

Samuel Foucher, Philippe Apparicio, Mickaël Germain, Yacine Bouroubi, Étienne Clabaut

# Traitement d'images satellites avec Python

Première édition

Samuel Foucher

Philippe Apparicio

Yacine Bouroubi

Mickaël Germain

Étienne Clabaut

2025-03-22

# Table des matières

<b>Préface</b>	<b>1</b>
Un manuel sous la forme d'une ressource éducative libre . . . . .	2
Comment lire ce manuel? . . . . .	4
Comment utiliser les données du livre pour reproduire les exemples? . . . . .	4
Structure du livre . . . . .	5
Remerciements . . . . .	5
Introduction aux images de télédétection . . . . .	6
Ressources en ligne . . . . .	6
Listes des <i>librairies</i> utilisés . . . . .	6
<b>À propos des auteurs</b>	<b>7</b>
<b>Partie 1. Importation, manipulation et visualisation de données spatiales</b>	<b>8</b>
<b>1 Introduction au langage Python</b>	<b>9</b>
1.1 Les distributions . . . . .	9
1.2 Les styles de programmation en Python . . . . .	10
1.2.1 Les outils de programmation . . . . .	10
1.3 Bonnes pratiques . . . . .	11
1.3.1 Création d'un environnement virtuel . . . . .	11
1.3.2 Création d'un environnement de travail local (avancé) . . . . .	11
1.4 Les structures de base en Python . . . . .	13
1.4.1 Les listes . . . . .	13
1.4.2 Les tuples . . . . .	13
1.4.3 Les ensembles (Sets) . . . . .	13
1.5 Dictionnaires . . . . .	13
1.6 Programmation objet . . . . .	14
<b>2 Importation et manipulation de données spatiales</b>	<b>15</b>
2.1 Préambule . . . . .	15
2.1.1 Objectifs . . . . .	15
2.1.2 Bibliothèques . . . . .	15
2.1.3 Données . . . . .	16
2.2 Importation d'images . . . . .	16
2.2.1 Formats des images . . . . .	17
2.2.2 Métadonnées des images . . . . .	21
2.3 Manipulation des images . . . . .	22
2.3.1 Manipulation de la matrice de pixels . . . . .	22

2.3.2	Information de base . . . . .	22
2.3.3	Découpage et indexation de la matrice . . . . .	24
2.3.4	Changement de projection cartographique (à venir) . . . . .	26
2.4	Données en géoscience . . . . .	26
2.4.1	xarray . . . . .	26
<b>3</b>	<b>Réhaussement et visualisation d'images</b>	<b>28</b>
3.1	Préambule . . . . .	28
3.1.1	Objectifs . . . . .	28
3.1.3	Bibliothèques . . . . .	28
3.1.4	Données . . . . .	29
3.2	Visualisation en Python . . . . .	30
3.3	Réhaussements visuels . . . . .	33
3.3.1	Statistiques d'une image . . . . .	33
3.3.2	Réhaussements linéaires . . . . .	37
3.3.3	Réhaussements non linéaires . . . . .	41
3.3.4	Composés colorés . . . . .	46
<b>Partie 2. Transformations des données satellitaires</b>		<b>48</b>
<b>4</b>	<b>Transformations spectrales</b>	<b>49</b>
4.1	Préambule . . . . .	49
4.1.1	Objectifs . . . . .	49
4.1.2	Librairies . . . . .	49
4.1.3	Images utilisées . . . . .	50
4.2	Qu'est ce que l'information spectrale? . . . . .	51
4.3	Indices spectraux . . . . .	52
<b>5</b>	<b>Transformations spatiales</b>	<b>58</b>
5.1	Préambule . . . . .	58
5.1.1	Objectifs . . . . .	58
5.1.2	Librairies . . . . .	58
5.1.3	Images utilisées . . . . .	59
5.2	Analyse fréquentielle . . . . .	60
5.2.1	La transformée de Fourier . . . . .	60
5.2.2	Filtrage fréquentiel . . . . .	61
5.2.3	L'aliasing . . . . .	62
5.3	Filtrage d'image . . . . .	65
5.3.1	Filtrage linéaire stationnaire . . . . .	66
5.4	Gestion des bordures . . . . .	69
5.4.1	Filtrage adaptatif . . . . .	71
5.5	Segmentation . . . . .	74
5.5.1	Super-pixel . . . . .	75
5.5.2	Fusion des segments par graphe de proximité . . . . .	78
5.5.3	Approche objet . . . . .	81

<b>Partie 3. Classifications d'images</b>	<b>83</b>
<b>6 Classifications d'images supervisées</b>	<b>84</b>
6.1 Préambule . . . . .	84
6.1.1 Objectifs . . . . .	84
6.1.2 Librairies . . . . .	84
6.1.3 Images utilisées . . . . .	85
6.2 Principes généraux . . . . .	86
6.2.1 Comportement d'un modèle . . . . .	86
6.2.2 Pipeline . . . . .	87
6.2.3 Construction d'un ensemble d'entraînement . . . . .	87
6.3 Analyse préliminaire des données . . . . .	94
6.4 Mesures de performance d'une méthode de classification . . . . .	98
6.5 Méthodes non paramétriques . . . . .	99
6.5.1 Méthode des parallélépipèdes . . . . .	99
6.5.2 Plus proches voisins . . . . .	102
6.5.3 Méthodes par arbre de décision . . . . .	107
6.6 Méthodes paramétriques . . . . .	111
6.6.1 Méthode Bayésienne naïve . . . . .	111
6.6.2 Analyse discriminante quadratique (ADQ) . . . . .	113
<b>Bibliographie</b>	<b>115</b>

# Liste des Figures

1	Licence Creative Commons du livre . . . . .	3
2	Téléchargement de l'intégralité du livre . . . . .	5
1.1	Client Jupyter Lab . . . . .	12
2.1	La librairie NumPy est le fondement de nombreuses bibliothèques scientifiques (d'après (Harris 2020)) . . . . .	23
2.2	Vue d'ensemble des opérations de base des matrices avec NumPy . . . . .	24
2.3	Organisation d'un Dataset dans xarray . . . . .	27
4.1	Positions des bandes spectrales pour quelques capteurs ( <a href="#">source</a> ) . . . . .	51
4.2	Visualisation des points d'une image Sentinel-2 pour trois classes . . . . .	57
5.1	Graphe d'adjacence de régions, d'après (Jaworek-Korjakowska (2018)). Chaque nœud est un segment, un lien est formé uniquement si les segments se touchent (par exemple le segment 6 ne touche que la région 5). La fonction <code>graph.rag_mean_color</code> produit un graphe à partir d'une segmentation et de l'image originale. Chaque nœud tient la couleur de chaque segment dans un attribut appelé ' <code>mean color</code> ' . . . . .	79
6.1	Exemples de sur et sous-apprentissage. . . . .	87
6.2	Étapes standards dans un entraînement. . . . .	88
6.3	Frontières de décision pour le classificateur K-NN . . . . .	104
6.4	Frontières de décision pour des arbres de décision de différente profondeur . . . . .	108
6.5	Frontières de décision pour un classificateur Bayésien naïf . . . . .	112

# Liste des Tables

2.1	Type de données de NumPy . . . . .	24
4.1	Noms des bandes Sentinel-2 . . . . .	53

# Préface

% Réinitialiser partie

**Résumé :** Ce livre vise à décrire une panoplie de méthodes de traitement d'images satellites avec le langage Python. Celles et ceux souhaitant migrer progressivement d'un autre logiciel d'imagerie et de télédétection vers Python trouveront dans cet ouvrage les éléments pour une transition en douceur. La philosophie de ce livre est de donner toutes les clefs de compréhension et de mise en œuvre des méthodes abordées dans Python. La présentation des méthodes est basée sur une approche compréhensive et intuitive plutôt que mathématique, sans pour autant négliger la rigueur mathématique ou statistique. Des rappels sur les fondements en télédétection pourront apparaître au besoin afin d'éclairer les approches techniques. Plusieurs éditions régulières sont prévues sachant que ce domaine évolue constamment.

Ce projet est en cours d'écriture et le contenu n'est pas complet



**Remerciements :** Ce manuel a été réalisé avec le soutien de la fabriqueREL. Fondée en 2019, la fabriqueREL est portée par divers établissements d'enseignement supérieur du Québec et agit en collaboration avec les services de soutien pédagogique et les bibliothèques. Son but est de faire des ressources éducatives libres (REL) le matériel privilégié en enseignement supérieur au Québec.

**Mise en page :** Samuel Foucher, Philippe Apparicio et Marie-Hélène Gadbois Del Carpio.

© Samuel Foucher, Philippe Apparicio, Yacine Bouroubi et Mickaël Germain.

**Pour citer cet ouvrage :** Foucher S., Apparicio P., Bouroubi Y., Germain M. et Clabaut, E. (2025). *Traitement d'images satellites avec Python*. Université de Sherbrooke, Département de géomatique appliquée. fabriqueREL. Licence CC BY-SA.



Sauf indications contraires, le contenu de ce manuel électronique est disponible en vertu des termes de la [Licence Creative Commons Attribution - Partage dans les mêmes conditions 4.0 International](#).

**Vous êtes autorisé·e à :**

**Partager** – copier, distribuer et communiquer le matériel par tous moyens et sous tous formats.

**Adapter** – remixer, transformer et créer à partir du matériel pour toute utilisation, y compris commerciale.

**Selon les conditions suivantes :**

**Paternité** – Vous devez citer le nom des auteurs originaux.

**Mêmes conditions** – Si vous remixez, transformez, ou créez à partir du matériel composant l'œuvre originale, vous devez diffuser l'œuvre modifiée avec la même licence.



