                 color="cb-RdBu";
 7                         else
 8                                 color="#FF0000";}
 9
10                 var values =[];
11                 for (var i = 0; i < data.features.length; i++){
12                 if (data.features[i].properties[attribute] == null) continue;
13                 if (!values.includes(data.features[i].properties[attribute]))
14                 {values.push(data.features[i].properties[attribute]);}}
15
16           if (order != null){
17                 if (order.length==values.length){
18                         for (var i=0; i<order.length;i++)
19                                 if (!values.includes(order[i])){
20                                 alert(order[i] + " is not in the data.");
21                                         return 0;}
22                         values=order;}
23                 else if (order.length < values.length){
24                         alert("Categories are missing, there is
                           "+values.length+" values in data.");
25                         return 0;}
26                 else
27                         {alert("Too much categories, there is
                           "+values.length+" values in data.");
28                         return 0;}}
29       else
30                 for (var i=0; i<values.length;i++){
31                         if(typeof(values[i]) != "number"){
32                                 alert("You have to define your order.");
33                                 return 0;}
34                         else {
35                                 const byValue = (a,b) => a - b;
36                                 values.sort(byValue);}}
```

```
38          // Diverging symbolisation
 39          if (diverging==true)
 40          {
 41          var paletteMap = new Map();
 42          var pal = palette(color,values.length);
 43          if(!pal){alert("Too many values for this palette");}
 44               for(var i = 0; i < pal.length; i++) {
 45                         paletteMap.set(values[i],pal[i]);}
 46
 47               L.geoJson(data,{
 48
 49               onEachFeature : function(feature, layer){
 50                                   var prop = Object.keys(feature.properties);
 51                                   var popup = "<p>";
 52                                   for(var i=0; i< prop.length;i++)
 53                                             {popup += prop[i] + " : " +
                                              feature.properties[prop[i]]
 54                                             + (prop[i+1] ? "<br>" : "</p>")};
 55                                   layer.bindPopup(popup);},
 56
 57               style: function(feature) {
 58                         return { color : "#" +
                           paletteMap.get(feature.properties[attribute]),
 59                         weight : first_size, opacity :1};}
 60
 61               }).addTo(map);
 62
 63               var legend = L.control({position: 'bottomright'});
 64               legend.onAdd = function (map) {
 65               var div = L.DomUtil.create('div', 'line');
 66               div.innerHTML = '<b><u>'+ attribute  +'</b></u></br></br>';
 67
 68               for (var i = 0; i < pal.length; i++) {
 69                         div.innerHTML +=
 70                         '<i style="background: #'+ pal[i] +'; width: '
 71                          +first_size*4+'px; height: '+first_size*2+'px"></i>'
                           +values[i];
 72               for(var j=0; j<(first_size/4);j++){
 73                                             div.innerHTML +='<br>';} }
 74               return div;};
 75               legend.addTo(map);
 76          }
 77
```

```
78          // Sequential symbolisation
 79          else{
 80          var sizeMap = new Map();
 81               for(var i = 0; i < values.length; i++) {
 82                         sizeMap.set(values[i],first_size*Math.pow(2,i));}
 83
 84               L.geoJson(data,{
 85               onEachFeature : function(feature, layer){
 86                                   var prop = Object.keys(feature.properties);
 87                                   var popup = "<p>";
 88                                   for(var i=0; i< prop.length;i++)
 89                                   {popup += prop[i] + " : " +
                                     feature.properties[prop[i]]
 90                                   +(prop[i+1] ? "<br>" : "</p>")};
 91                                   layer.bindPopup(popup) ;},
 92
 93               style: function(feature) {
 94                         return { color : color, weight :
                           sizeMap.get(feature.properties[attribute]),
 95                         opacity :1};}
 96               }).addTo(map);
 97
 98               var legend = L.control({position: 'bottomright'});
 99               legend.onAdd = function (map) {
100               var div = L.DomUtil.create('div', 'line');
101               div.innerHTML = '<b><u>'+ attribute  +'</b></u></br></br>';
102               for (var i = 0; i< values.length; i++) {
103                         var size = sizeMap.get(values[i]);
104                         div.innerHTML +=
105                                   '<i style="background: '+  color+'; width:
                                    '+first_size*20+'px; height: '
106                                    + size + 'px"></i>' +values[i];
107                         for(var j=0; j<((size+1)/10);j++){
108                         div.innerHTML +='<br>';}}
109               return div;};
110               legend.addTo(map);}
111     }
```

Annexe 7 : Polygonal ordonné

