



Université de Toulouse

MASTER 2 GEOMATIQUE

« Sciences Géomatiques en environneMent et Aménagement » (SIGMA)

<http://sigma.univ-toulouse.fr>

MEMOIRE DE RECHERCHE

Automatisation d'une méthode d'interpolation pour la modélisation de l'îlot de chaleur urbain sous QGIS

Gardes Thomas

Laboratoire LISST-CIEU



Maître de stage : Julia Hidalgo

Tuteur-enseignant : Sebastien Le Corre

Septembre 2017

Résumé

L'îlot de chaleur urbain (ICU) est un phénomène caractérisé par une élévation localisée des températures que l'on observe généralement dans un milieu urbain par rapport à une périphérie plus rurale.

Bien que ses causes et ses conséquences soient assez bien connues, l'ICU reste difficile à modéliser sur un territoire donné. Sa représentation nécessite en effet une connaissance précise des températures à une résolution spatiale la plus fine possible.

Dans le cadre du projet de recherche ANR MApUCE (Modélisation appliquée et droit de l'urbanisme : climat urbain et énergie), le laboratoire LISST-CIEU de l'université Toulouse 2 Jean Jaurès s'est intéressé à une méthode géostatistique de modélisation de l'ICU, basée sur la technique du « regression-kriging ».

En utilisant notamment un ensemble de variables développées durant le projet MApUCE et caractérisant le tissu urbain, il est ainsi possible d'estimer l'ICU par interpolation avec une précision acceptable.

Ce mémoire se base sur un stage de master 2 réalisé au LISST-CIEU avec pour objectifs d'affiner et d'automatiser la méthode de modélisation de l'ICU.

Une première partie décrit ainsi le fonctionnement, les atouts et les limites de la méthode d'interpolation par regression-kriging appliquée aux ICU. Le cas d'étude développé est celui de la ville de Toulouse, avec une résolution spatiale de 100m.

La seconde partie s'attache quant à elle à décrire le processus d'automatisation sous la forme d'un plug-in QGIS codé en python, avec une interface réalisée sous Qt designer.

Mots-clés : Géostatistique, îlot de chaleur urbain, interpolation, plug-in, Python, QGIS, regression-kriging

Abstract

The urban heat island (UHI) is a phenomenon characterized by a localized elevation of temperatures that is generally observed in an urban environment compared to a more rural periphery. Even if its causes and consequences are fairly well known, the UHI remains difficult to model on a given territory. Its representation requires precise knowledge of the temperatures at the finest possible spatial resolution. For the research project ANR MApUCE, the laboratory LISST-CIEU of Toulouse 2 Jean Jaurès University was interested in a geostatistical method for modeling ICU, based on regression-kriging. Using a set of variables developed during the MApUCE project and characterizing the land covering, it is possible to estimate the ICU by interpolation with an acceptable precision.

This document is based on a Master 2 internship at LISST-CIE. Its objectives was to refine and automate the methodology of UHI modeling.

A first part describes the operation, advantages and limitations of the regression-kriging interpolation method applied to the UHI. The study case developed is the city of Toulouse France, with a 100m spatial resolution .

The second part describes the automation process in the form of a QGIS plug-in coded in python, with an interface made on Qt designer.

Keywords : Geostatistics, interpolation, plug-in, python, QGIS, regression-kriging, Urban Heat Island

Remerciements

Je tiens à remercier Julia Hidalgo pour son accueil et son encadrement tout au long du stage mais aussi pour ses conseils, son soutien, sa bienveillance qui ont rendu cette expérience agréable et enrichissante.

Un grand merci également à Najla Touati pour sa disponibilité, sa patience, sa gentillesse et ses conseils précieux.

Merci aussi à Sebastien Le Corre pour son encadrement, sa disponibilité et ses idées apportées à la réalisation de ce mémoire et du stage.

Merci aux doctorants, futurs doctorants et stagiaires du LISST-CIEU : Delphine, Renaud, Guillaume, Zahra, Amira, Dung pour leur sympathie, leur soutien et leurs conseils. Je vous souhaite plein de réussite dans vos projets actuels et à venir.

D'une manière générale, merci à toute l'équipe du LISST, Boujemaa, Marie et tous ceux avec qui j'ai eu le plaisir d'échanger au cours de ce stage agréable et enrichissant.

Merci également à Adrien Napoly pour ses apports précieux sur la question des données NETATMO.

Un grand merci enfin à mes parents, ma famille, mes amis pour leur soutien indéfectible tout au long de ces années d'études qui s'achèvent (peut-être ?) avec ce mémoire.

Sommaire

Introduction	10
I. Une méthode d'interpolation géostatistique pour modéliser l'îlot de chaleur urbain. 15	
1. Modéliser les ICU : définition et état de l'art sélectif	15
2. Principes du « regression-kriging »	21
3. Application du regression-kriging à la modélisation des ICU : la méthode du LISST-CIEU.....	29
4. Analyse et discussion des résultats obtenus	42
Partie II : Automatisation de la méthode : de la chaîne de traitement au plug-in QGIS	53
1. Intérêt d'une automatisation de la méthode	53
2. Le script QGIS : une étape intermédiaire	56
3. Du script au plug-in QGIS.....	64
4. Discussion : constats et limites	81
Conclusion	87
Bibliographie	89
Table des matières.....	92
Table des figures	94
Index des acronymes et abréviations	96

Préambule

A l'heure où les préoccupations énergétiques, climatiques et écologiques prennent une place de plus en plus importante dans les politiques publiques menées à différentes échelles – de l'international au local – le rôle de la ville en tant que modèle d'organisation spatiale est souvent interrogé, tantôt en tant que facteur amplificateur du réchauffement climatique, tantôt comme foyer d'expérimentations énergétiques, écologiques et/ou sociales.

Les espaces urbains présentent en effet des caractéristiques uniques, notamment par leur capacité à agréger, concentrer. Concentrer les populations et les activités, mais aussi leurs supports et leurs conséquences : infrastructures, tissus urbains, émissions de gaz à effet de serre (GES), pollution atmosphérique...

Ceci fait des villes des objets d'étude climatologique à part entière, avec des phénomènes qui leur sont propres. L'un des plus célèbres est certainement celui d'îlots de chaleur urbains (ICU). Ces derniers correspondent à une élévation localisée des températures, pouvant amener plusieurs degrés de différences entre, par exemple, le cœur d'une ville et ses espaces plus périphériques. Ces îlots de chaleurs urbains trouvent leur sources dans différents facteurs et peuvent jouer un rôle dans des questionnements d'ordre divers, aussi bien énergétiques qu'écologiques, sociaux, sanitaires ou encore urbanistiques. Leur appréciation précise reste toutefois un exercice délicat, qui nécessite souvent de disposer d'un grand nombre de relevés de températures sur le territoire concerné.

Le projet ANR MApUCE porté par le laboratoire LISST CIEU s'intéresse principalement à la production de données climatiques urbaines et à leur intégration dans les documents d'urbanisme. Un de ses axes s'intéresse ainsi aux ICU, avec notamment le développement d'une méthode de modélisation basée sur des outils géomatiques.

Le stage sur lequel repose ce mémoire s'inscrit donc dans le cadre du projet MApUCE. Son premier objectif est de reprendre et affiner la méthode de modélisation des ICU précédemment développée. Le deuxième point consiste à automatiser cette méthode sous la forme d'un script python. L'étape finale sera d'aboutir à un plug-in QGIS fonctionnel permettant un usage simple de cette méthode en vue, notamment, d'une utilisation au sein des collectivités locales.

Au cours de ce mémoire, nous reviendrons donc sur le contexte et les enjeux liés au projet (introduction). Nous nous intéresserons ensuite à la construction de la méthode de modélisation géostatistique (I) puis aux tenants et aboutissants – notamment techniques – de son automatisation (II).

Introduction

A°) Territoire du stage, territoire de l'étude

Avant d'entrer plus avant dans le cœur du sujet, il convient de poser certains éléments introductifs utiles pour le lecteur, notamment celui qui ne serait pas familier avec la région toulousaine. Dans notre cas, le territoire qui accueille le stage et celui sur lequel porte l'essentiel du travail ne font qu'un : la commune de Toulouse et sa périphérie. Située dans le Sud-Ouest de la France, dans le département de Haute-Garonne, la commune de Toulouse compte 466 297 habitants, avec une aire urbaine particulièrement étendue, regroupant 1 312 304 habitants (recensement INSEE 2014).

En termes de morphologie physique et urbaine, la commune est divisée en deux rives par la Garonne. Elle ne présente pas de relief très marqué, mais on peut noter une zone de coteaux au sud qui avoisine les 250m d'altitude (sources : MNT de l'IGN) et un relief plus vallonné sur la partie Ouest du territoire. La commune comporte un centre historique de taille relativement modeste entourée d'une périphérie globalement résidentielle. Le dynamisme démographique et économique assez important de la région toulousaine est en grande partie lié au secteur de l'aéronautique, et notamment à l'entreprise Airbus. La zone d'étude est notamment délimitée par la disposition des capteurs utilisés dans la méthode d'interpolation, sur laquelle nous reviendrons dans la partie I. La carte ci-dessous illustre néanmoins cet espace et certaines de ses caractéristiques morphologiques qui pourront jouer un rôle dans les analyses présentées ci-après.



Figure 1 : Grandes caractéristiques du territoire d'étude.
Réalisation : Thomas Gardes, 2017

B°) La structure d'accueil

Le laboratoire LISST (Laboratoire Interdisciplinaire Solidarités, Sociétés, Territoires) est une Unité Mixte de Recherche en sciences humaines et sociales située sur le campus de l'université Toulouse 2 Jean Jaurès. Sous la tutelle de l'université, du CNRS, de l'EHESS et de l'ENSFEA, le laboratoire regroupe quatre équipes de recherche : les CAS (Centre d'anthropologie sociale), le CERS (Centre d'étude des rationalités et des savoirs), Dynamiques Rurales, et le CIEU (Centre Interdisciplinaire d'études urbaines). C'est au LISST-CIEU que se trouve rattaché le stage servant de base à se mémoire. Ses thématiques de recherche tournent principalement autour des transformations des villes, en particulier sous l'angle de l'habitat, de l'économie, de la ville durable et du climat urbain. Les travaux du CIEU se portent ainsi sur Toulouse et les villes de Midi-Pyrénées, mais aussi sur d'autres villes d'Europe, d'Amérique ou d'Afrique.

L'équipe est forte de 31 chercheurs et 23 chercheurs associés dans des disciplines variées telles que la géographie, l'aménagement du territoire, l'urbanisme, l'architecture, la géomatique, la climatologie, la géopolitique... Sur l'année 2017, le LISST CIEU compte 5 principaux projets de recherche en cours, dont le projet ANR MAPUCE, cadre du travail du stage.

C°) Le projet MAPUCE

Le projet ANR MAPUCE (« Modélisation appliquée et droit de l'Urbanisme : Climat et Energie ») est un projet démarré en mars 2014. Il associe notamment le LISST-CIEU au CNRM GAME (unité de recherche mixte du Centre National de Recherche Météorologique).

Le projet part du constat posé par la loi Grenelle II, qui a fait des documents de planification urbaine un cadre permettant d'intégrer aux politiques publiques les problématiques énergétiques et climatiques. L'objet du projet est ainsi « *d'intégrer dans les politiques urbaines et les documents juridiques les plus pertinents des données quantitatives de microclimat urbain, climat et énergie.* »¹ Cet objectif se décompose en deux temps :

1°) « Définir une stratégie de modélisation, à partir de bases de données nationales et à l'échelle de la France, du microclimat urbain, de la consommation d'énergie liée aux bâtiments et du comportement énergétique des habitants et usagers.

2°) proposer une méthodologie d'intégration dans les procédures juridiques et les politiques urbaines de données quantitatives de microclimat urbain, climat et énergie.

Les données quantitatives seront principalement obtenues à partir de simulations numériques, le projet visant notamment à développer une méthode automatisée de production de paramètres et indicateurs urbains concernant la simulation énergétique à l'échelle des quartiers. La méthode sera testée sur un panel de villes françaises (notamment Toulouse, Aix-en-Provence, La Rochelle), dans l'idée d'établir un « *diagnostic énergie-climat à l'échelle de la France pour le climat actuel ou futur* ».

¹ Site officiel UMR CNRM : <http://www.umar-cnrm.fr/spip.php?article787>, consulté le 12 mars 2017

En parallèle, un travail sera effectué « sur l'ensemble des documents d'urbanisme et juridiques adéquats pour identifier les potentiels leviers d'action et les échelles applicables, quels que soient les territoires et l'ingénierie en place. »

Il s'agit ainsi d'un projet assez vaste, organisé en plusieurs volets :



Figure 2 : L'organisation générale du projet MAPUCE (source : UMR-CRNM)

D°) Les objectifs du stage

Le stage s'inscrit dans la phase de conception d'indicateurs urbains liés au climat et à l'énergie. Il fait suite à une démarche entreprise en collaboration avec Toulouse Métropole lors de l'élaboration de son PLUi-H. Il s'agissait alors de développer une méthode permettant de modéliser l'îlot de chaleur urbain. La solution retenue consistait alors en une méthode d'interpolation géostatistique se basant notamment sur les données issues du projet CAPITOU, consistant en des relevés de température à fréquence horaire sur 27 stations situés dans le centre-ville de Toulouse et ses abords immédiats. A cela s'ajoute la prise en compte du relief – via un MNT de l'IGN – et de la « fraction de ville », c'est à dire, le degré d'artificialisation des sols. Cette fraction de ville est construite à partir des données ACCLIMAT (cf partie « données utilisées »), qui offrent une résolution spatiale de 250m. La première étape du stage consiste ainsi à s'approprier la méthode d'interpolation développée et à l'appliquer avec des données produites dans le cadre du projet MAPUCE et permettant de travailler à des résolutions spatiales plus fines. L'explicitation des tenants et aboutissants méthodologies (notamment géostatistiques) de la méthode fait également partie des attendus.

Il s'agit ensuite d'automatiser cette méthode sous la forme d'un script python, puis de l'adapter en un plug-in QGIS fonctionnel. Cette étape constitue le cœur du travail de stage.

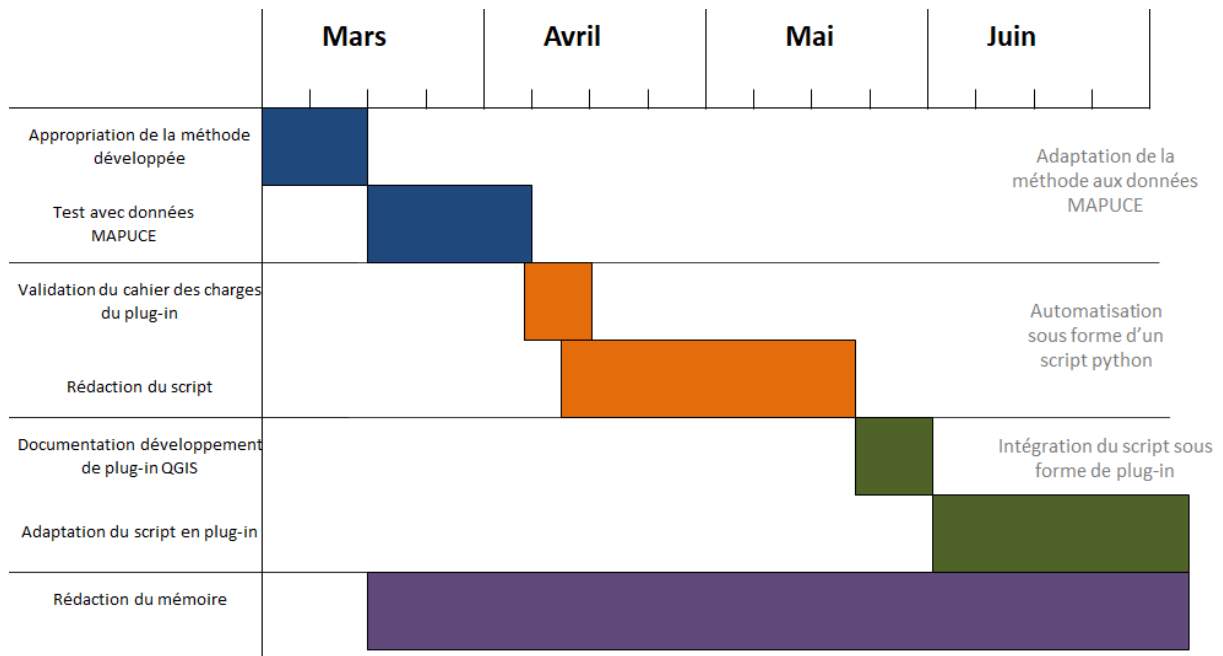


Figure 3 : Diagramme de Gantt envisagé en début de stage
 Réalisation : Thomas Gardes, 2017

**PARTIE I : Une méthode
d'interpolation géostatistique pour
modéliser l'îlot de chaleur urbain**

I. Une méthode d'interpolation géostatistique pour modéliser l'îlot de chaleur urbain

La première partie du stage (jusqu'à mi-avril environ) a été consacrée à l'appropriation, l'explicitation et l'amélioration de la méthode d'interpolation ébauchée en amont du stage et visant à modéliser l'îlot de chaleur urbain (ICU).

1. Modéliser les ICU : définition et état de l'art sélectif

a. Éléments de définition

L'îlot de chaleur urbain (ICU) ne fait pas vraiment l'objet d'une définition officielle que l'on pourrait trouver, par exemple, dans un dictionnaire. Il s'agit néanmoins d'un phénomène relativement bien connu. Un rapport de la Communauté Urbaine du Grand Lyon² le définit comme un « *secteur urbanisé où les températures de l'air et des surfaces sont supérieures à celles de la périphérie rurale* » Une revue de presse sur le web permet également de voir le phénomène décrit comme un « *microclimat artificiel provoqué par les activités humaines (centrales énergétiques, échangeurs de chaleur...) et l'urbanisme (surfaces sombres qui absorbent la chaleur, comme le goudron)*. »³ ou encore un « *effet de dôme thermique :[...] plus on s'approche du centre de la ville, plus il est dense et haut, et plus le thermomètre grimpe.* »⁴

Le phénomène d'ICU peut ainsi être représenté par un graphique :

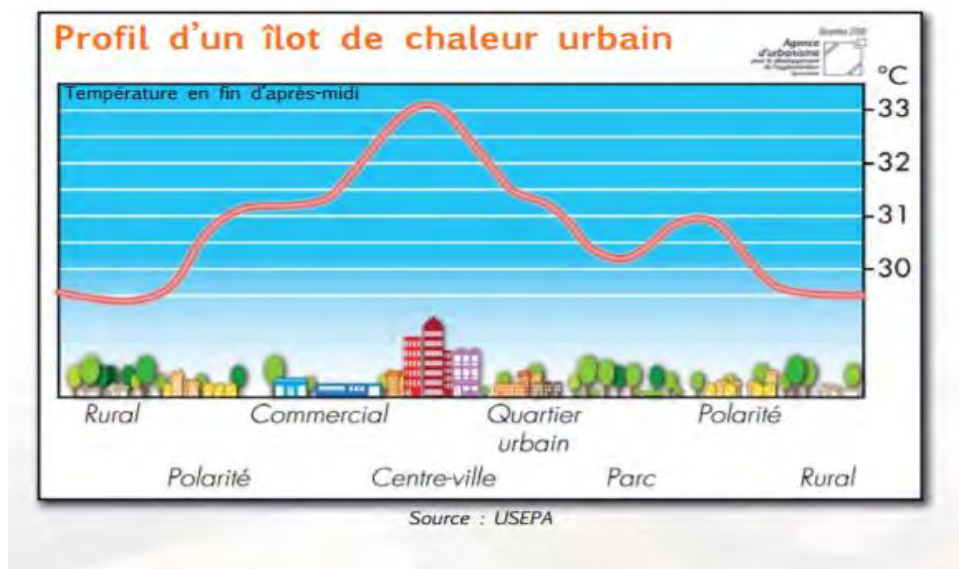


Figure 4 : Profil d'un îlot de chaleur urbain.

Source : USEPA, cité dans "Lutte contre les îlot de chaleur urbain", Communauté Urbaine du Grand Lyon

² « Lutte contre les îlots de chaleur urbain », Référentiel Conception et Gestion des Espaces Publics, Communauté Urbaine du Grand Lyon, 2010

³ <http://www.futura-sciences.com> : Développement durable : îlot de chaleur urbain ; <http://www.futura-sciences.com/planete/definitions/developpement-durable-ilot-chaleur-urbain-5473/>, consulté le 6 mai 2017

⁴ www.notre-planete.info : L'îlot de chaleur urbain, https://www.notre-planete.info/terre/climatologie_meteo/ilot-chaleur-urbain.php#, consulté le 6 mai 2017

Bien que l'on puisse discuter la généralité parfois sous-entendue selon laquelle le centre-ville serait toujours plus chaud que la périphérie, les éléments de définition proposés ci-dessus font apparaître un lien clair entre urbanisation et ICU, ce qui ouvre la question des causes à l'origine du phénomène.

b. Causes et conséquences

Le rapport de la communauté urbaine du Grand Lyon offre à ce propos un panorama assez complet. Le tableau suivant regroupe un grand nombre de critères pouvant intervenir dans la formation des ICU, et le graphique qui l'accompagne en propose une hiérarchisation :

(Sont notées en rouge, les interactions entre la composition urbaine, l'aménagement des espaces publics et les paramètres influençant les îlots de chaleur).

Phénomène microclimatique	Paramètres
Rétention de la chaleur	Propriétés radiatives et thermiques des matériaux (albédo) Géométrie des canyons urbains Exposition du relief au rayonnement solaire Exposition des canyons urbains et des façades au rayonnement solaire Absence d'ombrage Pollution atmosphérique
Perturbation de la dynamique des masses d'air	Topographie Géométrie des canyons urbains Rugosité du tissu urbain
Réduction de l'évapotranspiration	Imperméabilité des surfaces Rareté des masses d'eau Rareté de la végétation
Émission de chaleur par les activités anthropiques	Chaleur émise par les transports Chaleur émise par les bâtiments (hors industrie) Chaleur émise par l'industrie Chaleur émise par le métabolisme humain

Figure 5 : Facteurs pouvant entrer en compte dans la formation des ICU

Source : « Lutte contre les îlots de chaleur urbain », Référentiel Conception et Gestion des Espaces Publics, Communauté Urbaine du Grand Lyon, 2010

Les « canyons urbains » mentionnés dans le tableau correspondent à des lieux « fermés », avec des bâtiments hauts et rapprochés.

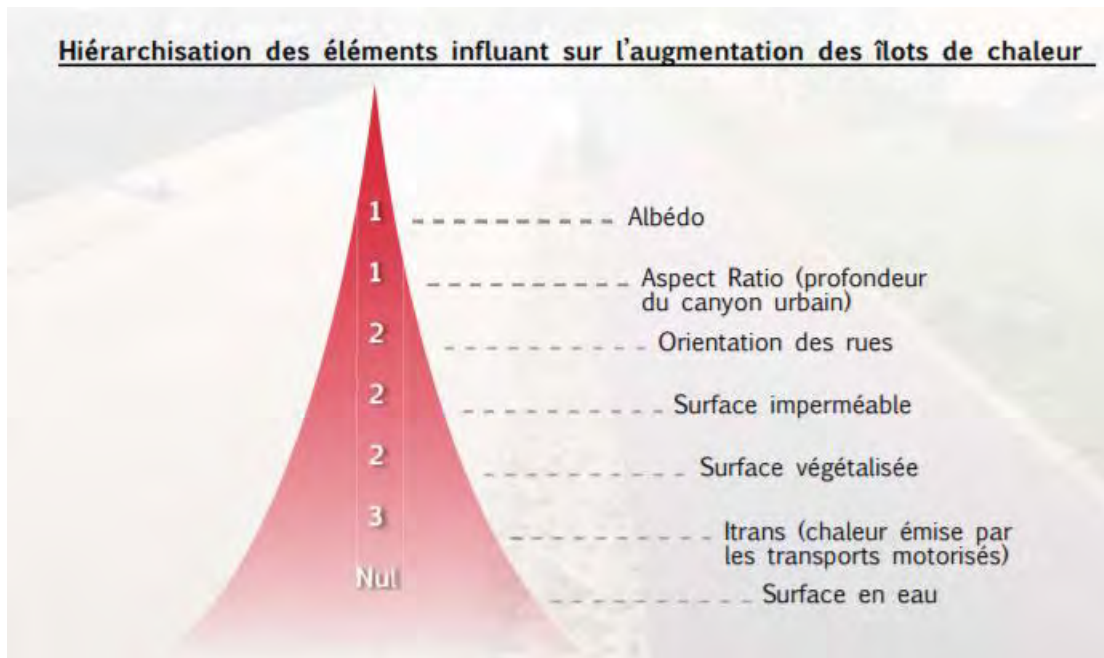


Figure 6 : Hiérarchisation des facteurs entrant en compte dans la formation des ICU. 1 correspondant à l'influence la plus forte.

Source : « Lutte contre les îlots de chaleur urbain », Référentiel Conception et Gestion des Espaces Publics, Communauté Urbaine du Grand Lyon, 2010

On constate ainsi que, si les causes des ICU sont variées, elles ont principalement trait aux activités humaines, à l'urbanisation, à l'artificialisation des sols. Les différences de température induites peuvent sembler négligeables (la figure 3 montre par exemple des différences d'environ 3°C), elles sont néanmoins lourdes de conséquences.

On peut citer par exemple un impact sanitaire, notamment lors des canicules et vagues de chaleur. Une étude de l'APUR⁵ explique ainsi que « *la canicule de 2003 a mis en exergue le caractère éminemment problématique de cette particularité climatique [les ICU, ndr], son effet amplificateur sur la mortalité a marqué les esprits et interroge de façon plus générale les pratiques urbaines et leurs effets sur l'expression du climat d'une ville.* » Leurs conséquences sont aussi météorologiques. Les ICU modifient l'humidité relative, peuvent faire diminuer ou augmenter les précipitations, voire les orages. Dès 1980, Philippe Duchêne-Marullaz expliquait dans un article⁶ : « *la ville perturbe principalement la circulation convective des masses d'air. Son influence est ainsi marquée sur les phénomènes violents comme les fortes averses, les orages ou encore les chutes de grêle. Les journées d'orage peuvent ainsi augmenter de 20 à 30 %* ». Les ICU ont aussi un impact écologique et environnemental, favorisant notamment le phénomène de « brises de campagne »⁷

⁵ Etude « Les îlots de chaleur urbain à Paris », APUR, Cahier numéro 1, décembre 2012, <http://www.apur.org/etude/ilots-chaleur-urbains-paris-cahier-1>, consulté le 6 mai 2017

⁶ Duchêne-Marullaz, Philippe. *Recherche exploratoire en climatologie urbaine*. CSTB, 1980, 86 p.

⁷ Olivier Cantat, « L'îlot de chaleur urbain parisien selon les types de temps », *Noroi* [En ligne], 191 | 2004/2, mis en ligne le 10 septembre 2008, consulté le 16 avril 2017. URL : <http://noroi.revues.org/1373> ; DOI : 10.4000/noroi.1373

Ces dernières peuvent se définir comme des vents thermiques faibles allant des zones froides vers les zones chaudes et favorisant ainsi la concentration des polluants atmosphériques dans les zones les plus urbanisées.

D'un point de vue d'aménageur, on peut également s'interroger sur un éventuel impact d'un ICU import sur l'appropriation des espaces publics concernés, notamment en cas de vagues de chaleurs importantes, où les lieux les plus exposés pourraient devenir difficilement fréquentables voire dangereux pour les publics les plus sensibles (personnes âgées, enfants en bas âges...).

On comprend ainsi que la nature du phénomène d'ICU peut poser des difficultés au moment de chercher à le représenter, le modéliser pour un espace précis sur un horaire précis. En effet, s'il s'agit ni plus de moins de connaître ou d'estimer une température à un endroit à un instant t, il ne s'agit pas de « simplement » mesurer une température au niveau d'une station de mesure ou de proposer une température moyenne pour un espace (comme on peut le faire fréquemment à l'échelle d'une ville, par exemple), mais bien d'estimer une température en chaque point formant le territoire d'analyse, ces points pouvant par exemple se matérialiser sous la forme des pixels d'un raster, une résolution spatiale plus fine permettant alors une représentation de l'ICU plus fine. Or, il est évidemment techniquement impossible de disposer de stations de mesure de température à chaque point de l'espace à mesurer, d'où l'intérêt de faire appel à des méthodes d'interpolation.

c. Etat de l'art sélectif

Nous nous proposons maintenant d'effectuer un bref état des lieux des méthodes existantes pour estimer et représenter les îlots de chaleur urbains. La littérature sur le sujet n'est pas particulièrement vaste, mais on peut néanmoins dénombrer plusieurs travaux recoupant la thématique. La méthode développée par le LISST-CIEU reposant principalement sur des techniques géostatistiques, c'est vers ce type de méthodologies que nous avons orienté cet état de l'art. En effet, si « *la plupart des études récentes ayant porté sur les ICU s'appuient sur des températures de surface télédétektées, les températures leur servant de base ne correspondent pas à celles que ressentent les gens* »⁸ ce qui importe dans différentes applications, notamment sanitaires.

Parmi les méthodes d'interpolation géostatistiques – qui permettent donc une estimation des ICU –, on peut citer notamment les travaux de Charabi, Bigot et Kergomard, en 2002⁹. Ces derniers ont cherché à spatialiser l'îlot de chaleur urbain de la métropole lilloise à l'aide d'un S.I.G en se basant sur des campagnes itinérantes de mesures de température et sur des données relatives au bâti issues du Plan de Gestion Parcellaire. La méthode générale consiste à déterminer un ensemble de variables affectant les températures à partir des données concernant le bâti. Les auteurs identifient entre autres le type d'occupation du sol, le taux de dégagement du ciel, la distance en mètre entre le point de mesure et le mur le plus proche...Un modèle d'interpolation par régression linéaire multiple est ensuite déterminé à l'aide d'un algorithme, fournissant une estimation des températures sur l'espace traité. Cette méthode intègre notamment un bon nombre de facteurs décrivant la morphologie urbaine au niveau des points à estimer.

⁸ Bergeron O., : « *Caractérisation de la variabilité intra-urbaine de la température selon une perspective géostatistique* », Huitième conférence internationale sur le climat urbain (International Conference on Urban Climate), du 6 au 10 août 2012, University College Dublin (Dublin) Irlande.

⁹ Charabi, Bigot et Kergomard, « *Interpolation et cartographie automatique de la température nocturne en milieu urbain* », Publication de l'association internationale de climatologie, Vol.14, 2002

On peut néanmoins critiquer l'usage d'une « simple » régression linéaire multiple, qui ne permet pas vraiment d'intégrer la répartition dans l'espace des points et de leur résidus, ce qui laisse penser que la méthode utilisée demeure sub-optimale en terme de précision.

Les travaux d'Onil Bergeron¹⁰ au Québec emploient ainsi une méthode de « regression-kriging », proche de celle développée par le LISST-CIEU. Celle-ci permet de mieux intégrer la dimension spatiale de l'analyse. L'article publié par Bergeron montre ainsi l'intérêt d'utiliser une méthode de co-krigeage pour spatialiser l'ICU sur la ville de Québec. Les données d'entrée proviennent de mesures de températures effectuées sur le terrain de façon itinérante ainsi que de bases de données concernant l'occupation des sols. Ici, plusieurs méthodes sont comparées (krigeage et co-krigeage) en vue de produire des cartes de température. Les résultats montrent une précision assez satisfaisante des méthodes de co-krigeage, malgré une marge d'erreur qui tend à croître assez fortement lorsque l'on s'éloigne de l'itinéraire de relevé des mesures. A noter que cette étude ne s'intéresse pas aux facteurs relevant de la géographie physique de l'espace, tels que le relief.

A l'inverse, les travaux de Lapparent et al.¹¹ mettent en avant l'influence de l'altitude sur la température. C'est ici une méthode de regression-krigeage qui est privilégiée, à l'instar de celle utilisée dans le projet MApUCE par le LISST-CIEU et que nous détaillerons dans les parties suivantes. Un modèle de régression linéaire simple est ici construit avec l'altitude pour variable explicative. Les résidus issus de cette régression font ensuite l'objet d'un krigeage et sont additionnés au résultat de la régression. L'article s'intéresse néanmoins plus à l'analyse des résultats obtenus d'un point de vue climatologique qu'à la construction du modèle statistique en lui-même, lequel n'intègre pas de donnée relative à l'urbanisation et l'occupation des sols.

Ces études sont celles qui, par leur méthodologie et leur thématique, se rapprochent le plus de l'approche adoptée par le LISST-CIEU et développée dans le cadre de ce mémoire. Le lecteur désireux d'approfondir le sujet pourrait néanmoins se référer à des travaux tels que ceux de Hudson et Wackernagel en 1994¹², qui emploient une méthode analogue (le « kriging with external drift ») pour cartographier des températures sur la quasi-totalité du territoire écossais. Un article de Di Piazza et al.¹³ propose également un comparatif de méthodes géostatistiques telles que les régressions linéaires, le krigeage ou les réseaux neuronaux artificiels pour estimer des températures en Sicile. Pour une application de ces méthodes dans un contexte thématique un peu différent, voir également Jabot et al. qui se sont intéressés aux méthodes de krigeage pour l'interpolation de températures en vue d'améliorer les prévisions de neige et précipitations en contexte Alpin¹⁴

¹⁰ Bergeron O., : « *Caractérisation de la variabilité intra-urbaine de la température selon une perspective géostatistique* », Huitième conférence internationale sur le climat urbain (International Conference on Urban Climate), du 6 au 10 août 2012, University College Dublin (Dublin) Irlande.

¹¹ Lapparent et al., « Mesures de température et spatialisation de l'îlot de chaleur urbain à Dijon », XXVIII Colloque de l'Association Internationale de Climatologie, Liège 2015

¹² Hudson, Warckernagel, « *Mapping temperature using kriging with external drift : theory and an example from Scotland* », International journal of climatology, Vol 14., 77-91, 1994

¹³ Di Piazza et al., « *Comparative Analysis of Spatial Interpolation Methods in the Mediterranean Area: Application to Temperature in Sicily* », Water, 2015

¹⁴ Jabot et al, « *Spatial interpolation of sub-daily air temperatures for improving snow and hydrological forecasts on Alpine catchments* », 68th Eastern Snow Conference, Montreal, Quebec, Canada, 2011.

Ce tour d'horizon montre la prévalence de la technique géostatistique appelée krigeage, et parfois adaptée en « régression krigeage » (regression-kriging). Cette méthode est celle retenue par le LISST-CIEU pour la modélisation des ICU avant le début du stage et se trouve donc au cœur du travail qui sera développé dans ce mémoire. Il convient donc dès maintenant d'en faire une présentation détaillée.

2. Principes du « regression-kriging »

La méthode de régression-krigeage (ou « regression-kriging », RK), est souvent qualifiée, dans la littérature, de « prédiction linéaire non-biaisée optimale » (« Best Linear Unbiased Predictor », abrégée « BLUP » ou « Best Linear Unbiased Estimation », abrégée « BLUE », Hengl, 2007).

Il s'agit d'une méthode combinant, comme son nom l'indique, une régression linéaire et un krigeage appliqué aux résidus. Il est donc nécessaire de bien comprendre ces deux notions avant d'appréhender le fonctionnement du RK.

a. Rappels sur la régression linéaire

Les modèles de régression sont généralement construits dans le but « d'expliquer ou de prédire la variance d'un phénomène (variable dépendante) à l'aide d'une combinaison de facteurs explicatifs (variables indépendantes) ». ¹⁵ Ils cherchent à « approximer une relation fonctionnelle trop complexe en général, par une fonction mathématique simple » ¹⁶.

Dans le cas d'une régression linéaire simple, on suppose une variable à expliquer Y qui peut être définie par une fonction affine d'une variable explicative X, de la forme :

$$Y = aX + b.$$

Les modèles statistiques intègrent en outre généralement la notion d'erreur, qui correspond à une partie de la variable à expliquer que le modèle ne peut justement pas expliquer. Ceci fournit des modèles de la forme suivante : Observation(i) = Modèle(i) + erreur (i)
Pour une régression linéaire simple, on obtient l'équation suivante :

$$Y_i = b_0 + b_1 X_i + e_i$$

Où Y est la variable à expliquer, X la variable explicative, e l'erreur associée à la valeur i et b₀ et b₁ des coefficients décrivant la relation entre X et Y.

Dans les faits, un phénomène s'explique néanmoins rarement par une seule et unique variable, de sorte que nous allons rapidement nous intéresser à la régression linéaire multiple. Celle-ci fonctionne sur un principe similaire mais mobilise plusieurs variables explicatives pour expliquer la variable Y. Son équation peut être envisagée comme une généralisation de celle de la régression linéaire simple. Ainsi, pour n variables explicatives, nous obtenons :

$$Y_i = b_0 + b_1 X_{1i} + b_2 X_{2i} + \dots + b_n X_{ni} + e_i.$$

b₀ correspond à l'ordonnée à l'origine, i.e, à la valeur de Y lorsque toutes les variables explicatives X_n sont égales à 0. Les coefficients b₁ à b_n donnent le poids explicatif de chaque variable au sein du modèle.

Ces coefficients – appelés paramètres du modèle – sont généralement déterminés par le critère des moindres carrés ordinaires (MCO, ou Ordinary Least Squares). Ce dernier correspond à la « minimisation de la somme des carrés des écarts (SC erreur, ou SS error) entre la valeur de Y observée et la valeur de Y estimée par l'équation de régression » ¹⁴.

¹⁵ Site web de l'université de Sherbrooke : usherbrooke.ca, « Régression multiple : rappels théoriques », <http://spss.espaceweb.usherbrooke.ca/pages/stat-inferentielles/regression-multiple.php>, consulté le 12 avril 2017

¹⁶ Confais, Le Guen, « Premier pas en régression linéaire multiple avec SAS », Revue MODULAD, n°35, 2006

Cette somme peut ainsi s'écrire sous la forme :

$$S(b_0, \dots, b_p) = \sum (Y_i - \hat{Y}_i)^2$$

Où S est somme du carré des écarts, Y_i la valeur mesurée (observation) et \hat{Y}_i la valeur estimée par le modèle.