Université de  
Sherbrooke



fabrique **REL**  
RESSOURCES ÉDUCATIVES LIBRES

## Un manuel sous la forme d'une ressource éducative libre

### Pourquoi un manuel sous licence libre?

Les logiciels libres sont aujourd’hui très répandus. Comparativement aux logiciels propriétaires, l'accès au code source permet à quiconque de l'utiliser, de le modifier, de le dupliquer et de le partager. Le logiciel Python, dans lequel sont mises en œuvre les méthodes de traitement d'images satellites décrites dans ce livre, est d'ailleurs à la fois un langage de programmation et un logiciel libre (sous la licence publique générale [GNU GPL2](#)). Par analogie aux logiciels libres, il existe aussi des **ressources éducatives libres (REL)** « dont la licence accorde les permissions désignées par les 5R (**Retenir — Réutiliser — Réviser — Remixier — Redistribuer**) et donc permet nécessairement la modification » ([fabriqueREL](#)). La licence de ce livre, CC BY-SA (figure 1), permet donc de :

- **Retenir**, c'est-à-dire télécharger et imprimer gratuitement le livre. Notez qu'il aurait été plutôt surprenant d'écrire un livre payant sur un logiciel libre et donc gratuit. Aussi, nous aurions été très embarrassés que des personnes étudiantes avec des ressources financières limitées doivent payer pour avoir accès au livre, sans pour autant savoir préalablement si le contenu est réellement adapté à leurs besoins.
- **Réutiliser**, c'est-à-dire utiliser la totalité ou une section du livre sans limitation et sans compensation financière. Cela permet ainsi à d'autres personnes enseignantes de l'utiliser dans le cadre d'activités pédagogiques.
- **Réviser**, c'est-à-dire modifier, adapter et traduire le contenu en fonction d'un besoin pédagogique précis puisqu'aucun manuel n'est parfait, tant s'en faut! Le livre a d'ailleurs été écrit intégralement dans Python avec [Quattro](#). Quiconque peut ainsi télécharger gratuitement le code source du livre sur [GitHub](#) et le modifier à sa guise (voir l'encadré intitulé *Suggestions d'adaptation du manuel*).
- **Remixer**, c'est-à-dire « combiner la ressource avec d'autres ressources dont la licence le permet aussi pour créer une nouvelle ressource intégrée » ([fabriqueREL](#)).
- **Redistribuer**, c'est-à-dire distribuer, en totalité ou en partie le manuel ou une version révisée sur d'autres canaux que le site Web du livre (par exemple, sur le site Moodle de votre université ou en faire une version imprimée).

La licence de ce livre, CC BY-SA (figure 1), oblige donc à :

- Attribuer la paternité de l'auteur dans vos versions dérivées, ainsi qu'une mention concernant les grandes modifications apportées, en utilisant la formulation suivante :

Samuel Foucher, Apparicio Philippe, Mickaël Germain, Yacine Bouroubi et Étienne Clabaut (2025). *Traitements d'images satellites avec Python*. Université de Sherbrooke, Département de géomatique appliquée. fabriqueREL. Licence CC BY-SA.

- Utiliser la même licence ou une licence similaire à toutes versions dérivées.

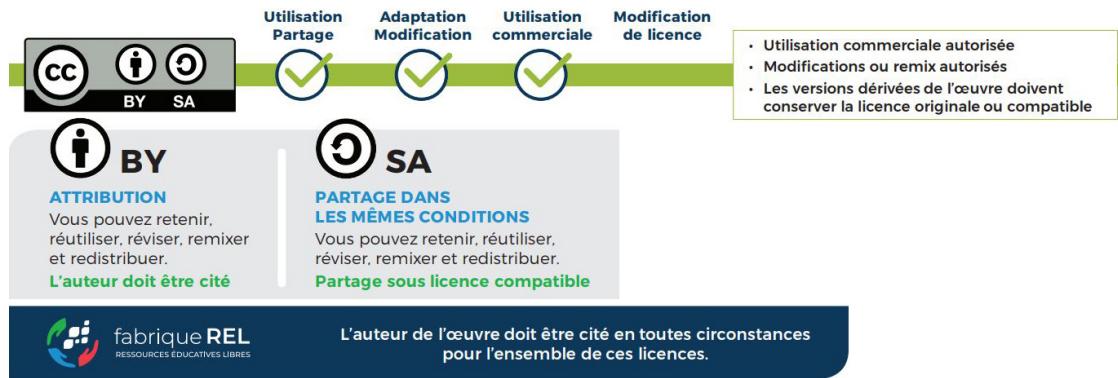


FIGURE 1 – Licence Creative Commons du livre

## Astuce

### Suggestions d'adaptation du manuel

Pour chaque méthode de traitement d'image abordée dans le livre, une description détaillée et une mise en œuvre dans Python sont disponibles. Par conséquent, plusieurs adaptations du manuel sont possibles :

- Conserver uniquement les chapitres sur les méthodes ciblées dans votre cours.
- En faire une version imprimée et la distribuer aux personnes étudiantes.
- Modifier la description d'une ou de plusieurs méthodes en effectuant les mises à jour directement dans les chapitres.
- Insérer ses propres jeux de données dans les sections intitulées *Mise en œuvre dans Python*.
- Modifier les tableaux et figures.
- Ajouter une série d'exercices.
- Modifier les quiz de révision.
- Rédiger un nouveau chapitre.
- Modifier des syntaxes en Python. Plusieurs *librairies* Python peuvent être utilisées pour mettre en œuvre telle ou telle méthode. Ces derniers évoluent aussi très vite et de nouvelles *librairies* sont proposées fréquemment! Par conséquent, il peut être judicieux de modifier une syntaxe Python du livre en fonction de ses habitudes de programmation en Python (utilisation d'autres *librairies* que ceux utilisés dans le manuel par exemple) ou de bien mettre à jour une syntaxe à la suite de la parution d'une nouvelle *librairie* plus performante ou intéressante.
- Toute autre adaptation qui permet de répondre au mieux à un besoin pédagogique.

## Comment lire ce manuel?

Le livre comprend plusieurs types de blocs de texte qui en facilitent la lecture.

### Package

#### Bloc *packages*

Habituellement localisé au début d'un chapitre, il comprend la liste des *packages* Python utilisés pour un chapitre.

### Objectif

#### Bloc objectif

Il comprend une description des objectifs d'un chapitre ou d'une section.

### Note

#### Bloc notes

Il comprend une information secondaire sur une notion, une idée abordée dans une section.

### Aller plus loin

#### Bloc pour aller plus loin

Il comprend des références ou des extensions d'une méthode abordée dans une section.

### Astuce

#### Bloc astuce

Il décrit un élément qui vous facilitera la vie : une propriété statistique, un *package*, une fonction, une syntaxe Python.

### Attention

#### Bloc attention

Il comprend une notion ou un élément important à bien maîtriser.

### Exercice

#### Bloc exercice

Il comprend un court exercice de révision à la fin de chaque chapitre.

## Comment utiliser les données du livre pour reproduire les exemples?

Ce livre comprend des exemples détaillés et appliqués en Python pour chacune des méthodes abordées. Ces exemples se basent sur des jeux de données ouverts et mis à disposition avec le livre. Ils sont disponibles sur le *repo GitHub* dans le sous-dossier **data**, à l'adresse <https://github.com/serie-tele-pyton/TraitementImagesVol1/tree/main/data>.

## Structure du livre

Une autre option est de télécharger le *repo* complet du livre directement sur *GitHub* (<https://github.com/serie-tele-pyton/TraitementImagesVol1>) en cliquant sur le bouton **Code**, puis le bouton **Download ZIP** (figure 2). Les données se trouvent alors dans le sous-dossier nommé **data**.

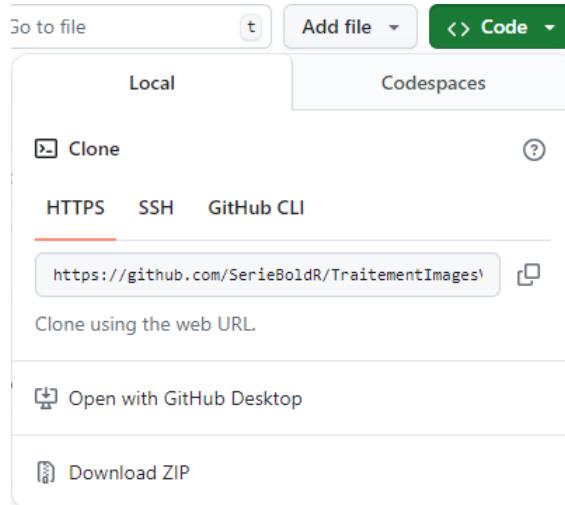


FIGURE 2 – Téléchargement de l'intégralité du livre

## Structure du livre

Le livre est organisé autour de quatre grandes parties.

**Partie 1. Importation et manipulation de données spatiales.** Dans cette première partie, nous voyons comment importer, manipuler, visualiser et exporter des données spatiales de type image (ou de type matriciel) avec Python, principalement avec les *packages rasterio*, *xarray* et *numpy* (chapitre 2). Ce chapitre vous permettra de maîtriser la manipulation à bas niveau de différents types d'imagerie. Différents exemples et exercices sont disponibles avec différents capteurs satellites (multi-spectral, RGB-NIR, SAR, etc.)

**Partie 2. Transformations des données spatiales.** Cette deuxième partie comprend deux chapitres : les transformations spectrales (chapitre 4) et les transformations spatiales (chapitre 5).

**Partie 3. Classifications d'images.** Cette troisième partie comprend deux chapitres : les classifications supervisées (chapitre 6) et non supervisées (à venir dans une prochaine édition).

**Partie 4. Données massives.** (à venir dans une édition future). Cette quatrième et dernière partie comprend un seul chapitre qui est dédié aux plateformes de mégadonnées, notamment Google Earth Engine.

## Remerciements

De nombreuses personnes ont contribué à l'élaboration de ce manuel.