```
1 //Polygon or MultiPolygon geometry
2     if(data.features[0].geometry["type"]=="Polygon" ||
      data.features[0].geometry["type"]=="MultiPolygon")
3       {
4                 if (color == null){
5                         if (diverging==true)
6                                 color="cb-RdBu";
7                         else
8                                 color="cb-Reds";}
9
10                var values =[];
11                for (var i = 0; i < data.features.length; i++){
12                if (data.features[i].properties[attribute] == null) continue;
13                if (!values.includes(data.features[i].properties[attribute]))
14                {values.push(data.features[i].properties[attribute]);}}
15
16            if (order != null){
17                if (order.length==values.length){
18                        for (var i=0; i<order.length;i++)
19                                if (!values.includes(order[i])){
20                                alert(order[i] + " is not in the data.");
21                                        return 0;}
22                        values=order;}
23                else if (order.length < values.length){
24                        alert("Categories are missing, there is
                          "+values.length+" values in data.");
25                        return 0;}
26                else
27                        {alert("Too much categories, there is
                          "+values.length+" values in data.");
28                        return 0;}}
29      else
30                for (var i=0; i<values.length;i++){
31                        if(typeof(values[i]) != "number"){
32                                alert("You have to define your order.");
33                                return 0;}
34                        else {
35                                const byValue = (a,b) => a - b;
36                                values.sort(byValue);}}
```

```
38          var paletteMap = new Map();
39          var pal = palette(color,values.length);
40
41          if(!pal){alert("Too many values for this palette");}
42               for(var i = 0; i < pal.length; i++) {
43                         paletteMap.set(values[i],pal[i]);}
44
45               L.geoJson(data,{
46                         onEachFeature : function(feature, layer){
47                         var prop = Object.keys(feature.properties);
48                         var popup = "<p>";
49                         for(var i=0; i< prop.length;i++)
50                          {popup += prop[i] + " : " +
                            feature.properties[prop[i]]+ (prop[i+1] ? "<br>" :
                            "</p>")};
51                         layer.bindPopup(popup) ;},
52
53                         style: function(feature) {
54                         return {color:"grey", weight : 1, fillColor : "#"
                           +paletteMap.get(feature.properties[attribute]),
55                         fillOpacity : 1 };}
56               }).addTo(map);
57
58               var legend = L.control({position: 'bottomright'});
59               legend.onAdd = function (map) {
60               var div = L.DomUtil.create('div', 'polygon');
61               div.innerHTML = '<b><u>'+ attribute  +'</b></u></br></br>';
62               for (var i = 0; i < pal.length; i++) {
63                         div.innerHTML +=
64                          '<i style="background: #'+ pal[i] +'"></i>'
                            +values[i]+ '<br><br>';}
65               return div;};
66               legend.addTo(map);
67     }
```

Annexe 8: Nominal quantitatif

```
function quantitatif(map,file,attribute,color=null,first_size=1, last_size=10,
classification=true, classif=null, numClasses=5, polygonColor="#ffffff",
diverging=false)
{
$.getJSON(file,function(data)
{
      //Point or MultiPoint geometry
    if (data.features[0].geometry["type"]=="Point"
    ||data.features[0].geometry["type"]=="MultiPoint")
      {
                if (color==null)
                         color="#FF0000";

                var values =[];
                for (var i = 0; i < data.features.length; i++){
                if (data.features[i].properties[attribute] == null) continue;
                   if(typeof(data.features[i].properties[attribute]) !=
                   "number") continue;
                values.push(data.features[i].properties[attribute]);}

          values.sort(function(a, b){return a - b});
          var min = values[0];
          var max = values[values.length-1];
          var middle = values[Math.round((values.length-1)/2)];

                var sizeMap = new Map();
                for(var i = 0; i < values.length; i++) {

        sizeMap.set(values[i],Math.sqrt((((Math.pow(last_size,2)-
Math.pow(first_size,2))/(max-min))*(values[i]-min))+Math.pow(first_size,2)));}