La courbe ci-dessous permet d'illustrer graphiquement ce principe :

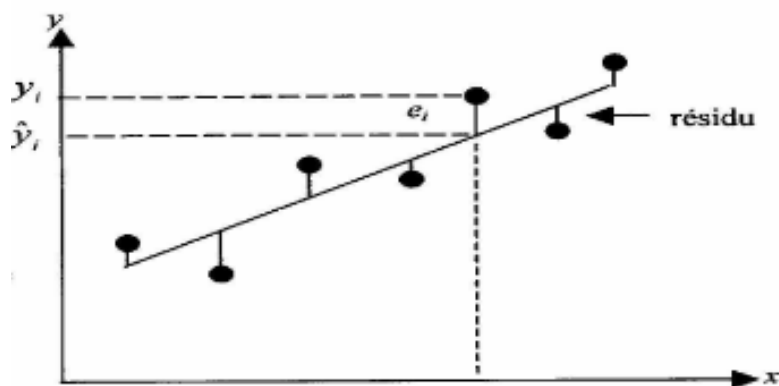


Figure 7 : Illustration du critère des moindres carrés
Source : <http://math.unice.fr/~diener/MAB07/MCO.pdf>

Les points correspondent aux valeurs observées, utilisées pour construire le modèle. La droite est celle issue de l'équation de régression, sur laquelle se situent toutes les valeurs de Y estimées. Les lignes reliant les points à la droite illustrent donc les écarts entre valeurs observées et valeurs prédites. Ajuster le modèle consiste à trouver les coefficients (paramètres du modèle) qui donnent la droite présentant le moins d'écart possible entre valeurs observées et prédites.

On utilise alors le carré de ces écarts pour éviter la compensation entre valeurs positives et négatives. La droite de régression « parfaite » serait donc celle passant par la totalité des points correspondant aux observations (somme des carrés des écarts SCE = 0), ce qui, bien entendu, n'est que rarement possible.

Estimation de la qualité du modèle

Pour évaluer la qualité d'une régression, outre la SCE utilisée pour déterminer les paramètres optimaux, les indicateurs les plus utilisés sont certainement les coefficients de corrélation et de détermination.

Le coefficient de corrélation (ou coefficient Bravais-Pearson), généralement noté r, correspond à « *la covariance entre la variable explicative x et la variable à expliquer y, rapportée au produit de leurs écarts-types.* »¹⁷

$$r = \frac{\sigma_{xy}}{\sigma_x \sigma_y}$$

¹⁷ Baudot J.Y, page web "Coefficient de détermination et R² ajusté", http://www.jybaudot.fr/Correl_regress/coeffdeterm.html, consulté le 12 avril 2017

Ce coefficient est compris entre -1 et 1. Un coefficient négatif indique qu'Y varie en sens inverse de X. Un coefficient égal à 0 indique que les variables sont totalement indépendantes. Plus la valeur tend vers -1 ou 1, plus la corrélation entre les deux variables est forte, dans un sens ou dans l'autre.

Le coefficient de détermination, souvent noté R^2 , est parfois décrit comme le carré du coefficient de corrélation. Ceci n'est cependant vrai que dans le cas d'une régression linéaire simple. Le calcul permettant de déterminer le R^2 (et valable également pour les régressions linéaires multiples) correspond en fait à la part de variance expliquée par le modèle dans la variance totale. Si l'on multiplie ces variances par l'effectif n, ceci revient à l'équation :

$$R^2 = 1 - \frac{SCR}{SCT}$$

Où SCR est la somme des carrés des résidus et SCT la somme des carrés totaux.

Le coefficient de détermination R^2 est donc compris en 0 et 1. Plus il tend vers 1, plus le modèle est explicatif. Un R^2 de 0.8 indique par exemple que 80% de la dispersion est expliquée par le modèle de régression ¹⁷.

b. Le krigeage

La notion de krigeage a été développée par l'ingénieur minier sud-africain Danie. G. Krige puis formalisée et démocratisée par le mathématicien et géologue français Georges Matheron. L'objectif d'origine était de déterminer la distribution spatiale de minerais à partir de plusieurs forages¹⁸. Parfois qualifié « d'interpolation optimale », le krigeage présente notamment l'intérêt de prendre en compte la répartition spatiale des données, élément clé en géostatistiques et en SIG en général.

Le krigeage est linéaire car son estimation est une combinaison linéaire pondérée des données disponibles. Il est « non-biaisé » car il tend à ramener à 0 la moyenne des résidus. Il est « optimal » car il cherche à minimiser la variance des erreurs¹⁹. On parle donc à son propos d'« estimateur linéaire non-biaisé optimal ».

La littérature distingue deux variantes principales du krigeage : le krigeage ordinaire et le krigeage simple. Bien que le premier soit de loin le plus fréquemment utilisé¹⁸ c'est néanmoins le second qui est traditionnellement appliqué dans la méthode de régression-krigeage (dans la mesure où il fait référence à un processus de moyenne connue, comme nous le verrons par la suite).

Qu'il soit simple ou ordinaire, le krigeage se base néanmoins sur un élément qui lui est essentiel : le variogramme. Ce dernier, également qualifié de « semi-variogramme » du fait d'un facteur ½ dans son équation, permet de caractériser la dissimilarité entre les valeurs en fonction de la distance qui les sépare spatialement. La construction mathématique du variogramme est très bien expliquée par G. Hudson, dans un article de 1994. Il explique que la dissimilarité entre deux valeurs peut s'écrire ainsi²⁰ :

$$\gamma^* = \frac{(z_2 - z_1)^2}{2}$$

¹⁸ Gratton Y., "Le krigeage : la méthode optimal d'interpolation spatiale", les articles de l'Institut d'Analyse Géographique, Juin 2002

¹⁹ Isaaks E., Srivastava R., "An introduction to applied geostatistics", Oxford University Press, New York, 1989, 561 p.

²⁰ Les equations présentées ici sont issues de l'article de G.Hudson, *Mapping Temperature Using Kriging with external drift : theory and example from scotland* », 1994

Il s'agit donc de la moitié du carré de la différence entre les valeurs z_2 et z_1 .

Lorsque cette dissimilarité dépend de la localisation spatiale de z_2 et z_1 et de leur orientation, décrite par un vecteur $h=x_2-x_1$, alors l'équation devient la suivante :

$$\gamma^*(\mathbf{h}) = \frac{1}{2} [z(\mathbf{x}_1 + \mathbf{h}) - z(\mathbf{x}_1)]^2$$

Si l'on calcule une moyenne de cette dissimilarité pour le nombre N de paires de points présentes dans le jeu de données et qui peuvent être séparées par un vecteur h (avec une tolérance définie concernant la longueur et la direction de ce vecteur, notamment dans le cas - fréquent - d'échantillons irrégulièrement répartis), nous obtenons alors l'équation du variogramme :

$$\gamma^*(\mathbf{h}) = \frac{1}{2N_h} \sum_{z=1}^{N_h} [z(\mathbf{x}_z + \mathbf{h}) - z(\mathbf{x}_z)]^2$$

Le krigeage se présente ensuite comme un problème d'interpolation spatiale classique, c'est-à-dire, un problème d'estimation d'une fonction $F(x)$, où $x = (x,y)$, en un point x_p du plan à partir de valeurs connues de F en un certain nombre, m , de points environnants x_i ¹⁸:

$$F(x_p) = \sum_{i=1}^m W_i \cdot F(x_i)$$

Mais, à la différence de la régression linéaire évoquée précédemment, le krigeage ajuste plutôt les poids (ici W_i), en fonction du degré de similarité entre les valeurs de F , c'est-à-dire en utilisant la covariance entre les points en fonction de la distance entre ces points¹⁸. Ce degré de similarité est exprimé par l'équation du variogramme. En ajustant une fonction à ce variogramme à l'aide de la méthode des moindres carrés, on obtient une fonction continue caractérisant la variance en fonction de la distance entre les points.

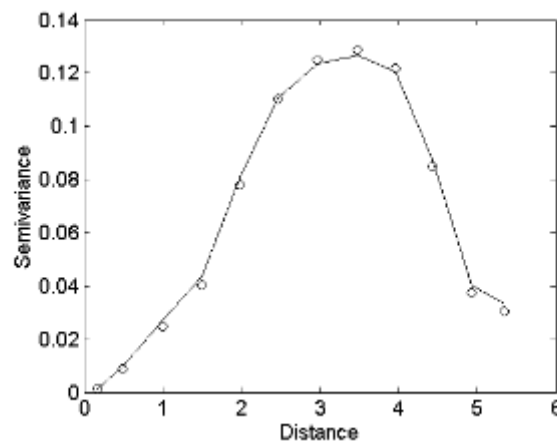


Figure 8 : Exemple de semi-variogramme

Figure extraite de Y.Gratton, 2002, montrant un exemple de semi-variogramme : les points sont obtenus à l'aide de l'équation du variogramme présentée plus haut. Une fonction continue (la ligne continue) a ensuite été ajustée à l'aide de la méthode des moindres carrés.

La différence entre krigeage simple et ordinaire réside dans le fait que le krigeage simple suppose que la moyenne du processus à estimer est connue.

A l'inverse, le krigeage ordinaire s'applique à des processus de moyenne inconnue, ce qui explique que l'on peut le décomposer ainsi :

- a) Estimation de la moyenne « m » inconnue du processus par krigeage ordinaire en utilisant les « n » points d'observations
- b) Estimation du point ou du bloc par krigeage simple en prenant pour moyenne connue la moyenne estimée par krigeage ordinaire, toujours en utilisant les « n » points d'observation²¹

Nous n'avons pas ici vocation à développer les équations inhérentes au fonctionnement du krigeage. Les voici néanmoins présentées dans le cas du krigeage simple. L'explication mathématique présentée maintenant est issue d'un cours de l'école polytechnique de Montréal, présenté par Denis Marcotte.

La variance d'estimation est notée :

$$\sigma_e^2 = \text{Var}[Z_v] + \sum_{i=1}^n \sum_{j=1}^n \lambda_i \lambda_j \text{Cov}[Z_i, Z_j] - 2 \sum_{i=1}^n \lambda_i \text{Cov}[Z_v, Z_i]$$

On cherche à choisir les « poids » (les λ_i) de sorte à minimiser cette variance d'estimation.

On dérive alors la variance d'estimation par rapport à chacun des λ_i et on pose les dérivées partielles égales à zéro.

On obtient alors un système linéaire de n équations à n inconnues, les « n » étant les λ_i

$$\sum_{j=1}^n \lambda_j \text{Cov}[Z_i, Z_j] = \text{Cov}[Z_v, Z_i] \quad \forall i = 1 \dots n$$

On obtient l'estimation par l'équation :

$$Z_v^* = m + \sum_{i=1}^n \lambda_i (Z_i - m)$$

m étant la moyenne connue du processus (pré-requis à l'usage du krigeage simple).

Ces équations sont ensuite écrites et traitées sous forme matricielle.

²¹ Marcotte D., "Krigage", Powerpoint réalisé pour l'école polytechnique de Montréal, 2003, <https://cours.etsmtl.ca/sys866/Cours/documents/krigeage-Marcotte.pdf>

c. Le regression-kriging

Les méthodes de regression-kriging et leurs dérivées ont des applications diverses, par exemple en géostatistique, pour prédire la distribution des caractéristiques d'un sol à partir d'échantillons. S'il s'intéresse à la littérature assez vaste concernant les méthodes de prédiction par krigeage avec régression, le lecteur aura tôt fait de tomber sur des expressions aussi diverses que *Universal Kriging* (Krigeage Universal, UK), *Kriging with External Drift* (KED) ou *Regression Kriging* (RK), souvent employées indifféremment l'une de l'autre, ce qui tend à alimenter une certaine confusion à leur égard. Or, si ces méthodes produisent en théorie des résultats extrêmement similaires, elles diffèrent néanmoins dans leur construction. Dans un article de 2003, Hengl, Heuvelink et Stein²² expliquent ainsi que le terme d'Universal Kriging - originellement décrit par G. Matheron - devrait être réservé aux cas où l'équation de la régression correspond uniquement à une fonction des coordonnées. Le terme de KED doit plutôt être employé lorsque la régression est définie à l'aide de variables auxiliaires extérieures. Dans ce cas, les paramètres associés à l'équation de régression et au krigeage des résidus restent déterminés simultanément, dans une même équation. Mais ils peuvent aussi être ajustés séparément, et additionnés ensuite.

Ce dernier cas fait plutôt référence au *Regression Kriging*. Dans un article de 2006, Hengl, Heuvelink et Rossiter écrivaient ainsi : *"One of these hybrid interpolation techniques is known as regression-kriging (RK) (Odeh et al., 1995 and Hengl et al., 2004b). It first uses regression on auxiliary information and then uses simple kriging (SK) with known mean (0) to interpolate the residuals from the regression model."*

Dans son ouvrage « *A practical guide to geostatistical mapping* », (2007), Hengl²³ explique que le RK peut être vu comme une combinaison d'une approche déterministe et d'une approche stochastique. Une équation de régression linéaire peut ainsi être déterminée et ajustée selon le critère des moindres carrés. Cette phase effectuée, les résidus peuvent ensuite être interpolés par krigeage simple et ajoutés à la régression.

Ce type de méthode ne convient toutefois pas à toutes les situations et Hengl suggère à ce propos un arbre de décision explicite :

²² Hengl, Heuvelink, Stein, "A generic framework for spatial prediction of soil variables based on regression kriging", 2003

²³ Hengl T., "A practical guide to geostatistical mapping on environmental variables", European Commission, Joint Research Center, Institute for environment and sustainability, 2007

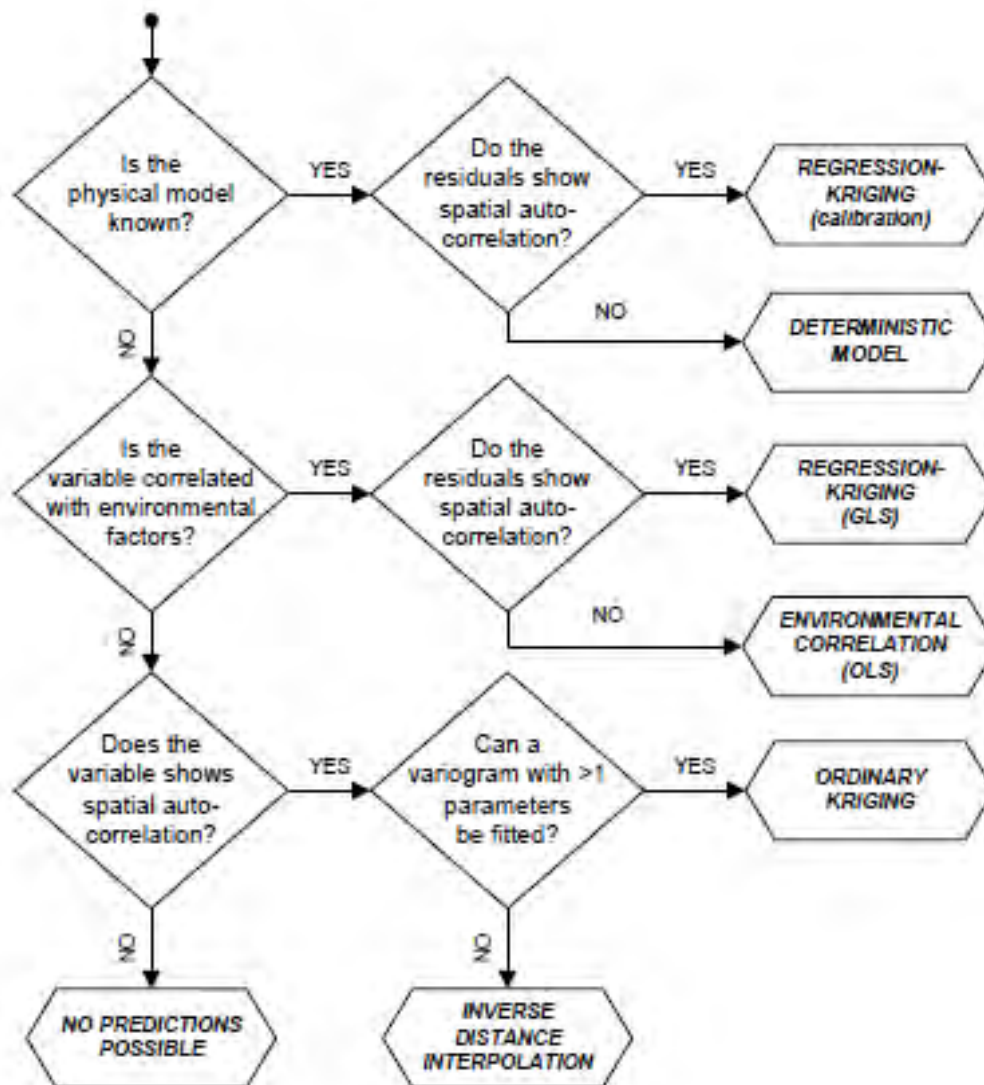


Fig. 2.3: Decision tree for selecting a suitable spatial prediction model.

Figure 9 : Arbre de décision pour la sélection d'un modèle de prédiction spatiale

Source: Hengl T., A practical guide to geostatistical mapping, 2007

Selon ce schéma, il est notamment nécessaire que les résidus issus de la régression montrent une auto-corrélation spatiale afin de pouvoir les intégrer par krigeage.

L'autocorrélation spatiale est le phénomène traduit dans la célèbre phrase: *"everything is related to everything else, but near things are more related than distant things"*²⁴. Concrètement, il s'agit d'une absence d'indépendance entre des observations géographiques, conduisant au fait que des variables spatialisées sont souvent soumises à des dépendances spatiales²⁵ qui sont d'autant plus fortes que leurs localisations sont proches.

²⁴ Miller H., « *Tobler's First Law and Spatial Analysis* », Annals of the Association of American Geographers, 2004

²⁵ IRSTEA, "Les indices d'autocorrélation spatiale, 2002, <https://oasis.irstea.fr/wp-content/uploads/2013/10/10-Autocorr%C3%A9lation.pdf>

La présence d'autocorrélation spatiale tend généralement à invalider la plupart des méthodes d'analyse statistique classiques, qui présupposent l'indépendance totale des observations.

D'où l'intérêt de faire usage du krigeage, qui permet justement de prendre en compte cette autocorrélation spatiale. Par définition, la moyenne des résidus (et non celle de leur carré) est nulle. Nous sommes donc dans une situation à moyenne connue (ici, 0) qui autorise l'usage du krigeage simple.

Ainsi, une estimation de la variable ciblée est réalisée à l'aide d'une régression linéaire multiple. Puis, les résidus issus de cette régression multiple sont interpolés par krigeage simple. Le résultat de cette interpolation est ensuite additionné à celui obtenu par régression linéaire pour donner la prédiction issue du RK.

Appliquons ce principe à notre cas d'étude pour mieux l'illustrer.

3. Application du regression-kriging à la modélisation des ICU : la méthode du LISST-CIEU

a. Principe général

Utiliser le Regression-kriging pour modéliser l'îlot de chaleur urbain dans le cadre de la méthode développée durant le projet MApUCE revient tout d'abord à effectuer une régression linéaire multiple en s'appuyant sur deux variables explicatives (exploratoires) : l'altitude (matérialisée par un MNT) et l'occupation des sols (un raster de fraction de ville). Les températures mesurées sur le terrain par un ensemble de stations servent de référence pour déterminer la variable à estimer : la température à un instant t..

De cette première régression découlent des résidus calculés au niveau des stations. Par nature, leur moyenne tend vers 0 (calcul de la droite de régression linéaire selon le critère des moindres carrés). Ils présentent en outre une autocorrélation spatiale qui autorise l'application du krigage. Le krigage est donc effectué puis le résultat de ce krigage (qui peut ici être matérialisé sous la forme d'un raster) est additionné au raster issu de la régression linéaire pour donner la prédiction finale.

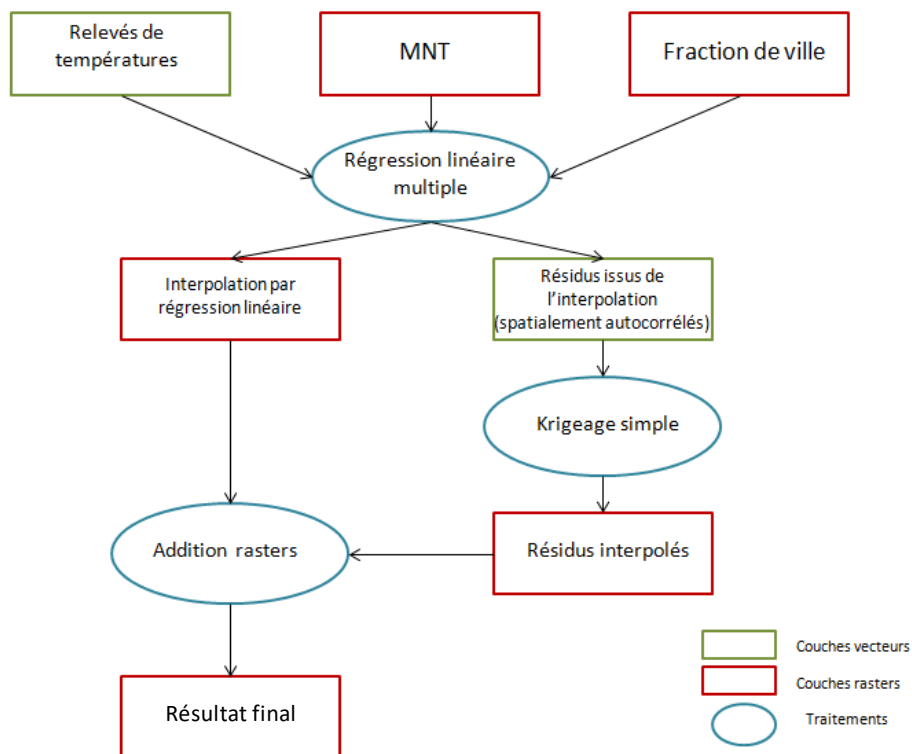


Figure 10 : Fonctionnement de la méthode de regression-kriging appliquée à la méthode de modélisation des ICU

Réalisation : Thomas Gardes, 2017

b. Données et logiciels

Les données utilisées pour mettre au point et tester la méthode proviennent de plusieurs sources. Les relevés de températures sont issus du programme de recherche CAPITOU. Ce dernier, qui visait à étudier l'atmosphère urbaine et la formation/transformation des aérosols urbains, a comporté une campagne de mesure menée sur la région toulousaine entre février 2004 et février 2005. Une classification de ces données atmosphériques par type de situations météorologiques -dite type de temps- (Hidalgo, 2014) a été effectuée. Sur les 11 types de temps présents sur la région Toulousaine, quatre ont été effectivement analysés dans le cadre de cette étude. Ces données sont utilisées sous forme de shapefile de 27 points (pour 27 stations) dont la table attributaire contient en champs la moyenne des températures par heure par type de temps.

Les données concernant l'occupation des sols sont quant à elles issues du projet MAPUCE. Leur importance est considérable puisque, nous l'avons vu, l'ICU est fortement influencé par l'artificialisation du sol et la structure du bâti (hauteur, densité...). Pour représenter cette influence, nous utilisons un ensemble de variables produites par les chercheurs du projet MAPUCE.

Cette base de données, dite « BD MAPUCE » repose notamment sur des données issues de l'IGN (BD Topo pour l'hydrologie et la végétation, INSEE pour la population et calcul de variables supplémentaires telles que la densité de route et de bâti).

Ces variables se matérialisent sous la forme d'une couche vecteur de polygones, correspondant à un découpage en USR du territoire concerné (ici en l'occurrence, la ville de Toulouse). Ces USR (Unités Surfiques de Références) sont un découpage irrégulier visant à représenter la « réalité du terrain ». En pratique, ces unités correspondent généralement aux « ilots » dans les zones urbaines, mais peuvent s'avérer bien plus vastes dans les espaces plus périphériques ou ruraux. Cette couche d'USR contient dans sa table attributaire 40 variables relatives à la structure du bâti et à l'occupation des sols (densité de bâti, de routes, de végétation, hauteur moyenne des bâtiments...). Parmi ces variables, trois ont été retenues pour construire le raster de fraction de ville utilisé dans la régression linéaire décrite précédemment : la densité de route, la densité de bâti et la surface d'enveloppe extérieure du bâti ramenée à la surface de l'USR :

Code	Echelle	Description
ROAD_DENS	USR	Densité surfacique du réseau routier, valeur minimale $= \frac{u_ROAD_A}{Area_{USR}}$
BUILD_DENS	USR	Densité surfacique des bâtiments $= \frac{\sum i_AREA}{Area_{USR}}$
EXT_ENV_AR	USR	Surface de façade extérieure (non contigue) totale des bâtiments de l'USR $= \sum (i_WALL_A - i_PWALL_A)$

Figure 11 : Récapitulatif des variables retenues pour la construction de la fraction de ville
Réalisation : Thomas Gardes, 2017

Ces variables ont été retenues parmi d'autres potentiellement intéressantes (surface végétalisée, hauteur moyenne du bâti de l'USR, volume du bâti de l'USR...) à la suite d'une série de tests de régression linéaire. En s'appuyant sur des critères tels que le coefficient de détermination (R^2) ajusté, et l'erreur quadratique moyenne, il s'est avéré que la combinaison des trois variables décrites ci-dessus était celle qui donnait les meilleurs résultats. Une rasterisation est donc effectuée sur chacune de ces variables, et les rasters correspondants sont additionnés pour donner la fraction de ville.

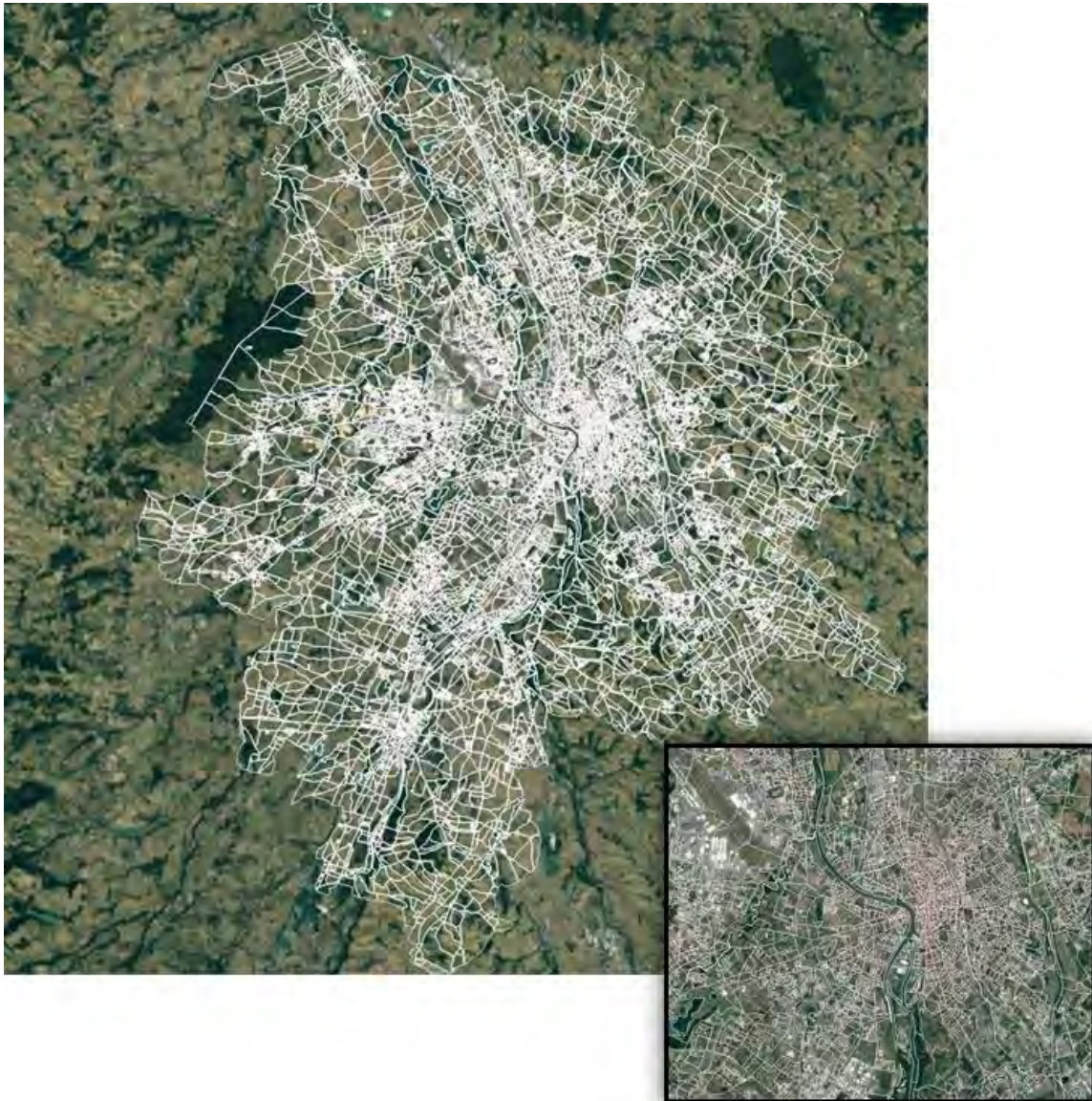


Figure 12 : Les USR : Vue générale et zoom sur le centre-ville
Réalisation : Thomas Gardes, 2017

Enfin, l'élévation est prise en compte via un Modèle Numérique de Terrain (MNT, au format raster) issu de la BD Alti de l'IGN (version 2006), avec une résolution spatiale d'origine de 25m.

Afin de tester les résultats produits par la méthode de modélisation, un jeu de données additionnel a également été utilisé : les relevés de températures « NETATMO ». Les données dites « NETATMO » proviennent d'un ensemble de stations météorologiques commercialisées par la société française NETATMO. Elles produisent notamment des mesures de températures instantanées, récupérées toutes les 5 minutes par un serveur NETATMO via la connexion wi-fi des usagers. Provenant principalement de particuliers, cette source de données relève du crowdsourcing passif (Muller et al. 2015). L'intérêt majeur est de disposer d'un nombre de stations bien plus conséquent que celui mobilisé en général dans les programmes scientifiques. 1034 stations NETATMO sont ainsi recensées à Toulouse (contre 27 stations CAPITOU), ce qui est particulièrement intéressant dans le cadre d'une méthode d'interpolation fondée sur la statistique.

Une contre-partie importante réside néanmoins dans la fiabilité et la qualité de ces données. En effet, le positionnement, l'usage et l'entretien de ces stations par les particuliers ne fait l'objet d'aucun contrôle. Si les stations NETATMO intègrent un module destiné à être placé à l'intérieur et un autre à l'extérieur, rien ne garantit que leur installation ne fasse pas l'objet d'un biais : par exemple, module installé dans un garage ou une véranda, module intérieur placé à l'extérieur ou inversement, etc... ce qui conduit à une nécessaire prudence dans l'exploitation des données de températures fournies. Les potentialités d'utilisation de ces données dans un cadre scientifique ont fait l'objet d'études (Meier et al. 2016). Dans le cadre d'un stage de recherche à l'Université Technique de Berlin, Adrien Napoly a notamment travaillé sur une méthode de filtrage des données NETATMO²⁵. Cette dernière permet d'éliminer efficacement les stations présentant un biais de température identifiable. Nous constaterons toutefois par la suite que des stations sujettes à un « biais spatial » demeurent parfois présentes dans les échantillons (températures plausibles en elles-mêmes mais aberrantes compte tenu de la spatialisation des stations, par exemple, deux stations distantes de moins de 30m l'une de l'autre avec deux ou trois degrés d'écart).

Ce jeu de données constitue néanmoins un échantillon très intéressant car statistiquement plus significatif que celui de CAPITOU et autorisant par exemple l'usage de méthodes de type cross-validation.

²⁵ Napoly A., *Utilisation des données participatives Netatmo pour l'étude du climat urbain. Application aux villes de Toulouse, Paris et Berlin*, INP Toulouse, Ecole Nationale de Météorologie, mémoire de Master II, 2017

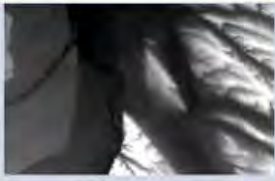




Donnée	Type	Source	Date de production	Illustration
MNT	Raster	IGN	2006	
Relevés de température CAPITOUL	Vecteur (points)	Projet CAPITOUL	2005	
Relevés de température NETATMO	Vecteur (points)	Stations NETATMO	2016	
Données MApUCE par USR	Vecteur (polygones)	Projet MApUCE	2015	
Fraction de ville	Raster	Construite à partir des données MApUCE	2017 (construite dans le cadre de la modélisation des ICU)	

Figure 13 : Vue d'ensemble des données utilisées
Réalisation : Thomas Gardes, 2017

Ce tableau récapitulatif permet notamment de constater un important décalage temporel entre les relevés de températures et les données MApUCE d'occupation du sol. Cette dernière peut avoir été modifiée de façon non-négligeable en 10 ans. Ceci ne pose pas de réel problème au moment d'appliquer la méthode de régression-kriging. C'est néanmoins une information à garder en tête au moment d'interpréter les résultats et d'en estimer la qualité.

Concernant les logiciels employés, deux principaux sont à citer : SAGA GIS dans sa version 2.3.1 et QGIS en 2.14.12 LRT.

SAGA GIS a d'abord été utilisé pour son panel d'algorithmes géostatistiques, facilitant l'usage du « régression-kriging ». La méthode a donc principalement été testée et développée sous cet environnement.

QGIS intègre néanmoins une bibliothèque d'algorithmes issus de SAGA GIS suffisamment fournie pour permettre de reproduire la méthode. QGIS a donc ensuite été privilégié pour sa plus grande popularité et sa meilleure accessibilité. La méthode y a été reproduite, testée, affinée, en vue de l'automatisation sous la forme d'un plug-in, décrite en partie II.

c. Décomposition de la méthode et de sa mise en œuvre sous SAGA GIS

Théoriquement, en se basant sur les explications proposées précédemment, il est possible d'appliquer la méthode sous SAGA GIS de deux façons :

- i) Réaliser « manuellement » une régression linéaire, suivie d'un krigeage simple sur les résidus puis d'une addition raster
- ii) Utiliser l'algorithme « regression-kriging » qui effectue l'ensemble de ces traitements.

La solution i) semble la plus intéressante pour étudier la mise en œuvre des différentes étapes. Il peut ensuite être intéressant de la comparer à la solution ii).

Réalisation « manuelle » de la méthode : enchaînement de deux algorithmes

La régression linéaire multiple s'effectue ici en utilisant les rasters de MNT et de fraction de ville avec les relevés de températures ici du projet CAPITOUL. L'exemple retenu et choisi aléatoirement est celui de l'horaire 20H00 UTC pour la période juin-juillet-août (jja). L'algorithme utilisé sous SAGA GIS 2.3.1 s'appelle « Multiple Regression Analysis (Points and Predictor Grids) ». Sa configuration est illustrée ci-dessous :

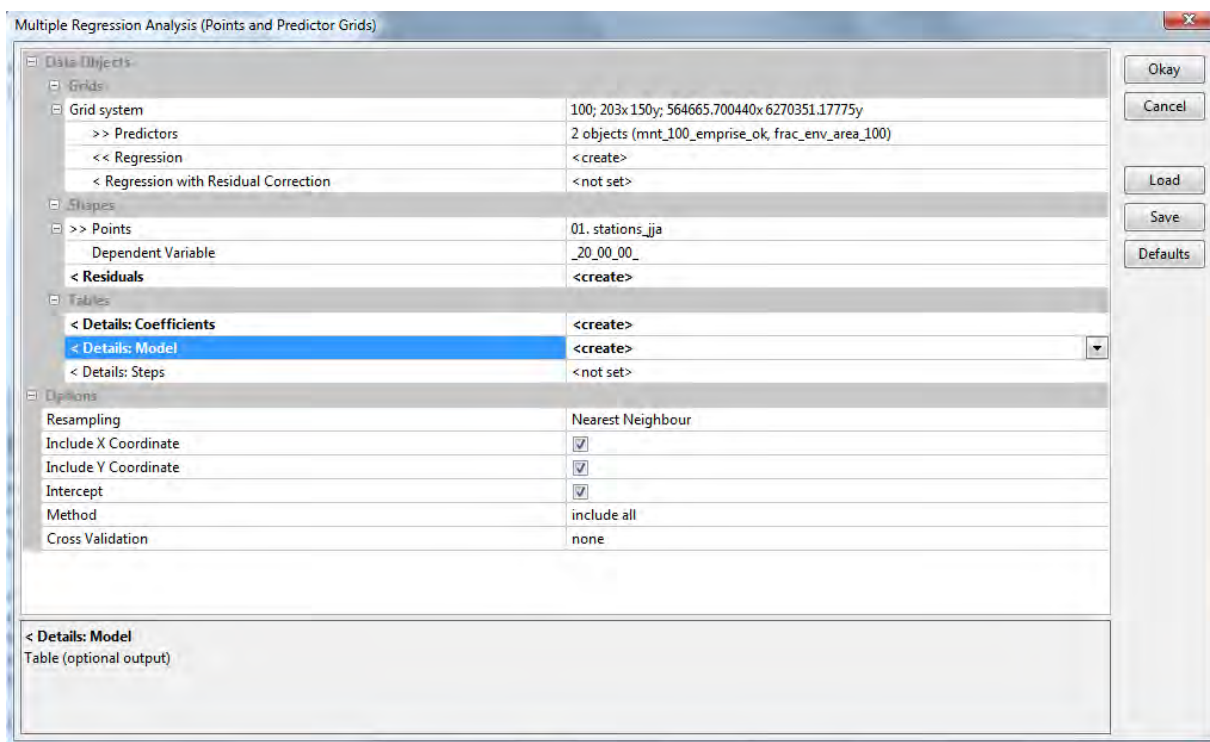


Figure 14 : Configuration de l'algorithme Multiple Regression Analysis sous SAGA GIS 2.3.1 (capture d'écran)

Les « predictors » correspondent aux deux variables exploratoires utilisées (ici, les rasters donc). On demande à créer une régression (paramètre « regression » sur « create »).

La couche de points choisie correspond donc aux stations CAPITOUL, période jja, 20H00 UTC. On crée une couche de résidus et on demande à produire le « model », qui permet de vérifier des critères tels que le R^2 ajusté ou l'erreur quadratique.