Ce projet a bénéficié du soutien pédagogique et financier de la **fabriqueREL** (ressources éducatives libres). Les différentes rencontres avec le comité de suivi nous ont permis de comprendre l'univers des ressources éducatives

libres (REL) et notamment leurs **fameux 5R** (Retenir — Réutiliser — Réviser — Remixer — Redistribuer), de mieux définir le besoin pédagogique visé par ce manuel, d'identifier des ressources pédagogiques et des outils pertinents pour son élaboration. Ainsi, nous remercions chaleureusement les membres de la *fabriqueREL* pour leur soutien inconditionnel :

- José-Miguel Escobar-Zuniga, bibliothécaire à l'Université de Sherbrooke.
- Marianne Dubé, coordonnatrice de la *fabriqueREL*, Université de Sherbrooke.
- Claude Potvin, conseiller en formation, Service de soutien à l'enseignement, Université Laval.

Nous remercions chaleureusement les personnes étudiantes du **Baccalauréat en géomatique appliquée à l'environnement** et du **Microprogramme de 1er cycle en géomatique appliquée** du Département de géomatique appliquée de l'**Université de Sherbrooke** qui utilisent le manuel dans le cadre de cours.

## **Introduction aux images de télédétection**

L'imagerie numérique a pris une place importante dans notre vie de tous les jours depuis une quinzaine d'années. Ces images sont prises généralement au niveau du sol (imagerie proximale) avec seulement trois couleurs dans le domaine de la vision humaine (rouge, vert et bleu). Dans la suite du manuel, on parlera d'images du domaine de la vision par ordinateur ou images en vision pour faire plus court.

Les images de télédétection ont des particularités et des propriétés qui les différencient des images “classiques”, dont les cinq principales sont:

1. Les images sont géoréférencées. Cela signifie que pour chaque pixel nous pouvons y associer une position géographique ou cartographique.
2. Le point de vue est très différent. Ces images sont prises avec une vue d'en haut (Nadir) ou oblique avec une distance qui peut être très grande; on parle d'images distales.
3. Elles possèdent plus que 3 bandes. Contrairement aux images en vision, les images de télédétection possèdent bien souvent plus que trois bandes. Il n'est pas rare de trouver quatre bandes (Pléiade), 13 bandes (Sentinel-2, Landsat) et même 200 bandes pour des capteurs hyperspectraux.
4. Elles peuvent être calibrées. Les valeurs numériques de l'image peuvent être converties en quantités physiques (luminance, réflectance, section efficace, etc.) via une fonction de calibration.
5. Elles sont de grande taille. Il n'est pas rare de manipuler des images qui font plusieurs dizaines de milliers de pixels en dimension.

## **Ressources en ligne**

### **Listes des *librairies* utilisés**

Dans ce livre, nous utilisons de nombreux *packages* Python que vous pouvez installer en une seule fois (voir section 1.3.1) ou chapitre par chapitre.

# À propos des auteurs

**Samuel Foucher** est professeur au Département de géomatique appliquée de l'Université de Sherbrooke. Il y enseigne aux programmes de 1<sup>er</sup> et 2<sup>e</sup> cycles de géomatique les cours *Traitement numérique des images de télédétection*, *Base de données géospatiales* et *Apprentissage profond appliqué à l'observation de la Terre*. Ses intérêts de recherche portent sur le traitement d'images et l'application de l'IA aux données géospatiales.

**Philippe Apparicio** est professeur titulaire au Département de géomatique appliquée de l'Université de Sherbrooke. Il y enseigne aux programmes de 1<sup>er</sup> et 2<sup>e</sup> cycles de géomatique les cours *Transport et mobilité durable*, *Modélisation et analyse spatiale* et *Géomatique appliquée à la gestion urbaine*. Durant les dernières années, il a offert plusieurs formations aux Écoles d'été du Centre interuniversitaire québécois de statistiques sociales (**CIQSS**). Géographe de formation, ses intérêts de recherche incluent la justice et l'équité environnementale, la mobilité durable, les pollutions atmosphérique et sonore, et le vélo en ville. Il a publié une centaine d'articles scientifiques dans différents domaines des études urbaines et de la géographie mobilisant la géomatique et l'analyse spatiale.

**Mickaël Germain** est professeur agrégé au Département de géomatique appliquée de l'Université de Sherbrooke.

**Yacine Bouroubi** est professeur agrégé au Département de géomatique appliquée de l'Université de Sherbrooke.

**Étienne Clabaut** est professeur associé au Département de géomatique appliquée de l'Université de Sherbrooke.

# **Partie 1. Importation, manipulation et visualisation de données spatiales**

# 1 Introduction au langage Python

Dans ce chapitre, nous présentons quelques éléments essentiels du langage Python qui nous seront utiles dans ce manuel. Python est un langage très riche et peut aboutir à des projets logiciels très sophistiqués. Il est important de comprendre que la programmation Python n'est pas ici une fin en soi, mais plutôt un outil de scriptage et de manipulation des données satellitaires.

## Objectif

### Objectifs d'apprentissage visés dans ce chapitre

À la fin de ce chapitre, vous devriez être en mesure de :

- connaître les principales distributions de Python;
- installer un environnement d'exécution du code de cet ouvrage;
- comprendre les structures de base du langage Python;

Python, créé par [Guido van Rossum](#) en 1991, est un langage de programmation polyvalent et facile à apprendre, souvent comparé à un couteau suisse numérique pour sa simplicité et sa polyvalence. Comme un outil multifonction, Python peut être utilisé pour une variété de tâches, du développement web à l'analyse de données, en passant par l'intelligence artificielle.

## 1.1 Les distributions

Il existe plusieurs [distributions](#) du langage Python, ces distributions sont comme différentes saveurs de votre glace préférée - chacune a ses propres caractéristiques uniques, mais elles sont toutes fondamentalement Python. Voici un aperçu des principales distributions :

- [CPython](#) est la distribution “vanille” officielle, comme la recette originale de Python. Elle est ainsi le choix idéal pour la compatibilité et la conformité aux standards.
- [Anaconda](#). Pensez-y comme à un sundae tout garni. Elle vient avec de nombreuses bibliothèques scientifiques préinstallées, ce qui est idéal pour l'analyse de données et l'apprentissage automatique (*machine learning*).
- [Miniconda](#) est une distribution légère de Python qui vous permet d'ajouter au besoin d'autres bibliothèques.
- [PyPy](#) : est une version turbo de Python, optimisée pour la vitesse.

Chaque distribution a ses forces, que ce soit la simplicité, la vitesse ou des fonctionnalités spécifiques. Le choix dépend donc de vos besoins, comme choisir entre une glace simple ou une glace royal (banana split) élaboré.

## 1.2 Les styles de programmation en Python

Il existe plusieurs approches pour programmer en Python. La plus directe est en version interactive en tapant `python` et de rentrer des commandes ligne par ligne.

### 1.2.1 Les outils de programmation

Un code python prend la forme d'un simple fichier texte avec l'extension `.py` et peut être modifié avec un simple éditeur de texte. Cependant, il n'y aura pas de rétroactions immédiates de l'interpréteur Python, ce qui rend la correction d'erreurs (débogage) beaucoup plus laborieux.

Un IDE (*Integrated Developement Environnement*) est comme une boîte à outils complète pour les programmeurs, vous trouverez :

- Un éditeur de texte amélioré pour écrire votre code, avec des fonctionnalités comme la coloration syntaxique qui rend le code plus lisible.
- Un compilateur qui transforme votre code en instructions que l'ordinateur peut comprendre.
- Un débogueur pour trouver et corriger les erreurs, tel un détective numérique.
- Des outils d'automatisation qui effectuent des tâches répétitives, comme un assistant virtuel pour le codage.
- L'accès à la documentation des différentes librairies.

Ces outils intégrés permettent aux développeurs de travailler plus efficacement, en passant moins de temps à jongler entre différentes applications et plus de temps à produire du code.

Voici quelques options populaires :

- **PyCharm**: est un des outils les plus utilisés dans l'industrie. Il offre une multitude de fonctionnalités comme l'autocomplétion intelligente et le débogage intégré, idéal pour les grands projets. Cependant, cet outil peut être assez gourmand en mémoire et en CPU. Notez qu'il existe une version gratuite et une version professionnelle.
- **Visual Studio Code** : gratuit, léger mais puissant, il est personnalisable avec des extensions pour Python.
- **Spyder** : logiciel libre et gratuit, orienté vers les applications scientifiques.
- **Jupyter Notebooks** : imaginez un cahier interactif pour le code. Idéal pour l'analyse de données et l'apprentissage, il permet de mélanger code, texte et visualisations. Des services gratuits dans le **cloud** sont disponibles comme Google Colab et Kaggle. Ces environnements sont néanmoins moins appropriés pour des grands projets et le débogage. Jupyter peut souffrir de problèmes de reproductibilité dus à l'exécution arbitraire des cellules.
- **Marimo** se veut une version plus moderne des *notebooks jupyter*. Contrairement à Jupyter, Marimo garantit la cohérence entre le code, les sorties et l'état du programme. Il analyse intelligemment les relations entre les cellules et réexécute automatiquement celles qui sont affectées par des changements, éliminant ainsi les problèmes d'état caché.

## 1.3 Bonnes pratiques

Python est un langage très dynamique, qui évolue constamment. Cela pose certains défis pour la gestion du code à long terme. Il est fortement conseillé d'utiliser des environnements virtuels pour gérer vos différentes bibliothèques (*libraries*). Voici quelques bonnes pratiques à suivre :

1. **N'installez pas la toute dernière version de Python** : Il est recommandé d'installer 1 ou 2 versions antérieures, par exemple si 3.13 est la version plus récente, installer plutôt la version 3.11. Les versions trop récentes peuvent être instables. La version de python désirée peut être spécifiée au moment de la création d'un environnement virtuel (voir plus bas). Vous pouvez afficher la liste des versions de python avec la commande `conda search --full-name python`.
2. **N'utilisez pas de version obsolète de Python**. Cela peut sembler contradictoire avec le point précédent mais c'est l'excès inverse. Si vous utilisez une version trop ancienne alors toutes vos librairies cesseront d'évoluer et peuvent devenir obsolètes.
3. **Utilisez des environnements virtuels**. Pensez-y comme à des compartiments séparées pour chaque projet. Cela évite les conflits entre les différentes versions de bibliothèques (*libraries*) et garde votre système propre. Par exemple, si vous souhaitez vérifier une nouvelle version de Python, utilisez un environnement : `conda create --name test python=3.11`
4. **Vérifiez l'installation**. Après l'installation, ouvrez un terminal et tapez `python --version` pour vous assurer que tout fonctionne correctement.