                L.geoJson(data,{
                pointToLayer: function (feature,latlng){
                var marker = L.circleMarker (latlng,{radius
                :sizeMap.get(feature.properties[attribute]),
                fillColor:color,color:"#000000", weight: 1,fillOpacity:1});
                var prop = Object.keys(feature.properties);
                var popup = "<p>";
                for(var i=0; i< prop.length;i++)
                   {popup += prop[i] + " : " + feature.properties[prop[i]]+
                   (prop[i+1] ? "<br>" : "</p>")};
                marker.bindPopup(popup) ;
                return marker}
                }).addTo(map);
```

```
39                var legend = L.control({position: 'bottomright'});
40                legend.onAdd = function (map) {
41                         var div = L.DomUtil.create('div', 'point');
42                div.innerHTML = '<b><u>'+ attribute  +'</b></u></br></br>';
43                         var classes = [max,middle,min];
44                         var symbolsContainer = L.DomUtil.create('div',
                          'symbolsContainer');
45                         var sum = sizeMap.get(classes[0]) +
                          sizeMap.get(classes[1]);
46
47                   $(symbolsContainer).attr('style','margin-left
                     :'+(sizeMap.get(max)*2)+'px;margin-right :'+
                     (sizeMap.get(max)+10)+'px');
48                   for (var i = 0; i < 3;i++) {
49                   var legendCircle = L.DomUtil.create('div', 'legendCircle');
50                   var radius = Math.sqrt((((Math.pow(last_size,2)-
                     Math.pow(first_size,2))/(max-min))*(classes[i]-
                     min))+Math.pow(first_size,2));
51                         if (i==0) {var margin=-radius*2;}
52                         else{var margin = -radius-last_radius-2;}
53                                   last_radius = radius;
54                   $(legendCircle).attr('style', 'width: ' + radius*2 + 'px;
                     height: ' + radius*2 + 'px;margin-left:'
55                   + margin+'px; background :'+color);
56                   $(legendCircle).append("<span class='legendValue'>"
                     +(Math.round(classes[i]*100)/100)+"</span>");
57                   $(symbolsContainer).append(legendCircle);}
58                   $(div).append(symbolsContainer);
59                                   return div;};
60                         legend.addTo(map);
61       }
```

Annexe 9: Linéaire quantitatif

```
1 //LineString or MultiLineString geometry
 2     if (data.features[0].geometry["type"]=="LineString" ||
       data.features[0].geometry["type"]=="MultiLineString")
 3       {
 4                 if (color==null)
 5                           color="#FF0000";
 6
 7                 var values =[];
 8                 for (var i = 0; i < data.features.length; i++){
 9                 if (data.features[i].properties[attribute] == null) continue;
10                    if(typeof(data.features[i].properties[attribute]) !=
                      "number") continue;
11                 values.push(data.features[i].properties[attribute]);}
12
14           values.sort(function(a, b){return a - b});
15           var min = values[0];
16           var max = values[values.length-1];
17           var middle = values[Math.round((values.length-1)/2)];
18
19                 var sizeMap = new Map();
20                 for(var i = 0; i < values.length; i++) {
21                    sizeMap.set(values[i],(((last_size-first_size)/(max-
                      min))*(values[i]-min))+first_size);}
23
24                 L.geoJson(data,{
25                 onEachFeature : function(feature, layer){
26                           var prop = Object.keys(feature.properties);
27                           var popup = "<p>";
28                           for(var i=0; i< prop.length;i++)
29                                    {popup += prop[i] + " : " +
                                    feature.properties[prop[i]]+ (prop[i+1] ?
                                    "<br>" : "</p>")};
30                                     layer.bindPopup(popup) ;},
31
32                 style: function(feature) {
33                    return { color : color, weight :
                      sizeMap.get(feature.properties[attribute]), opacity :1};}
34                 }).addTo(map);
35
36                 var legend = L.control({position: 'bottomright'});
37                 var classes=[max,middle,min];
38                 legend.onAdd = function (map) {
39                 var div = L.DomUtil.create('div', 'line');
40                 div.innerHTML = '<b><u>'+ attribute  +'</b></u></br></br>';
```

```
41                for (var i = 0; i< 3; i++) {
42                        var size = first_size*5*((i+2)/2);
43                        div.innerHTML +=
44                         '<i style="background: '+  color+'; width:  15px;
                          height: '+ sizeMap.get(classes[i])
45                        + 'px"></i>' + (Math.round(classes[i]*100)/100);
46                        for(var j=0; j<(sizeMap.get(classes[i])/5);j++){
47                        div.innerHTML +='<br>';} }
48                return div;};
49                legend.addTo(map);
50        }
```

Annexe 10: Polygonal quantitatif – classification