Le ré-échantillonnage se fait au plus proche voisin (« Nearest Neighbour »), méthode classique adaptée à la situation (pas de transformation géométrique complexe, etc...).

On choisit d'inclure les coordonnées X et Y qui constituent des variables exploratoires au même titre que les rasters d'entrée.

L'« intercept » fait référence à l'ordonnée à l'origine, incluse dans l'équation de régression. La documentation sur cette option est inexistante. Il semble néanmoins que la décocher force l'intercept à 0, ce qui, par expérience et dans le cas qui nous concerne, semble affecter négativement le coefficient de détermination. Nous la laisserons donc cochée. La méthode « include all » permet la prise en compte de l'ensemble du jeu de données. Nous n'avons pas ajouté ici de méthode de cross validation, bien que cela soit envisageable.

La régression produit le résultat suivant :

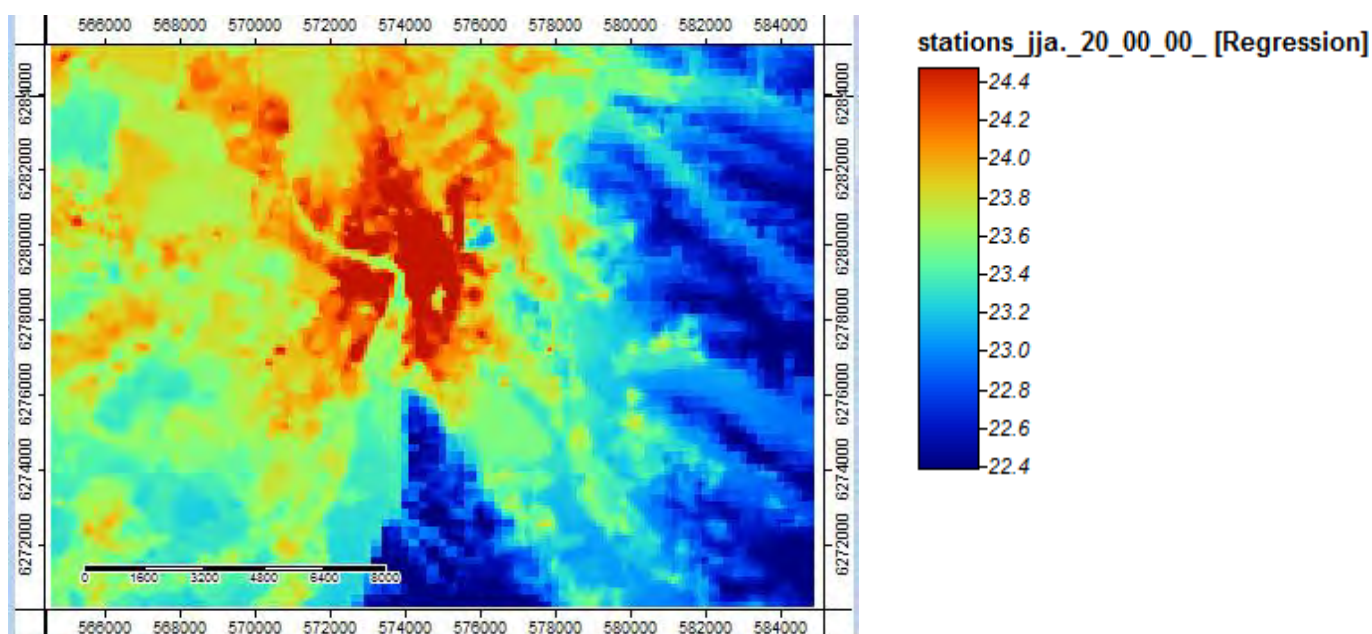


Figure 15 : Résultat obtenu par régression linéaire multiple, selon la configuration de la figure précédente (capture d'écran)

Le coefficient de détermination R^2 obtenu sur cette régression est de 0.720, ce qui montre une bonne corrélation entre les variables explicatives et la variable à expliquer.

Pour se référer à l'arbre de décision proposé par Hengl, nous sommes dans la situation où notre variable à expliquer est corrélée à des facteurs environnementaux (R^2 assez élevé) et où les résidus montrent de l'autocorrélation spatiale, ce qui permet d'utiliser un krigeage simple sur ces résidus.

L'algorithme de SAGA « Simple Kriging » est donc configuré comme suit :

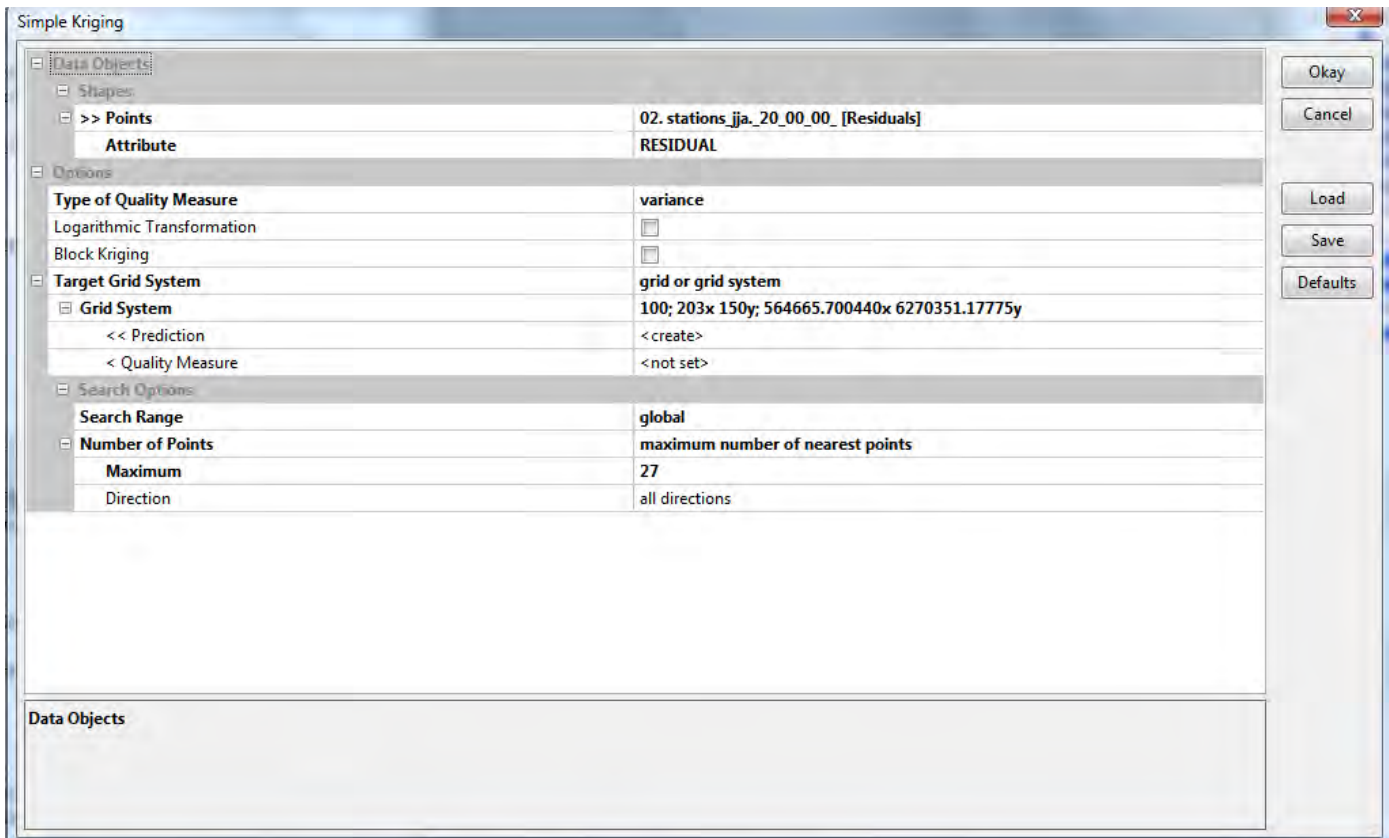


Figure 16 : Configuration de l'algorithme Simple Kriging sous SAGA GIS 2.3.1 (capture d'écran)

Nous n'effectuons pas de transformation logarithmique et nous prenons pour emprise celle de la prédiction issue de la régression.

Le search range correspond à la distance à laquelle les données vont être cherchées pour être utilisées dans le krigeage. Ici, on fixe le paramètre sur « global » pour utiliser l'ensemble des données, ces dernières permettant de couvrir l'ensemble de l'emprise déterminée. Le nombre maximal de points (ici de relevés de températures donc) à prendre en compte pour l'estimation d'un point peut également être fixé. Cela permet notamment de réduire les temps de traitement dans le cas de jeux de données importants. Ce n'est ici pas nécessaire et nous prendrons en compte l'intégralité du jeu de données, soit 27 points.

A noter qu'il est possible de configurer un Search Range local et de lui spécifier une distance suffisante pour couvrir l'intégralité de l'emprise (ici, une valeur de 30 000 suffit). On peut alors également choisir un nombre minimal de points à intégrer pour l'estimation. Cependant, des tests basés sur des différences entre les rasters produits dans les différentes configurations montrent que, dans notre cas, les résultats obtenus sont strictement identiques.

L'algorithme affiche alors une fenêtre intitulée « variogram ».

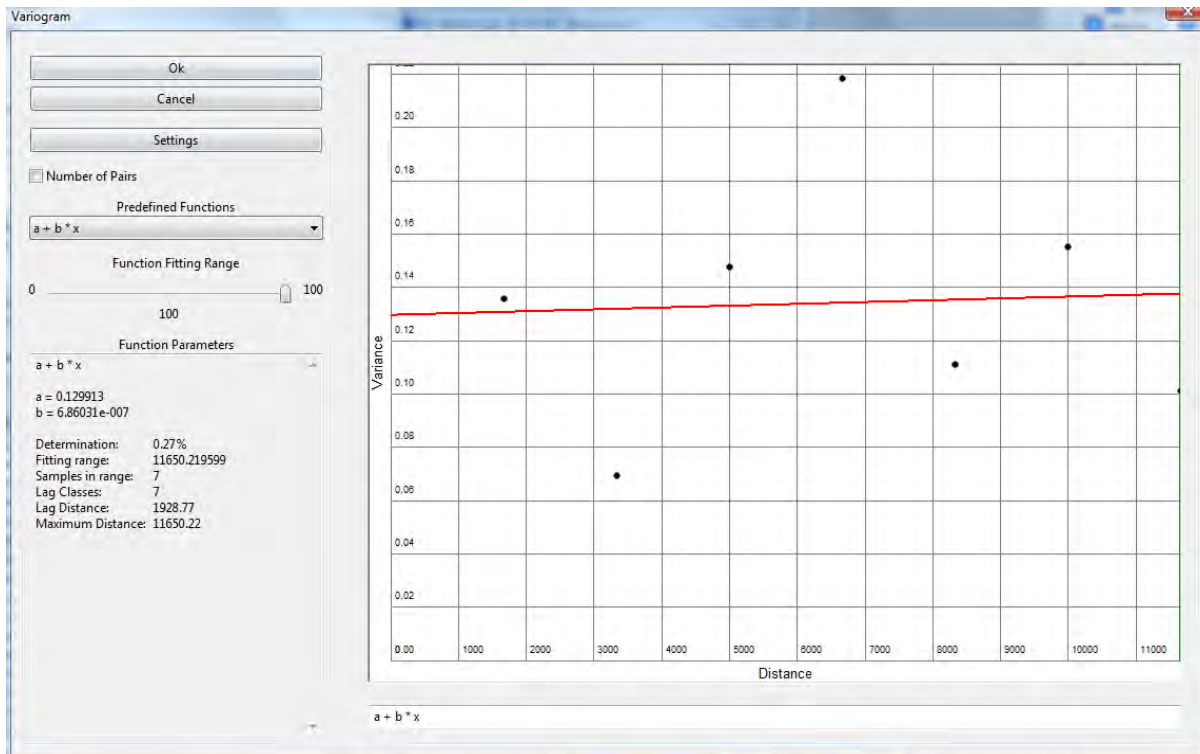


Figure 17 : Fenêtre "Variogram" lors de l'usage de l'algorithme Simple Kriging, sous SAGA GIS (capture d'écran)

Comme l'explique le développeur de l'algorithme, Olaf Conrad, contacté à ce propos: *"it tells only, how good the variogram model (the 'red line', i.e. the function used to calculate the weights for the subsequent kriging interpolation of the residuals of the mlra model) is fitted to the so called 'experimental variogram' (the 'black dots', i.e. the variances within increasing lag distances)"*

Il affirme que la "determination value", visible dans le cadre de gauche, est « *of minor importance in this context* ». Celle-ci sert à caractériser la façon dont la fonction de variogramme est corrélée au variogramme expérimental.

Nous obtenons donc le résultat suivant :

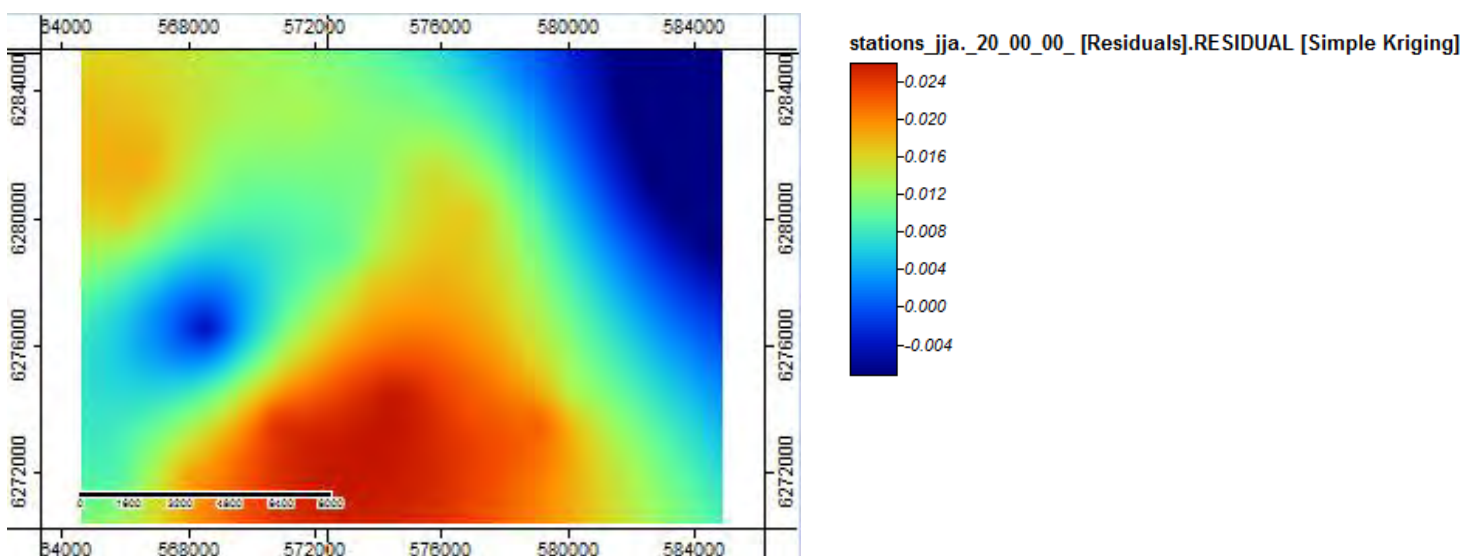


Figure 18 : Résultat obtenu suite au krigeage simple des résidus issus de la régression linéaire (capture d'écran)

Nous pouvons désormais additionner les résidus krigés à la prédiction issue de la régression linéaire afin de l'affiner.

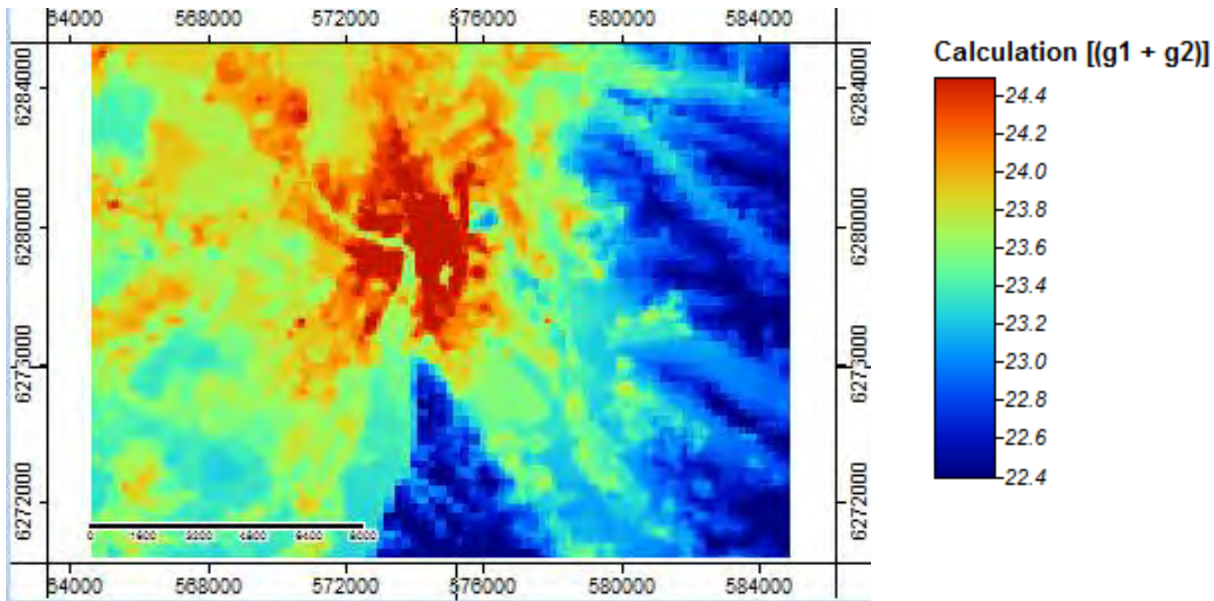


Figure 19 : Résultat de l'addition entre le raster de résidus krigés et le résultat de la régression linéaire (capture d'écran)

Nous obtenons ainsi le résultat final du regression-kriging. Ce dernier semble visuellement peu diverger du résultat de la régression, mais une soustraction raster effectuée sous QGIS 2.14 montre des différences pouvant approcher les 2°C à certains endroits.

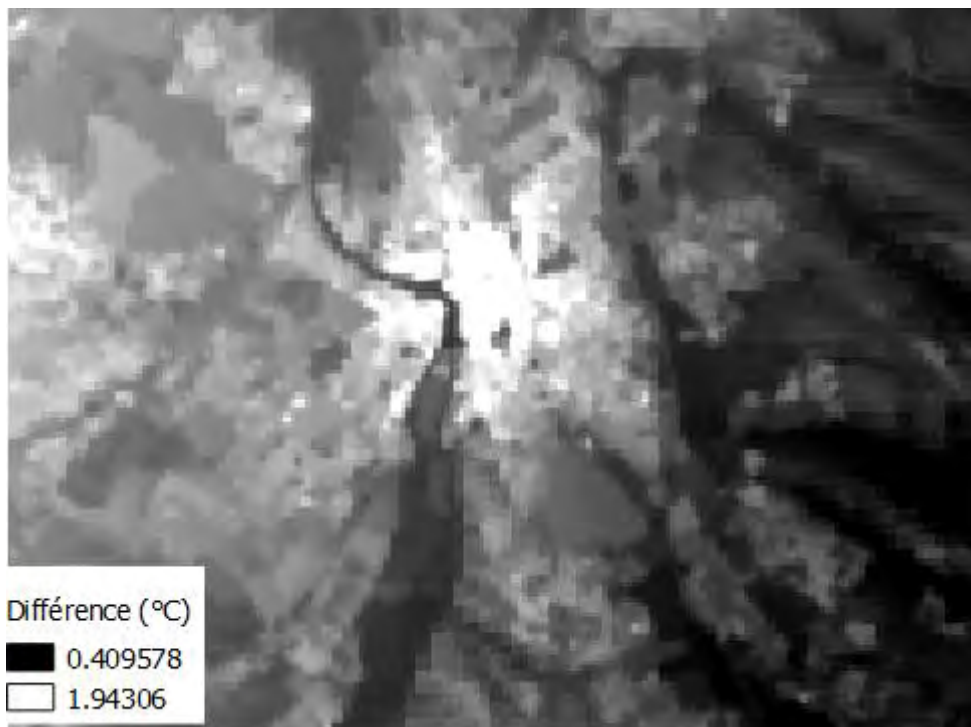


Figure 20 : Soustraction entre le raster issu de la régression linéaire et le résultat final avec les résidus krigés (capture d'écran)

La méthode automatisée : l’algorithme “regression-kriging”

A titre de comparaison, il est intéressant de réaliser la même démarche avec l’algorithme regression-kriging, qui regroupe les traitements effectués précédemment.

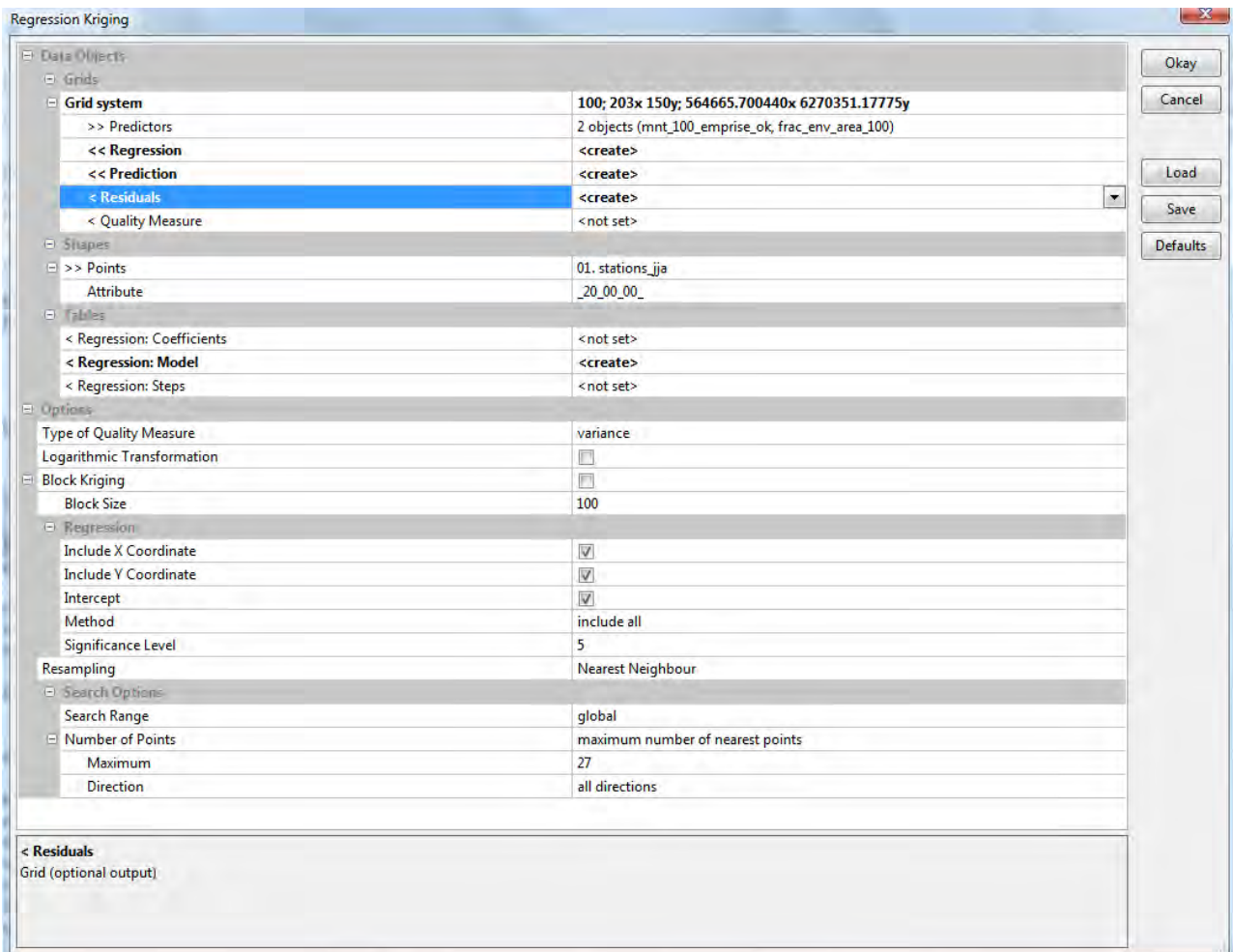
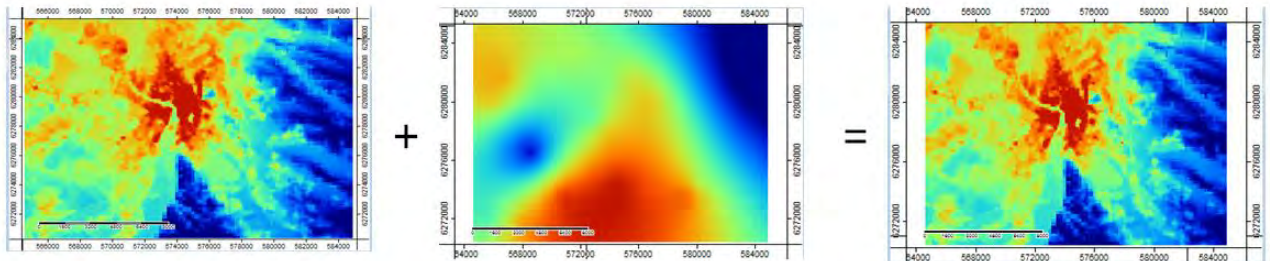


Figure 21 : Configuration de l’algorithme Regression Kriging sous SAGA GIS 2.3.1 (capture d’écran)

L'interface se présente comme une combinaison de celle des algorithmes Multiple Regression Analysis et Simple Kriging. Nous la remplissons donc de la même façon que précédemment. On peut maintenant comparer les résultats obtenus en sortie à ceux issus du traitement « manuel ». L'algorithme regression-kriging produit en sortie trois rasters : la régression, les résidus krigés, et la somme des deux (appelée « prédiction »).

De g. à d. : Regression, résidus krigés et rendu final, pour l'algorithme Multiple Regression Analysis suivi du Simple Kriging



De g. à d. : Regression, résidus krigés et rendu final, pour l'algorithme Regression Kriging

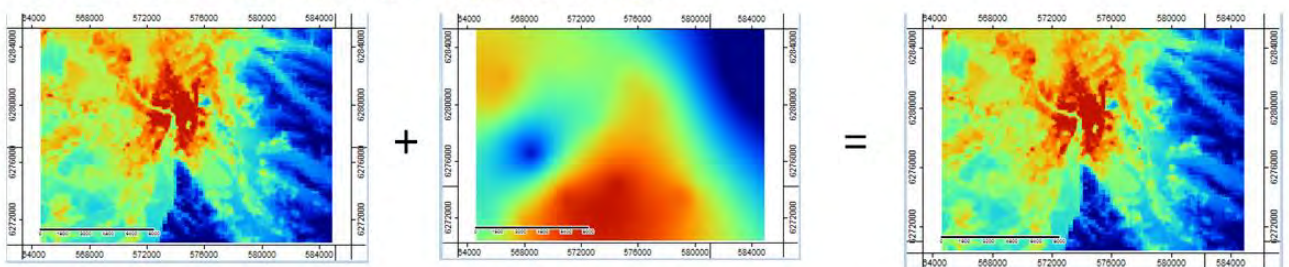


Figure 22 : Comparaison des résultats obtenus selon les deux méthodes testées

Réalisation : Thomas Gardes, 2017

Visuellement, les résultats paraissent identiques. En outre, le modèle de régression linéaire est, logiquement, le même, avec un même R^2 . Pour pousser plus loin la comparaison, une soustraction raster a été réalisée sous QGIS, entre les rendus finals des deux méthodes. On obtient le résultat suivant :

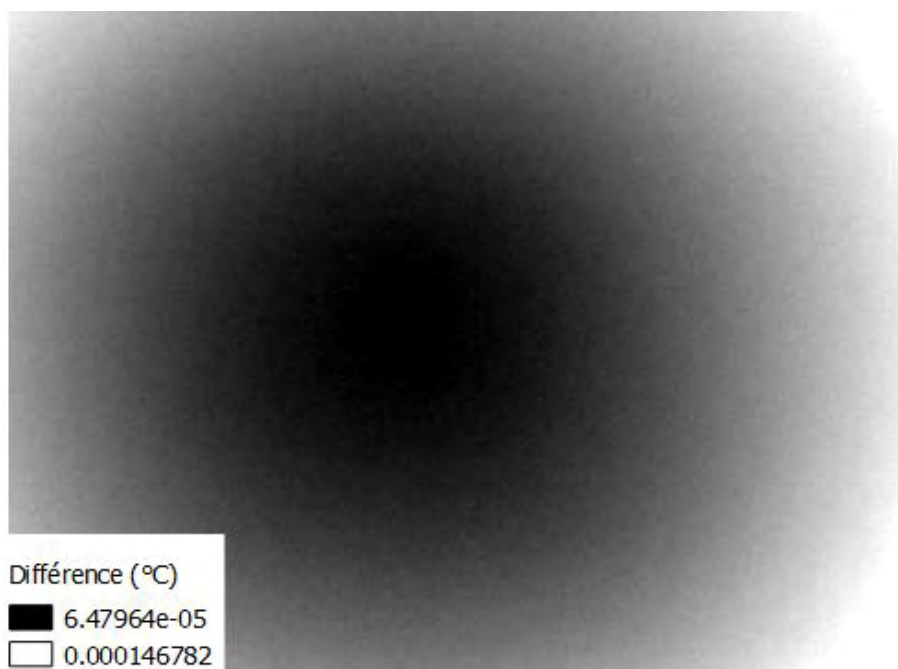


Figure 23 : Résultat de la soustraction raster entre les résultats finals obtenu selon les deux méthodes (capture d'écran)

On constate que les deux rasters sont finalement différents. Mais ces différences sont si petites – variant entre 0,00006 et 0.0001 – que l'on peut les qualifier de négligeables. Il semblait néanmoins pertinent de s'interroger sur l'origine de ces différences. Après expérimentation, il s'avère donc que l'algorithme « regression-kriging » de SAGA utilise non pas un krigeage simple, comme généralement recommandé dans la méthodologie du regression-kriging mais un krigeage ordinaire.

Si l'on effectue ainsi une régression linéaire multiple suivie d'un krigeage ordinaire des résidus (algorithme « Ordinary Kriging ») et —que l'on additionne les résultats, une soustraction raster montre alors un résultat strictement identique à celui obtenu par l'algorithme « regression-kriging ».

Si les différences entre les résultats obtenus par krigeage ordinaire et par krigeage simple apparaissent infinitésimales dans notre contexte, elles peuvent néanmoins susciter un questionnement d'ordre méthodologique quant au choix qui a été fait d'utiliser le krigeage ordinaire plutôt que simple et ouvre également une piste de questionnement dans le cadre d'une automatisation de la méthode : est-il plus pertinent d'automatiser l'algorithme « regression-kriging » ou plutôt un enchaînement utilisant une régression linéaire multiple et un krigeage simple sur les résidus ?

4. Analyse et discussion des résultats obtenus

Les résultats produits par la méthode ont fait l'objet d'analyses et de questionnements. L'un des éléments clés étant l'évaluation de la qualité des résultats obtenus et donc, de la fiabilité de la méthode.

Pour ce faire, plusieurs indicateurs ont été retenus. Ils se basent en partie sur une comparaison entre les valeurs mesurées par les stations CAPITOUL (valeurs de référence) et celles prédites par l'interpolation au même point (valeurs estimées).

Cette comparaison est effectuée notamment par le biais du plug-in QGIS « Point Sampling Tool », Celui-ci permet de sélectionner en entrée un champ d'une couche de points et une couche raster et produit en sortie une couche de points contenant en table attributaire deux champs : les valeurs du champ de points en entrée et celles du raster au niveau des points.

Pour étudier la qualité des résultats de l'interpolation, on indique donc en entrée le raster d'interpolation final et le champ de la couche de stations CAPITOUL correspondant à l'horaire modélisé. Un troisième champ est ensuite ajouté manuellement : la valeur absolue de la soustraction entre valeur des stations CAPITOUL et valeur de l'interpolation. La différence entre les deux est ainsi facilement quantifiable. L'outil QGIS « Statistiques basiques sur les champs numériques » appliqué sur ce champ permet ainsi d'obtenir facilement la différence moyenne, les écarts minimums et maximums, la somme des écarts, leur médiane...

Il est également possible de s'intéresser au coefficient de détermination R^2 et à l'erreur quadratique moyenne, qui sont deux des indicateurs liés à la régression linéaire multiple. Ils ne peuvent toutefois servir qu'à qualifier cette étape et non pas celle du krigeage et donc, du résultat final.

Dans une optique d'ajustement de la méthode, de nombreuses modélisations ont été effectuées et testées, notamment sur des journées complètes (24 modélisations, heures par heures). Le choix a néanmoins été fait ici de synthétiser en présentant les résultats obtenus à des horaires clés. Ces horaires ont été choisis en fonction des caractéristiques des types de temps par saison telles que déterminées sur Toulouse par Julia Hidalgo (N.Touati, J.Hidalgo, 2016).

Les plages horaires privilégiées pour la modélisation sont donc résumées dans le tableau suivant :

		Heure (UTC)	Heure local (LT)	T _{max et min zone urbaine} (°C)
Printemps (Cluster 2)	Jour	16	18	24.95
	Nuit	6	8	14.44
Été (Cluster 9)	Jour	16	18	27.99
	Nuit	5	7	17.08
Été – jour chaud (Cluster 9)	Jour	16	18	34.01
	Nuit	4	6	18.79
Automne (Cluster 3)	Jour	14	15	11.73
	Nuit	5	6	7.1
Hiver (Cluster 5)	Jour	15	16	10.77
	Nuit	5	6	5.71

Figure 24 : Horaires retenus pour la modélisation de l'ICU

Source : J. Hidalgo, N. Touati, 2016

Par soucis de concision et de clarté, nous commenterons dans cette section deux exemples particulièrement parlants. L'intégralité des 10 cartes de températures produites aux 10 horaires du tableau précédent est néanmoins présente en annexe 1.

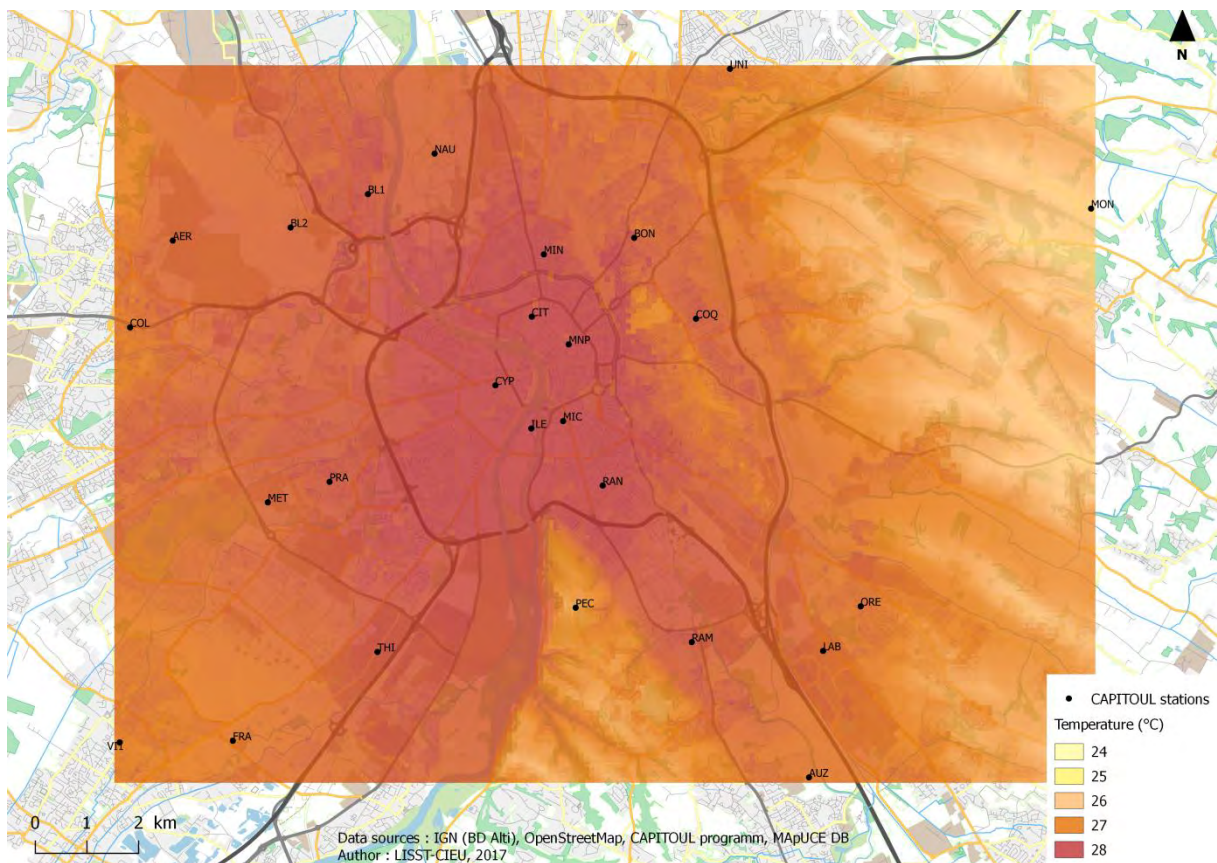


Figure 25 : Température de l'air à Toulouse à 16H UTC pour la période été, a 100m de résolution spatiale
 Réalisation : Thomas Gardes, 2017

Indicateurs été, 16H UTC	
R ² (coefficient de détermination)	0.622
Ecart moyen (°C)	0.426
Ecart minimal (°C)	0.091
Ecart maximal (°C)	1.226

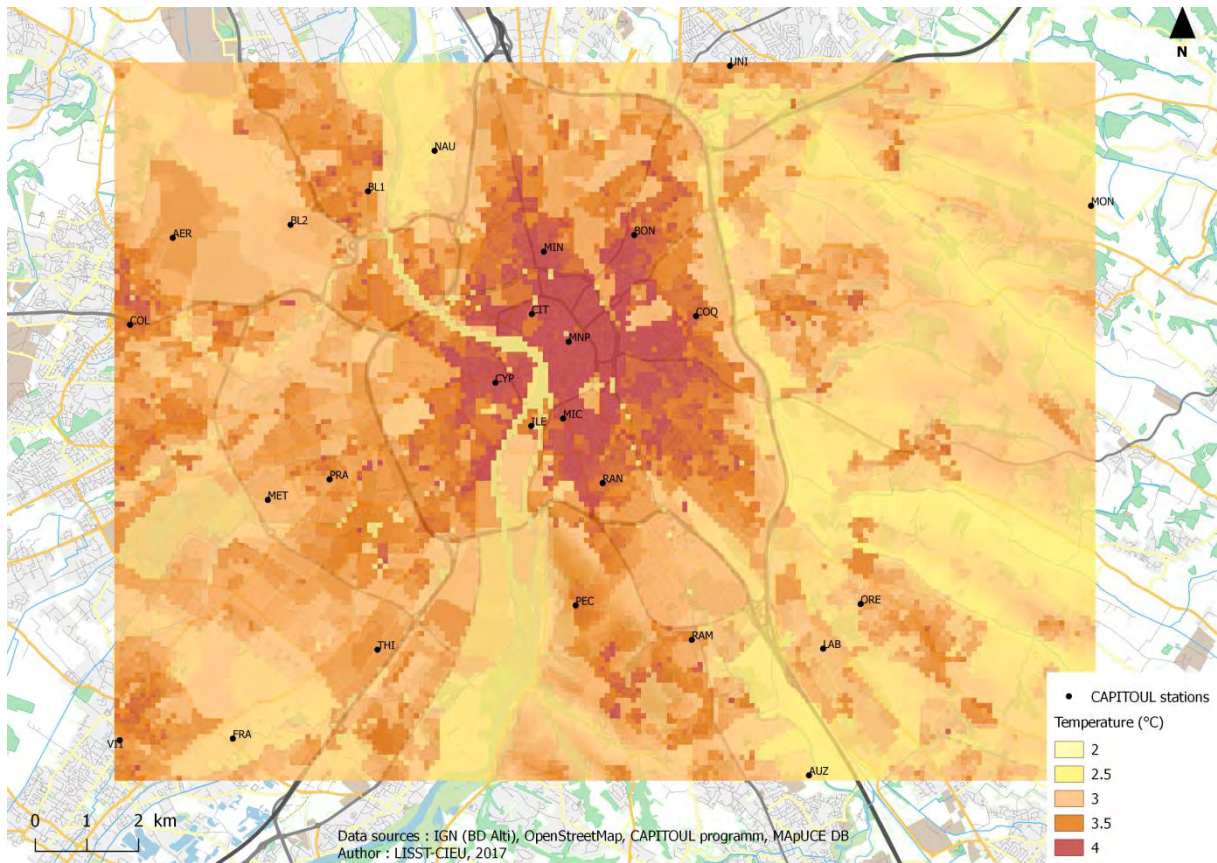


Figure 26 : Température de l'air à Toulouse à 05H UTC pour la période hiver, a 100m de résolution spatiale
Réalisation : Thomas Gardes, 2017

Indicateurs hiver, 05H UTC	
R² (coefficient de détermination)	0.597
Ecart moyen (°C)	0.253
Ecart minimal (°C)	0.001
Ecart maximal (°C)	0.708

Globalement, les rendus montrent une importante prise en compte des particularités morphologiques du territoire. La carte d'hiver à 5H UTC montre un ICU très visible sur le centre-ville, de part et d'autres de la Garonne. La température décroît progressivement lorsqu'on s'éloigne vers la périphérie. On observe des « tâches » de température plus élevées au niveau de certaines zones d'activité, par exemple (vers Colomiers notamment).