### 1.3.1 Création d'un environnement virtuel

Il y a deux façons d'installer un environnement virtuel selon votre distribution de Python:

1. **Option 1.** Vous utilisez **Anaconda** ou **Miniconda**. La commande `conda` est utilisée pour créer un environnement test avec Python 3.10:

```
1 conda env -n test python=3.10
2 conda activate test
```

2. **Option 2.** Vous utilisez **CPython**

```
1 conda env -n test python=3.10
2 conda activate test
```

### 1.3.2 Création d'un environnement de travail local (avancé)

**Note:** les notebooks peuvent fonctionner localement uniquement sous Linux ou avec WSL2.

Les notebooks Python fonctionnent par défaut dans l'environnement **Google Colab**. Si vous souhaitez faire fonctionner ces notebook localement, vous pouvez installer un environnement local avec un serveur **Jupyter**. Il suffit de suivre les étapes suivantes:

1. Installer **WSL2** sous **Windows**

2. Installer vscode

3. Installer Miniconda

4. Faire une installation du contenu du livre soit en utilisant une commande `git clone` ou en récupérant le `.zip` du livre

5. Ouvrir WSL2 et placer vous dans le répertoire du livre `TraitemenImagesPythonVol1`. Assurez vous que vous avez accès à conda en tapant `conda --version`

6. Lancer la commande `conda env create -f jupyter_env.yaml`

7. Activer le nouvel environnement: `conda activate jupyter_env`

8. Le serveur jupyter peut ensuite être lancé avec la commande suivante: `jupyter lab --ip='*' --NotebookApp.token='' --NotebookApp.password=''`

Une fenêtre devrait alors apparaître dans votre fureteur. Dans le menu de gauche vous pouvez accéder aux notebooks dans le répertoire `notebooks`:

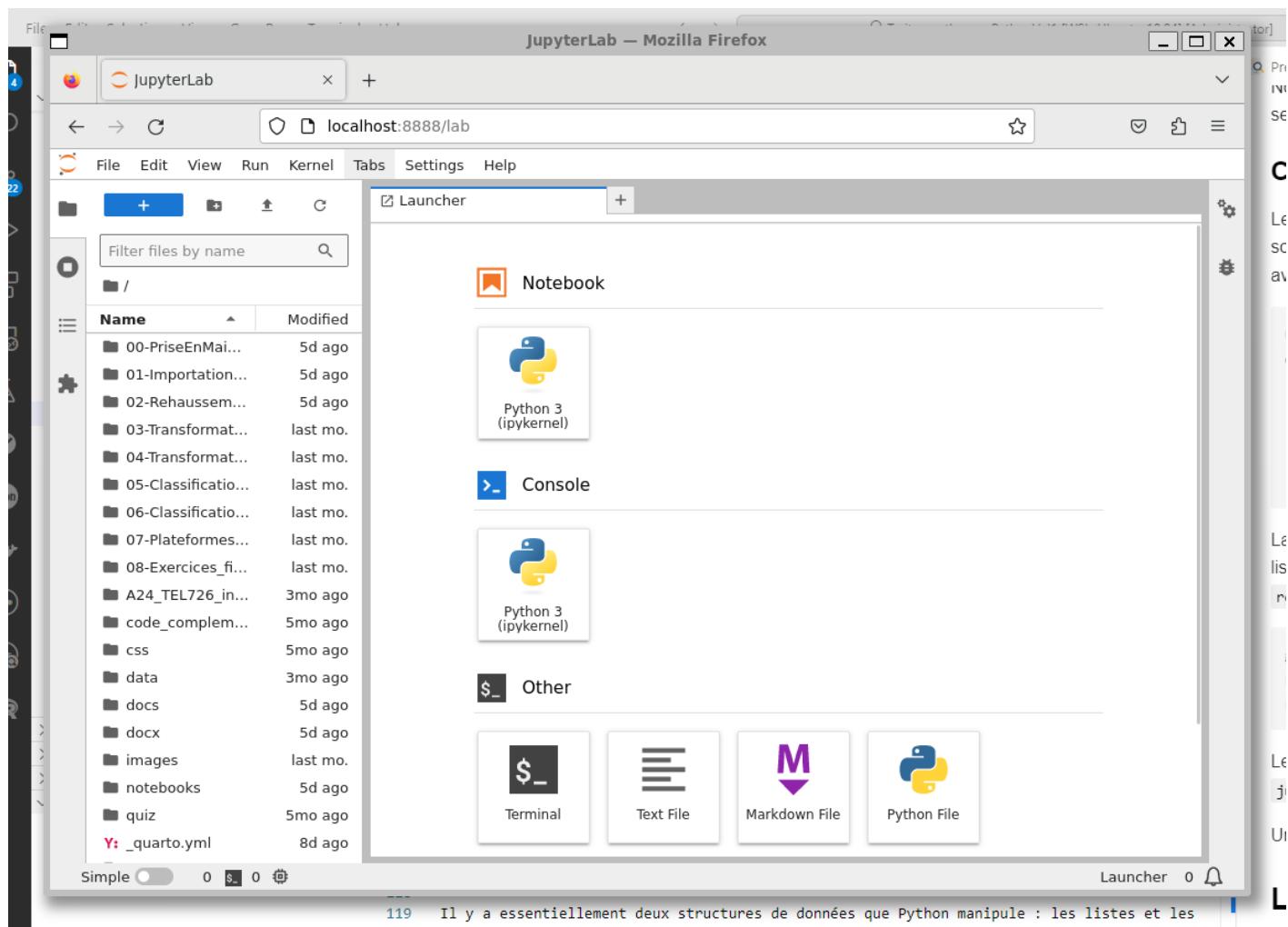


FIGURE 1.1 – Fenêtre principale du serveur Jupyter Lab.

## 1.4 Les structures de base en Python

Il y a essentiellement deux structures de données que Python manipule : les listes et les dictionnaires.

### 1.4.1 Les listes

Les listes sont comme des boites extensibles où vous pouvez ranger différents types d'objets:

- Représentées par des crochets : [1, 2, 3, "python"].
- Ordonnées et modifiables (*mutable*), vous pouvez récupérer une valeur par sa position avec [].
- Permettent les doublons (deux fois la même valeur).
- Idéales pour stocker des collections d'éléments que vous voulez modifier

### 1.4.2 Les tuples

Les tuples sont similaires aux listes, mais les boîtes sont scellées:

- Représentés par des parenthèses : (1, 2, 3, "python").
- Ordonnés mais non modifiables (*immutable*).
- Permettent les doublons.
- Souvent utilisés pour stocker des données qui ne doivent pas changer (comme des paramètres).

### 1.4.3 Les ensembles (Sets)

Les ensembles sont comme des boites magiques qui ne gardent qu'un exemplaire de chaque objet:

- Représentés par des accolades : {1, 2, 3}.
- Non ordonnés et modifiables.
- N'autorisent pas les doublons.
- Utiles pour éliminer les doublons et effectuer des opérations mathématiques sur des ensembles.

## 1.5 Dictionnaires

Les dictionnaires sont comme des boites avec des étiquettes sur chacune d'elle :

- Représentés par des accolades avec des paires clé-valeur : {"nom": "Python", "année": 1991}.
- Non ordonnés et modifiables.
- Les clés doivent être uniques, mais les valeurs peuvent être dupliquées
- Utiles pour stocker des données associatives ou pour créer des tables de recherche rapide

## 1.6 Programmation objet

La programmation orientée objet (POO) en Python est comme construire avec des blocs LEGO. Chaque objet est un bloc LEGO avec ses propres caractéristiques (attributs) et capacités (méthodes). Les classes sont les plans pour créer ces blocs. Par exemple, une classe “Voiture” pourrait avoir des attributs comme “couleur” et “vitesse”, et des méthodes comme “démarrer” et “accélérer”.

Python rend la POO accessible avec des fonctionnalités conviviales:

1. **Encapsulation:** comme emballer un cadeau, elle cache les détails internes d'un objet.
2. **Héritage:** permet de créer de nouvelles classes basées sur des classes existantes, comme un enfant héritant des traits de ses parents.
3. **Polymorphisme:** permet à différents objets de répondre au même message de manière unique, comme si différents animaux répondaient différemment à “fais du bruit”.

Ces caractéristiques font de Python un excellent choix pour apprendre et appliquer les concepts de la POO, rendant le code plus organisé et réutilisable

### Package

#### Liste des *packages* utilisés dans ce chapitre

- Pour importer et manipuler des fichiers géographiques :
  - **numpy** pour manipuler des données matricielles.
  - **rasterio** pour importer et manipuler des données matricielles.
- Pour construire des cartes et des graphiques :
  - **matplotlib** est certainement le *package* le plus complet pour l'affichage général.
  - **seaborn** pour construire des graphiques plus détaillés en particulier pour les statistiques.

# 2 Importation et manipulation de données spatiales

## 2.1 Préambule

Assurez-vous de lire ce préambule avant d'exécutez le reste du notebook.