```
1 //Polygon or MultiPolygon geometry
 2     if(data.features[0].geometry["type"]=="Polygon" ||
       data.features[0].geometry["type"]=="MultiPolygon")
 3       {
 4                //Symbolisation with classification
 5                if (classification==true)
 6                {
 7                        if (color==null){
 8                                if(diverging==true)
 9                                        color="RdBu";
10                                else
11                                color="Reds";}
12
13                        var values =[];
14                        for (var i = 0; i < data.features.length; i++){
15                        parseInt(data.features[i].properties[attribute]);
16                        if (data.features[i].properties[attribute] == null)
                          continue;
17                        if(typeof(data.features[i].properties[attribute]) !=
                          "number") continue;
18                        values.push(data.features[i].properties[attribute]);}
19
21                    var brew = new classyBrew();
22                    brew.setSeries(values);
23                          brew.setNumClasses(numClasses);
24                          brew.setColorCode(color);
25
26                          if (classif==null){
27                                  var max_TAI=0,
28                                          min_entropy=1;
29                                  var type =['jenks', 'equal-interval',
                                          'quantile'];
```

```
30                                  for (var i=0;i<type.length;i++){
31                                          brew.classify(type[i]);
32                                          var indice =
                                            best_classification(brew.getSeries(),
                                            brew.getBreaks(),numClasses);
33                                          var TAI=indice[0],
34                                                  entropy=indice[1];
35
36                                        if (TAI>max_TAI && entropy<min_entropy)
37                                          {
38                                                  max_TAI=TAI;
39                                                  min_entropy=entropy;
40                                                  classif=type[i];
42                                          }}
43                                  }
44                          brew.classify(classif);
45
46                          L.geoJson(data,{
47                          onEachFeature : function(feature, layer){
48                                  var prop = Object.keys(feature.properties);
49                                  var popup = "<p>";
50                                  for(var i=0; i< prop.length;i++)
51                                  {popup += prop[i] + " : " +
                                    feature.properties[prop[i]]+ (prop[i+1] ?
                                    "<br>" : "</p>")};
52                                  layer.bindPopup(popup) ;},
53
54                          style: function(feature) {
55                                  return {  fillColor:
                                    brew.getColorInRange(feature.properties[attribut
                                    e]),weight: 1, color: 'black',fillOpacity: 1};}
56                          }).addTo(map);
57
58                          var legend = L.control({position: 'bottomright'});
59                          legend.onAdd = function (map) {
60                          var div = L.DomUtil.create('div', 'polygon');
61                           div.innerHTML = '<b><u>'+ attribute
                             +'</b></u></br></br>';
62                          classes = brew.getBreaks();
63                          for (var i = 0; i < classes.length; i++) {
64                                   div.innerHTML +=
65                                   classes[i+1] ?'<i style="background:' +
                                  brew.getColorInRange(classes[i+1]) + '"></i> ' +
66                                  (Math.round(classes[i]*100)/100)+   ' &ndash; '
                                  + (Math.round(classes[i+1]*100)/100) +
                                  '</br></br>' : "" ;}
67                          return div;}
68                          legend.addTo(map);
69                  }
```

Annexe 11: Polygonal quantitatif – point centré

```
1 //Symbolisation with point in the center of gravity
 2                else
 3                {
 4                         if (color==null)
 5                                  color="#FF0000";
 6
 7                         var values =[];
 8                         for (var i = 0; i < data.features.length; i++){
 9                                 if (data.features[i].properties[attribute] ==
                                   null) continue;
10
        if(typeof(data.features[i].properties[attribute]) != "number") continue;
11                     values.push(data.features[i].properties[attribute]);}
12
13                      values.sort(function(a, b){return a - b});
14                      var min = values[0];
15                      var max = values[values.length-1];
16                      var middle = (max-min)/2;
17
18                     var sizeMap = new Map();
19                    for(var i = 0; i < values.length; i++) {
20
         sizeMap.set(values[i],Math.sqrt((((Math.pow(last_size,2)-
Math.pow(first_size,2))/(max-min))*(values[i]-min))+Math.pow(first_size,2)));}
21
22                L.geoJson(data,{
23                    style: function(feature) {
24                         return {color:"black", weight : 1, fillColor :
                           polygonColor, fillOpacity : 1 };}
25                    }).addTo(map);
26
27                L.geoJson(data,{
28                    onEachFeature: function(feature,layer){
29                        var centroid =turf.centroid(feature);
30                        var long = centroid.geometry.coordinates[0];
31                        var lat = centroid.geometry.coordinates[1];
32                        L.circleMarker([lat,long],{radius
                          :sizeMap.get(feature.properties[attribute]),fillColor:co
                          lor,color:"#000000", weight: 1, fillOpacity:1
                          }).addTo(map);
34                                  var prop = Object.keys(feature.properties);
35                                  var popup = "<p>";
36                                  for(var i=0; i< prop.length;i++)
37                                  {popup += prop[i] + " : " +
                                   feature.properties[prop[i]]+ (prop[i+1] ?
                                   "<br>" : "</p>")};
38                                  layer.bindPopup(popup);},
```