La carte d'été à 16H UTC montre un ICU beaucoup plus diffus. Les caractéristiques morphologiques tendent à se démarquer, comme les coteaux de Rangueil au sud, un peu plus frais, ou la zone de relief à l'est. La zone de températures élevées est plus vaste que dans le cas de l'hiver, ce qui peut s'expliquer notamment par la saison et l'heure plus avancée de la journée (l'ICU tend à se « diffuser » au cours de la journée).

Concernant la qualité des modèles, on observe des R^2 acceptables, aux alentours de 60% de détermination dans les deux cas. Si cela n'est pas perceptible dans les exemples présentés ici, une étude plus approfondie des R^2 sur la journée aux différentes saisons montre que ce coefficient tend à décroître à la mi-journée pour remonter ensuite. On peut formuler l'hypothèse que cette variation est due au fait que, au fur et à mesure que la journée avance, la diffusion de la température fait que celle-ci est moins directement dépendante du relief et de la fraction de sol artificialisée, qui sont les variables utilisées pour la modélisation. Ceci mériterait cependant une étude plus approfondie.

Les écarts entre température mesurée au niveau des stations et température estimée par interpolation sont satisfaisants. Ils n'excèdent les 1°C que pour deux stations de la situation été, et ne les dépassent que légèrement (1.2°C d'écart maximal). La moyenne de 0.426°C d'écart est acceptable. Le cas hiver montre des résultats sensiblement plus précis, puisque l'écart maximal est de seulement 0.7°C , soit assez en dessous du seuil symbolique des 1°C . La moyenne des écarts est basse : 0.253°C , et les écarts les plus bas sont tout à fait négligeables (de l'ordre de 0.001°C).

Nous retrouvons ici une précision semblable à celle obtenue avec des modèles physiques à des résolutions spatiales similaires. (J.Hidalgo, SAGEO)

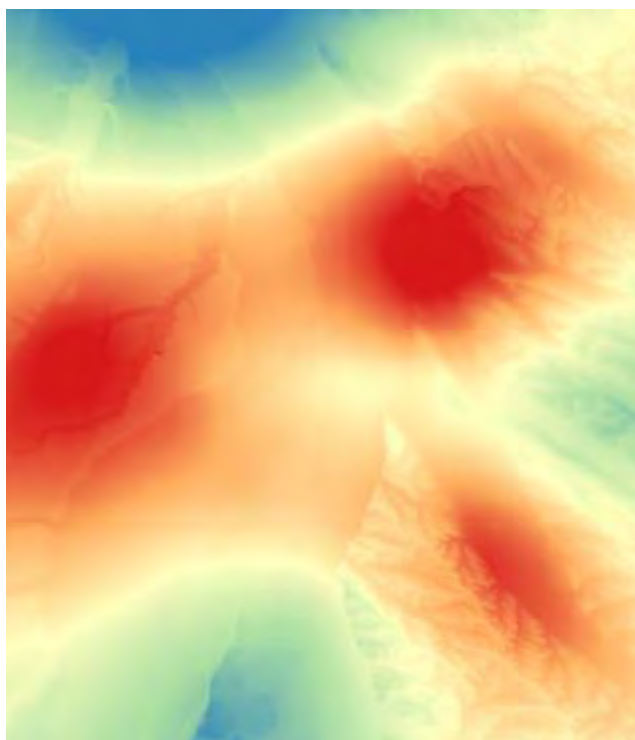
La première critique à formuler ici concerne toutefois le nombre de points impliqués. 27 points constituent en effet un seuil assez bas pour appliquer une méthode statistique, et empêche des méthodes d'estimation de la qualité telles que la cross-validation. D'où l'intérêt d'exploiter le jeu de données NETATMO.

Les données NETATMO pour Toulouse ont été récupérées grâce à Adrien Napoly, à des dates sélectionnées pour être particulièrement représentatives des types de temps présents dans les données CAPITOUL. Deux dates présentant des données en quantité et qualité suffisantes ont été retenues : le 14 décembre 2016 (type de temps automne) et le 16 août 2016 (type de temps été). Les données NETATMO les plus anciennes datant de 2015, il est en revanche impossible d'effectuer une comparaison directe avec les résultats des stations CAPITOUL, qui datent de 2005.

Sur les deux dates retenues, deux horaires ont été sélectionnés pour modélisation – les mêmes que pour les types de temps correspondant avec CAPITOUL : 16H et 5H UTC pour le 16 août, 14H et 5H UTC pour le 14 décembre.

Les 1034 stations de chaque horaire ont donc été extraites des fichiers CSV d'origine, comprenant la localisation de chaque station (en longitude/latitude) et la température associée. Les CSV ont ensuite été importés sous QGIS. Les stations n'ayant pas de température associée à cet horaire (défaut de la donnée) ont été supprimées et une première interpolation a été effectuée.

Pour chaque horaire considéré, cette interpolation a néanmoins fourni des résultats de piètre qualité (rendu visuel du raster de température, différence importante entre température interpolée et mesurée, R^2 faible, RMSE importante...), comme illustré ci-après :



Ecart minimal	6.604
Ecart maximal	0.004
Ecart Moyen	1.688

Figure 27 : Raster obtenu après interpolation sur les données NETATMO non filtrée à 18H le 16 août 2016 (capture d'écran)

On constate notamment des écarts importants entre températures mesurées et températures estimées, pouvant dépasser les 6°C, avec plus d'un degré d'écart en moyenne.

En vérifiant les stations présentant les écarts température mesurée/température estimée les plus importants, on constate rapidement des valeurs « aberrantes » : certaines stations distantes de moins de 30m l'une de l'autre présentent parfois plus de 2°C d'écart, ce qui laisse entrevoir des biais de la nature de ceux évoqués précédemment.

Développer un filtrage approfondi des données NETATMO ne rentrait ni dans les objectifs du stage ni dans ses possibilités en termes de temporalités. Devant la volonté de rendre les échantillons NETATMO exploitables, la méthode suivante a donc été appliquée :

- Une première interpolation a été effectuée sur les données non-filtrées (sauf retrait des stations n'ayant pas de température associée, soit environ la moitié des stations par horaire)
- La comparaison entre T° mesurée et T° estimée aux différentes stations a été réalisée via le plug-in QGIS Point Sampling Tool.
- Les stations présentant les écarts les plus importants (supérieurs à 3°C) ont été supprimées
- Une nouvelle interpolation a été effectuée sur les stations restantes.
- Une nouvelle comparaison est effectuée et la méthode est ainsi répétée jusqu'à ce que l'essentiel des points jugés « aberrants » ait été supprimé. L'objectif fixé est de conserver au minimum 100 stations par horaire, afin de légitimer l'usage de méthodes de cross-validation.

Cette démarche présente un biais évident dans la mesure où elle tend à « adapter les résultats à la méthode » et non l'inverse. Elle permet néanmoins, appliquée dans ce contexte avec le seul objectif de simplement retenir un échantillon à la fois quantitativement significatif et qualitativement

exploitable, d'obtenir rapidement un résultat utilisable en vue de tests en cross-validation. Un nombre satisfaisant de stations possédant des mesures de températures valables a ainsi pu être sélectionné pour chaque horaire :

Horaire	Nombre de stations
16/08/16 : 5H UTC	328
16/08/16 : 16H UTC	128
14/12/16 : 14H UTC	330
14/12/16 : 5H UTC	267

Figure 28 : Nombre de stations NETATMO par horaire modélisé

Une interpolation a été réalisée pour chaque échantillon. Puis, chaque échantillon a été divisé en vue de la cross-validation. La cross-validation (ou « validation croisée ») est une méthode d'estimation de la fiabilité d'un modèle basée sur l'échantillonnage.

La cross-validation appliquée ici est dite « *testset validation* ». Il s'agit d'une méthode particulièrement simple : l'échantillon (ici les stations) est divisé en deux parties. L'une d'elle est utilisée pour l'interpolation. L'autre va servir à la vérification. En l'occurrence, on va mesurer l'écart entre température estimée par interpolation et température mesurée sur des stations **n'ayant pas servi à l'interpolation**. Ceci était jusqu'à présent impossible en raison du trop faible nombre de stations sur l'échantillon CAPITOU.

Les ensembles de stations filtrées ont donc été divisés aléatoirement en deux groupes, à l'aide de l'algorithme de sélection aléatoire de QGIS. Un groupe contenant 70% des stations a été utilisé pour l'interpolation, les 30% restants, pour vérification.

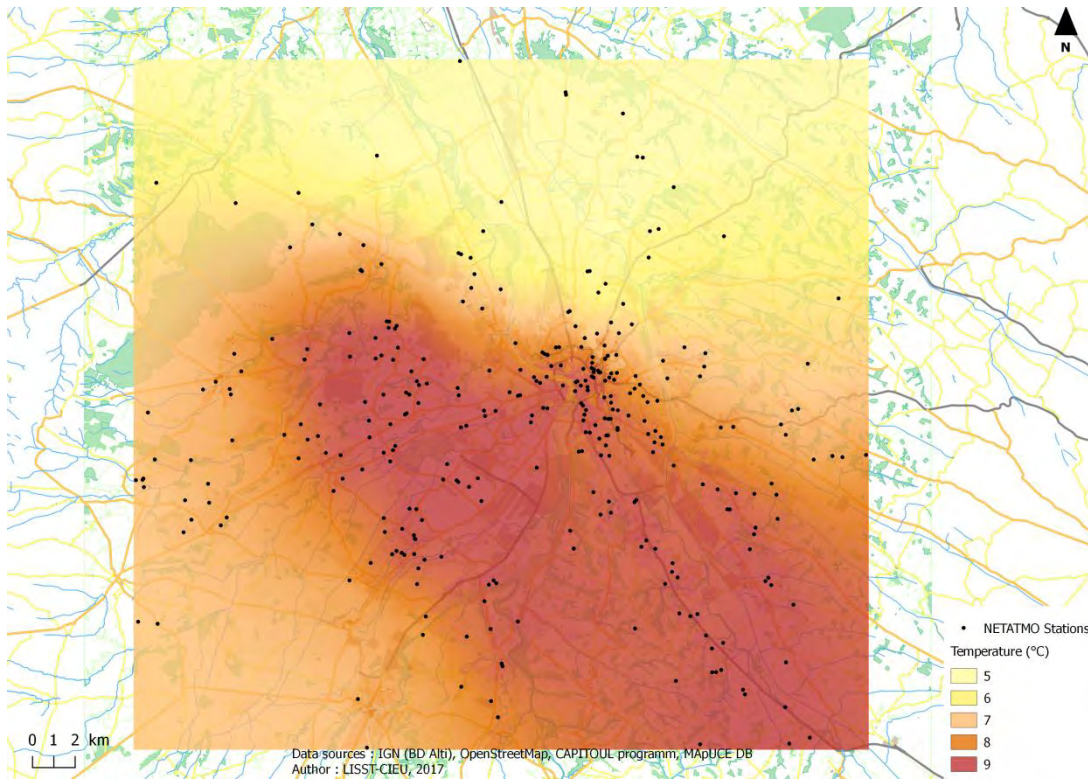


Figure 29 : Température de l'air à Toulouse à 05H UTC le 14/12/16 (automne), à 100m de résolution spatiale, données NETATMO
 Réalisation : Thomas Gardes, 2017

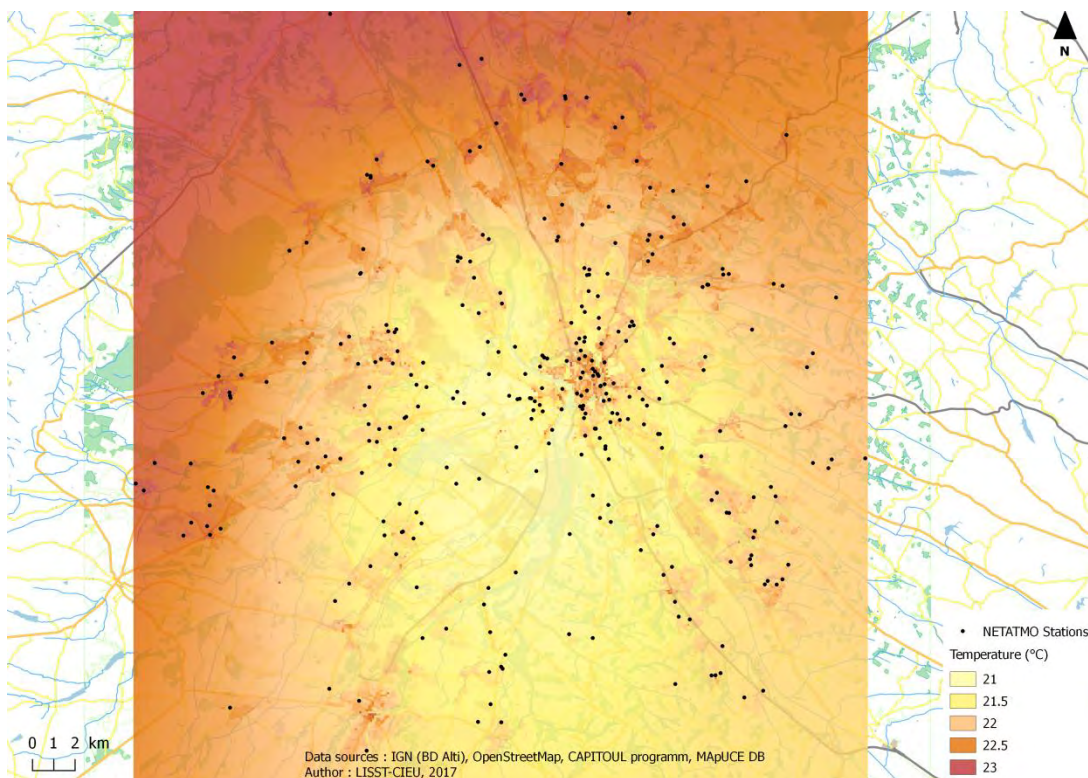


Figure 30 : . Température de l'air à Toulouse à 16H UTC le 16/08/16 (été), à 100m de résolution spatiale, données NETATMO
 Réalisation : Thomas Gardes, 2017

	Eté	Automne
Ecart minimal	0.000	0.002
Ecart maximal	1.114	0.964
Ecart moyen	0.475	0.364

Figure 31 : Indicateurs pour l'étude des résultats obtenus en cross-validation à partir des données NETATMO

Réalisation : Thomas Gardes, 2017

L'aspect des cartes obtenues peut déstabiliser au premier abord. Il convient tout d'abord de noter que l'emprise couverte par le jeu de données NETATMO est bien plus grande que celle des données CAPITOU. Bien que les données aient été découpées de sorte à rester dans l'emprise couverte par le MNT et les données MApUCE, on peut constater que les coins du raster de sortie souffrent généralement d'une faible densité de stations, qui rend la prédiction de température uniforme et imprécise dans ces zones. Néanmoins, si l'on se concentre sur l'emprise équivalente à celle couverte par les données CAPITOU, on remarque des résultats qui semblent visuellement en cohérence avec ceux produits en utilisant lesdites données.

La carte du 14/12/16 montre une sorte de « nuage » de température élevé, qui couvre la zone centre-ouest jusqu'au sud-est et peut sembler curieux compte tenu des résultats observés dans les autres modélisations et de ce que l'on connaît de la morphologie du territoire. Ce résultat (à nuancer quelque peu compte tenu d'assez faible amplitude de température représentée sur la carte) semble imputable en partie à la méthode de discrétisation choisie mais surtout à la qualité des données d'entrées, dans la mesure où de tels phénomènes ne s'observent pas sur les autres horaires NETATMO modélisés et que la précision de l'interpolation (écarts températures mesurées/estimées) reste satisfaisante.

En effet, la vérification par cross-validation fournit des résultats corrects, avec des écarts moyens en dessous des 0.5°C. Sur l'exemple d'été, seules 3 stations sur 98 utilisées pour validation présentent un écart supérieur à 1°C (avec un écart maximal de 1.114). Pour le cas d'automne, aucune des 80 stations de validation ne montre plus d'un degré d'écart avec son résultat interpolé.

D'une manière générale, les résultats obtenus aux différents tests sont encourageants et montrent une méthode statistique à la fiabilité comparable à celle de modèles physiques. En toute logique, on constate que les résultats sont néanmoins fortement assujettis à la qualité des données fournies en entrée. Pour un résultat optimal, l'interpolation doit idéalement pouvoir compter sur un nombre élevé de points fiables, mais il reste possible d'obtenir des résultats intéressants avec des jeux de données restreints, en témoignent ceux obtenus à partir du programme CAPITOU.

Plusieurs remarques et critiques méritent néanmoins d'être formulées.

Tout d'abord, la prise en compte de la fraction de sol artificialisé se fait uniquement par le biais des données MApUCE. Les variables retenues pour la représenter ont fait l'objet d'un processus de test mais elles ne garantissent pas une prise en compte optimale.

Il pourrait être intéressant d'explorer plus en détail l'intégration de variables supplémentaires, comme les cours d'eau, l'albédo des surfaces, etc. L'avantage étant que la méthode développée reste

assez flexible pour intégrer aisément ce type de variables à l'analyse, via notamment la régression linéaire multiple.

Un travail approfondi sur les résolutions spatiales et leur impact sur la qualité de modélisation reste également à effectuer. La résolution de travail de 100m retenue ici s'avère fonctionnelle compte tenu de la taille des USR sur Toulouse, et fournit déjà une précision intéressante. Néanmoins, des jeux de données plus précis concernant l'occupation des sols pourraient permettre de travailler à des résolutions plus fines, puisque le MNT, facilement disponible à 25m de résolution spatiale, ne constitue pas une limite.

Une piste de réflexion pourrait également se porter sur la visualisation des résultats en sortie. Les cartes présentées précédemment font en effet le choix d'une palette de couleurs chaudes – exprimant des variations de températures positives – réparties en 5 classes (qui permettent de conserver une bonne lisibilité) avec un pas fixé à 0.5 ou 1°C entre chaque classe, selon l'amplitude à représenter. Si cette discrétisation permet de tenir un discours accessible en utilisant des seuils assez parlants, elle n'est pas nécessairement optimale puisqu'elle tend à faire disparaître des nuances que l'algorithme de Jenks, par exemple, laisserait mieux paraître. L'aspect uniforme de la température observé à l'est ou à l'ouest sur certaines cartes s'explique ainsi en partie par cet effet (et en partie par la densité de stations plus faible sur ces zones). En outre, lorsque l'amplitude à représenter est faible (moins de 5°C), les couleurs pourraient laisser penser à un œil non-averti, que les écarts de température sont importants, alors qu'ils n'excèdent parfois pas les 2°C entre la nuance de jaune la plus pâle et le rouge le plus foncé. Il est toutefois difficile de répondre à cette problématique tout en conservant suffisamment de nuances et de lisibilité. Une réflexion plus poussée sur les façons de rendre compte et de représenter les résultats obtenus pourrait donc s'avérer bénéfique. Signalons enfin que les données à l'application de cette méthode demeurent parfois difficilement accessibles (notamment les relevés de température), ce qui pose la question de ses applications.

Usages possibles

Concluons donc cette partie par une brève réflexion sur les possibilités d'application de la méthode développée. En effet, bien que disposant d'une précision a priori acceptable, l'interpolation demeure une méthode statistique, et, à ce titre, conserve une part de probabilisme qui ne doit pas être négligée. Si la précision dépend en grande partie de la qualité et quantité des données utilisées, il est peu probable d'imaginer que les cartes produites puissent servir dans des situations où il est nécessaire de connaître des températures exactes au dixième de degré près (voire même au degré près dans certains cas, des écarts de l'ordre de 1°C étant parfois constatés entre données de référence et interpolées). Les températures estimées ne valent donc pas tant pour leur valeur absolue que pour leur valeur relative, qui permet de déterminer, avec une précision spatiale intéressante, les zones « chaudes » et les zones « froides ». L'étude de l'îlot de chaleur urbain et notamment de son évolution dans le temps peut donc profiter de tels résultats. Ces derniers pourraient notamment être exploités en matière d'aménagement du territoire et d'urbanisme, sur des questions énergétiques par exemple, mais aussi de santé publique. Citons en guise d'illustration cette carte produite par l'AUAT en 2015, qui s'appuyait sur les premiers résultats ébauchés par le LISST-CIEU au cours du projet MApUCE :

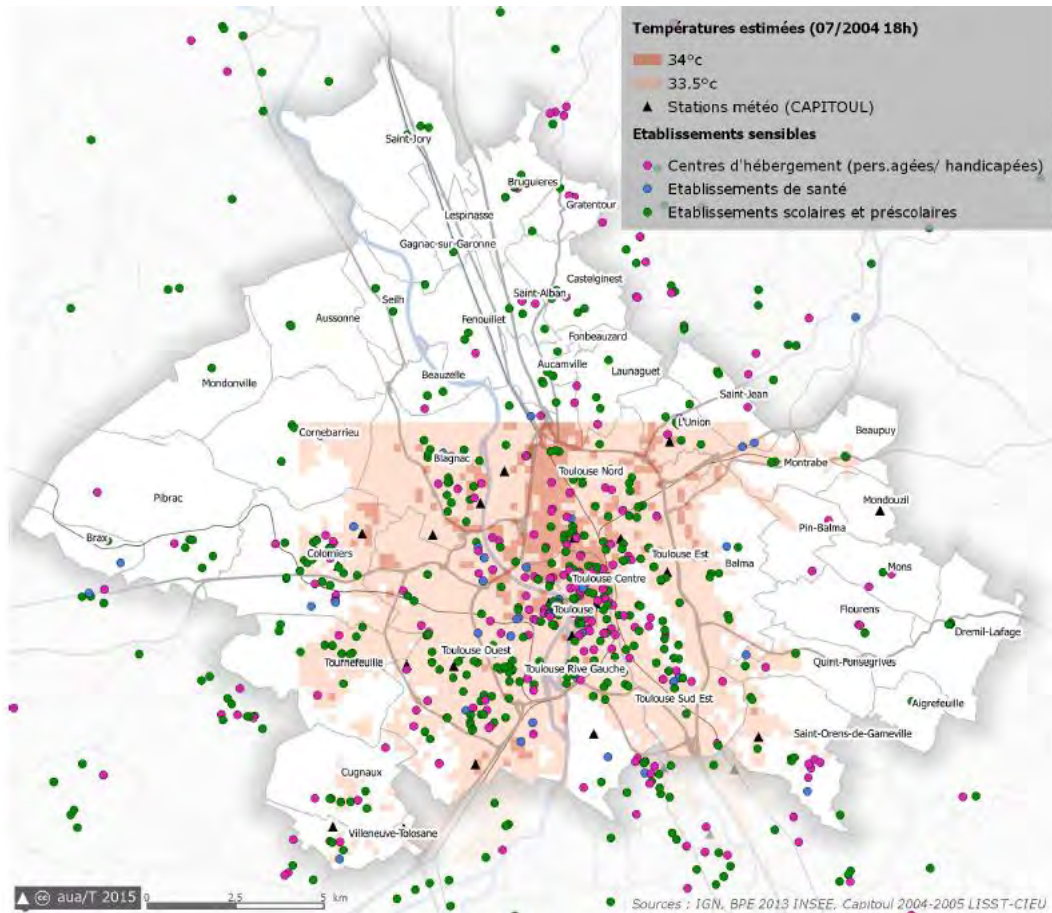


Figure 32 : Les établissements "sensibles" de Toulouse par rapport à l'ICU
 Réalisation : AUA/T 2015. Source : J.Hidalgo, N.Toutai, 2016

Celle-ci met en relation les établissements accueillant un public dit « sensible » aux fortes chaleurs avec les zones où les températures sont justement les plus élevées. De telles études pourraient bénéficier de la précision plus importante obtenue en exploitant les données MApUCE. A noter enfin que l'un des objectifs du projet est de pousser la réflexion concernant l'intégration de données climatiques et environnementales dans les documents d'urbanisme et que la question des usages de la méthode d'interpolation rentre dans ce cadre.

Quoi qu'il en soit, il apparaît clair que le principal public susceptible d'avoir recours à cette méthode se trouve dans les collectivités territoriales et/ou les bureaux d'études, qui sont à la fois les plus directement concernés par les usages possibles et les plus à même d'avoir accès aux données nécessaires. Ceci implique néanmoins un usage par un public potentiellement non-spécialiste d'une méthode qui demeure relativement complexe à mettre en œuvre manuellement. D'où l'intérêt de s'orienter vers une automatisation.

**PARTIE II : Automatisation de la
méthode : de la chaîne de traitement au
plug-in QGIS**

Partie II : Automatisation de la méthode : de la chaîne de traitement au plug-in QGIS

1. Intérêt d'une automatisation de la méthode

a. Principes et outils de l'automatisation

Selon l'une des définitions proposées par le dictionnaire Larousse, l'automatisation consiste en « l'exécution totale ou partielle de tâches techniques par des machines fonctionnant sans intervention humaine ». Dans le cadre de la géomatique, les « tâches techniques » sont souvent un ensemble de traitements sur un SIG, et les « machines » sont généralement des ordinateurs.

L'automatisation fait partie des tâches récurrentes qui peuvent être demandées à un géomaticien, et ce dans des contextes très variés. D'une manière générale, il s'agit de simplifier et accélérer un traitement ou un ensemble de traitements plus ou moins complexe(s), de sorte à en faciliter l'utilisation. Une recherche rapide suffit à mettre en exergue différents exemples d'applications : automatisation d'un bilan hydrique sous ArcGIS²⁶, automatisation des inventaires de sites hydroélectriques en SIG²⁷, automatisation de chaînes de traitement GRASS pour la télédétection²⁸...les champs thématiques sont diversifiés, ce qui pose également la question des outils à mobiliser.

Ces derniers peuvent être des logiciels tels que FME (développé par Veremes) mais prennent souvent la forme de langages de programmation, le python étant souvent utilisé lorsque l'automatisation concerne des SIG tels que QGIS ou ArcGis. L'automatisation peut ainsi prendre des formes variées, qui dépendent évidemment des traitements à automatiser mais aussi des publics ciblés. Nous n'opérerons pas les mêmes choix selon qu'il s'agit de faciliter l'exécution de traitements répétitifs pour son usage personnel, d'automatiser une chaîne de traitements pour accélérer le travail d'un géomaticien dans une entreprise, ou encore, de permettre aux employés d'une collectivité territoriale d'effectuer des traitements relativement complexes sans connaissances particulières en géomatique.

Les deux premiers cas peuvent par exemple se satisfaire d'un model builder ou d'un script python, échangeables « en main propre » (via clé USB, disque dur externe...) mais nécessitant souvent un minimum de connaissance du logiciel concerné pour être installés/utilisés. Le troisième exemple imaginé, pourrait plutôt être traité par un plug-in, nettement plus complexe à développer mais pouvant se révéler bien plus souple d'utilisation.

²⁶ LAMY C., DUBREUIL V., *Impact du changement climatique sur les sécheresses en Bretagne. Automatisation d'un bilan hydrique avec ArcGis et Python*, Revue internationale de géomatique, vol 24/3, 2014

²⁷ Offre d'emploi publiée sur le site Wizbbii : <https://www.wizbbii.com/company/electrabel-gdf/job/stage-automatisation-des-inventaires-de-sites-hydroelectriques-sous-sig>, consulté le 21 mai 2017

²⁸ Offre de stage du CNRS : <https://cnrsformation.cnrs.fr/stage-17261-Automatisation-des-traitements-geomatiques-avec-le-SIG-GRASS%AO-application-a-la-teledetection-et-aux-enquetes-de-terrain.html>, consulté le 21 mai 2017

Dans le cas de la méthode géostatistique explicitée en I, deux principaux logiciels sont impliqués : SAGA GIS et QGIS. Les publics concernés peuvent être divers et il convient donc de bien définir les objectifs ciblés et les moyens à mettre en œuvre avant d'entamer l'automatisation proprement dite.

b. Cahier des charges pour l'automatisation de la méthode de modélisation des îlots de chaleur urbain

La question des outils mobilisables est ici réduite par le choix d'utiliser des outils libres et gratuits. Ces derniers – QGIS et SAGA GIS - étaient déjà le support de la conception de la méthode et seront donc logiquement celui de son automatisation. Si des traitements de SAGA GIS sont indispensables, ce logiciel reste à la fois moins répandu que QGIS, moins accessible et moins pratique lorsqu'il s'agit d'automatiser des traitements. QGIS incluant en outre un certain nombre de traitements issus de SAGA, c'est sur QGIS que sera réalisée l'automatisation. Celle-ci prendra la forme d'un plug-in, qui semble en effet mieux adapté aux publics ciblés. Un « cahier des charges » a ainsi été établi afin d'assurer la cohérence entre les objectifs visés et les moyens mis en œuvre.

i) Les publics ciblés

Le plug-in visant à modéliser l'îlot de chaleur urbain sur un territoire déterminé semble avant tout destiné au domaine public, et en particulier aux collectivités locales. L'intérêt premier de spatialiser un ICU est en effet de pouvoir intégrer cette information dans les politiques d'aménagement – ce qui rejoint en outre les préoccupations du projet MApUCE.

On peut néanmoins imaginer une utilisation dans le cadre privé, notamment par des bureaux d'étude en aménagement/urbanisme, travaillant, par exemple, sur l'élaboration de documents d'urbanisme tels que les PLU, SCOT, etc...

Enfin, on peut envisager un usage dans le cadre de la recherche sur les thématiques du climat urbain.

Dans tous les cas, le public ciblé n'est pas nécessairement spécialiste ni en S.I.G, ni en géostatistique ou en climatologie. Il faut donc définir des objectifs adaptés pour le plug-in.

ii) Les objectifs

- Etre d'un usage simple, adapté à un public non-spécialiste.
- Etre utilisable avec un minimum de données, accessibles notamment pour les collectivités locales.
- Produire des fichiers de sortie assez simples à interpréter, et bien documentés.
- Ne pas se cantonner aux données MApUCE et pouvoir intégrer des données propres aux différents usagers, notamment sur l'occupation des sols (fractions de ville)
- Etre transparent sur la méthode utilisée et permettre de mesurer la qualité des résultats produits.
-

iii) Les données d'entrées

Au minimum, les données d'entrée devront être au nombre de trois : une couche de points correspondant aux relevés de températures, un raster d'altitude (MNT) et un raster de fraction de ville.

Peu de variations semblent possibles concernant les deux premières couches. En revanche, on peut imaginer que la fraction de ville puisse être produite de différentes façons, en fonction des besoins et possibilités de l'utilisateur :

- Directement à partir des données MApUCE. Si celles-ci sont disponibles pour le territoire concerné, il peut s'agir de la méthode la plus simple : l'utilisateur indique la couche vecteur MApUCE et la fraction de ville est produite et utilisée de façon totalement transparente pour l'usager. Eventuellement, on peut envisager de laisser à l'utilisateur le choix d'intégrer ou non certaines variables dans la fraction de ville (fraction 2D ou 3D, etc ;..)
- A partir des données dont dispose l'utilisateur. Le plug-in pourrait ainsi permettre à l'usager d'utiliser une fraction de ville qu'il aurait lui-même produite. Ceci afin de ne pas limiter l'usage du plug-in aux territoires couverts par les données MApUCE.

iv) Les fichiers en sortie

Au minimum, la sortie devra être le raster représentant l'ICU. On peut néanmoins envisager d'y intégrer différentes couches intermédiaires : dans le cas où la fraction de ville est produite à partir des données MApUCE par le plug-in, le raster correspondant pourrait se trouver en sortie.

On peut également envisager de sortir le résultat de la régression linéaire et des résidus krigés. L'intérêt paraît limité pour un usage en collectivité, mais est peut-être intéressant dans le domaine de la recherche.

Enfin, les indicateurs de la régression tels que le R^2 ajusté peuvent être ajoutés aux couches de sortie, par souci de transparence vis-à-vis de l'utilisateur.

D'une manière générale, on peut envisager que le fait de sortir ou non telle ou telle couche soit laissé au choix de l'utilisateur.

v) Les paramètres utilisateurs

Il est possible de laisser l'utilisateur configurer certains paramètres de la régression. Cela semble même nécessaire pour rendre le plug-in adaptable à toutes les situations.

Pour ne pas complexifier inutilement l'usage du plug-in, des paramètres par défaut convenant à la majorité des situations seront suggérés. On peut néanmoins laisser la main à l'utilisateur sur des paramètres tels que le search radius (global ou local avec distance choisie) et le nombre de points à inclure dans l'analyse.

La résolution pourrait aussi être laissée à l'appréciation de l'utilisateur, pour lui permettre éventuellement de profiter de fractions de ville « maison » à des résolutions plus fines que 100m.

Ces notions définies, il est désormais possible d'entamer l'automatisation proprement dite.

2. Le script QGIS : une étape intermédiaire

Avant d'aller vers la conception d'un plug-in – développement final de l'automatisation – le choix a été fait de réaliser un script QGIS effectuant l'essentiel des tâches à automatiser. Ce choix se base sur deux principaux arguments :

- Les scripts QGIS peuvent être développés en python et le script écrit lors de cette étape pourra donc être directement réutilisé lors du développement du plug-in, moyennant quelques ajustements.
- L'éditeur de script de QGIS est plutôt simple d'utilisation et permet notamment de tester directement son script sans passer par la phase de conception d'une interface par exemple.

L'écriture d'un script QGIS se pose donc comme une étape intermédiaire avant le développement d'un plug-in, permettant de fixer au préalable l'essentiel du code. Les différents choix de conception et les étapes composant le script vont donc être détaillés ci-dessous.