### 2.1.1 Objectifs

Dans ce chapitre, nous abordons quelques formats d'images ainsi que leur lecture. Ce chapitre est aussi disponible sous la forme d'un notebook Python:



#### Objectif

##### Objectifs d'apprentissage visés dans ce chapitre

À la fin de ce chapitre, vous devriez être en mesure de :

- connaître les principales bibliothèques Python pour lire une image;
- accéder à l'information d'une image avant de la lire;
- comprendre les principaux formats pour une image
- manipuler la matrice de la donnée d'une image avec numpy

### 2.1.2 Bibliothèques

Les bibliothèques qui vont être explorées dans ce chapitre sont les suivantes:

- SciPy
- NumPy
- opencv-python · PyPI
- scikit-image
- Rasterio
- Xarray
- rioxarray

Dans l'environnement Google Colab, seul `rioxarray` et `gdal` doivent être installés:

```
1 !apt-get update
2 !apt-get install gdal-bin libgdal-dev
3 !pip install -q rioxarray
```

Vérifier les importations:

```

1 import numpy as np
2 import rioxarray as rxr
3 from scipy import signal
4 import xarray as xr
5 import xrscipy
6 import matplotlib.pyplot as plt

```

### 2.1.3 Données

Nous utilisons ces images dans ce chapitre:

```

1 import gdown
2
3 gdown.download(
4     'https://drive.google.com/uc?export=download&confirm=pbef&id=1a6Ypg0g10y4AJt9XWkWfnR12NW1XhNg_',
5     output= 'RGBNIR_of_S2A.tif')
6 gdown.download(
7     'https://drive.google.com/uc?export=download&confirm=pbef&id=1a4PQ68Ru8zBphbQ22j0sgJ4D2quw-Wo6',
8     output= 'landsat7.tif')
9 gdown.download(
10    'https://drive.google.com/uc?export=download&confirm=pbef&id=1_zwCLN-x7XJcNHJCH6Z8upEdUXtVtvS1',
11    output= 'berkeley.jpg')
12 !wget https://raw.githubusercontent.com/sfoucher/TraitemenImagesPythonVol1/refs/heads/main/images/
13   modis-aqua.PNG -O modis-aqua.PNG

```

Vérifiez que vous êtes capable de les lire:

```

1 with rxr.open_rasterio('berkeley.jpg', mask_and_scale= True) as img_rgb:
2     print(img_rgb)
3 with rxr.open_rasterio('RGBNIR_of_S2A.tif', mask_and_scale= True) as img_rgnir:
4     print(img_rgnir)

```

## 2.2 Importation d'images

La première étape avant tout traitement est d'accéder à la donnée image pour qu'elle soit manipulée par le langage Python. L'imagerie satellite présente certains défis notamment en raison de la taille parfois très importante des images. Il existe maintenant certaines bibliothèques, comme **Xarray**, qui visent à optimiser la lecture et l'écriture de grandes images. Il est donc conseillé de toujours garder un oeil sur l'espace mémoire occupé par les variables Python représentant les images. La librairie principale en géomatique qui permettre d'importer (et d'exporter) de l'imagerie est la librairie **GDAL** qui rassemble la plupart des formats sous forme de *driver* (ou pilote en français).

Dans le domaine de la géomatique, il faut prêter attention à trois caractéristiques principales des images:

1. **La matrice des données** elle-même qui contient les valeurs brutes des pixels. Cette matrice sera souvent un cube à trois dimensions. En Python, ce cube sera le plus souvent un objet de la librairie **NumPy** (voir section).
2. **La dynamique des images** c.-à.-d le format de stockage des valeurs individuelles (octet, entier, double, etc.). Ce format décide principalement de la résolution radiométrique et des valeurs minimales et maximales supportées.
3. **Le nombre de bandes** spectrales de l'image qui est souvent supérieur à trois et peut atteindre plusieurs centaines de bandes pour certains capteurs (notamment hyperspectraux).
4. **La métadonnée** qui va transporter l'information auxiliaire de l'image comme les dimensions et la position de l'image, la date, etc. Cette donnée auxiliaire prendra souvent la forme d'un dictionnaire Python. Elle contiendra aussi l'information de géoréférence.

Les différents formats se distinguent principalement sur la manière dont ces trois caractéristiques sont gérées.

### 2.2.1 Formats des images

Il existe de nombreux formats numériques pour la donnée de type image parfois appelé donnée matricielle ou donnée *raster*. La librairie GDAL rassemble la plupart des formats matriciels rencontrés en géomatique (voir [Raster drivers — GDAL documentation](#) pour une liste complète).

On peut distinguer deux grandes familles de format:

1. Les formats de type **RVB** issus de l'imagerie numérique grand public comme **JPEG**, **png**, etc. Ces formats ne supportent généralement que trois bandes au maximum (rouge, vert et bleu) et des valeurs de niveaux de gris entre 0 et 255 (format dit 8 bits ou **uint8**).
2. **Les géo-formats** issus des domaines scientifiques ou techniques comme GeoTIFF, HDF5, NetCDF, etc. qui peuvent inclure plus que trois bandes et des dynamiques plus élevées (16 bits ou même float).

Les formats RVB restent très utilisés en Python notamment par les bibliothèques dites de vision par ordinateur (*Computer Vision*) comme OpenCV et sickit-image ainsi que les grandes bibliothèques en apprentissage profond (PyTorch, Tensorflow).

#### Package

##### Installation de gdal dans un système Linux

— Pour installer GDAL :

```
!apt-get update  
!apt-get install gdal-bin libgdal-dev
```

### 2.2.1.1 Formats de type RVB

Les premiers formats pour de l'imagerie à une bande (monochrome) et à trois bandes (image couleur rouge-vert-bleu) sont issus du domaine des sciences de l'ordinateur. On trouvera, entre autres, les formats pbm, png et jpeg. Ces formats supportent peu de métadonnées et sont placées dans un entête (*header*) très limité. Cependant, ils restent très populaires dans le domaine de la vision par ordinateur et sont très utilisés en apprentissage profond en particulier. Pour la lecture des images RVB, on peut utiliser les bibliothèques Rasterio, [PIL](#) ou [OpenCV](#).

#### Lecture avec la librairie PIL

La librairie PIL retourne un objet de type `PngImageFile`, l'affichage de l'image se fait directement dans la cellule de sortie.

```
1 from PIL import Image
2 img = Image.open('modis-aqua.PNG')
3 img
```



### Lecture avec la librairie OpenCV

La librairie **OpenCV** est aussi très populaire en vision par ordinateur. La fonction **imread** donne directement un objet de type NumPy en sortie.

```
1 import cv2
2 img = cv2.imread('modis-aqua.PNG')
3 img
```

```
array([[[17, 50, 33],
       [15, 49, 31],
       [14, 48, 30],
       ...,
       [23, 56, 36],
       [23, 55, 36],
       [22, 55, 36]],
      [[18, 51, 34],
       [16, 50, 32],
```

```
[15, 49, 32],
...,
[27, 59, 40],
[28, 60, 41],
[27, 60, 41]],

[[18, 53, 35],
[18, 52, 34],
[18, 51, 34],
...,
[31, 64, 44],
[34, 66, 47],
[33, 65, 46]],

...,

[[34, 74, 48],
[35, 73, 48],
[34, 70, 46],
...,
[41, 74, 54],
[41, 73, 54],
[41, 73, 54]],

[[36, 76, 50],
[36, 74, 49],
[35, 71, 47],
...,
[37, 70, 51],
[38, 71, 51],
[38, 71, 51]],

[[36, 76, 50],
[35, 73, 48],
[33, 69, 45],
...,
[31, 63, 44],
[33, 65, 46],
[33, 66, 46]]], dtype=uint8)
```

### Lecture avec la librairie RasterIO

Rien ne nous empêche de lire une image de format RVB avec **RasterIO** comme décrit dans ci-dessous. Vous noterez cependant les avertissements concernant l'absence de géoréférence pour ce type d'image.

```
1 import rasterio
2 img= rasterio.open('modis-aqua.PNG')
3 img
```

```
<open DatasetReader name='modis-aqua.PNG' mode='r'>
```

#### 2.2.1.2 Le format GeoTiff

Le format GeoTIFF est une extension du format TIFF (Tagged Image File Format) qui permet d'incorporer des métadonnées géospatiales directement dans un fichier image. Développé initialement par Dr. Niles Ritter au Jet Propulsion Laboratory de la **NASA** dans les années 1990, GeoTIFF est devenu un standard de facto pour le stockage et l'échange d'images géoréférencées dans les domaines de la télédétection et des systèmes d'information géographique (SIG). Ce format supporte plus que trois bandes aussi longtemps que ces bandes sont de même dimension.

Le format GeoTIFF est très utilisé et est largement supporté par les bibliothèques et logiciels géospatiaux, notamment **GDAL** (*Geospatial Data Abstraction Library*), qui offre des capacités de lecture et d'écriture pour ce format. Cette compatibilité étendue a contribué à son adoption généralisée dans la communauté géospatiale.

## Standardisation par l'OGC

Le standard GeoTIFF proposé par l'Open Geospatial Consortium (OGC) en 2019 formalise et étend les spécifications originales du format GeoTIFF, offrant une norme robuste pour l'échange d'images géoréférencées. Cette standardisation, connue sous le nom d'OGC GeoTIFF 1.1 (2019), apporte plusieurs améliorations et clarifications importantes.

### 2.2.1.3 Le format COG

Une innovation récente dans l'écosystème GeoTIFF est le format *Cloud Optimized GeoTIFF (COG)*, conçu pour faciliter l'utilisation de fichiers GeoTIFF hébergés sur des serveurs web HTTP. Le COG permet aux utilisateurs et aux logiciels d'accéder à des parties spécifiques du fichier sans avoir à le télécharger entièrement, ce qui est particulièrement utile pour les applications basées sur l'infonuagique.