```
40          style: function(feature) {return {opacity : 0, fillOpacity : 0 };}
41                  }).addTo(map);
42
43              var legend = L.control({position: 'bottomright'});
44              legend.onAdd = function (map) {
45                      var div = L.DomUtil.create('div', 'point');
46                      div.innerHTML = '<b><u>'+ attribute
                        +'</b></u></br></br>';
47                      var classes = [max,middle,min];
48                      var symbolsContainer = L.DomUtil.create('div',
                                'symbolsContainer');
49                      var sum=sizeMap.get(classes[0])
                                +sizeMap.get(classes[1]);
50                      var last_radius=0;
51
52                      $(symbolsContainer).attr('style','margin-left
                                :'+(sizeMap.get(max)*2)+'px;margin-right :'+
                                (sizeMap.get(max)+10)+'px');
53                      for (var i = 0; i < 3;i++) {
54                      var legendCircle = L.DomUtil.create('div',
                                'legendCircle');
55                      var radius = Math.sqrt((((Math.pow(last_size,2) -
                                 Math.pow(first_size,2))/(max -
                                 min))*(classes[i]-min))
                                 +Math.pow(first_size,2));
56                      if (i==0) {var margin=-radius*2;}
57                      else{var margin = -radius-last_radius-2;}
58                      last_radius = radius;
59                      $(legendCircle).attr('style', 'width: ' + radius*2 +
                                'px; height: ' + radius*2 + 'px;margin-
                                left:'
60                              + margin +'px; background :'+color);
61                      $(legendCircle).append("<span
                                class='legendValue'>"+(Math.round(classes[i]*10
                                0)/100)+"</span>");
62                      $(symbolsContainer).append(legendCircle);}
63                      $(div).append(symbolsContainer);
64                      return div;};
65                      legend.addTo(map);
66              }
```

Annexe 12: Fonction best_classification

```
function best_classification(series,breaks,numClasses)
{
	var H_max = Math.log(numClasses)/Math.log(2);
	var classes = new geostats(series);
	var sum_j=0, sum=0,H=0;
	var pos=0;
	for(var j=1;j<breaks.length;j++){
		var values=[];
		var k=0;
		while(series[pos]<=breaks[j])
		{
			values[k]=series[pos];
			pos++;
			k++;
		}
		var classes_j = new geostats(values);
		var sum_i=0;
		for(var i=0; i<values.length;i++)
		{
			sum_i += Math.abs(values[i]-classes_j.mean());
		}
		sum_j += sum_i;
		var pj = classes_j.pop()/classes.pop();
		H += pj * (Math.log(pj)/Math.log(2));
	}
	for(var i=0; i<series.length;i++)
	{
		sum += Math.abs(series[i]-classes.mean());
	}
	var TAI = 1 - (sum_j/sum);
	var entropy = 1 - (-H/H_max);
	return [TAI, entropy];

}
```

Annexe 13: Continu

```
function continu(map, file, attribute, colorPalette=null)
{
        if (colorPalette != null)
                var palette = C.linearPalette(colorPalette);

        var layer;
        CovJSON.read(file).then(function (coverage) {
        layer = new C.Grid(coverage, { parameter: attribute, palette : palette})
          .on('afterAdd', function () {
          C.legend(layer).addTo(map);}).
          addTo(map);
        });

        map.on('click', function (e) {
          new C.DraggableValuePopup({
            layers: [layer] }).
          setLatLng(e.latlng).openOn(map);
        });
}
```