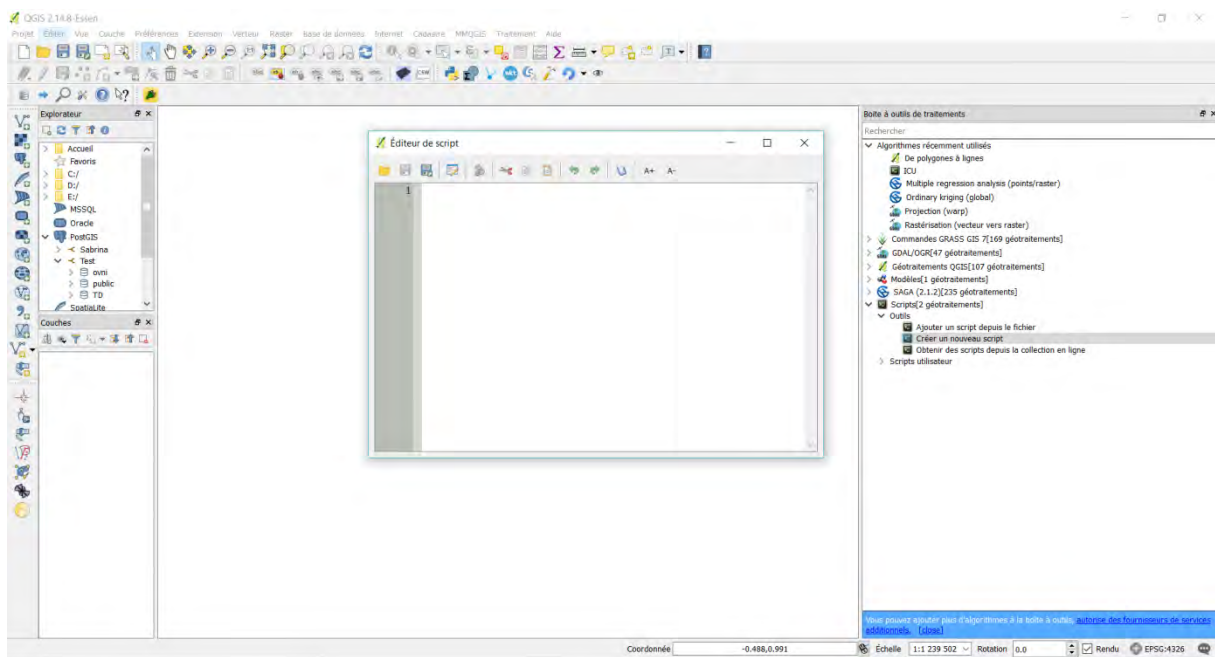


Figure 33 : L'éditeur de scripts de QGIS 2.14 (capture d'écran)

a. Structure générale du script

Les schémas de la page suivante décrivent les principales étapes par lesquelles passe le script pour finalement réaliser la méthode détaillée précédemment. A noter que le choix a été fait d'utiliser indépendamment et successivement les algorithmes de SAGA pour la régression linéaire multiple puis celui pour le krigeage et d'ajouter les résultats plutôt que d'utiliser directement l'algorithme kriging-regression. Si nous avons vu dans le II de la partie 1 que les résultats étaient identiques, il s'avère en effet que l'algorithme kriging-regression de SAGA n'est actuellement pas intégré à QGIS, ce qui a pour effet d'en complexifier l'automatisation.

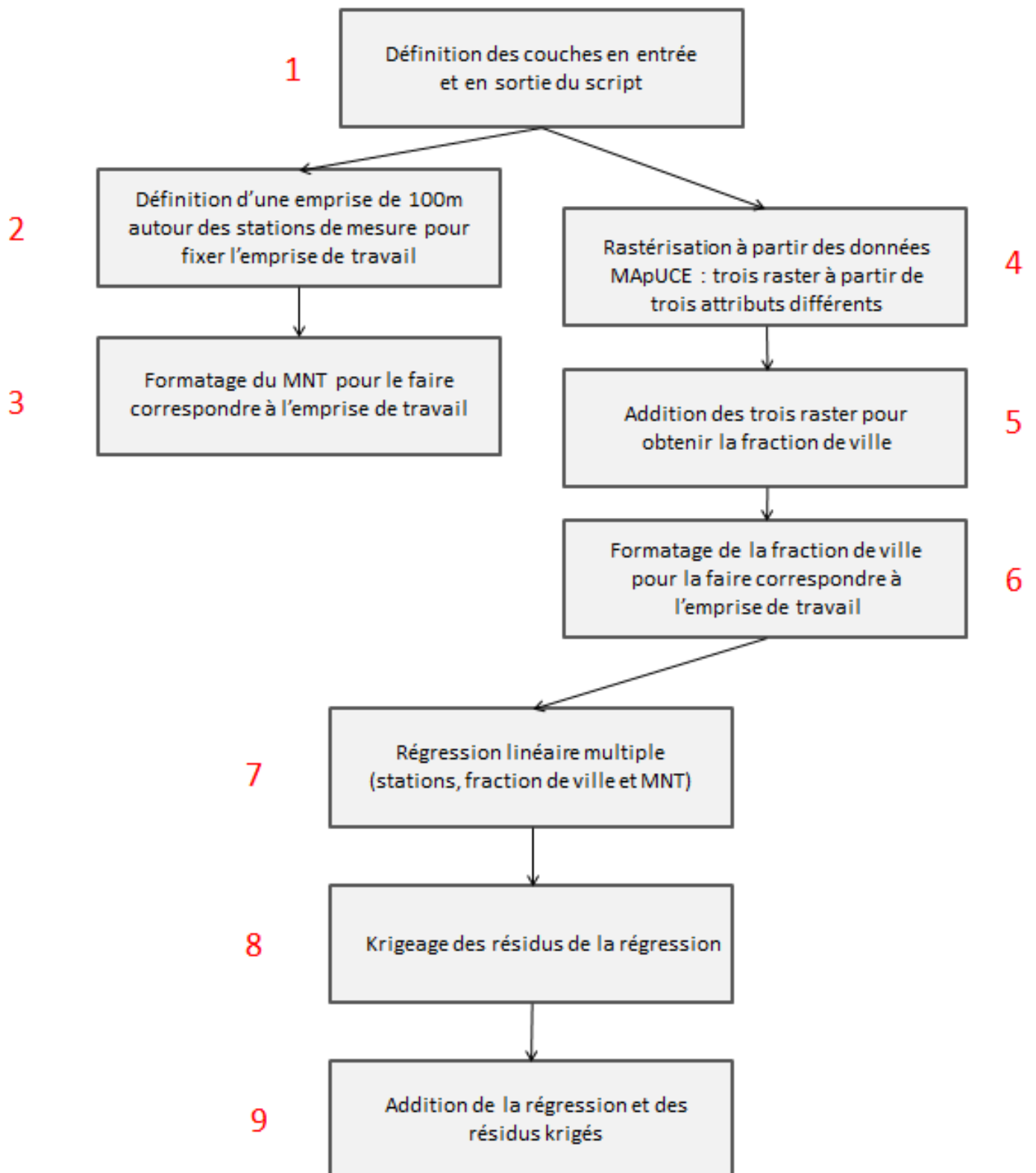


Figure 34 : Ordre et description des principales étapes réalisées au cours du déroulement du script.
Réalisation : Thomas Gardes, 2017

Le déroulement général étant posé, nous allons maintenant expliciter les principaux points techniques mis en œuvre en python pour réaliser ces différentes étapes.

b. Définir les inputs et les outputs

Inputs et outputs peuvent être définis de façon très simple via l'éditeur de script, en utilisant le symbole `##`. Nous avons d'ores et déjà réfléchi aux données nécessaires en entrées et à celles que l'on souhaite proposer en sortie, et nous connaissons leur type (vecteur, raster). On définit donc des noms de variables précédés de `##`, auxquels on associe un type, en précisant « output » s'il doit s'agir d'un output. L'éditeur de script produit alors automatiquement une interface, comme le montre l'image ci-dessous montrant la correspondance entre le code utilisé et la fenêtre produite :

```
6
7 ##ICU=name
8
9 # Inputs
10
11 ##mnt=raster
12 ##stations=vector
13 ##fieldTemp=field stations
14 ##mapuce=vector
15
16 #Outputs
17 ##mnt_clip=output raster
18 ##regression=output raster
19 ##frac_ville=output raster
20 ##lr_model=output table
21 ##lr_residuals=output vector
22 ##residuals_kriging=output raster
```

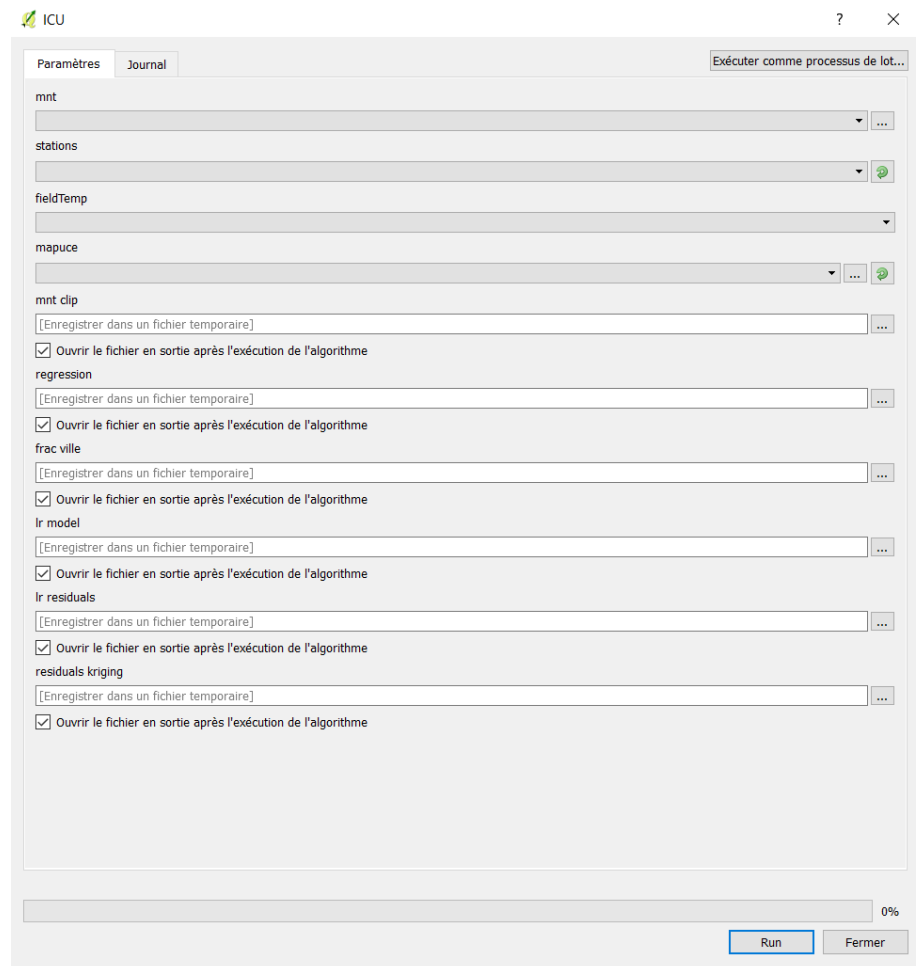


Figure 35 : Définition des inputs et outputs via l'éditeur de script de QGIS, en python (capture d'écran)

La description complète des types d'inputs et outputs possibles est disponible via la documentation de QGIS (https://docs.qgis.org/2.6/fr/docs/user_manual/processing/console.html)

Les variables définies comme données d'entrées peuvent maintenant être utilisées dans les différents algorithmes mobilisés pour le traitement.

Attention toutefois, si l'interface produite automatiquement par l'éditeur de script s'avère très pratique pour le test et la conception du script, ce code ne fonctionne pas dans le cadre d'un plug-in, comme nous le verrons par la suite.

c. Appeler les traitements de QGIS en python : la bibliothèque « processing »

Nous l'avons vu dans le I), la méthode à automatiser fait notamment appel à des algorithmes de QGIS et SAGA GIS. Dans le cadre d'un script python, les différents algorithmes de QGIS peuvent être appelés assez simplement, via la commande `processing.runalg()`. Celle-ci prend en paramètres le nom de l'algorithme à utiliser et les différents paramètres inhérents à celui-ci.

Exemple : nous avons défini à l'étape précédente une couche d'entrée « stations », de type vecteur (qui seront en l'occurrence des points représentant les stations de mesures de températures). L'un des premiers traitements consiste à définir une emprise de 100m autour de ces stations, emprise qui sera utilisée dans le reste du script. Ceci peut être fait de la façon suivante :

```
import processing

emprise=processing.runalg("qgis:fixeddistancebuffer", stations, 100,5,1,
None) #buffer de 100m autour de la couche stations
```

L'algorithme « buffer » est appelé par le nom `qgis:fixeddistancebuffer` (str). Les paramètres suivants correspondent respectivement à la couche cible (ici, la couche vectorielle définie précédemment via la variable `stations`), la taille du buffer (int), le nombre de segments (int), le fait de dissoudre ou non les buffers (bool) et enfin, le fichier de sortie.

Il est possible d'obtenir les détails de n'importe quel algorithme en tapant dans la console python de QGIS : `processing.alghelp(« nomdelalgorithme »)`. La liste complète des algorithmes disponibles peut être affichée via la commande `processing.alglist()`. Elle inclut des traitements issus de GRASS, SAGA ou GDAL.

On remarquera que, dans l'exemple précédent, le paramètre correspondant à l'output est défini par « None ». Ceci permet de gérer les couches que nous ne souhaitons pas sortir au final comme des temporaires afin de les utiliser ultérieurement dans le script. On peut s'y référer en indiquant le nom de la variable contenant le traitement (ici, `emprise`) suivi du nom du paramètre entre crochets (ici `['OUTPUT']`) mais les algorithmes de SAGA utilisent par exemple le terme `USER_GRID` pour désigner la même chose). Pour reprendre l'exemple précédent, si nous souhaitions à présent appliquer sur les buffers produits un autre buffer négatif (pour une dilatation/érosion par exemple), nous devrions alors écrire :

```
erosion=processing.runalg("qgis:fixeddistancebuffer",emprise['OUTPUT'],
-50,5,1, None) #buffer de -50m autour de la couche en sortie de la variable
emprise
```

Ce mode de fonctionnement est relativement simple à mettre en œuvre une fois qu'il a été assimilé et permet de traduire des chaînes de traitements assez complexes. Ce n'est cependant généralement pas suffisant pour réaliser un script entier et d'autres problèmes peuvent se poser assez vite.

d. Manipuler des « objets QGIS » en python

Si nous effectuons un « print » de la variable `emprise['OUTPUT']`, nous n'obtenons pas une couche vecteur (`QgsVectorLayer`) mais son chemin d'accès. Or, nous souhaitons à présent récupérer l'emprise de la couche produite (le buffer de 100m autour des stations) pour la définir comme emprise des prochains rasters à produire. Ceci permet d'adapter le script à toutes sortes de données (quel que soit le nombre de stations de mesure de température et leur disposition, l'interpolation se fera toujours dans l'emprise de ces stations avec une marge de 100m) mais ne peut pas se faire directement sur un chemin d'accès. Nous avons donc recours à la solution suivante :

```
objEmp = processing.getObject(emprise['OUTPUT']) #recuperer buffer stations
sous forme de QgsVectorLayer
```

Un print de `objEmp` indique maintenant :

```
15 <qgis._core.QgsVectorLayer object at 0x00000000199F0AE8>
```

Nous avons récupéré non pas le chemin d'accès vers la couche (layer) mais le layer lui-même, sur lequel il est désormais possible de récupérer l'emprise à l'aide de la méthode `extent()` :

```
rectangle= objEmp.extent() #recuperer l emprise de la couche buffer
```

A noter que, pour pouvoir être utilisée dans les algorithmes de GDAL – ce que nous devons faire par la suite – cette emprise doit être obtenue sous la forme d'une chaîne de caractère (str, pour « string ») dont l'ordre doit être modifié pour correspondre à celui de GDAL, qui diffère de celui de QGIS (l'emprise est définie par des coordonnées Xmin Xmax / Ymin Ymax qui ne sont pas données dans le même ordre dans un logiciel ou dans l'autre). Ceci est réalisé via des fonctions python classiques pour la manipulation des chaînes de caractères :

```
coordinates = rectangle.toString(16) #convertir emprise sous forme de
chaîne de caractere
replace = coordinates.replace(":",",",1) #formater la chaîne pour
utilisation dans la fonction clip raster by extent
split = replace.split(",") #split de la chaîne pour re-composition...
extent= split[0]+","+split[2]+","+split[1]+","+split[3] #...concatenation
pour que l ordre des coordonnes decrivant l emprise corresponde a l ordre
utilise dans clip raster by extent
```

Le recours à la fonction `processing.getObject()` se fait ainsi plusieurs fois au cours du script, notamment lorsque l'on souhaite effectuer des opérations sur des couches temporaires issues de traitements appelées via `processing.runalg()`. C'est le cas lorsque l'on désire utiliser la calculatrice raster.

e. Usage de la calculatrice raster de QGIS en python

La calculatrice raster de QGIS ne peut être appelée via `processing.runalg()`. En revanche, dans les versions 2.xx du logiciel, elle fait l'objet de classes pouvant être importées depuis la bibliothèque `qgis.analysis` :

```
from qgis.analysis import QgsRasterCalculator, QgsRasterCalculatorEntry
```

Le script fait usage de la calculatrice raster à deux reprises : lorsqu'il s'agit de calculer la fraction de ville à partir d'attributs des données MApUCE préalablement rasterisés (addition de 3 rasters) et au moment d'additionner les résultats de la régression linéaire multiple avec ceux du krigeage de ses résidus. C'est le premier cas qui sert d'exemple ici.

Nous définissons tout d'abord une liste, qui va correspondre aux inputs de la calculatrice :

```
entries=[] #initialisation de la liste des entrees
```

Puis nous définissons les rasters à additionner :

```
rast1=QgsRasterCalculatorEntry() #creation premier raster

rast1.ref = 'rast@1' #definir l appellation du premier raster
objBuild=processing.getObject (build['OUTPUT']) #recuperer raster sous forme
d objet Qgis
rast1.raster=objBuild #definir le raster
rast1.bandNumber=1 #definir la bande du raster a utiliser

entries.append(rast1) #ajout du raster aux entrees
```

Le premier raster (qui correspond à l'output de la variable `build` définie plus haut dans le script) est ainsi défini et ajouté à la liste d'inputs (`entries`). La même démarche est suivie pour les deux rasters suivants, appelés fort à propos `rast2` et `rast3`.

Les inputs sont prêts et stockés sous forme de liste dans la variable `entries`. Le calcul proprement dit est paramétré puis effectué par les lignes suivantes :

```
#parametrage de la calculatrice raster
calc=QgsRasterCalculator('rast@1+rast@2+rast@3', outputCalc , 'GTiff',
objBuild.extent(), objBuild.width(), objBuild.height(),entries)

#execution du calcul
calc.processCalculation()
```

'`rast@1+rast@2+rast@3`' fait référence au calcul à effectuer sur les différents raster – ici, une simple addition. `outputCalc` est un chemin d'accès pour l'output défini préalablement dans le script. `GTiff` correspond au format de l'image à produire. Les trois paramètres suivants donnent respectivement l'emprise, la largeur et la hauteur de la sortie. Elles sont ici calquées sur celle d'un des raster en entrée (les trois ayant au préalable été dimensionnés à l'identique) et récupérées dans la variable `objBuild`. `entries` correspond enfin à la liste de raster que nous avons définie.

La commande `calc.processCalculation()` permet enfin d'exécuter le calcul défini dans la variable `calc`.

Nous avons ainsi effectué un tour d'horizon des principales solutions techniques déployées pour la réalisation de la chaîne de traitement en python. Pour des raisons de lisibilité, il ne semble pas pertinent de développer ici l'ensemble du script. Néanmoins, l'essentiel des bibliothèques, fonctions, méthodes employées aux différentes étapes se trouve résumé dans le tableau suivant.

Un model-builder complet détaille également les différents traitements exécutés dans le script.

Etape n°	Etape	Bibliothèques / classes importées	Principales méthodes utilisées	Objectif(s)
1	Définition des inputs et outputs	-	Définition des inputs et outputs selon la méthode propre à l'éditeur de script (##)	Définition des inputs et outputs
2	Définition de l'emprise de travail	Processing ; Qgis.core	processing.runalg ; processing.getObject ; fonctions de manipulation de chaînes de caractères (split, replace...)	Définir un buffer de 100m et récupérer l'objet vectoriel en sortie pour en extraire l'emprise et la formater en vue d'un usage dans les algorithmes suivants
3	Formatage du MNT par rapport à l'emprise de travail	Processing	processing.runalg	Usage des algorithmes SAGA et QGIS nécessaire au découpage et ré-échantillonnage du MNT
4	Rastérisation à partir des données MAPUCE	Processing	Processing.runalg	Appel des algorithmes GDAL de rastérisation
5	Addition de raster pour former la fraction de ville	qgis.analysis : QgsRasterCalculator, QgsRasterCalculatorEntry ; processing	processing.getObject ; QgsVectorLayer.dataProvider(). dataSourceUri ; processCalculation	Définition des inputs et outputs de la calculatrice raster et exécution du calcul de la fraction de ville
6	Formatage de la fraction de ville par rapport à l'emprise de travail	Processing	processing.runalg	Usage des algorithmes SAGA et QGIS nécessaire au découpage et ré-échantillonnage de la fraction de ville
7	Régression linéaire multiple	Processing	processing.getObject ; processing.runalg ; QgsVectorLayer.dataProvider(). dataSourceUri ;	Formatage des inputs et exécution de la régression linéaire par l'appel de l'algorithme SAGA
8	Krigeage des résidus	Processing	processing.getObject ; processing.runalg ; QgsVectorLayer.dataProvider(). dataSourceUri ;	Formatage des inputs et exécution du krigeage par l'appel de l'algorithme SAGA
9	Addition des résultats de la régression et du krigeage	qgis.analysis : QgsRasterCalculator, QgsRasterCalculatorEntry ; processing	processing.getObject ; QgsVectorLayer.dataProvider(). dataSourceUri ; processCalculation	Définition des inputs et outputs de la calculatrice raster et exécution du calcul final

Figure 36 : Principales classes et méthodes utilisées dans la conception du script sous l'éditeur de QGIS

Réalisation : Thomas Gardes, 2017

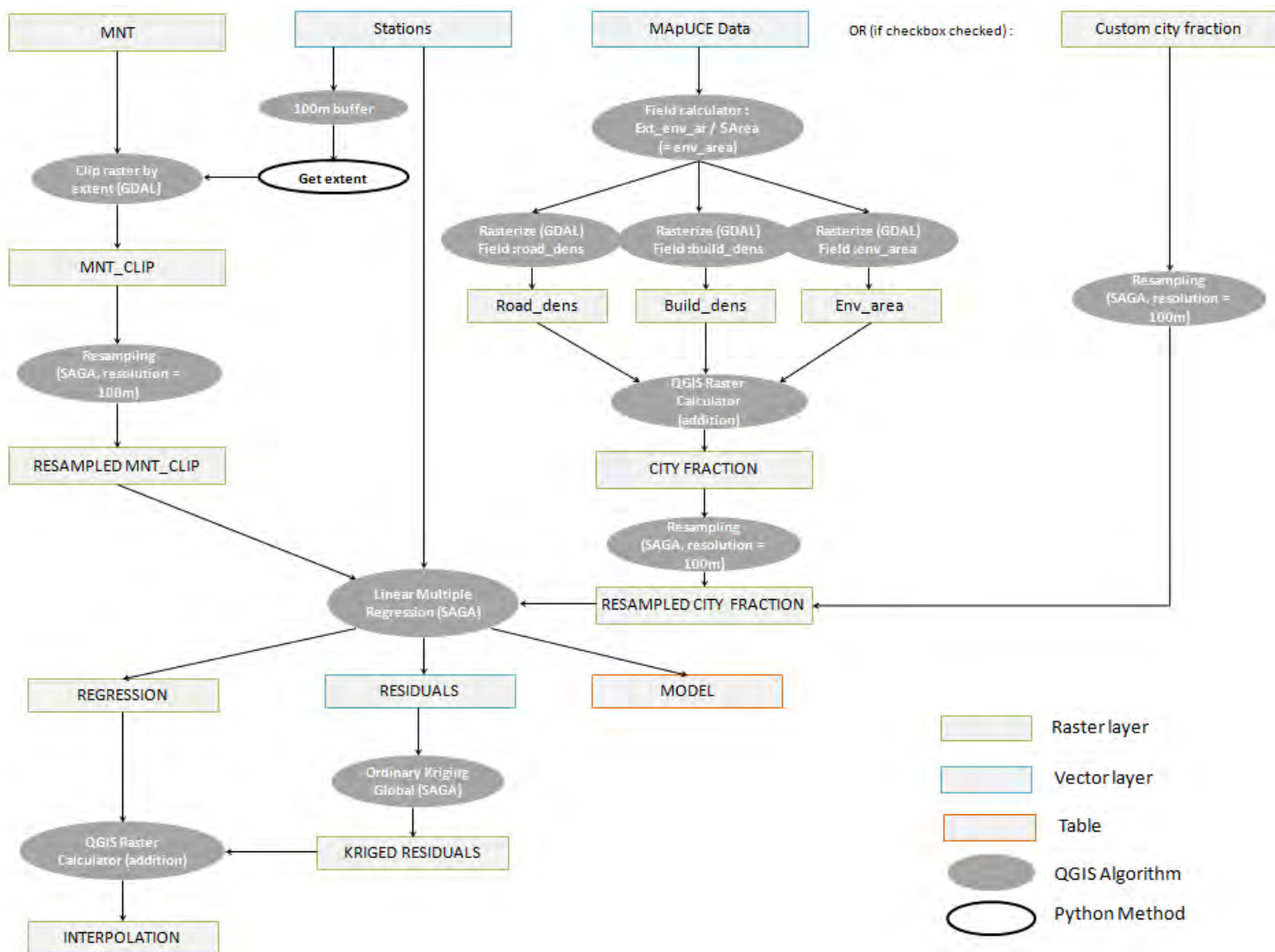


Figure 37 : Model-builder du script QGIS
Réalisation : Thomas Gardes, 2017

3. Du script au plug-in QGIS

a. Quels outils pour quels ajouts ?

Si le script est fonctionnel et permet le calcul de l'îlot de chaleur urbain avec le minimum de données d'entrées que nous avons évoqué dans le cahier des charges, il reste un outil plutôt confidentiel (la transmission devant plutôt s'effectuer en « mains propres » depuis un périphérique de stockage par exemple), peu documenté et muni d'une interface plutôt sommaire. Pour correspondre à des usages et des publics plus vastes, des ajustements sont donc nécessaires, d'où l'intérêt du développement d'un plug-in QGIS.

Qu'est-ce qu'un plug-in ?

Si le terme de « plug-in » ne semble pas (encore ?) faire l'objet d'une définition officielle dans les dictionnaires de langue française, on peut néanmoins trouver des explications sur de nombreux sites grands publics. Littéralement traduit *depuis* l'anglais, le terme pourrait devenir « *brancher dedans* ». Le site web l'internaute le définit ainsi : « *Un plug-in est un outil composé d'un ensemble de fichiers informatiques et qui permet d'installer des nouvelles fonctionnalités en marge d'un logiciel auquel il est rattaché. On parle parfois de module d'extension ou de « plugiciel » pour le désigner* ». ²⁹ Cette définition explicite plutôt bien ce dont on va parler ici. Notre plug-in pour QGIS vise en effet à ajouter une fonction au logiciel – en l'occurrence l'estimation d'un îlot de chaleur urbain selon la méthode développée précédemment. QGIS, en tant que logiciel libre (i.e dont le code source est librement accessible) facilite en outre le développement de ce type d'extension. Il intègre notamment un « dépôt officiel » d'extensions permettant leur téléchargement et leur installation directement depuis l'interface du logiciel

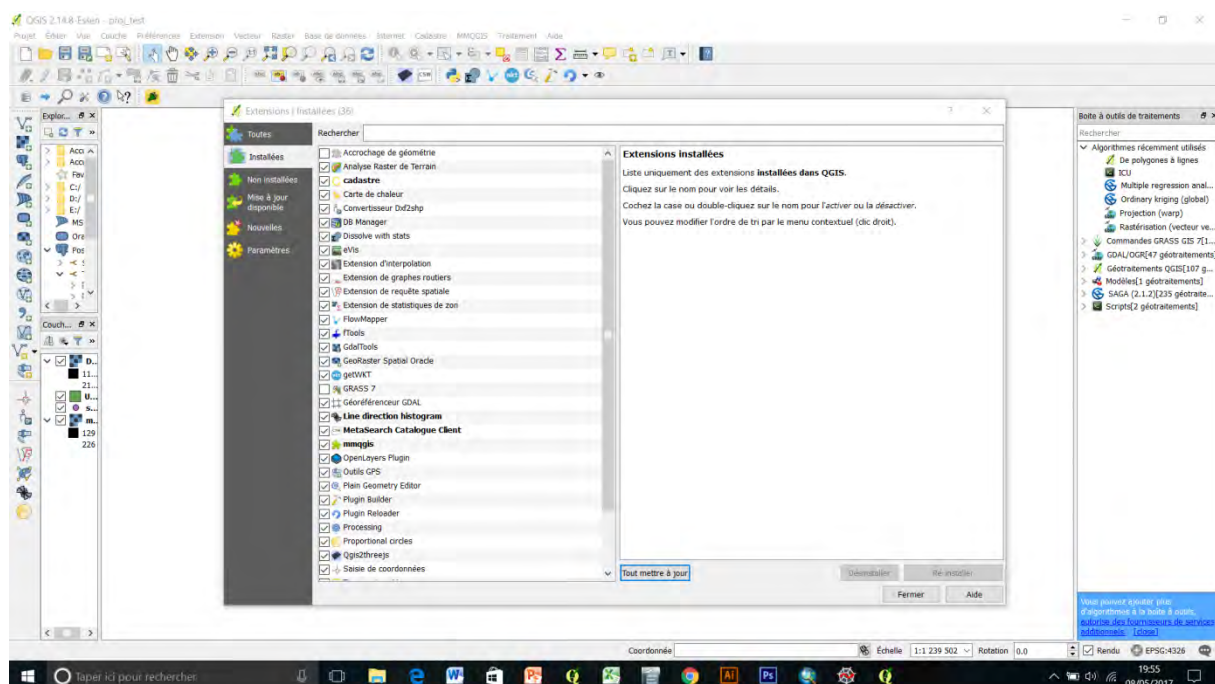


Figure 38 : L'interface de téléchargement et d'installation des plug-in QGIS (capture d'écran)

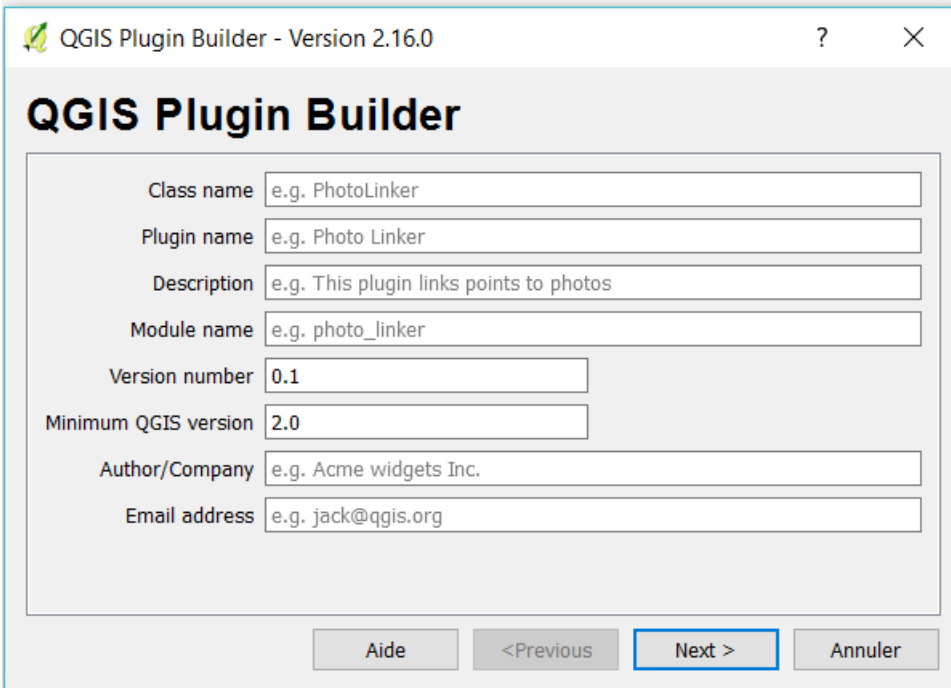
²⁹ L'internaute : <http://www.linternaute.com/dictionnaire/fr/definition/plugin/>, consulté le 2 avril 2017

Une fois conçu et développé, notre plug-in pourra donc éventuellement être ajouté à ce dépôt officiel.

b. Développer un plug-in QGIS : les outils

Le développement d'un plug-in pour QGIS peut mobiliser un certain nombre d'outils (logiciels, sites web...), en fonction des objectifs visés et des habitudes/préférences des développeurs. Le développement du plug-in présenté ici ne fait pas exception à la règle. L'essentiel du développement se faisant en langage python, un éditeur permettant l'usage de ce langage est donc une base incontournable. Si plusieurs IDE python (environnements de développements) peuvent être envisagés (Wing IDE, Sublime Text...), le développement s'est ici fait tout simplement via le NotePad++, éditeur de texte bien connu intégrant une coloration syntaxique et un ensemble de fonctionnalités de bases jugées ici suffisantes.

La conception du plug-in à proprement parler s'est ensuite basée sur un autre plug-in, appelé « Plug-in Builder ». Ce « plug-in pour créer des plug-in » est disponible dans le dépôt officiel des extensions QGIS, avec la mention expérimentale. A partir d'un formulaire à remplir, il permet de générer un ensemble de fichiers de base essentiels au bon fonctionnement d'un plug-in.



The screenshot shows the 'QGIS Plugin Builder' window. The title bar reads 'QGIS Plugin Builder - Version 2.16.0'. The main content area has a heading 'QGIS Plugin Builder' and a form with the following fields and values:

Class name	e.g. PhotoLinker
Plugin name	e.g. Photo Linker
Description	e.g. This plugin links points to photos
Module name	e.g. photo_linker
Version number	0.1
Minimum QGIS version	2.0
Author/Company	e.g. Acme widgets Inc.
Email address	e.g. jack@qgis.org

At the bottom of the form, there are four buttons: 'Aide', '<Previous', 'Next >', and 'Annuler'.

Figure 39 : Extrait de l'interface du "Plug-in Builder" (capture d'écran)

Nous reviendrons plus en détail sur l'usage de ce plug-in dans une partie ultérieure. Un autre plug-in indispensable a également été utilisé tout au long du développement : « plug-in reloader ». Celui-ci, utile exclusivement à des fins de développement, sert à recharger un plug-in préalablement choisi en un seul clic, ce qui permet de prendre directement en compte des modifications apportées dans le code sans avoir à fermer puis relancer QGIS.

Enfin, tout plug-in intégrant une interface qui lui est spécifique, celle utilisée ici a été conçue à l'aide du logiciel Qt Designer. Ce dernier propose une interface graphique justement dédiée au développement d'interfaces / fenêtres qui semblent ici tout à fait adaptées à nos besoins et produit des fichiers .ui compilables en python et utilisables dans un plug-in QGIS.

« Plug-in Builder », NotePad++ et Qt Designer sont donc les principaux logiciels qui ont été mobilisés aux différents stades de développement du plug-in.

A noter qu'un compte a également été créé sur le site web Git Hub, dédié au stockage et au partage de code, et qui intègre des espaces dédiés à QGIS. Un espace a donc été créé pour le plug-in, sur lequel ont été uploadés, à rythmes réguliers, les fichiers composant le plug-in à ces différentes étapes, notamment dans un but de sauvegarde.

c. Les étapes du développement

Création d'un template via « plug-in builder »

Une fois le script QGIS rendu fonctionnel (cf partie précédente), la première phase du passage vers le plug-in s'est fait via le plug-in « Plug-in Builder ».

Ce dernier demande d'abord un ensemble d'informations, telles qu'un nom de plug-in, un nom de classe principale (pour les fichiers python), un nom et un contact de développeur...

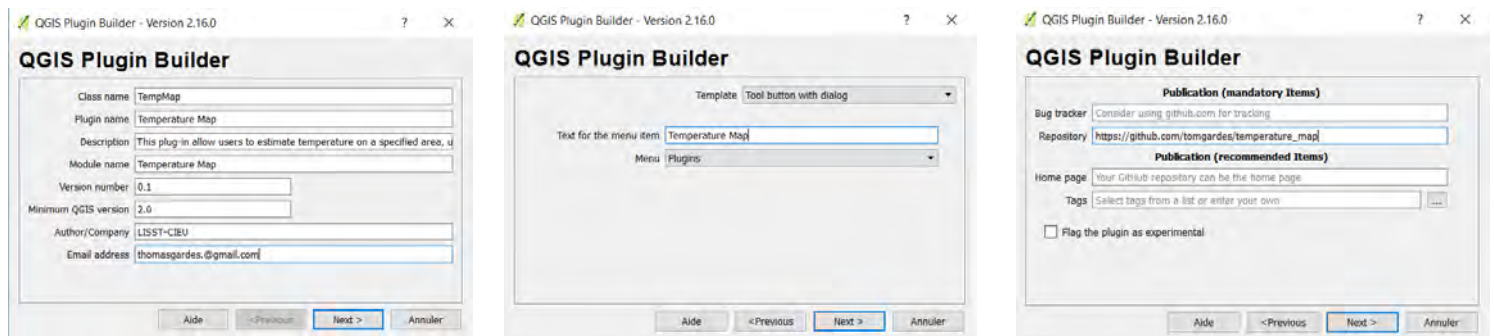


Figure 40 : Exemples de fenêtres issues de "Plug-in Builder" permettant de générer les fichiers de base pour le développement d'un plug-in (capture d'écran)

Dans un premier temps, c'est le nom « Temperature Map » qui a été retenu (si le plug-in vise plutôt à modéliser un îlot de chaleur urbain, il est en effet plus exact de qualifier son rendu en sortie de « carte de température »), abrégé en « TempMap » dans le nom de classe qui sera utilisé dans le principal fichier python. Ces étapes remplies, le plug-in génère en effet automatiquement un ensemble de fichiers, véritable template à personnaliser ensuite en fonction de ses besoins.

Ces fichiers sont visibles dans l'illustration de la page suivante et peuvent être résumés comme suit :

- Un fichier .py « init » contenant une unique classe « classFactory » permettant notamment l'intégration du plug-in à QGIS
- Un fichier .png « icon » contenant une icône par défaut à afficher dans l'interface de QGIS
- Un fichier de ressources .py

- Un fichier .py intitulé temp_map (en fonction du nom spécifié précédemment) et contenant les classes principales qui permettront notamment au plug-in d'accomplir les tâches pour lesquelles il est conçu.
- Un fichier .py « temp_map_dialog » permettant notamment la génération d'une interface
- Un fichier .ui « temp_map_dialog_base » à remplacer ensuite par un .ui contenant l'interface réalisée sous Qt Designer.