## 2.2.2 Métadonnées des images

La manière la plus directe d'accéder à la métadonnée d'une image est d'utiliser les commandes `rio info` de la librairie Rasterio ou `gdalinfo` de la librairie `gdal`. Le résultat est imprimé dans la sortie standard ou sous forme d'un dictionnaire Python.

```
1 !gdalinfo RGBNIR_of_S2A.tif
```

```
Warning 1: TIFFReadDirectory:Sum of Photometric type-related color channels and ExtraSamples doesn't match SamplesPerPixel. Defining non-color
    ↪ channels as ExtraSamples.
Driver: GTiff/GeoTIFF
Files: RGBNIR_of_S2A.tif
       RGBNIR_of_S2A.tif.aux.xml
Size is 2074, 1926
Coordinate System is:
PROJCS["WGS 84 / UTM zone 18N",
  GEOGCS["WGS 84",
    DATUM["WGS_1984",
      SPHEROID["WGS 84",6378137,298.257223563,
        AUTHORITY["EPSG","7030"]],
      AUTHORITY["EPSG","6326"]],
    PRIMEM["Greenwich",0,
      AUTHORITY["EPSG","8901"]],
    UNIT["degree",0.0174532925199433,
      AUTHORITY["EPSG","9122"]],
    AUTHORITY["EPSG","4326"]],
  PROJECTION["Transverse_Mercator"],
  PARAMETER["latitude_of_origin",0],
  PARAMETER["central_meridian",-75],
  PARAMETER["scale_factor",0.9996],
  PARAMETER["false_easting",500000],
  PARAMETER["false_northing",0],
  UNIT["metre",1,
    AUTHORITY["EPSG","9001"]],
  AXIS["Easting",EAST],
  AXIS["Northing",NORTH],
  AUTHORITY["EPSG","32618"]]
Origin = (731780.00000000000000,5040800.00000000000000)
Pixel Size = (10.00000000000000,-10.00000000000000)
Metadata:
AREA_OR_POINT=Area
TIFFTAG_IMAGEDESCRIPTION=subset_RGBNIR_of_S2A_MSIL2A_20240625T153941_N0510_R011_T18TYR_20240625T221903
TIFFTAG_RESOLUTIONUNIT=1 (unitless)
TIFFTAG_XRESOLUTION=1
TIFFTAG_YRESOLUTION=1
Image Structure Metadata:
INTERLEAVE=BAND
Corner Coordinates:
Upper Left  ( 731780.000, 5040800.000) ( 72d 2' 3.11"W, 45d28'55.98"N)
Lower Left   ( 731780.000, 5021540.000) ( 72d 2'35.69"W, 45d18'32.70"N)
Upper Right  ( 752520.000, 5040800.000) ( 71d46' 9.19"W, 45d28'30.08"N)
Lower Right  ( 752520.000, 5021540.000) ( 71d46'44.67"W, 45d18' 6.95"N)
Center       ( 742150.000, 5031170.000) ( 71d54'23.16"W, 45d23'31.71"N)
Band 1 Block=2074x1926 Type=UInt16, ColorInterp=Gray
Min=86.000 Max=15104.000
Minimum=86.000, Maximum=15104.000, Mean=1426.625, StdDev=306.564
```

```

Metadata:
  STATISTICS_MAXIMUM=15104
  STATISTICS_MEAN=1426.6252674912
  STATISTICS_MINIMUM=86
  STATISTICS_STDDEV=306.56427126942
  STATISTICS_VALID_PERCENT=100
Band 2 Block=2074x1926 Type=UInt16, ColorInterp=Undefined
  Min=1139.000 Max=14352.000
  Minimum=1139.000, Maximum=14352.000, Mean=1669.605, StdDev=310.919
Metadata:
  STATISTICS_MAXIMUM=14352
  STATISTICS_MEAN=1669.6050060032
  STATISTICS_MINIMUM=1139
  STATISTICS_STDDEV=310.91935787639
  STATISTICS_VALID_PERCENT=100
Band 3 Block=2074x1926 Type=UInt16, ColorInterp=Undefined
  Min=706.000 Max=15280.000
  Minimum=706.000, Maximum=15280.000, Mean=1471.392, StdDev=385.447
Metadata:
  STATISTICS_MAXIMUM=15280
  STATISTICS_MEAN=1471.3923473736
  STATISTICS_MINIMUM=706
  STATISTICS_STDDEV=385.44654593014
  STATISTICS_VALID_PERCENT=100
Band 4 Block=2074x1926 Type=UInt16, ColorInterp=Undefined
  Min=1067.000 Max=15642.000
  Minimum=1067.000, Maximum=15642.000, Mean=4393.945, StdDev=1037.934
Metadata:
  STATISTICS_MAXIMUM=15642
  STATISTICS_MEAN=4393.94485025
  STATISTICS_MINIMUM=1067
  STATISTICS_STDDEV=1037.933939728
  STATISTICS_VALID_PERCENT=100

```

Le plus simple est d'utiliser la fonction `rio info`:

```
1 !rio info RGNIR_of_S2A.tif --indent 2 --verbose
```

## 2.3 Manipulation des images

### 2.3.1 Manipulation de la matrice de pixels

La donnée brute de l'image est généralement contenue dans un cube matricielle à trois dimensions (deux dimensions spatiales et une dimension spectrale). Comme exposé précédemment, la librairie dite “*fondationnelle*” pour la manipulation de matrices en Python est [NumPy](#). Cette librairie contient un nombre très important de fonctionnalités couvrant l'algèbre linéaire, les statistiques, etc.; elle constitue la fondation de nombreuses bibliothèques en traitement numérique (voir (figure 2.1))

### 2.3.2 Information de base

Les deux informations de base à afficher sur une matrice sont 1) les dimensions de la matrice et 2) le format de stockage (le type). Pour cela, on peut utiliser le code ci-dessous, dont le résultat nous informe que la matrice a trois dimensions et une taille de `(442, 553, 3)` et un type `uint8` qui représente 1 octet (8 bit). Par conséquent, la matrice a `442` lignes, `553` colonnes et `3` canaux ou bandes. Il faut prêter une attention particulière aux valeurs minimales et maximales tolérées par le type de la donnée comme indiqué dans le (tableau 2.1) (voir aussi [Data types — NumPy v2.1 Manual](#)).

```
1 import cv2
2 img = cv2.imread('modis-aqua.PNG')
3 print('Nombre de dimensions: ',img.ndim)
```

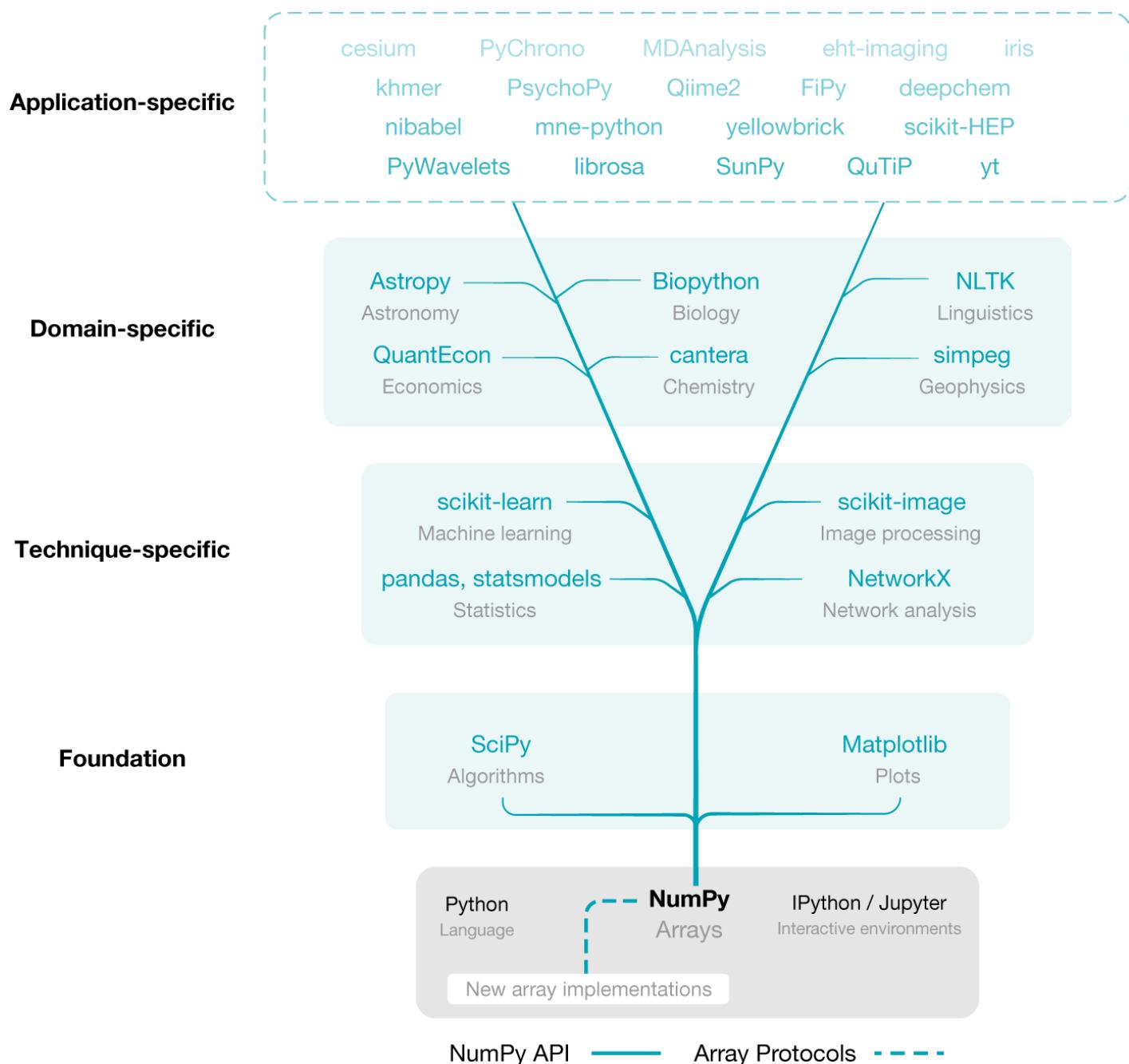


FIGURE 2.1 – La librairie NumPy est le fondement de nombreuses bibliothèques scientifiques (d'après (Harris 2020)).

```

4 print('Dimensions de la matrice: ',img.shape)
5 print('Type de la donnée: ',img.dtype)