Nom	Modifié le	Type	Taille
help	30/04/2017 10:33	Dossier de fichiers	
i18n	30/04/2017 10:33	Dossier de fichiers	
scripts	30/04/2017 10:33	Dossier de fichiers	
test	30/04/2017 10:33	Dossier de fichiers	
init.py	22/04/2017 16:51	Python File	2 Ko
init.pyc	24/04/2017 09:23	Compiled Python ...	2 Ko
help_icon_25px.png	27/04/2017 16:16	Fichier PNG	2 Ko
icon.png	13/02/2017 17:46	Fichier PNG	2 Ko
Makefile	22/04/2017 16:51	Fichier	8 Ko
metadata.txt	22/04/2017 16:51	Document texte	1 Ko
pb_tool.cfg	22/04/2017 16:51	Fichier CFG	3 Ko
plugin_upload.py	13/02/2017 17:46	Python File	4 Ko
pylintrc	13/02/2017 17:46	Fichier	9 Ko
README.html	22/04/2017 16:51	Chrome HTML Do...	2 Ko
README.txt	22/04/2017 16:51	Document texte	1 Ko
resources.py	22/04/2017 16:57	Python File	6 Ko
resources.pyc	24/04/2017 09:23	Compiled Python ...	2 Ko
resources.qrc	22/04/2017 16:51	Fichier QRC	1 Ko
temp_map.py	30/04/2017 10:43	Python File	21 Ko
temp_map.pyc	30/04/2017 11:16	Compiled Python ...	15 Ko
temp_map_dialog.py	24/04/2017 15:08	Python File	2 Ko
temp_map_dialog.pyc	28/04/2017 11:24	Compiled Python ...	2 Ko
temp_map_dialog_base.ui	28/04/2017 11:48	Fichier UI	16 Ko

Figure 41 : Les fichiers générés par Plug-in Builder (capture d'écran)

Le rôle de « Plug-in Builder » est désormais terminé mais il reste néanmoins une étape à réaliser qui lui est directement liée : la compilation. Cette dernière est, d'une manière générale en informatique, une étape permettant de « traduire » un code écrit dans un langage de haut niveau – et donc assez facilement compréhensible pour un individu – en un code directement fonctionnel et exécutable par la machine.

Ici, la compilation va concerner deux fichiers : le fichier de ressources et le fichier .ui. La compilation est exécutée assez simplement en ligne de commande, ici à partir du shell de l'OSGeo4W.

On accède au répertoire contenant le plug-in à l'aide de la commande « cd » suivie du chemin d'accès. Puis on utilise la commande « pyrcc4 » pour compiler le fichier de ressources (fichier .rc) et « pyuic4 » pour le fichier .ui. On la fait suivre du paramètre –o qui demande l'écriture d'un fichier en sortie, suivi du nom de l'output et de celui de l'input. L'illustration suivante montre la compilation du fichier resources.qrc :

```

OSGeo4W Shell
nircmd      xmlpatterns
nircmdc     xmlpatternsvalidator
ogdi_import xmlwf
ogdi_info  xxmklink
ogr2ogr

epsg_tr      gdal_pansharp    ps2pdf14
esri2wkt     gdal_polygonize ps2pdfxx
gcps2vec     gdal_proximity  python_qgis
gcps2wld     gdal_retile     pyuic4
gdal2tiles  gdal_sieve      qgis-browser-grass7
gdal2xyz     grass72         qgis-browser
gdalchksum  gsettings       qgis-designer
gdalcompare liblas          qgis_grass7
gdalident   make-bat-for-py qgis
gdalimport  mkgraticule     rgb2pct
gdalmove    o-help          saga-ltr_gui
gdal_auth   o4w_env         setup_test
gdal_calc   pct2rgb         setup
gdal_edit   ps2pdf          ps2pdf12
gdal_fillnodata gdal_merge     ps2pdf13

GDAL 2.1.3, released 2017/20/01
C:\Users\thoma\Desktop>cd C:\Users\thoma\.qgis2\python\plugins\TempMap
C:\Users\thoma\.qgis2\python\plugins\TempMap>pyrcc4 -o resources_rc.py resources.qrc

```

Figure 42 :
Compilation d'un
fichier resources.qrc
dans l'OSGeo4W Shell
(capture d'écran)

Cette étape est indispensable pour poursuivre le développement du plug-in.

Nous verrons par la suite qu'il peut également être nécessaire d'effectuer d'autres compilations à d'autres étapes du développement, dans certaines conditions (ajout d'un fichier .ui, modification du fichier de ressources...).

Conception d'une interface graphique sous Qt Designer

Une fois la base réalisée à l'aide de Plug-in Builder, l'étape suivante consiste à concevoir l'interface graphique, qui sera ensuite reliée au script.

Pour ce faire, c'est le logiciel Qt Designer, dans sa version 2.14.12 qui a été utilisé. Celui-ci permet de créer une interface de façon assez intuitive, à l'aide d'un ensemble de widgets personnalisables.

Le design de l'interface a évolué au fur et à mesure du développement. Une première version basée sur les réflexions préalables issues du cahier des charges a ainsi été produite puis ajustée durant la phase d'ajout du script python principal au plug-in.

Nous nous proposons de revenir ici sur la version finale retenue. Celle-ci se base sur un système d'onglets, au nombre de quatre.

Le premier, « Configuration », concerne la sélection des inputs et la définition des outputs. Cet onglet seul est suffisant à l'utilisation du plugin. Un deuxième onglet, « Parameters », permet d'affiner certains réglages, notamment celui de la résolution de sortie des rasters (fixée par défaut à 100, ce qui correspond plutôt bien à un usage avec les données MAPUCE).

Un troisième onglet « Help » comporte un texte d'aide complet décrivant le fonctionnement général du plug-in, les algorithmes utilisés et le rôle des différents inputs et outputs. Cette aide a été rédigée en anglais et en français, l'onglet « Help » étant alors subdivisé en deux onglets : « English » et « Français ».

Le dernier onglet, intitulé « About », rassemble, en français et en anglais, des informations de base sur l'origine du plug-in, le programme de recherche MAPUCE et les personnes à contacter en cas de remarques/questions.

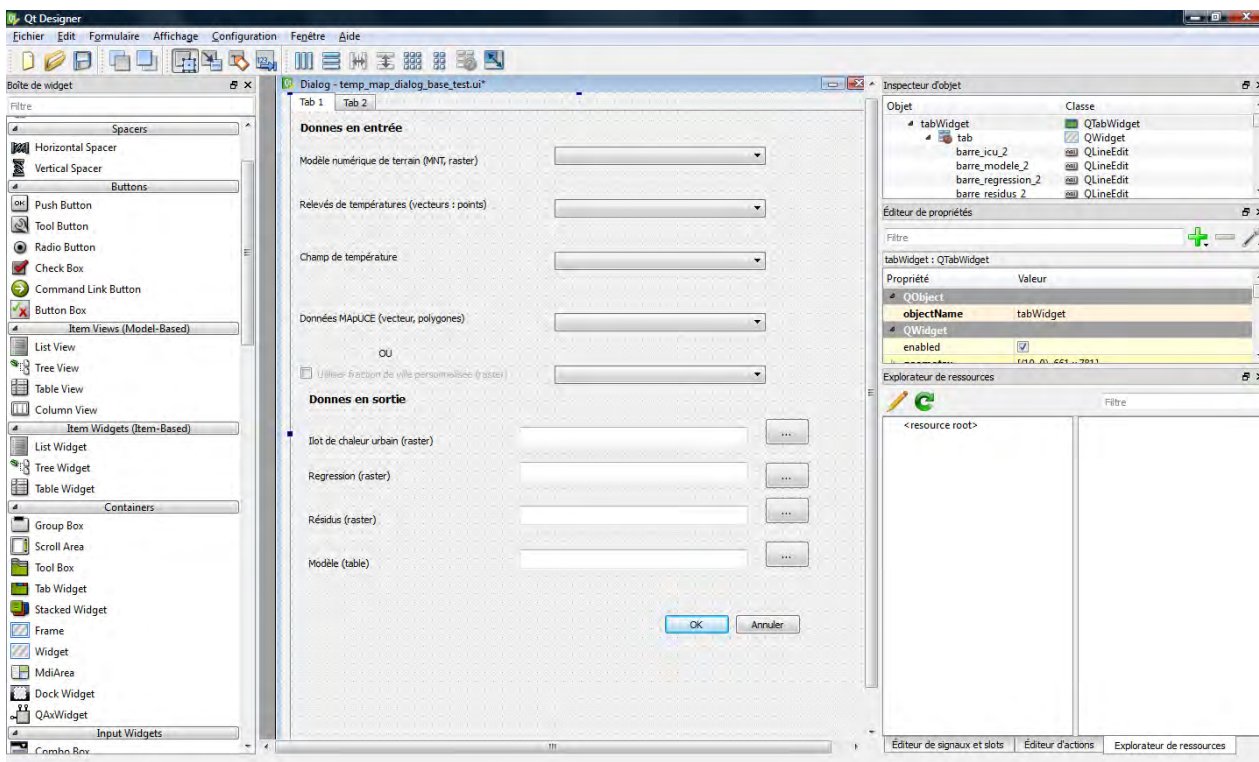


Figure 43 : Extrait de l'interface durant sa conception, sous Qt Designer (capture d'écran)

L'essentiel du travail de conception de l'interface s'est concentré sur l'onglet « Configuration ». Celui-ci se devait en effet d'être facilement compréhensible et de limiter au maximum les éventuelles erreurs de configuration de la part de l'utilisateur.

Les inputs sont ainsi au nombre de 5 : un MNT, un vecteur de relevé de températures, un champ de température issu de ce vecteur, une couche vecteur de données MApUCE ou une couche raster de fraction de ville laissée au choix de l'utilisateur (dans le cas où celui-ci ne peut pas ou ne souhaite pas utiliser les données MApUCE). Pour limiter au maximum les risques de confusion entre ces deux derniers champs – dont seul l'un des deux doit être rempli – une « case à cocher » (checkbox) a été ajoutée. La cocher, active l'utilisation d'une fraction de ville personnalisée et rend inutilisable le champ dédié aux données MApUCE. La laisser décocher, permet l'usage des données MApUCE (situation par défaut) et interdit l'usage d'une fraction personnalisée.

D'une manière générale, les inputs sont choisis via des « combo box », menus déroulants rassemblant les couches ouvertes dans le projet QGIS lors de l'utilisation du plug-in (ou les champs issus de ces couches). Les chemins d'enregistrement des outputs correspondent quant à eux à un widget de type « Line Edit ». Un chemin d'accès peut ainsi y être entré librement, mais peut aussi être sélectionné via un browser à l'aide d'un bouton « parcourir » ajouté devant chaque output. Laisser ces champs vides revient à enregistrer les outputs comme fichiers temporaires.

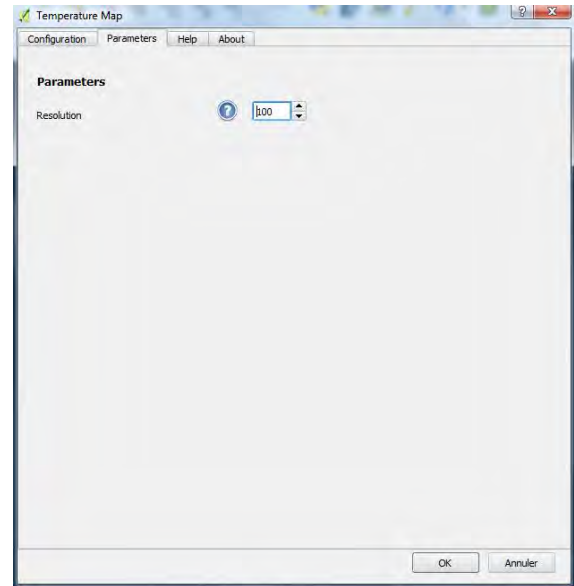
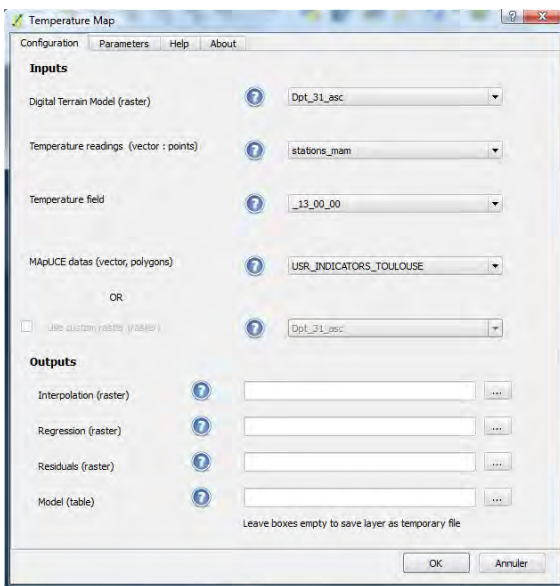
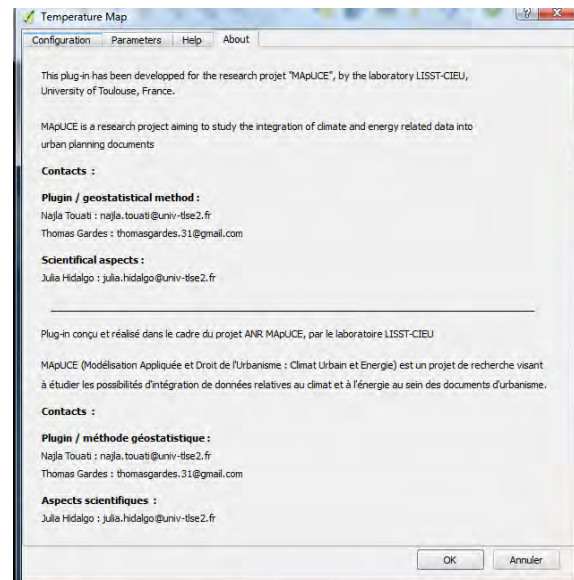
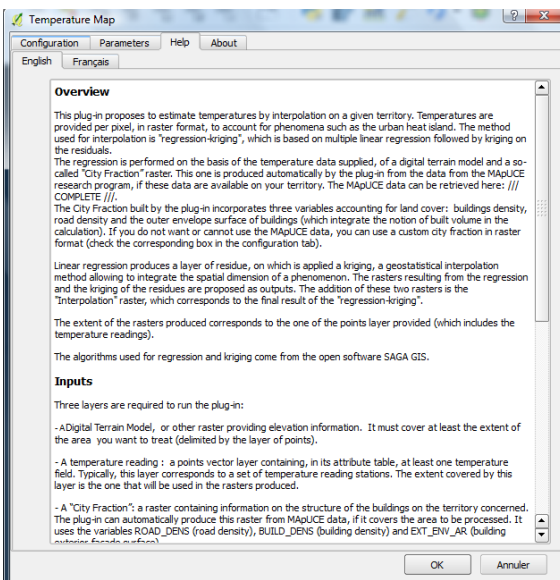


Figure 44 : Rendu final des différents onglets, sous QGIS (captures d'écran)



Le tableau suivant résume l'ensemble des widgets Qt Designer utilisés dans l'interface :

Nom du widget	Description	Nombre	Rôle dans le plug-in
Tab Widget	Permet de générer des onglets	2	Définition des 4 onglets principaux puis des deux sous-onglets de la rubrique « Help »
Label	Permet d'écrire un texte non modifiable ou d'insérer une image	42	Description des inputs, outputs et paramètres, insertion des icônes d'aide, contenu de l'onglet « About ».
Combo Box	Un menu déroulant permettant de sélectionner un élément	5	Sélection des différents inputs par l'utilisateur
Line Edit	Une ligne de texte pouvant être remplie par l'utilisateur	4	Sélection des différents chemins d'outputs par l'utilisateur
Push Button	Bouton cliquable	4	Permet à l'utilisateur d'utiliser la fonction « parcourir » pour sélectionner ses chemins d'output
Spin Box	Contient une valeur numérique uniquement, qui peut être réglée à l'aide d'un curseur ou entrée manuellement par l'utilisateur	1	Définition de la valeur du paramètre « Résolution »
Check Box	« Case à cocher », distinguant deux états : cochée et non cochée	1	Permet à l'utilisateur de choisir d'utiliser ou non une fraction de ville personnalisée. Limite la confusion entre fraction personnalisée et données MAPUCE.
Text Browser	Zone de texte non modifiable par l'utilisateur	2	Utilisé pour afficher l'aide dans les sous-onglets « English » et « Français » de l'onglet « Help »

Figure 45 : Tableau récapitulatif des widgets Qt Designer utilisés
Réalisation : Thomas Gardes, 2017

A noter que l'ajout d'images est normalement possible via le widget Label. Cette méthode a été utilisée ici pour afficher les différentes icônes d'aide visibles sur les onglets « Configuration » et « Parameters ». En passant la souris sur cette icône, l'utilisateur obtient un message d'aide décrivant les différents inputs, outputs et paramètres. Cependant, un bug d'affichage faisait que, si le widget était bien présent dans l'interface lors de son ouverture sous QGIS, l'image restait cependant invisible.

Après recherches, ce problème a été contourné par l'ajout d'une fonction python dans le fichier principal temp_map.py. Cette dernière, baptisée « resolve » par son auteur (« Nathan W. ») et visible sur les forums de stackexchange³⁰ permet de récupérer le chemin d'accès à un fichier dont le nom est défini.

```
def resolve(name, basepath=None):
    """Function used to get around a bug making pictures on the dialog doesn't
    display"""
    if not basepath:
        basepath = os.path.dirname(os.path.realpath(__file__))
    return os.path.join(basepath, name)
```

On l'utilise ici pour récupérer le chemin du fichier contenant l'image de l'icône d'aide (stockée dans le dossier du plug-in sous le nom « help.png »). Puis ce chemin est associé aux différents labels Qt designers correspondants, de la façon suivante :

```
#Set 'help' pictures for each combo box
pathIcon = resolve('help.png')
self.dlg.infoMnt.setPixmap(QtGui.QPixmap(pathIcon))
self.dlg.infoTemp.setPixmap(QtGui.QPixmap(pathIcon))
self.dlg.infoField.setPixmap(QtGui.QPixmap(pathIcon))
self.dlg.infoMapuce.setPixmap(QtGui.QPixmap(pathIcon))
```

Cette « astuce » permet de contourner le bug décrit précédemment. Elle permet également d'entrevoir une étape clé du développement d'un plug-in : le lien entre l'interface et le script.

Adapter le script à l'interface et l'interface au script

Nous l'avons vu, le script python assurant l'essentiel du fonctionnement du plug-in a été écrit en amont, avant même le développement du plug-in à proprement parler. (cf. partie 2, II). L'enjeu ici est donc plutôt de faire en sorte que ce script s'adapte à l'interface conçue sous Qt designer, de sorte à ce que les différents widget puissent, par exemple, appeler certaines fonctions python lors du clic, ou récupérer certains contenus comme les couches ouvertes dans un projet QGIS. Par exemple, la forme `##mnt=raster` qui permettait de définir les inputs et outputs dans l'éditeur de script de QGIS n'est ici plus valable. Nous nous proposons donc de revenir sur les principaux ajustements et les nombreux ajouts apportés au script d'origine afin de l'adapter au plug-in.

Le fichier python principal temp_map.py généré par plug-in builder inclut une classe principale TempMap comportant un ensemble de fonctions et méthodes prédéfinies telles que le `__init__`. Une fonction baptisée « run » est quant à elle laissée à la personnalisation du développeur. Celle-ci correspond notamment aux différents événements se déroulant après le lancement du plug-in. D'autres fonctions et lignes de codes peuvent bien sûr être ajoutées à d'autres endroits du fichier, mais l'essentiel des lignes ajoutées dans notre cas l'ont été dans la fonction run (malgré des exceptions qui seront mentionnées).

L'import du fichier correspondant à l'interface Qt designer dans le fichier python principal est également assuré par Plug-in Builder, ici via la ligne :

```
from temp_map_dialog import TempMapDialog
```

³⁰ Forum Stack Exchange : « *getting a plugin path using python in qgis* », post de Nathan W, <https://gis.stackexchange.com/questions/130027/getting-a-plugin-path-using-python-in-qgis/130031#130031>, consulté le 13 avril 2017

La classe TempMapDialog, qui contient en outre les différents widgets positionnés sous Qt Designer et traduits en python est également référencée de la façon suivante, dans la fonction add_action créée par Plug-in Builder : self.dlg = TempMapDialog()
self.dlg nous permet ainsi de faire référence aux widgets de l'interface dans le script.

Définir inputs et outputs en utilisant les widgets Qt Designer

La première tâche réalisée durant cette étape du développement a été de lier les « combo box » (menus déroulants) de Qt designer permettant le choix des inputs aux couches ouvertes dans le projet QGIS lors de l'utilisation du plug-in. L'idée est d'opérer ce lien de façon sélective : les couches de type raster seront ajoutées uniquement dans les menus déroulants des inputs de type raster, les couches vecteurs de type point uniquement dans les menus déroulants demandant des points...

Ceci est fait tout d'abord en créant une connexion au canevas de QGIS. On l'utilise ensuite pour récupérer l'ensemble des couches (layers) dans une liste :

```
#add layers to interface
mapCanvas = self.iface.mapCanvas() #connect to map canvas
ayers = self.iface.legendInterface().layers() #get all layers from map
canvas
```

Ensuite, plusieurs listes sont initialisées en vue de récupérer les différents layers selon leur type :

```
#Initialize lists for diffents types of layers
rasterList=[]
rasterLayerList = []
pointList = []
pointLayerList = []
polyList = []
polyLayerList = []
```

Les listes contenant la mention « Layer » récupèrent l'objet layer, les autres uniquement leurs noms, pour affichage.

Il faut ensuite opérer une distribution des layers dans les différentes listes en fonction de leur type. Ceci est fait en utilisant l'attribut .type() permettant de caractériser les objets QGIS de type VectorLayer et des structures conditionnelles :

```
#Add layers in lists depending of their types
for i in range (mapCanvas.layerCount()-1,-1,-1):
    layer = layers[i]
    if layer.type() == QgsMapLayer.VectorLayer :

        if layer.wkbType() == QGis.WKBPoint :
            pointList.append(layer.name()) #add name for combo boxes
            pointLayerList.append(layer) #add layers for processing
        else :
            polyList.append(layer.name())
            polyLayerList.append(layer)

    else :
        rasterList.append(layer.name())
        rasterLayerList.append(layer)
```


Pour chaque layer présent dans le canevas (donc ouvert dans le projet QGIS en cours), on vérifie s'il s'agit d'un vecteur. Si c'est le cas, on vérifie s'il s'agit d'un point. Si oui, on l'ajoute à la liste de points, sinon à la liste de polygones (qui récupère donc également les vecteurs de type ligne).

S'il ne s'agit pas d'un vecteur, on l'ajoute aux listes de raster.

Ces listes doivent ensuite être chargées dans les différentes combo box pour permettre la sélection.

On utilise alors l'attribut `addItem()` du widget combo box :

```
# Load lists in combo boxes
self.dlg.menu_mnt.addItem(rasterList)
self.dlg.menu_temperature.addItem(pointList)
self.dlg.menu_mapuce.addItem(polyList)
self.dlg.menu_fraction.addItem(rasterList)
```

`self.dlg` permet de faire référence aux widgets du fichier de l'interface Qt Designer. `menu_mnt`, `menu_temperature...` correspond aux noms donnés aux différents widgets lors de la conception de l'interface.

Les menus déroulants permettant de choisir une couche de MNT, de relevés de températures, de fraction de ville ou de données MAppUCE sont ainsi remplis. Remplir le menu proposant un champ de la couche de relevés de températures est par contre un peu plus délicat. En effet, il est souhaitable de faire en sorte que ce menu ne propose que les champs de la couche qui est indiquée dans le menu « relevé de températures », et puisse donc s'ajuster de façon dynamique en fonction de la valeur indiquée dans ce champ.

Nous utilisons pour cela le principe des signaux. Ces derniers permettent de lier une méthode/fonction définie à certaines actions choisies³¹. En l'occurrence, nous définissons la méthode suivante :

```
def layer_field() :
    """ Method used to change the content of the 'field' combo box depending of
    the layer set in "temperature" combo box """
    selectedLayerIndex = self.dlg.menu_temperature.currentIndex() #get
name of layer set in 'temperature' combo box
    selectedLayer = pointLayerList[selectedLayerIndex] #get layer set in
'temperature' combo box
    fields=selectedLayer.pendingFields() #get fields from the layer
    fieldnames=[field.name() for field in fields] #get fields names
    self.dlg.menu_field.clear() #clear combo box before add items
    self.dlg.menu_field.addItem(fieldnames) #add items (fields names) to
'field' combo box
```

Celle-ci vérifie le nom du layer présent dans le menu « relevés de températures », récupère le layer correspondant, puis les noms des champs qui y sont associés (attributs `.pendingFields()` puis `.name()`) et les ajoute à la combo box « Champ de température ».

Nous utilisons ensuite le `signal.connect` pour appeler cette nouvelle fonction lors de la mise à jour du contenu du menu « relevés de températures » :

```
self.dlg.menu_temperature.currentIndexChanged.connect(layer_field) #connect
layer_field method to temperature combo box
```

`.currentIndexChanged` fait ici référence à un changement de contenu (d'index) du menu « relevés de températures ».

³¹ SourceForge, PyQt 4.12 Reference Guide, « *New Style Signals and Slots Supports* », http://pyqt.sourceforge.net/Docs/PyQt4/new_style_signals_slots.html, consulté le 27 avril 2017

Un principe similaire est utilisé pour gérer la checkBox (case à cocher) « utiliser une fraction de ville personnalisée ». Celle-ci à un double rôle : lorsqu'elle est cochée, le script n'utilise plus les données indiquées dans le menu de données MAPUCE mais celles spécifiées dans celui « fraction de ville personnalisée » pour l'interpolation. Ceci se passe néanmoins après que l'utilisateur a configuré et lancé le plug-in et nous le verrons plus loin, dans la gestion du déroulement du script.

Ici, nous souhaitons par contre faire en sorte que le fait de cocher la case active le menu « fraction de ville personnalisée » et désactive celui « données MAPUCE », et vice-versa. Une autre méthode est alors ajoutée au fichier temp_map.py :

```
def state_changed() :
    """Method used to perform real time changes on the interface when checkBox
    is checked or unchecked"""
    #Check if checkBox is checked
    if self.dlg.checkBox.isChecked() :
        self.dlg.menu_mapuce.setEnabled(False) #Disable 'mapuce' combo
        box
        self.dlg.menu_fraction.setEnabled(True) #Enable 'fraction'
        combo box
        self.dlg.label_frac.setStyleSheet('color : black') #Set color
        of label 'fraction' to black
        self.dlg.label_mapuce.setStyleSheet('color : grey') #Set color
        of label 'mapuce' to grey
    else : #if checkBox not checked
        self.dlg.menu_mapuce.setEnabled(True) #Enable 'mapuce' combo
        box
        self.dlg.menu_fraction.setEnabled(False) #Disable 'fraction'
        combo box
        self.dlg.label_frac.setStyleSheet('color : grey') #Set color of
        label 'fraction' to grey
        self.dlg.label_mapuce.setStyleSheet('color : black') #Set color
        of label 'mapuce' to black
```

Cette méthode vérifie si la case est cochée (self.dlg.checkBox.isChecked()). Si elle l'est, la combo box « données MAPUCE » est désactivée (le widget « combo box » dispose d'un attribut .setEnabled(True or False) permettant de « griser » et empêcher l'usage de cette combo box). En outre, le texte associé (Qt Label) est lui aussi grisé. Si la case n'est pas cochée, c'est l'inverse qui se produit : le menu « fraction personnalisée » est désactivé et son texte est grisé. C'est d'ailleurs la situation par défaut à l'ouverture du plug-in. Cette fonction est ensuite connectée au widget checkBox de la même manière que précédemment :

```
#connect state_changed method to checkBox
self.dlg.checkBox.stateChanged.connect(state_changed)
```

Nous obtenons alors le résultat suivant :



Figure 46 : Impact dynamique de la checkBox sur l'interface (capture d'écran)

Enfin, le chargement des inputs possibles se faisant à chaque ouverture du plug-in, il convient de nettoyer le contenu des « combo box » avant un nouveau chargement. Des lignes permettant cela ont été ajoutées en préalable au code évoqué ci-dessus :

```
#clear all combo boxes
self.dlg.menu_mnt.clear()
self.dlg.menu_temperature.clear()
self.dlg.menu_field.clear()
self.dlg.menu_mapuce.clear()
self.dlg.menu_fraction.clear()
```

La gestion des outputs relève d'une démarche relativement similaire. Le widget « Line Edit » utilisé pour les chemins de sauvegarde des outputs permet directement à l'utilisateur d'entrer un chemin manuellement. Nous souhaitons cependant permettre à l'utilisateur de sélectionner le chemin via une fenêtre de navigation. Pour cela, de nouvelles méthodes ont été définies - une par output – sur ce modèle :

```
def outputICU(self) :
    """Define path to save output files"""
    filename = QFileDialog.getSaveFileName(self.dlg, "Select output
file", "", '*.tif')
    self.dlg.barre_icu.setText(filename)
```

La méthode `getSaveFileName` de la classe `QFileDialog` permet de générer la fenêtre de navigation souhaitée. On lui passe notamment en paramètres le titre de la fenêtre souhaité ? et le format du fichier à enregistrer. Le chemin ainsi défini est ensuite associé au widget Line Edit correspondant (appelé ici « barre_icu »), via la ligne `self.dlg.barre_icu.setText(filename)`.

Cette méthode est ensuite connectée à l'action de cliquer sur le bouton « parcourir » correspondant :

```
#Connect buttons "browse" to line edit
self.dlg.bouton_icu.clicked.connect(self.outputICU)
```

On obtient le résultat suivant :

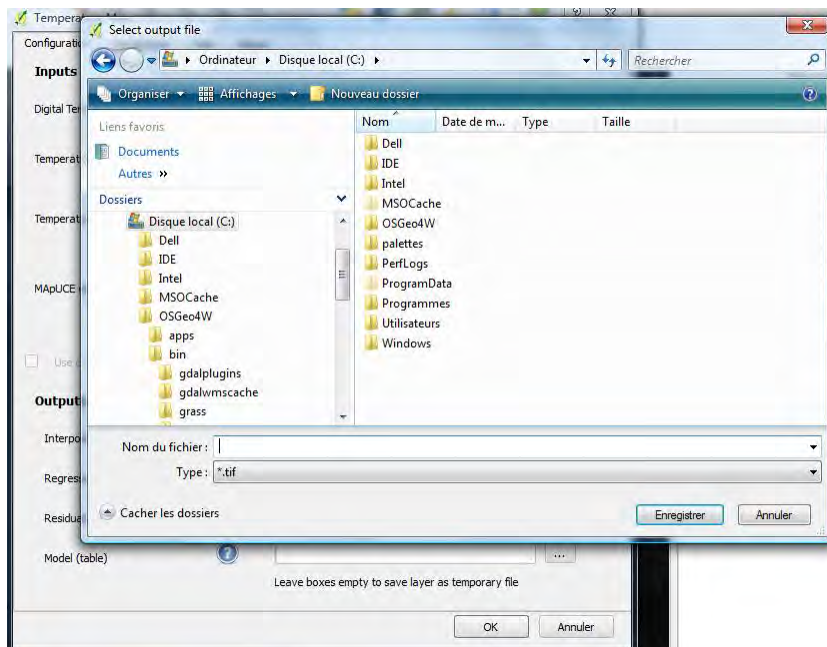


Figure 47 : Sélection d'un répertoire pour l'enregistrement d'un output en cliquant sur le bouton "parcourir" (capture d'écran)

Ces étapes achevées, il est désormais possible de remplir l'interface Qt Designer avec les informations souhaitées :

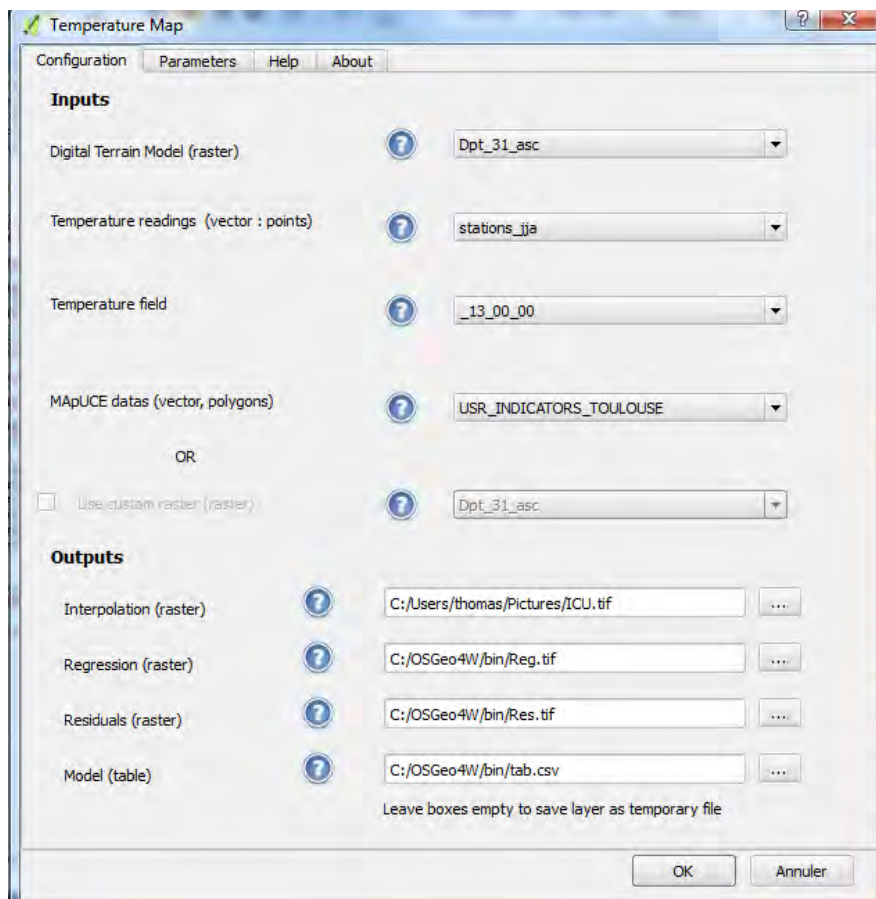


Figure 48 : Exemple de configuration du plug-in avant lancement (capture d'écran)

Récupérer les informations indiquées et les utiliser dans le déroulement du script

Si l'interface est maintenant « fonctionnelle » dans le sens où les différents widgets sont reliés à des fonctions et méthodes les rendant utilisables, il reste qu'un appui sur le bouton « OK » censé lancer le traitement ne déclenche encore, à ce stade, absolument rien.

Pour remédier à cela, nous nous basons sur les lignes suivantes, proposées par défaut à l'intérieur de la fonction run() générée par Plug-in Builder :

```
# Run the dialog event loop
result = self.dlg.exec_()
# See if OK was pressed
if result :
```

Les lignes de codes indiquées en indentation dans le « if result : » seront exécutées lors de l'appui sur « OK ». C'est donc ici que sera placé l'essentiel du script réalisé précédemment.

L'une des premières étapes à effectuer lors du lancement du plug-in est la récupération des inputs et des chemins d'outputs. En effet, nous avons au préalable permis à l'utilisateur d'indiquer ou de récupérer depuis QGIS les données qu'il souhaite utiliser. Il peut alors cliquer sur « OK » et sa tâche sera terminée. Celle du plug-in continue néanmoins, puisqu'il lui faut désormais récupérer les données indiquées et les utiliser comme variables pour alimenter les différents traitements définis.

Ceci est fait en allant chercher dans les listes de layer préalablement définies les layers correspondants à ceux indiqués par l'utilisateur :

```
#set layers and fields depending of current values of combo boxes
fieldTemp = fieldnames[selectedFieldIndex]
field = fields[selectedFieldIndex]
stations = pointLayerList[selectedTempIndex]
mnt = rasterLayerList[selectedMntIndex]
mapuce=polyLayerList[selectedMapuceIndex]
frac_user=rasterLayerList[selectedFracIndex]
```

A noter que les noms donnés à ces variables correspondent à ceux qui avaient été utilisés dans l'éditeur de script de QGIS (cf II.) pour définir les inputs, ceci afin de faciliter l'adaptation de ce script (il n'est ainsi pas nécessaire de réécrire les noms des variables utilisées dans les différents algorithmes appelés, par exemples, par le biais de `processing.runalg()`).

Le script déroulé ensuite est donc très similaire à celui conçu sur l'éditeur de QGIS, avec quelques ajouts permettant notamment une gestion différente selon que l'utilisateur utilise les données MAPUCE ou une fraction de ville personnalisée. Ceci est réalisé à l'aide d'une structure conditionnelle qui vérifie si la checkBox est cochée ou non :

```
if self.dlg.checkBox.isChecked() :

    getPathTemp = objEmp.dataProvider().dataSourceUri() #recuperation du
chemin d un fichier stocke comme temporaire
    splitPathTemp= getPathTemp[:-10] #suppression du nom du fichier en
fin de chaine
    outputCalc2 = splitPathTemp + 'frac_user.tif' #remplacement du nom du
fichier par le nom de la sortie voulu pour le raster calculator
    resamplFracVille = processing.runalg("saga:resampling", frac_user,
True, 0,0, extent,100,outputCalc2) #dimensionnement de la fraction de ville
utilisateur

else :

#Construction de la fraction de ville

#Rasterisation a 100m de resolution a partir des donnees mapuce
    build=processing.runalg("gdalogr:rasterize",mapuce,"BUILD_DENS",1,res
olution,resolution,False,5,"",4,75,6,1,False,0,"",None)#rasterisation
variable densite de bati
    [...]
```

L'extrait de code ci-dessus montre les différents traitements effectués selon si la case « utiliser une fraction de ville personnalisée » est cochée ou non.

Si la case est cochée, on récupère un chemin d'accès vers un fichier temporaire dans lequel on va stocker le résultat d'un « resampling » visant à mettre la fraction de ville personnalisée aux bonnes dimensions pour l'interpolation (la récupération et l'usage d'un chemin d'accès temporaire étant nécessaires pour éviter un bug lié ensuite à l'algorithme de régression linéaire de SAGA).

Si la case n'est pas cochée, on se tourne vers la construction de la fraction de ville à partir des données MAPUCE, qui n'est pas décrite ici dans son intégralité (cf. II pour plus de détails à ce sujet).

A ce stade, le plug-in produit alors des résultats similaires à ceux produits par le script rédigés dans l'éditeur de QGIS. Pourtant, un important travail de finition reste à réaliser, pour en rendre la prise en main agréable, en intégrant, par exemple, la gestion de possibles erreurs de la part de l'utilisateur ou la question de la compatibilité avec les différentes versions de QGIS.

Communiquer avec l'utilisateur

Réaliser l'interface et la rendre fonctionnelle en intégrant le script préalablement conçu aux fichiers python du plug-in constitue le cœur du travail de conception et de développement mais demeure insuffisant au moment de rendre un « produit fini ».

Il faut en effet intégrer un certain nombre d'éléments indispensables pour permettre un usage dans de bonnes conditions. Parmi ces éléments, on peut citer ceux relatifs à la communication avec l'utilisateur.

Cette dernière passe tout d'abord par la rédaction d'une aide complète. Celle-ci fait l'objet d'un onglet dédié, traduit en anglais et en français. Son contenu complet est visible en annexe 2.

L'aide détaille le fonctionnement général du plug-in, en expliquant l'intérêt de la méthode choisie ainsi que les différents algorithmes utilisés. Elle explique également le rôle des différents inputs, outputs et paramètres. Cette rubrique d'aide est complétée dans l'interface par des icônes en « points d'interrogation » placées devant les différents champs à remplir. Passer la souris sur ces icônes affiche une explication relative au fonctionnement du champ correspondant. Ceci est réalisé à l'aide de la fonction « tool tip » du widget « label » de Qt Designer.

La communication avec l'utilisateur passe également par l'intégration de messages d'erreurs. La structure du plug-in fait que les erreurs possibles pour l'utilisateur sont relativement limitées : une fois les inputs correctement remplis, le plug-in doit en effet s'exécuter sans erreur. Le remplissage de ces inputs avec des couches valides est fait automatiquement à partir des couches ouvertes dans le projet. Remplir un champ raster avec une couche vecteur est donc impossible. De même, le système de check-box fait qu'il est impossible d'indiquer à la fois une fraction de ville personnalisée et des données MApUCE (le plug-in utilisant forcément l'une OU l'autre). Deux cas principaux d'erreurs de remplissage des inputs peuvent cependant encore être envisagés :

- L'utilisateur n'a chargé dans son projet aucune couche raster ou aucune couche vecteur, de sorte que certains champs restent vides
- L'utilisateur a indiqué un même raster dans la rubrique « MNT » et dans la rubrique « Fraction de ville personnalisée » (les deux seuls champs supportant des formats raster).

Des messages d'erreur ont donc été intégrés pour répondre à ces situations. Ceux-ci sont réalisés en important la classe QMessageBox, qui permet d'afficher des messages d'avertissement ou d'information. La vérification du remplissage des différents inputs est faite au moment de l'exécution du plug-in (lorsque l'utilisateur a cliqué sur « OK » et avant que ne démarrent les premiers traitements) sur le modèle suivant :

```
#check MNT combo box
if self.dlg.menu_mnt.currentText() == "" : #if combo box is empty
    QMessageBox.warning(None, "Temperature Map", str("Veuillez indiquer un
raster de MNT valide"))
```

La structure conditionnelle vérifie si le champ est vide (s'agissant d'une combo box, il est soit vide soit correctement rempli, un remplissage avec une couche non-valide étant impossible). S'il est vide, on affiche alors un message d'avertissement : QMessageBox.warning, qui prend notamment en paramètres le titre de la fenêtre à afficher, le message qu'elle doit contenir. Les différents inputs sont ainsi vérifiés successivement (structure en « elif »). On s'assure également que les raster de MNT et de fraction de ville sont différents à l'aide du code suivant :

```
#check if fraction and MNT are different from each other (only when
checkBox is checked)
elif self.dlg.menu_fraction.currentText() ==
self.dlg.menu_mnt.currentText() and BoxChecked :
    QMessageBox.warning(None, "Temperature Map", str("La fraction de
ville et le MNT doivent etre deux rasters differents"))
```

La condition « BoxChecked » étant une variable booléenne définie préalablement et utilisée pour vérifier si la checkBox est cochée ou non (cette vérification de remplissage des inputs raster n'ayant en effet de sens que lorsque la checkBox est cochée). Dans les cas où ces erreurs sont renvoyées, l'exécution du plug-in est bien sûr annulée.

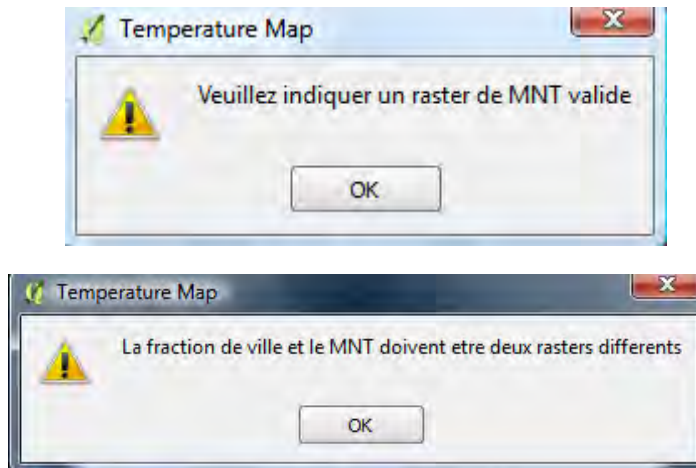


Figure 49 : Exemples de messages utilisés pour communiquer avec l'utilisateur (capture d'écran)

Une autre partie de la communication avec l'utilisateur passe par le suivi de l'exécution du traitement. Deux éléments ont été ajoutés en ce sens : l'affichage du temps de traitement et une barre de progression.

L'affichage du temps de traitement est réalisé de la même manière que les messages d'erreurs. On importe dans le script la bibliothèque « time », qui va permettre d'enregistrer un temps en début et en fin d'exécution du plug-in. Une soustraction de ces deux temps donne une variable contenant la durée d'exécution en secondes :

```
# See if OK was pressed
if result:

    tps1 = time.clock() #initialize time count
```

[Déroulement du script...]

```
tps2 = time.clock()
duration = round(tps2 - tps1,3)
QMessageBox.information(None, "Temperature Map", "Traitement
effectue en "+str(duration)+"s")
```

L'ajout d'une barre de progression s'est avéré plus complexe. Si un widget Qt Designer « Progress Bar » existe bel et bien, le problème se posait du placement de cette barre. En effet, l'ajout à l'interface d'origine n'était pas pertinent puisque, par nature, cette boîte de dialogue disparaît après que l'utilisateur a cliqué sur « OK » pour déclencher l'exécution du plug-in.

Le choix a donc été fait d'ajouter une nouvelle fenêtre Qt Designer (donc un nouveau fichier .ui), qui serait appelée lorsque l'utilisateur clique sur OK. La fenêtre a donc été réalisée sous Qt Designer, en intégrant simplement un widget « progress bar ».

Ce nouveau fichier (appelé « progress.ui ») a ensuite été compilé en python à l'aide du shell OsGeo4W. La classe principale Ui_Form issue de cette compilation est ensuite importée dans le fichier python principal temp_map.py. Il est alors possible de déclencher l'appel de cette nouvelle fenêtre lors de l'appui sur OK via le code suivant :

```
if result :
```

```
    #show progression window when "OK" is pressed
    widget = QtGui.QWidget()
    ui = Ui_Form()
    ui.setupUi(widget)
    widget.show()
    progress=ui.progressBar #define progress bar widget added from ui
file
    progress.setMaximum(100) #set maximum in percentage for progress bar
```

La valeur maximale de la barre est fixée à 100, afin d'être donnée en pourcentage. Son remplissage est effectué manuellement, à différentes étapes clés du script, par l'attribut « setValue » du widget « progress bar : »

```
progress.setValue(30)
```

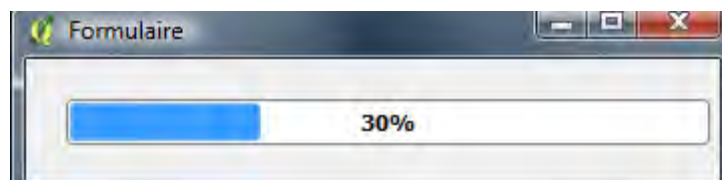
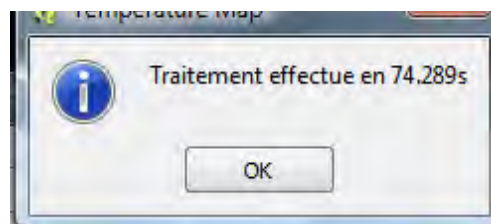


Figure 50 : La barre de progression et le temps d'exécution qui s'affichent respectivement pendant et après l'exécution du plug-in, sous QGIS (capture d'écran)



Enfin, et toujours dans l'optique de rendre l'usage du plug-in plus simple et plus agréable, le choix a été fait de modifier le système de projection des couches en sortie, données par défaut en EPSG:4326. Le but est alors de faire correspondre ce système à celui d'une des couches indiquées par l'utilisateur. C'est la couche de MNT qui a été choisie arbitrairement, à partir des constats suivants :

- Il n'est pas nécessaire que les différents inputs aient le même CRS pour que le plug-in fonctionne
- Soit l'utilisateur a défini un même CRS pour tous ses inputs, et le layer choisi pour référence de CRS n'a pas d'importance
- Soit l'utilisateur a laissé des CRS différents et il faut donc en choisir un arbitrairement, n'ayant pas les moyens de savoir lequel l'utilisateur souhaite privilégier en sortie.

Cet ajustement est donc fait en récupérant le CRS au moment de l'exécution du plug-in :

```
crsRefLayer = layers[0] #get crs value from the first layer
idCrsRef = crsRefLayer.crs().authid()
epsgSplit = idCrsRef.split(':')
epsg0 = epsgSplit[1]
epsg = int(epsg0)
```

Ce CRS est ensuite attribué aux couches en sortie (exemple ici du raster « INTERPOLATION ») :

```
iface.addRasterLayer(outputPath1, "INTERPOLATION")
interObj = processing.getObject(outputPath1)
interObj.setCrs(QgsCoordinateReferenceSystem(epsg,
QgsCoordinateReferenceSystem.EpsgCrsId))
```

Ce travail effectué, le plug-in est prêt à entrer en phase de test.

4. Discussion : constats et limites

Le plug-in est évidemment testé en continu, aux différentes étapes du développement. L'ajout de chaque fonctionnalité fait l'objet d'un test, suivi de la correction des éventuels problèmes constatés. Il serait laborieux et peu pertinent de décrire ici ces tests réguliers, qui ont forcément émaillé les différentes étapes du développement expliqué ci-dessus. Nous nous concentrerons donc sur la phase de test « finale », effectuée après l'essentiel du développement avec plusieurs objectifs :

- « Mettre en difficulté » le plug-in en le testant avec différentes données, valides comme invalides
- Comparer les résultats obtenus par le plug-in avec ceux obtenus par une application « manuelle » de la méthode
- Tester les temps d'exécution sur différentes machines avec différents jeux de données
- Tester la compatibilité avec différentes versions de QGIS

a. Exactitude des résultats produits

Les résultats obtenus en utilisant la méthode d'interpolation ici automatisée sont déjà exposés et discutés dans la partie I de ce mémoire. L'objectif ici n'est donc pas de revenir sur ces résultats mais de confirmer que les résultats produits par le plugin sont bien identiques à ceux observés lors de la validation de la méthode.

Pour ce faire, un jeu de données a été retenu arbitrairement : stations CAPITOUL sur la période jja à 13H, MNT de la BD Alti à 25m de résolution spatiale et données MApUCE par USR. La méthode d'interpolation a été reproduite manuellement, puis le plug-in a été utilisé avec les mêmes données d'entrée.

Les résultats respectifs ont fait l'objet d'une comparaison, d'abord visuelle, ensuite par soustraction raster (un des raster étant soustrait à l'autre). Le constat est probant : le résultat de la soustraction est un raster uniforme, de valeur 0, indiquant des résultats parfaitement identiques.

Le même test a été reproduit avec d'autres données d'entrées : des stations CAPITOUL à d'autres horaires et des fractions de villes personnalisées (qui sont traitées différemment des fractions de ville MAppUCE dans le plug-in). Là encore, la comparaison des résultats a montré que le plug-in respectait bien la méthode d'interpolation explicitée en partie I.

b. Compatibilité avec les différentes versions de QGIS

Mise en compatibilité avec les versions actuelles

Une fois confirmée la conformité des résultats, la phase de test suivante a consisté à vérifier la comptabilité du plug-in avec les différentes versions de QGIS actuellement en usage, et, le cas échéant, à ajuster le code de sorte à le rendre compatible.

La compatibilité est déjà avérée avec la version LRT 2.14.8, qui est la version de développement.

Les tests ont donc porté sur les versions antérieures (à partir de la 2.10) et supérieures (jusqu'à la dernière version en usage de la 2.18).

Le tableau présenté plus loin dans cette sous-partie résume l'ensemble des versions testées et les ajustements effectués, le cas échéant, pour mettre en compatibilité.

D'une manière générale, les plus gros problèmes concernent les versions antérieures à la 2.14. En effet, les algorithmes de krigeage implémentés dans ces versions apparaissent peu fonctionnels et diffèrent fortement de l'algorithme utilisé lors du développement et de l'automatisation de la méthode. Ceci ne pouvant pas être modifié par le plug-in, ces versions n'ont donc pas pu être mises en compatibilité.

Les versions supérieures à la 2.14 présentent des difficultés liées à l'appel des algorithmes en python dans le fichier principal du plug-in. En effet, en fonction des versions, le nombre d'arguments obligatoires pour appeler un même algorithme peut varier. Ceci a été contourné en intégrant dans le code une détection de la version de QGIS utilisée au lancement du plug-in. Des structures conditionnelles aux différentes étapes concernées ont ensuite été intégrées, de sorte à appeler les algorithmes avec le nombre d'argument correspondant pour les différentes versions testées.

Le numéro de version est obtenu à l'aide du bloc d'instruction suivant :

```
#Check QGIS version
version = qgis.utils.QGis.QGIS_VERSION #get QGIS version as string
versionSplit = version.split('.') #split version string
NumVersion = versionSplit[1] #get only number of version after the 2.xx
(get the "xx")
NumVersion2= versionSplit[2] #get only third version number (get the "xx"
in 2.14.xx)
qgisVersion = int(NumVersion) #get version number as integer
qgisVersion2 = int(NumVersion2)
```

L'adaptation aux différentes versions peut se faire à travers des structures de ce type :

```
if qgisVersion <= 14 : #check qgisVersion used

#Allow to get around a difference between version following 2.14 : gdal
rasterize algorithm take one more argument in the latests versions.
build=processing.runalg("gdalogr:rasterize",mapuce,"BUILD_DENS",1,resolutio
n,resolution,False,5,"",4,75,6,1,False,0,"",None
```

Le tableau suivant résume les versions testées et leur compatibilité.

Version testée	Compatible	Ajustements effectués
2.10.1	✗	Algorithme de krigeage incompatible
2.12.3	✗	Algorithme de krigeage incompatible
2.14.8	✓	-
2.16.3	✓	Ajustement du nombre d'arguments de algorithmes de rasterisation de GDAL
2.18.0	✓	Ajustement du nombre d'arguments de algorithmes de rasterisation de GDAL

Figure 51 : Compatibilité du plug-in avec les différentes versions de QGIS testées
Réalisation : Thomas Gardes, 2017

Les versions 2.18.6 et 2.18.10 ont également été testées à leur sortie (qui a eu lieu pendant, voire après le développement du plug-in). Le plug-in semble compatible avec la 2.18.6 mais pas avec la 2.18.10, dernière version en date au moment où ces lignes sont écrites. En effet, après quelques recherches, celle-ci semble présenter un problème de compatibilité avec les outils de SAGA qui y sont intégrés par défaut, ce qui se manifeste par l'impossibilité d'y utiliser l'outil de régression linéaire multiple.

Et ensuite ? L'arrivée de QGIS 3

Si la compatibilité du plug-in avec les différentes versions de QGIS a pu être vérifiée pour les versions diffusées au moment du développement, la question se pose pour les versions à venir.

L'annonce de la sortie prochaine de QGIS 3 peut légitimement cristalliser les interrogations. En effet, s'il reste impossible de déterminer la compatibilité tant que QGIS 3 n'est pas disponible, les annonces sur le sujet laissent penser que, dans le meilleur des cas, une adaptation du script sera nécessaire.

Selon les informations officielles fournies à ce stade³², l'API sera remaniée en profondeur, rendant obsolète les classes et fonctions utilisées jusqu'alors.

De même, le support de Qt4 et Python en version 2.xx ne devrait pas être maintenu, au profit de Qt5 et Python 3.

Ces changements majeurs laissent supposer que le plug-in ne sera pas compatible en l'état avec cette version, dans la mesure où il se base sur l'API QGIS actuelle. Néanmoins, on peut s'interroger sur les éventuels moyens qui pourraient être mis en œuvre par les développeurs du logiciel pour assurer la compatibilité avec les nombreuses extensions et plug-in développés jusqu'alors.

³² Site officiel QGIS.ch, *Les plans pour QGIS 3.0*, consulté le 06/07/2017, <https://www.qgis.ch/fr/nouvelles/les-plans-pour-qgis-3.0>

La version 3.0 devrait être disponible au 29 septembre 2017, une réponse définitive pourra donc être apportée à ce moment-là.

Quoi qu'il en soit, il est plus que vraisemblable de supposer que le plug-in devra faire l'objet d'ajustements multiples pour maintenir une compatibilité avec les versions récentes à long terme. Ceux-ci peuvent prendre la forme de ceux déjà effectués (modification de l'appel de tel ou tel algorithme) mais pourront aussi solliciter des modifications plus profondes, qui ne peuvent pas vraiment être prédites à l'heure actuelle.

c. Les temps de traitements

Une bonne connaissance des potentialités d'utilisation du plug-in passe également par la bonne connaissance des temps de traitements. Ceux-ci sont tributaires de 3 variables principales :

- La version de QGIS utilisée
- La configuration matérielle du PC utilisé (en particulier mémoire vive et processeur)
- Le nombre d'entités à traiter

Des tests ont donc été effectués pour comparer l'impact de ces différents éléments sur le temps de traitement. Les durées ont été mesurées en relevant celles fournies par le plug-in. Ce dernier commence un décompte lorsque le bouton « OK » lançant l'exécution du plug-in est cliqué, et s'arrête lorsque le dernier traitement est réalisé. Le temps de traitement est ensuite affiché à destination de l'utilisateur.

Versions et temps de traitement

Le tableau suivant résume l'impact des versions de QGIS sur les temps de traitement.

Les tests ont été effectués sur la même configuration (Intel Core 2 Duo CPU T7250 2.00 Ghz 3.00 Go de RAM) avec le même jeu de données (27 stations CAPITOUL, USR MAPUCE)

Version	2.14.12	2.16.3	2.18.6
Durée de traitement (secondes)	76.216	75.365	46.862

Figure 52 : Temps de traitement en fonction des versions de QGIS
Réalisation : Thomas Gardes, 2017

On constate notamment la faculté de la version 2.18 à proposer un temps de traitement significativement réduit, vraisemblablement dû à une meilleure rapidité d'exécution des algorithmes QGIS sur cette version.

Configuration, nombre d'entités et temps de traitements

Les tests sur les différentes configurations ont également été effectués avec un nombre d'entités à traiter différent. Concrètement, il s'agit de faire varier le nombre de points de relevés de températures. Cette donnée est en effet celle qui semble impacter le plus fortement la durée de traitement. Les tests ont donc été réalisés avec 27 points (données CAPITOUL), puis avec 100, 500 et 1000 points.

Ces 3 dernières couches de points correspondent à des relevés fictifs, créés via l’outil QGIS « Points aléatoires dans les limites de la couche ». Un champ de température leur a été ajouté via la calculatrice de champs, en utilisant la fonction random pour assigner un nombre décimal entre 15 et 18, afin de simuler un relevé de températures. Les résultats produits ne peuvent pas être exploités (dans la mesure où la répartition des valeurs de température n’est pas réaliste), mais le temps de traitement relevé doit correspondre à celui d’un relevé réel comprenant le même nombre de points.

Nombre d’entités	Intel Core 2 Duo CPU T7250 2.00 GHz 3.00 Go de RAM	Intel Core i3 M380 2.53 GHz 4.00 Go de RAM	Intel Celeron N2840 2.16 GHz 4.00 Go de RAM	Intel Core i5-6300HQ 2.30 GHz 8.00 Go de RAM
27	47.447 sec	42.336 sec	43.845 sec	22.329 sec
100	71.249 sec	72.148 sec	75.478 sec	26.475 sec
500	158.222 sec	147.656 sec	138.905 sec	42.417 sec
1000	492.387 sec	365.113 sec	368.118 sec	97.327 sec

Figure 53 : Temps de traitements en secondes sur différentes configurations de test
Réalisation : Thomas Gardes, 2017

d. Remarques et discussion

D’une manière générale, le plug-in paraît correspondre au cahier des charges préalablement établi. Néanmoins, plusieurs points méritent d’être signalés et discutés.

L’accessibilité des données nécessaires au fonctionnement du plug-in a déjà été évoquée en partie I, elle peut néanmoins constituer une limite à un usage « démocratisé » de l’outil, dans la mesure où des relevés de température ne sont pas aisément disponibles pour toutes les communes.

Au rang des ajustements possibles, le choix a été délibérément fait de laisser un minimum d’options au choix de l’utilisateur et de proposer une configuration « par défaut » simplifiant au maximum l’utilisation du plug-in. On pourrait cependant réfléchir à la pertinence d’intégrer un plus grand nombre des paramètres configurables par l’utilisateur. Par exemple, l’utilisateur pourrait être amené à ajuster l’emprise des rasters de sortie (la fixer sur celle de son MNT ou des données MApUCE plutôt que sur celle des points par exemple). De tels ajustements tendraient néanmoins à alourdir l’usage du plug-in et paraissent superflus compte tenu du public de non-spécialiste visé.

La question de la compatibilité avec les versions de QGIS mérite également d’être soulevée. Nous l’avons vu, le plug-in est actuellement compatible depuis la version 2.14 (maintenue à long terme) jusqu’à la 2.18 (la dernière en date). Ceci semble couvrir une large gamme d’usagers disposant d’une version un minimum actualisée, mais exclut également les usagers de versions plus anciennes. En outre, aucune garantie solide ne peut pour l’instant être formulée quant à la compatibilité avec les versions futures, la « révolution » de QGIS 3.0 semblant bien partie pour bouleverser les habitudes de développement.

Pour finir, évoquons la question de la mise à disposition. A l'heure actuelle, le code est notamment stocké en ligne via un espace GitHub. L'objectif à terme est d'intégrer le plug-in au dépôt officiel des extensions QGIS. Ceci ne devrait cependant pas être effectué avant quelques mois. En effet, un article est actuellement en cours de rédaction sur le sujet de la méthode d'interpolation et du plugin, avec pour objectif une publication dans une revue scientifique. La mise à disposition du plug-in dans le dépôt officiel QGIS ne devrait pas intervenir avant cette publication.

Conclusion

Au final, l'expérience de stage développée dans ce mémoire s'est avérée particulièrement enrichissante. Humainement tout d'abord, avec un parcours émaillé de rencontres et d'opportunités, et professionnellement, grâce une mission riche et variée. Celle-ci a en effet permis l'acquisition et le développement de compétences dans des domaines tels que la géostatistique (régression linéaire, krigeage, régression-kriging), le développement python en général et orienté QGIS en particulier (généralités du langage, architecture d'un plug-in, usage de Qt Designer, de l'API QGIS, de bibliothèques python spécifiques), la conception de chaînes de traitements en SIG, la sémiologie graphique ou encore la recherche d'informations issues de sources telles que les forums liés à la programmation et au développement.

La liberté laissée dans l'organisation du travail a également donné au stage une certaine dimension « gestion de projet ». On peut ainsi noter que le GANTT correspondant au travail effectif diffère de celui, prévisionnel, présenté en début de rapport :

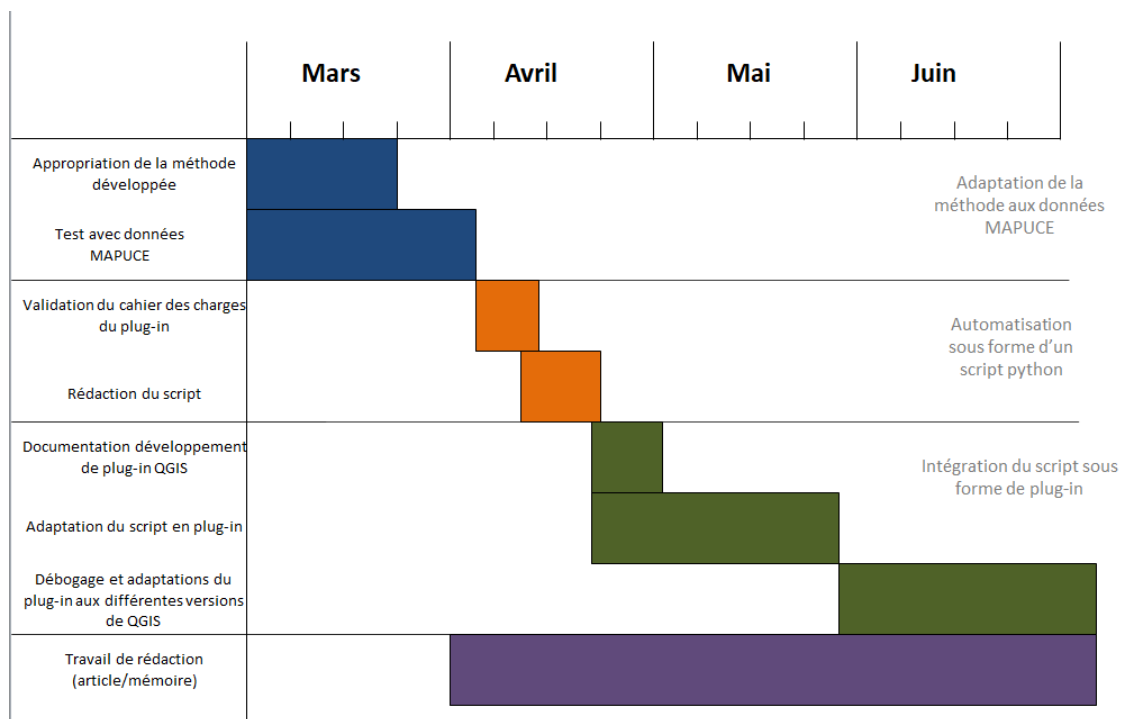


Figure 54 : GANTT récapitulatif de l'organisation effective du stage
Réalisation : Thomas Gardes, 2017

Le travail produit est ainsi un condensé des savoirs acquis durant l'année d'étude et des nombreux apports issus de la période de stage, qui s'est avérée particulièrement formatrice. Néanmoins, plusieurs aspects aussi bien techniques que thématiques restent encore à développer. La question de la représentation visuelle de l'îlot de chaleur urbain modélisé n'a ainsi été abordée que de façon très superficielle et mériterait des approfondissements. Ce travail avait à l'origine été envisagé comme une possible mission de stage complémentaire, mais elle semble finalement mériter d'y dédier un stage à part entière. On pourrait également imaginer étoffer le plug-in avec des fonctionnalités permettant de rapatrier directement les données MAPUCE sur QGIS à partir d'une base de données en ligne. Réaliser

des tests sur d'autres territoires semble aussi une prochaine étape incontournable. Celle-ci n'a pas pu être réalisée durant le stage faute d'un accès à des données exploitables. Néanmoins les questionnements d'avenir résident surtout dans la façon dont la méthode d'interpolation et le plug-in qui l'automatise pourront être exploités à un niveau opérationnel. Quelle appropriation par les publics ciblés ? Quels apports concrets à l'analyse et l'élaboration de politiques urbaines ? Quels problèmes et quelles limitations pourront être rencontrés à l'usage ?

D'une manière générale, l'avenir de la méthode proposée ici paraît lié à la place qu'occupent et qu'occuperont dans un avenir proche les données climatiques dans les politiques et documents d'urbanisme, ce qui rejoint les principales préoccupations du projet ANR MApUCE.

Cet avenir passe aussi inévitablement par la mise à disposition du plug-in dans le dépôt officiel des extensions QGIS et par un suivi à long-terme afin de prendre en compte les retours d'utilisateurs et les évolutions du support logiciel, l'arrivée de QGIS 3.0 en tête.

Bibliographie

Ouvrages

CHARRE J., *Statistique et territoire*, Reclus, 2000

HENGL, T., *A Practical Guide to Geostatistical Mapping of Environmental Variables*. European Commission, 2007.

JOURNEL A.G, HUIJBREGTS J., *Mining Geostatistics*, Academic Press, 1978

Articles et rapports scientifiques

BERGERON O., : « *Caractérisation de la variabilité intra-urbaine de la température selon une perspective géostatistique* », Huitième conférence internationale sur le climat urbain (International Conference on Urban Climate), du 6 au 10 août 2012, University College Dublin (Dublin) Irlande.

CHARBAI, BIGOT, KERGOMARD, « Interpolation et cartographie automatique de la température nocturne en milieu urbain », Publication de l'association internationale de climatologie, Vol.14, 2002

ELKETTANI, Y. « Deux méthodes complémentaires au krigeage ordinaire, pour l'estimation d'un processus spatial stationnaire de moyenne inconnue », *Revue de Statistique Appliquée, Tome 53 (2005) no. 4* , p. 31-47 .

FU, W. et al.. « Using Moran's I and GIS to Study the Spatial Pattern of Forest Litter Carbon Density in a Subtropical Region of Southeastern China ». *Biogeosciences* 11, n° 8 (29 avril 2014)

GRATTON, Y. « Le krigeage: la méthode optimale d'interpolation spatiale ». *Les articles de l'Institut d'Analyse Géographique* 1 (2002).

GUILLOT, G.. « Introduction à la géostatistique ». *Bulletin of the World Health Organization* 81, n° 3 (2003): 197–204.

HENGL, Tomislav, HEUVELINK G., ROSSITER D., . « About regression-kriging: From equations to case studies ». *Computers & Geosciences* 33 (s. d.): 1301-15.

HIDALGO J., TOUATI N., « Vers une modélisation opérationnelle de l'îlot de chaleur urbain à l'aide d'outils SIG », Conférence SAGEO 2016, Nice, 6-9 décembre 2016

HUDSON, GORDON, et WACKERNAGEL. « Mapping temperature using kriging with external drift: theory and an example from Scotland ». *International journal of Climatology* 14, n° 1 (1994): 77–91.

JABOT, E., I. ZIN, T. LEBEL, A. GAUTHERON, et C. OBLED. « Spatial interpolation of sub-daily air temperatures for improving snow and hydrological forecasts on Alpine catchments ». *68th Easter Snow Conference, McGill University, Montreal, Quebec, Canada 2011*

LAMY C, DUBREUIL V. « Impact du changement climatique sur les sécheresses en Bretagne. Automatisation d'un bilan hydrique avec ArcGis et Python ». *Revue internationale de géomatique* 24, n° 3 (30 septembre 2014): 355-75.

MATHERON G., « Le krigeage universel ». *Les Cahiers du Centre de Morphologie Mathématique de Fontainebleau*, 1969.

PIAZZA et al. « Comparative Analysis of Spatial Interpolation Methods in the Mediterranean Area: Application to Temperature in Sicily ». *Water* 7, n° 5 (27 avril 2015): 1866-88

Thèses et mémoires

BENZIANI I F, « Etude par krigeage des données piézométriques de la nappe du plateau de Mostaganem », Université d'Oran, 2009

NAPOLY A., *Utilisation des données participatives Netatmo pour l'étude du climat urbain. Application aux villes de Toulouse, Paris et Berlin*, INP Toulouse, Ecole Nationale de Météorologie, mémoire de Master II, 2017.

Pages web et forum

Communauté Urbaine du Grand Lyon, « Identifier les îlots de chaleur urbains pour réduire l'impact sanitaire des vagues de chaleur », Consulté le 17 avril 2017
<http://blogs.grandlyon.com/plan-climat/files/downloads/2010/09/23509293-ICU-synthese-PDF.pdf>.

DDTM Nord, « Conception d'un plug-in pour QGIS ». Consulté le 18 avril 2017.
http://www.geoinformations.developpement-durable.gouv.fr/fichier/pdf/CETENP_2011-10-27_Concevoir_Plugin_Python_QGIS_cle1f6963.pdf?arg=177828825&cle=b663023619b5e68c2fbe7f3d7dedbb239e70f484&file=pdf%2FCETENP_2011-10-27_Concevoir_Plugin_Python_QGIS_cle1f6963.pdf.

Developpez.com « PyQt : concevoir visuellement des interfaces avec Qt Designer ». Consulté le 5 mai 2017. <http://tcuvelier.developpez.com/tutoriels/pyqt/qt-designer/>.

DRAPEAU, L. « Statistiques et Interpolations dans les SIG », 2000.,
http://www.dphu.org/uploads/attachments/books/books_359_0.pdf.

IRSTEA. « Les indices d'auto-correlation spatiale »,

MARCOTTE D., « Le Krigeage » (powerpoint) ». Consulté le 4 avril 2017,
<https://cours.etsmtl.ca/sys866/Cours/documents/krigeage-Marcotte.pdf>.

QGIS Tutorial « Building a Python Plugin — QGIS Tutorials and Tips ». Consulté le 10 avril 2017.
http://www.qgistutorials.com/it/docs/building_a_python_plugin.html.

Forum ESRI GeoNet, "Regression kriging" ». Consulté le 2 avril 2017.
<https://geonet.esri.com/thread/137579.v>

Forum Stackexchange « getting a plugin path using Python in QGIS - Geographic Information Systems Stack Exchange ». Consulté le 12 mai 2017. <https://gis.stackexchange.com/questions/130027/getting-a-plugin-path-using-python-in-qgis/130031>.

Forum Stackexchange« pyqgis - Performing Raster Calculator functions using Python and open source modules? - Geographic Information Systems Stack Exchange ». Consulté le 5 avril 2017. <http://gis.stackexchange.com/questions/84550/performing-raster-calculator-functions-using-python-and-open-source-modules>.

Forum Stackexchange “Why does including latitude and longitude in a GAM account for spatial autocorrelation?” » Consulté le 2 avril 2017. <http://stats.stackexchange.com/questions/35510/why-does-including-latitude-and-longitude-in-a-gam-account-for-spatial-autocorre>.

SourceForge, « New-style Signal and Slot Support — PyQt 4.12 Reference Guide ». Consulté le 27 avril 2017. http://pyqt.sourceforge.net/Docs/PyQt4/new_style_signals_slots.html.