```

Nombre de dimensions: 3  
 Dimensions de la matrice: (442, 553, 3)  
 Type de la donnée: uint8

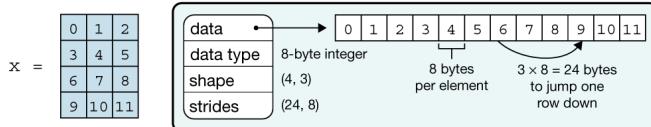
TABLEAU 2.1 – Type de données de NumPy

dtype	Nom	Taille (bits)	Min	Max
uint8	char	8	0	255
int8	signed char	8	-127	128
uint16	unsigned short	16	0	-32768
int16	short	16	0	655355

### 2.3.3 Découpage et indexation de la matrice

L’indexation et le découpage (*slicing*) des matrices dans NumPy sont des techniques essentielles pour manipuler efficacement les données multidimensionnelles en Python, offrant une syntaxe puissante et flexible pour accéder et modifier des sous-ensembles spécifiques d’éléments dans les tableaux (voir figure 2.2). Indexer une matrice consiste à accéder à une valeur dans la matrice pour une position particulière, la syntaxe générale est `matrice[ligne, colonne, bande]` et est similaire à la manipulation des `listes` en Python. Les indices commencent à **0** et se terminent à la **taille-1** de l’axe considéré.

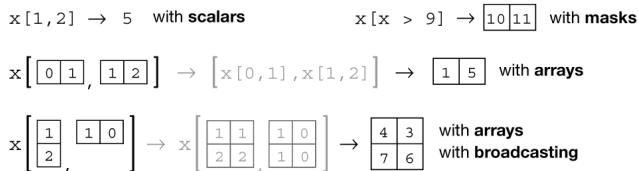
#### a Data structure



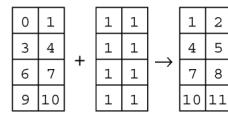
#### b Indexing (view)



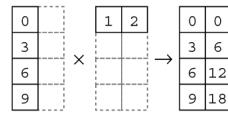
#### c Indexing (copy)



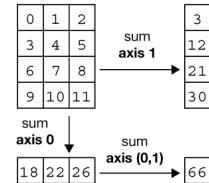
#### d Vectorization



#### e Broadcasting



#### f Reduction



```
In [1]: import numpy as np
```

```
In [2]: x = np.arange(12)
```

```
In [3]: x = x.reshape(4, 3)
```

```
In [4]: x
```

```
Out[4]:
```

```
array([[ 0,  1,  2],
       [ 3,  4,  5],
       [ 6,  7,  8],
       [ 9, 10, 11]])
```

```
In [5]: np.mean(x, axis=0)
Out[5]: array([4.5, 5.5, 6.5])
```

```
In [6]: x = x - np.mean(x, axis=0)
```

```
In [7]: x
```

```
Out[7]:
```

```
array([[-4.5, -4.5, -4.5],
       [-1.5, -1.5, -1.5],
       [ 1.5,  1.5,  1.5],
       [ 4.5,  4.5,  4.5]])
```

FIGURE 2.2 – Vue d’ensemble des opérations de base des matrices avec NumPy

Le découpage (ou *slicing* en anglais) consiste à produire une nouvelle matrice qui est un sous-ensemble de la matrice d’origine. Un découpage se fait avec le symbole ‘:’, la syntaxe générale pour définir un découpage

est `[début:fin:pas]`. Si on ne spécifie pas `début` ou `fin` alors les valeurs 0 ou `dimension-1` sont considérées implicitement. Quelques exemples: \* choisir un pixel en particulier avec toutes les bandes: `matrice[1,1,:]` \* choisir la colonne 2: `matrice[:,2,:]`

La syntaxe de base pour le découpage (*slicing*) des tableaux NumPy repose sur l'utilisation des deux-points `(:)` à l'intérieur des crochets d'indexation. Cette notation permet de sélectionner des plages d'éléments de manière concise et intuitive. La structure générale du découpage est `matrice[start:stop:step]`, où : 1. `start` représente l'index de départ (inclus) 2. `stop` indique l'index de fin (exclu) 3. `step` définit l'intervalle entre chaque élément sélectionné

Si l'un de ces paramètres est omis, NumPy utilise des valeurs par défaut : 0 pour `start`, la taille du tableau pour `stop`, et 1 pour `step`. Par exemple, pour un tableau unidimensionnel `array`, on peut extraire les éléments du deuxième au quatrième avec `array[1:4]`. Pour sélectionner tous les éléments à partir du troisième, on utiliserait `array[2:]`. Cette syntaxe s'applique également aux tableaux multidimensionnels, où chaque dimension est séparée par une virgule. Ainsi, pour une matrice 2D `m`, `m[0:2, 1:3]` sélectionnerait une sous-matrice 2x2 composée des deux premières lignes et des deuxième et troisième colonnes. L'indexation négative est également supportée, permettant de compter à partir de la fin du tableau. Par exemple, `a[-3:]` sélectionnerait les trois derniers éléments d'un tableau.

```

1 import cv2
2 img = cv2.imread('modis-aqua.PNG')
3 img_col = img[:,1,:]
4 print('Nombre de dimensions: ',img_col.ndim)
5 print('Dimensions de la matrice: ',img_col.shape)

```

Nombre de dimensions: 2  
Dimensions de la matrice: (442, 3)

### Aller plus loin

#### Une vue versus une copie

Avec NumPy, les manipulations peuvent créer des vues ou des copies. Une vue est une simple représentation de la même donnée originale alors qu'une copie est un nouvel espace mémoire.

Par défaut, un découpage créé une vue.

On peut vérifier si l'espace mémoire est partagé avec `np.shares_memory(arr, slice_arr)`.

On peut toujours forcer une copie avec la méthode `copy()`

#### 2.3.3.1 Masquage

L'utilisation d'un masque est un outil important en traitement d'image car la plupart des images de télédétection contiennent des pixels non valides qu'il faut exclure des traitements (ce que l'on appelle le *no data* en Anglais). Il y a plusieurs raisons possibles pour la présence de pixels non valides:

1. L'image est projetée dans une grille cartographique et certaines zones, généralement situées en dehors de l'empreinte au sol du capteur, sont à exclure.
2. La présence de nuages que l'on veut exclure.
3. La présence de pixels erronés dus à des problèmes de capteurs.
4. La présence de valeurs non numériques (*not a number* ou *nan*)

La librairie NumPy fournit des mécanismes pour exclure automatiquement certaines valeurs.

### 2.3.4 Changement de projection cartographique (à venir)

## 2.4 Données en géoscience

Les données en géoscience contiennent beaucoup de métadonnées et peuvent être composées de différentes variables avec différentes unités, résolution, etc. Ces données sont aussi souvent étiquetées avec des dates sur certains axes, des coordonnées géographiques, des identifiants d'expériences, etc. Par conséquent, utiliser seulement des matrices est souvent incomplet (Hoyer et Hamman 2017).

Calibration, unités, données manquantes, données éparses.

### 2.4.1 xarray

**Xarray** est une puissante bibliothèque Python qui améliore les matrices multidimensionnelles de type numpy en y ajoutant des étiquettes, des dimensions, des coordonnées et des attributs. Elle fournit deux structures de données principales : **DataArray** (un tableau étiqueté à n dimensions) et **Dataset** (une base de données de tableaux multidimensionnels en mémoire).

Les caractéristiques principales sont les suivantes:

- Opérations sur les dimensions nommées au lieu des numéros d'axe
- Sélection et opérations basées sur les étiquettes
- Diffusion automatique de tableaux basée sur les noms de dimensions
- Alignement de type base de données avec des étiquettes de coordonnées
- Suivi des métadonnées grâce à des dictionnaires Python

#### 2.4.1.1 Avantages

La bibliothèque réduit considérablement la complexité du code et améliore la lisibilité du code pour les applications de calcul scientifique dans divers domaines, notamment la physique, l'astronomie, les géosciences, la bio-informatique, l'ingénierie, la finance et l'apprentissage profond. Elle s'intègre de manière transparente avec NumPy et pandas tout en restant compatible avec l'écosystème Python au sens large.

#### 2.4.1.2 DataArray

Un tableau multidimensionnel étiqueté avec des propriétés clées :

- **valeurs** : Les données réelles du tableau
- **dims** : Dimensions nommées (par exemple, « x », « y », « z »)
- **coords** : Dictionnaire de tableaux étiquetant chaque point
- **attrs** : Stockage de métadonnées arbitraires
- **name** : Identifiant facultatif

### 2.4.1.3 Dataset

Un conteneur de type dictionnaire de **DataArrays** avec des dimensions alignées, contenant :

- **dims** : Dictionnaire de correspondance entre les noms des dimensions et les longueurs
- **data\_vars** : Dictionnaire des variables du dataArray
- **coords** : Dictionnaire des variables de coordonnées
- **attrs** : Stockage des métadonnées

Les principales différences sont les suivantes :

- **dataArray** contient un seul tableau avec des étiquettes
- Le **Dataset** contient plusieurs DataArrays alignés.

Ces trois structures prennent en charge les opérations de type dictionnaire et les calculs de coordination tout en conservant les métadonnées.