Table des matières

Introduction	10
A°) Territoire du stage, territoire de l'étude	10
B°) La structure d'accueil	11
C°) Le projet MAPUCE	11
D°) Les objectifs du stage	12
I. Une méthode d'interpolation géostatistique pour modéliser l'îlot de chaleur urbain. 15	
1. Modéliser les ICU : définition et état de l'art sélectif	15
a. Eléments de définition	15
b. Causes et conséquences	16
c. Etat de l'art sélectif	18
2. Principes du « regression-kriging »	21
a. Rappels sur la régression linéaire	21
b. Le krigeage	23
c. Le regression-kriging	26
3. Application du regression-kriging à la modélisation des ICU : la méthode du LISST-CIEU.....	29
a. Principe général	29
b. Données et logiciels	30
c. Décomposition de la méthode et de sa mise en œuvre sous SAGA GIS.....	34
4. Analyse et discussion des résultats obtenus	42
Partie II : Automatisation de la méthode : de la chaîne de traitement au plug-in QGIS.....	53
1. Intérêt d'une automatisation de la méthode	53
a. Principes et outils de l'automatisation	53
b. Cahier des charges pour l'automatisation de la méthode de modélisation des îlots de chaleur urbain	54

2. Le script QGIS : une étape intermédiaire	56
a. Structure générale du script.....	56
b. Définir les inputs et les outputs	58
c. Appeler les traitements de QGIS en python : la bibliothèque « processing »	59
d. Manipuler des « objets QGIS » en python	60
e. Usage de la calculatrice raster de QGIS en python	61
3. Du script au plug-in QGIS.....	64
a. Quels outils pour quels ajouts ?	64
Qu'est-ce qu'un plug-in ?	64
b. Développer un plug-in QGIS : les outils	65
c. Les étapes du développement.....	66
Création d'un template via « plug-in builder »	66
Conception d'une interface graphique sous Qt Designer	68
Adapter le script à l'interface et l'interface au script	71
Communiquer avec l'utilisateur	78
4. Discussion : constats et limites	81
a. Exactitude des résultats produits.....	81
b. Compatibilité avec les différentes versions de QGIS.....	82
c. Les temps de traitements.....	84
d. Remarques et discussion.....	85
Conclusion.....	87
Bibliographie.....	89
Table des matières.....	92
Table des figures.....	94

Table des figures

Figure 1 : Grandes caractéristiques du territoire d'étude.	10
Figure 2 : L'organisation générale du projet MApUCE.....	12
Figure 3 : Diagramme de Gantt envisagé en début de stage.....	13
Figure 4 : Profil d'un îlot de chaleur urbain.	15
Figure 5 : Facteurs pouvant entrer en compte dans la formation des ICU	16
Figure 6 : Hiérarchisation des facteurs entrant en compte dans la formation des ICU.....	17
Figure 7 : Illustration du critère des moindres carrés	22
Figure 8 : Exemple de semi-variogramme	24
Figure 9 : Arbre de décision pour la sélection d'un modèle de prédiction spatiale	27
Figure 10 : Fonctionnement de la méthode de regression-kriging appliquée à la méthode de modélisation des ICU	29
Figure 11 : Récapitulatif des variables retenues pour la construction de la fraction de ville	30
Figure 12 : Les USR : Vue générale et zoom sur le centre-ville	31
Figure 13 : Vue d'ensemble des données utilisées	33
Figure 14 : Configuration de l'algorithme Multiple Regression Analysis sous SAGA GIS 2.3.1	34
Figure 15 : Résultat obtenu par régression linéaire multiple, selon la configuration de la figure précédente	35
Figure 16 : Configuration de l'algorithme Simple Kriging sous SAGA GIS 2.3.1	36
Figure 17 : Fenêtre "Variogram" lors de l'usage de l'algorithme Simple Kriging, sous SAGA GIS (.....	37
Figure 18 : Résultat obtenu suite au krigeage simple des résidus issus de la régression linéaire	37
Figure 19 : Résultat de l'addition entre le raster de résidus krigés et le résultat de la régression linéaire	38
Figure 20 : Soustraction entre le raster issu de la regression linéaire et le résultat final avec les résidus krigés.....	38
Figure 21 : Configuration de l'algorithme Regression Kriging sous SAGA GIS 2.3.1	39
Figure 22 : Comparaison des résultats obtenus selon les deux méthodes testées	40
Figure 23 : Résultat de la soustraction raster entre les résultats finals obtenu selon les deux méthodes	40
Figure 24 : Horaires retenus pour la modélisation de l'ICU	42
Figure 25 : Température de l'air à Toulouse à 16H UTC pour la période été, a 100m de résolution spatiale	43
Figure 26 : Température de l'air à Toulouse à 05H UTC pour la période hiver, a 100m de résolution spatiale	44
Figure 27 : Raster obtenu après interpolation sur les données NETATMO non filtrée à 18H	46
Figure 28 : Nombre de stations NETATMO par horaire modélisé.....	47
Figure 29 : Température de l'air à Toulouse à 05H UTC le 14/12/16 (automne), à 100m de résolution spatiale, données NETATMO	48
Figure 30 : . Température de l'air à Toulouse à 16H UTC le 16/08/16 (été), à 100m de résolution spatiale, données NETATMO	48
Figure 31 : Indicateurs pour l'étude des résultats obtenus en cross-validation à partir des données NETATMO	49
Figure 32 : Les établissements "sensibles" de Toulouse par rapport à l'ICU	51
Figure 33 : L'éditeur de scripts de QGIS 2.14	56

Figure 34 : Description des principales étapes réalisées au cours du déroulement du script.....	57
Figure 35 : Définition des inputs et outputs via l'éditeur de script de QGIS, en python.....	58
Figure 36 : Principales classes et méthodes utilisées dans la conception du script sous l'éditeur de QGIS	62
Figure 37 : Model-builder du script QGIS	63
Figure 38 : L'interface de téléchargement et d'installation des plug-in QGIS	64
Figure 39 : Extrait de l'interface du "Plug-in Builder"	65
Figure 40 : Exemples de fenêtres issues de "Plug-in Builder" permettant de générer les fichiers de base pour le développement d'un plug-in	66
Figure 41 : Les fichiers générés par Plug-in Builder	67
Figure 42 : Compilation d'un fichier ressources.qrc dans l'OSGeo4W Shell	67
Figure 43 : Extrait de l'interface durant sa conception, sous Qt Designer	68
Figure 44 : Rendu final des différents onglets, sous QGIS	69
Figure 45 : Tableau récapitulatif des widgets Qt Designer utilisés	70
Figure 46 : Impact dynamique de la checkBox sur l'interface	74
Figure 47 : Sélection d'un répertoire pour l'enregistrement d'un output en cliquant sur le bouton "parcourir"	75
Figure 48 : Exemple de configuration du plug-in avant lancement	76
Figure 49 : Exemples de messages utilisés pour communiquer avec l'utilisateur	79
Figure 50 : La barre de progression et le temps d'exécution qui s'affichent respectivement pendant et après l'exécution du plug-in, sous QGIS	80
Figure 51 : Compatibilité du plug-in avec les différentes versions de QGIS testées	83
Figure 52 : Temps de traitement en fonction des versions de QGIS	84
Figure 53 : Temps de traitements en secondes sur différentes configurations de test	85
Figure 54 : GANTT récapitulatif de l'organisation effective du stage	87

Index des acronymes et abréviations

ICU : Îlot de chaleur urbain

LISST-CIEU : Laboratoire Interdisciplinaire Solidarités, Sociétés, Territoires / Centre Interdisciplinaire d'Études Urbaines

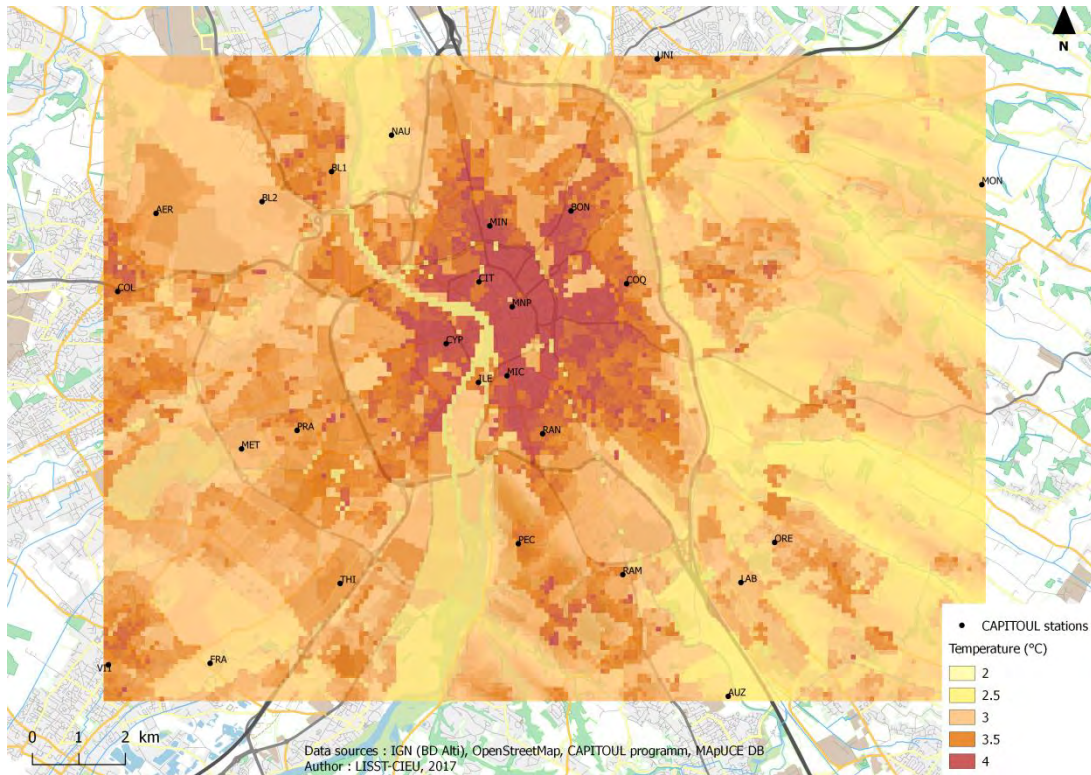
MApUCE : Modélisation Appliquée et Droit de l'Urbanisme : Climat et Energie

RK : Regression-Kriging

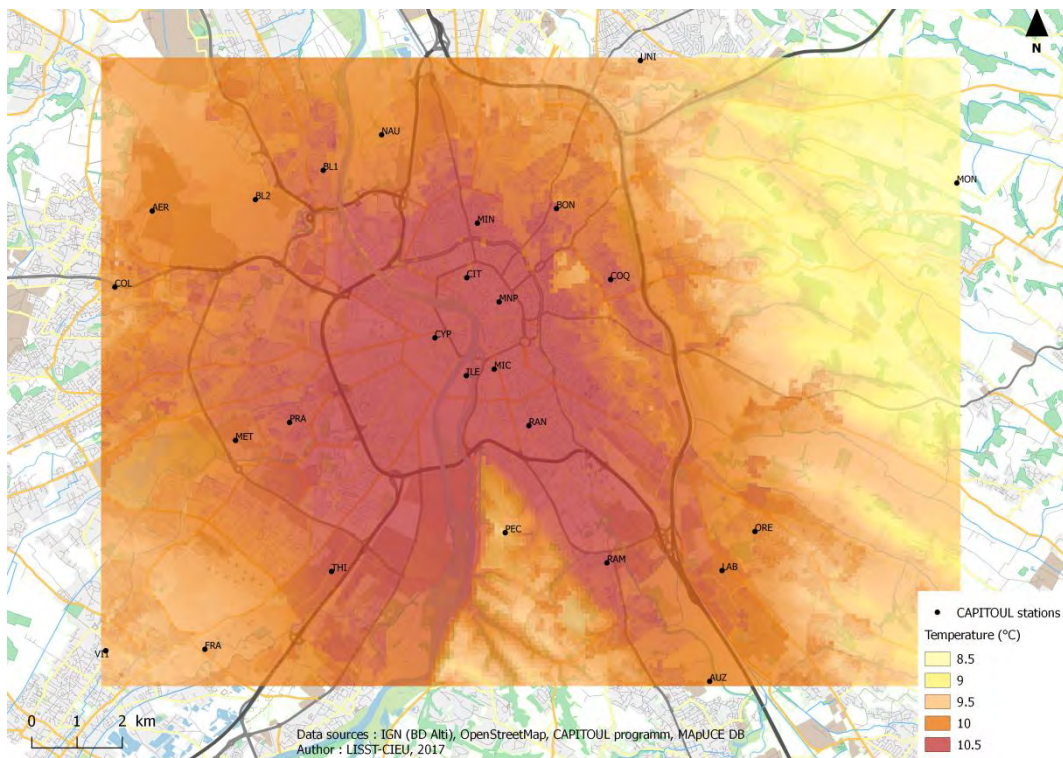
RL : Régression linéaire (LR en anglais)

Annexes

Annexe 1 : Cartes de températures modélisées aux dix horaires de test retenus (cf. p43)



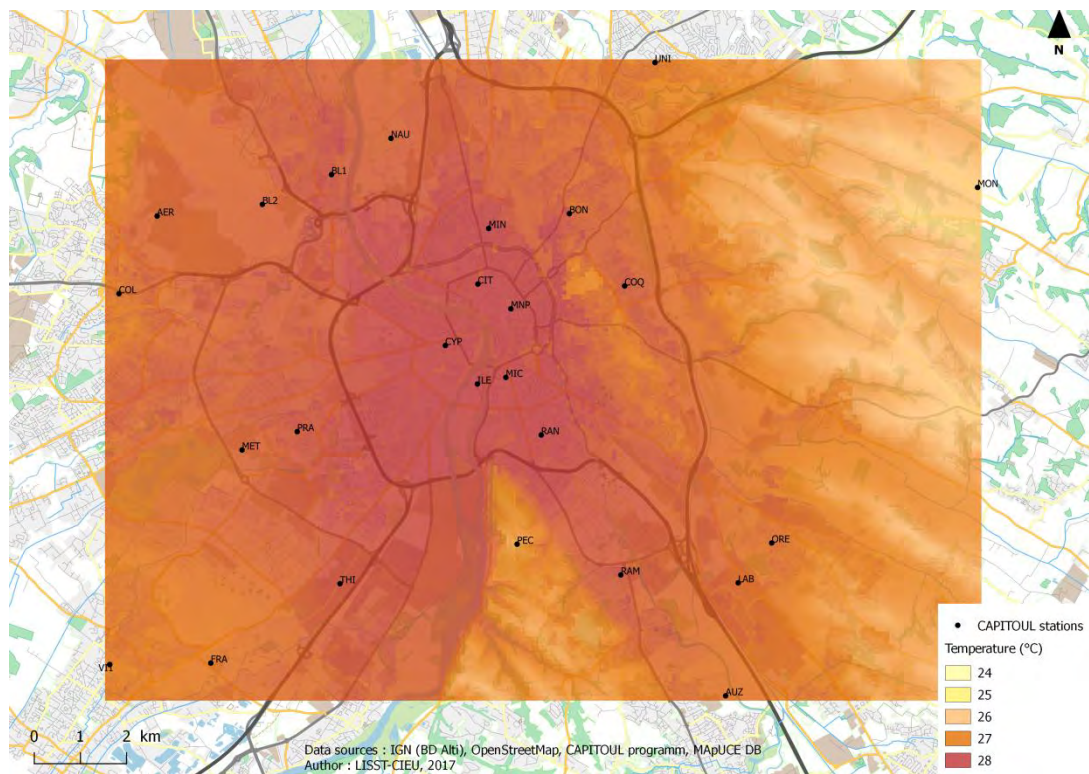
Température de l'air à Toulouse à 5H UTC, période djf à 100m de résolution spatiale



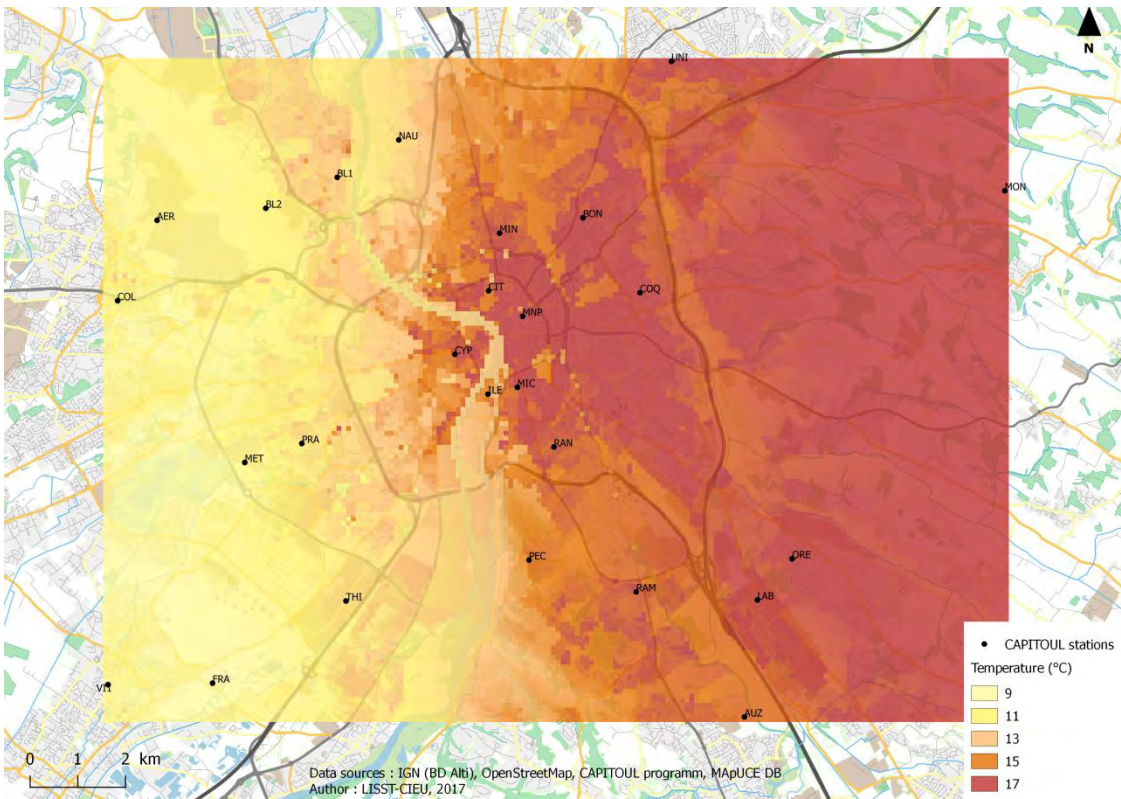
Température de l'air à Toulouse à 15H UTC, période djf à 100m de résolution spatiale



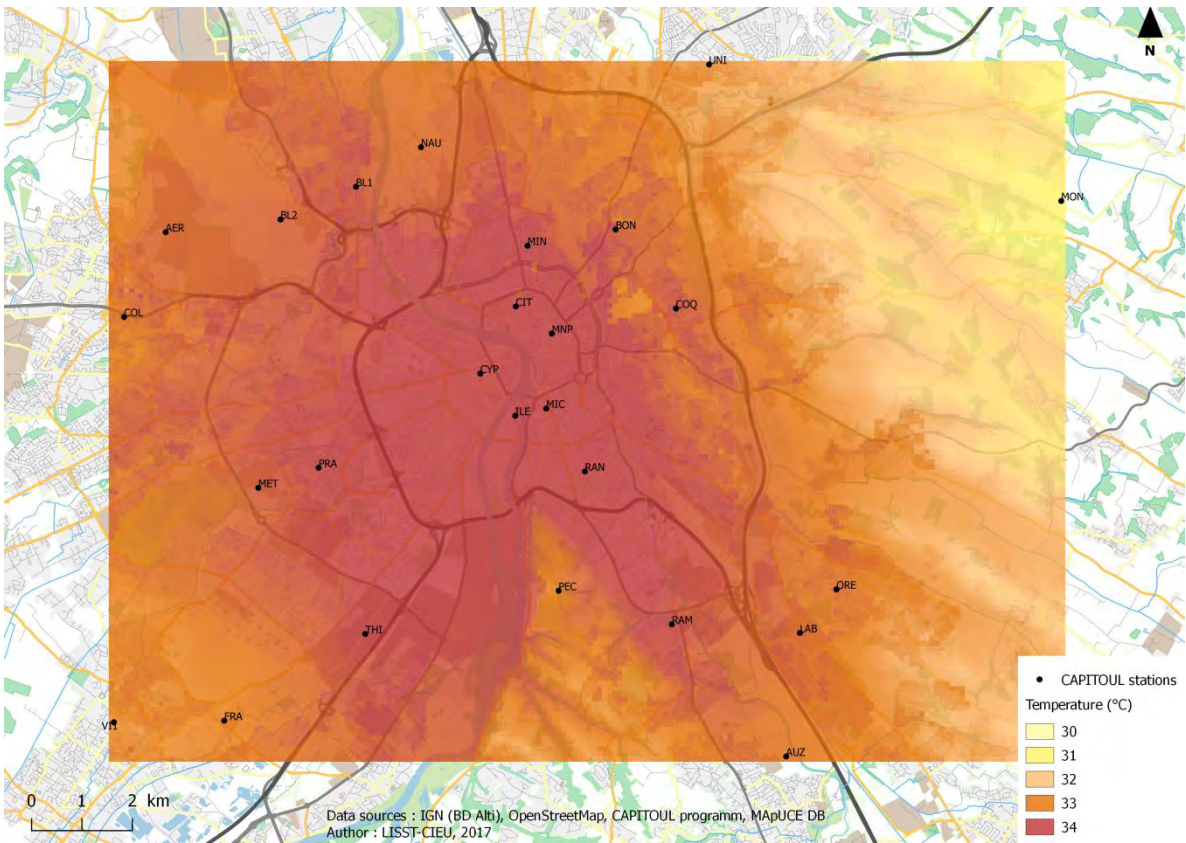
Température de l'air à Toulouse à 5H UTC, période jja à 100m de résolution spatiale



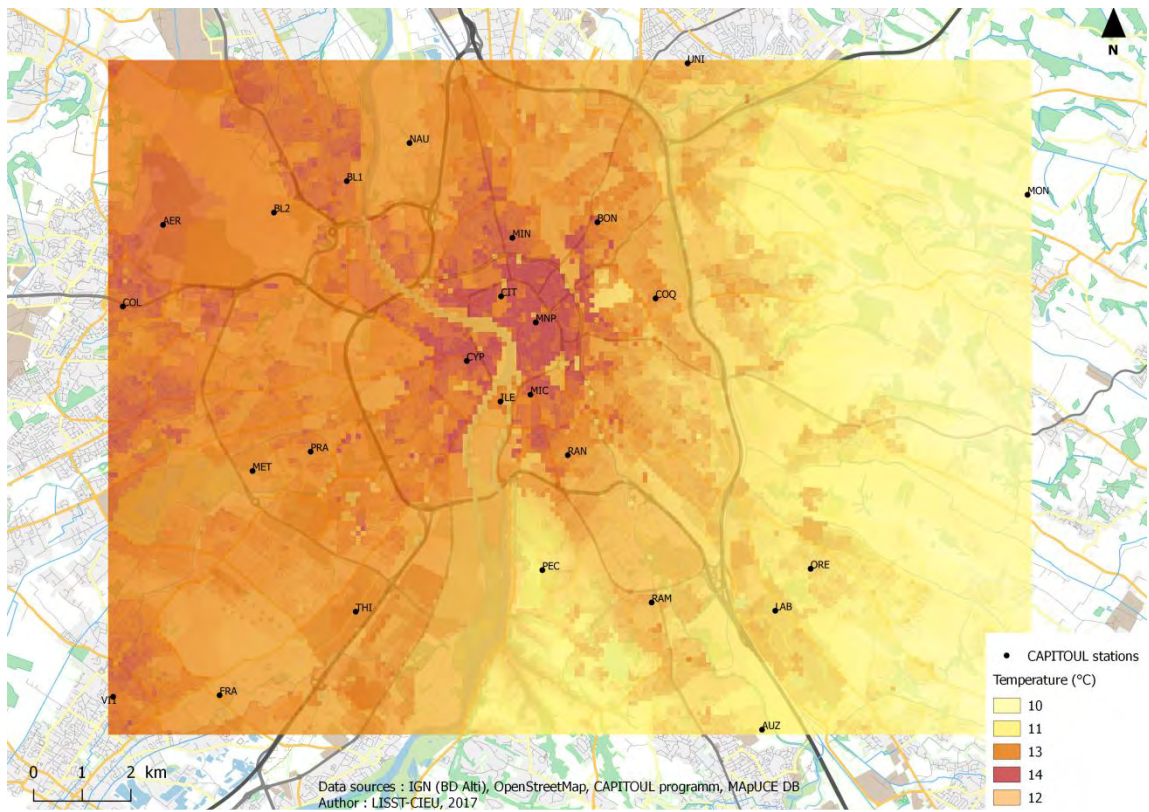
Température de l'air à Toulouse à 16H UTC, période jja à 100m de résolution spatiale



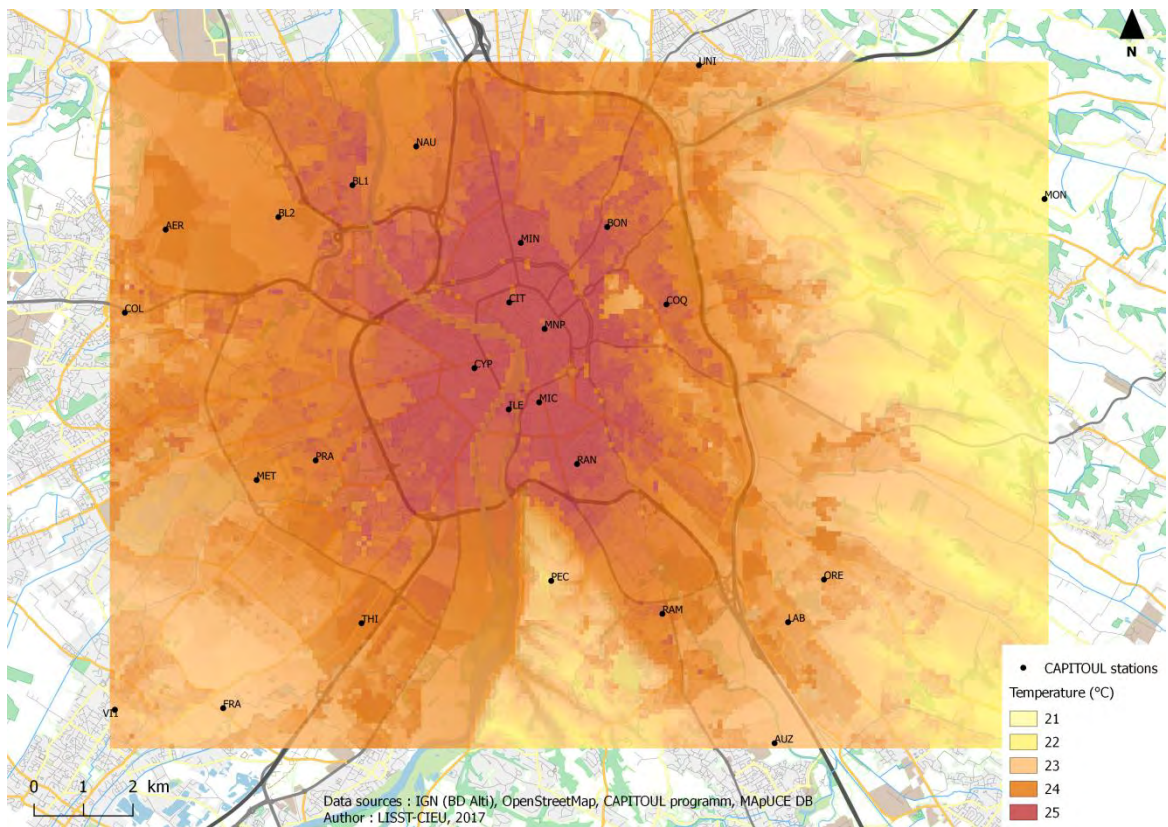
Température de l'air à Toulouse à 4H UTC, période jja chaud à 100m de résolution spatiale



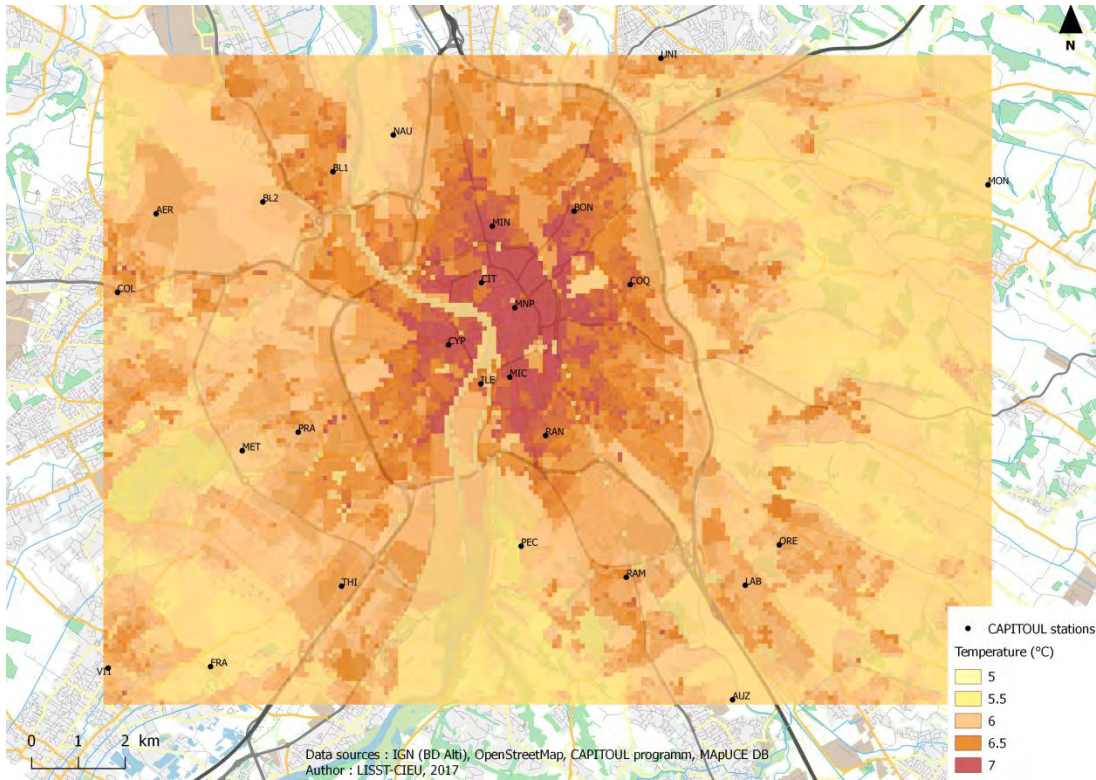
Température de l'air à Toulouse à 16H UTC, période jja chaud à 100m de résolution spatiale



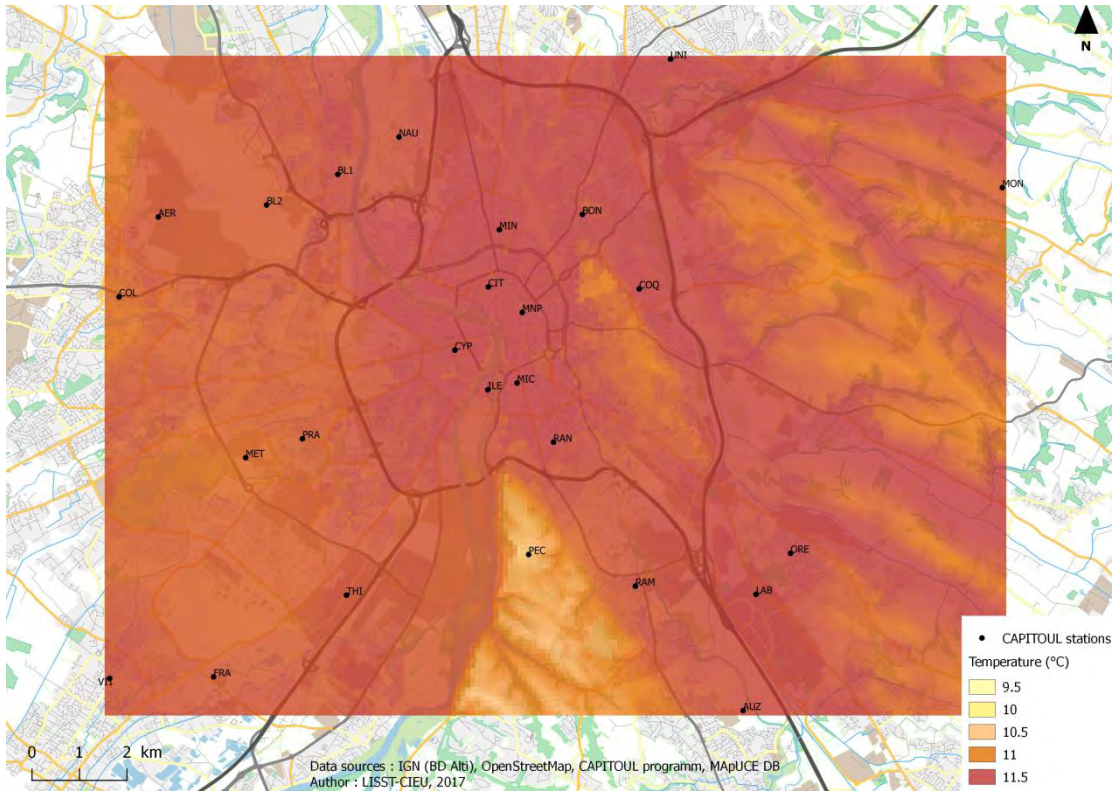
Température de l'air à Toulouse à 6H UTC, période mam à 100m de résolution spatiale



Température de l'air à Toulouse à 16H UTC, période mam à 100m de résolution spatiale



Température de l'air à Toulouse à 5H UTC, période son à 100m de résolution spatiale



Température de l'air à Toulouse à 14H UTC, période son à 100m de résolution spatiale

Annexe 2 : Aide du plug-in

Principe général

Ce plug-in se propose d'estimer par interpolation les températures sur un territoire donné. Les températures sont fournies par pixel, sous la forme d'un raster, dans le but de rendre compte de phénomènes comme l'îlot de chaleur urbain.

La méthode utilisée pour l'interpolation est le « regression-kriging », méthode qui se fonde sur une régression linéaire multiple suivie d'un krigeage des résidus.

La régression est effectuée à partir des données de température fournies ainsi que d'un modèle numérique de terrain et d'un raster dit de « fraction de ville ». Ce dernier est produit automatiquement par le plug-in à partir des données issues du programme de recherche MAPUCE, si vous disposez de ces données sur votre territoire. Les données MAPUCE peuvent être récupérées ici : [///A REMPLIR///](#).

La fraction de ville construite par le plug-in intègre trois variables rendant compte de l'artificialisation des sols : la densité de bâti, la densité de route et la surface d'enveloppe extérieure des bâtiments (qui permet d'intégrer la notion de volume du bâti dans le calcul).

Si vous ne souhaitez pas ou ne pouvez pas utiliser les données MAPUCE, il est possible d'utiliser une fraction de ville personnalisée, au format raster (cochez pour cela la case correspondante dans l'onglet configuration).

La régression linéaire produit notamment une couche de résidu, sur laquelle est ensuite appliquée un krigeage, qui est une méthode d'interpolation géostatistique favorisant notamment la prise en compte de la dimension spatiale d'un phénomène. Les rasters issus de la régression et du krigeage des résidus sont proposés en sortie. L'addition de ces deux rasters forme le raster « Interpolation », qui correspond au résultat final du « regression-kriging ».

L'emprise des rasters produits correspond à celle de la couche de points fournie (qui comporte les relevés de température). Les algorithmes utilisés pour la régression et le krigeage proviennent du logiciel libre SAGA GIS.

Les inputs

Trois couches sont nécessaires pour l'exécution du plug-in :

- Un **Modèle Numérique de Terrain**, ou autre raster fournissant une information relative à l'élévation. Celui-ci doit couvrir au minimum l'étendue de la zone à traiter (délimitée par la couche de points).
- Un **relevé de températures** : il s'agit d'une couche vectorielle de type points contenant, dans sa table attributaire, au moins un champ de température. Typiquement, cette couche correspond à un ensemble de stations de relevés de températures. L'emprise couverte par cette couche est celle qui sera utilisée dans les rasters produits.
- Une **fraction de ville** : il s'agit d'un raster contenant des informations sur la structure du bâti sur le territoire concerné. Le plug-in peut produire automatiquement ce raster à partir des données MAPUCE, si celles-ci couvrent la zone à traiter. Il utilise pour ce faire les variables ROAD_DENS (densité de route), BUILD_DENS (densité de bâti) et EXT_ENV_AR (surface de façade extérieure des bâtiments). Une fraction de ville personnalisée peut également être utilisée en lieu et place de celle produite à partir des données MAPUCE. Pour cela, cochez la case « utiliser une fraction de ville personnalisée » et indiquez un raster correspondant.

Les outputs

Quatre fichiers sont disponibles en sortie :

- **[INTERPOLATION]** : il s'agit du résultat final des traitements effectués par le plug-in. C'est un raster fournissant la température estimée pour chaque pixel.
- **[REGRESSION]** : Résultat intermédiaire : il s'agit du raster produit par régression linéaire à partir des informations (températures, MNT, fraction de ville) fournies. Il n'intègre pas le krigeage des résidus.
- **[RESIDUS]** : Résultat intermédiaire : Il s'agit des résidus issus de la régression linéaire auxquels a été appliqués un krigeage ordinaire. Cette méthode d'interpolation permet d'estimer ces résidus sur l'ensemble du territoire à traiter et de les intégrer dans le calcul final (raster Interpolation)
- **[MODELE]** : une table au format csv contenant des indicateurs mathématiques sur la qualité de la régression linéaire effectuée, notamment le R^2 , le R^2 ajusté, l'erreur quadratique moyenne...

Détail des algorithmes utilisés :

Les rasters de MNT et de fraction de ville sont découpés et mis à une même résolution à l'aide des algorithmes [*Clip Raster by Extent*] de GDAL/OGR et [*Resampling*] de SAGA GIS.

Dans le cas où les données MAPUCE sont utilisées, un raster est déterminé à partir de ces dernières en utilisant l'algorithme [*Rasterize*] de GDAL/OGR puis la calculatrice raster de QGIS.

Une régression linéaire multiple est effectuée via l'algorithme [*Multiple Regression Analysis (point/grids)*] de SAGA GIS, utilisant en entrée les rasters MNT et fraction de ville ainsi que les relevés de température. Les résidus issus de cette régression font ensuite l'objet d'un krigeage via l'algorithme [*Ordinary Kriging (global)*] de SAGA GIS. Le raster issu de ce krigeage et celui issu de la régression linéaire sont ensuite additionnés via la calculatrice raster de QGIS.

Résumé

L'îlot de chaleur urbain (ICU) est un phénomène caractérisé par une élévation localisée des températures que l'on observe généralement dans un milieu urbain par rapport à une périphérie plus rurale.

Bien que ses causes et ses conséquences soient assez bien connues, l'ICU reste difficile à modéliser sur un territoire donné. Sa représentation nécessite en effet une connaissance précise des températures à une résolution spatiale la plus fine possible.

Dans le cadre du projet de recherche ANR MApUCE (Modélisation appliquée et droit de l'urbanisme : climat urbain et énergie), le laboratoire LISST-CIEU de l'université Toulouse 2 Jean Jaurès s'est intéressé à une méthode géostatistique de modélisation de l'ICU, basée sur la technique du « regression-kriging ».

En utilisant notamment un ensemble de variables développées durant le projet MApUCE et caractérisant le tissu urbain, il est ainsi possible d'estimer l'ICU par interpolation avec une précision acceptable.

Ce mémoire se base sur un stage de master 2 réalisé au LISST-CIEU avec pour objectifs d'affiner et d'automatiser la méthode de modélisation de l'ICU.

Une première partie décrit ainsi le fonctionnement, les atouts et les limites de la méthode d'interpolation par regression-kriging appliquée aux ICU. Le cas d'étude développé est celui de la ville de Toulouse, avec une résolution spatiale de 100m.

La seconde partie s'attache quant à elle à décrire le processus d'automatisation sous la forme d'un plug-in QGIS codé en python, avec une interface réalisée sous Qt designer.

Mots-clés : Géostatistique, îlot de chaleur urbain, interpolation, plug-in, Python, QGIS, regression-kriging

Abstract

The urban heat island (UHI) is a phenomenon characterized by a localized elevation of temperatures that is generally observed in an urban environment compared to a more rural periphery.

Even if its causes and consequences are fairly well known, the UHI remains difficult to model on a given territory. Its representation requires precise knowledge of the temperatures at the finest possible spatial resolution. For the research project ANR MApUCE, the laboratory LISST-CIEU of Toulouse 2 Jean Jaurès University was interested in a geostatistical method for modeling ICU, based on regression-kriging. Using a set of variables developed during the MApUCE project and characterizing the land covering, it is possible to estimate the ICU by interpolation with an acceptable precision.

This document is based on a Master 2 internship at LISST-CIE. Its objectives was to refine and automate the methodology of UHI modeling.

A first part describes the operation, advantages and limitations of the regression-kriging interpolation method applied to the UHI. The study case developed is the city of Toulouse France, with a 100m spatial resolution .

The second part describes the automation process in the form of a QGIS plug-in coded in python, with an interface made on Qt designer.

Keywords : Geostatistics, interpolation, plug-in, python, QGIS, regression-kriging, Urban Heat Island