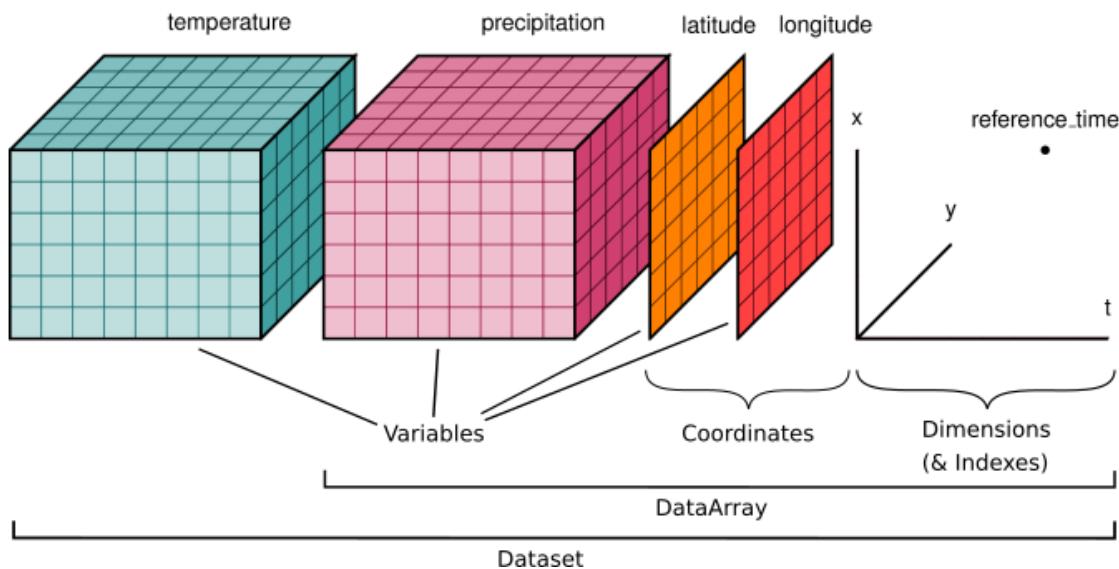


FIGURE 2.3 – Organisation d'un Dataset dans xarray

# 3 Réhaussement et visualisation d'images

Assurez-vous de lire ce préambule avant d'exécuter le reste du notebook.

## 3.1 Préambule

### 3.1.1 Objectifs

Dans ce chapitre, nous abordons quelques techniques de réhaussement et de visualisation d'images. Ce chapitre est aussi disponible sous la forme d'un notebook Python:



#### Objectif

##### Objectifs d'apprentissage visés dans ce chapitre

À la fin de ce chapitre, vous devriez être en mesure de :

- exploiter les statistiques d'une image pour améliorer la visualisation;
- calculer les histogrammes de valeurs;
- appliquer une transformation linéaire ou non linéaire pour améliorer une visualisation;
- comprendre le principe des composés colorés;

### 3.1.2

### 3.1.3 Bibliothèques

Les bibliothèques qui vont être explorées dans ce chapitre sont les suivantes:

- SciPy
- NumPy
- opencv-python · PyPI
- scikit-image
- Rasterio
- Xarray
- rioxarray

Dans l'environnement Google Colab, seul `rioxarray` et GDAL doivent être installés:

```

1 %%capture
2 !apt-get update
3 !apt-get install gdal-bin libgdal-dev

```

Dans l'environnement **Google Colab**, il convient de s'assurer que les librairies sont installées:

```

1 %%capture
2 !pip install -qU matplotlib rioxarray xrscipy scikit-image

```

Vérifier les importations:

```

1 import numpy as np
2 import rioxarray as rxr
3 from scipy import signal
4 import xarray as xr
5 import xrscipy
6 import matplotlib.pyplot as plt

```

### 3.1.4 Données

Nous utiliserons les images suivantes dans ce chapitre:

```

1 %%capture
2 import gdown
3
4 gdown.download(
    ↵ 'https://drive.google.com/uc?export=download&confirm=pbef&id=1a6Ypg0g10y4AJt9XWKWfnR12NW1XhNg_',
    ↵ output= 'RGBNIR_of_S2A.tif')
5 gdown.download(
    ↵ 'https://drive.google.com/uc?export=download&confirm=pbef&id=1a603L_ab0fU7h94K22At8qtBuLMGErwo',
    ↵ output= 'sentinel2.tif')
6 gdown.download(
    ↵ 'https://drive.google.com/uc?export=download&confirm=pbef&id=1_zwCLN-x7XJcNHJCH6Z8upEdUXtVtvsl',
    ↵ output= 'berkeley.jpg')
7 gdown.download(
    ↵ 'https://drive.google.com/uc?export=download&confirm=pbef&id=1dM6IVqjba6GHwTLmI7CpX8GP2z5txUq6',
    ↵ output= 'SAR.tif')

```

Vérifiez que vous êtes capable de les lire :

```

1 with rxr.open_rasterio('berkeley.jpg', mask_and_scale= True) as img_rgb:
2     print(img_rgb)
3 with rxr.open_rasterio('sentinel2.tif', mask_and_scale= True) as img_s2:
4     print(img_s2)
5 with rxr.open_rasterio('RGBNIR_of_S2A.tif', mask_and_scale= True) as img_rgbnir:
6     print(img_rgbnir)
7 with rxr.open_rasterio('SAR.tif', mask_and_scale= True) as img_SAR:
8     print(img_SAR)

```

## 3.2 Visualisation en Python

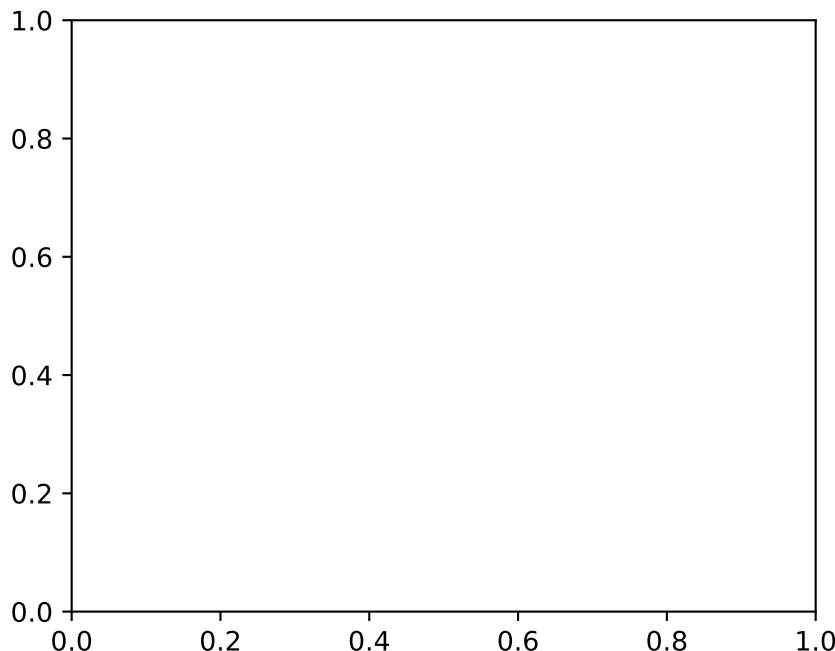
ID'emblee, il faut mentionner que Python n'est pas vraiment fait pour visualiser de la donnée de grande taille, le niveau d'interactivité est aussi assez limité. Pour une visualisation interactives, il est plutôt conseillé d'utiliser un outil comme [QGIS](#). Néanmoins, il est possible de visualiser de petites images avec la librairie [matplotlib](#) qui est la librairie principale de visualisation en Python. Cette librairie est extrêmement riche et versatile, nous ne présenterons que les bases nécessaires pour démarrer. Le lecteur désirant aller plus loin pourra consulter les nombreux tutoriels disponibles comme [celui-ci](#).

La fonction de base pour créer une figure est `subplots`, la largeur et la hauteur en pouces de la figure peuvent être contrôlées via le paramètre `figsize`:

```

1 import matplotlib.pyplot as plt
2 fig, ax= plt.subplots(figsize=(5, 4))
3 plt.show()

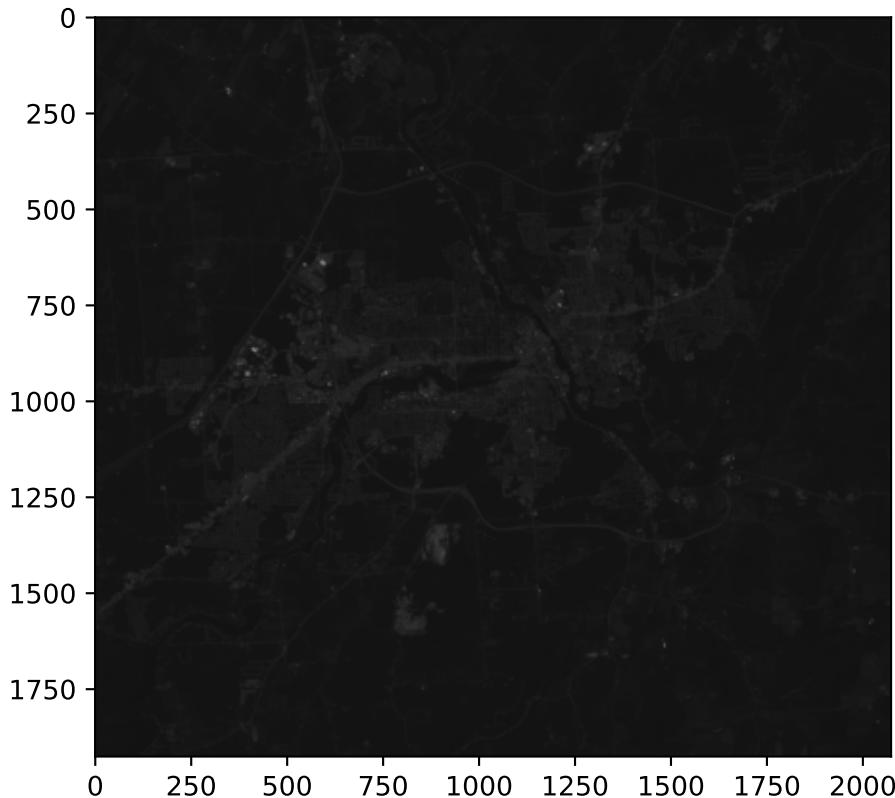
```



### 3 Réhaussement et visualisation d'images

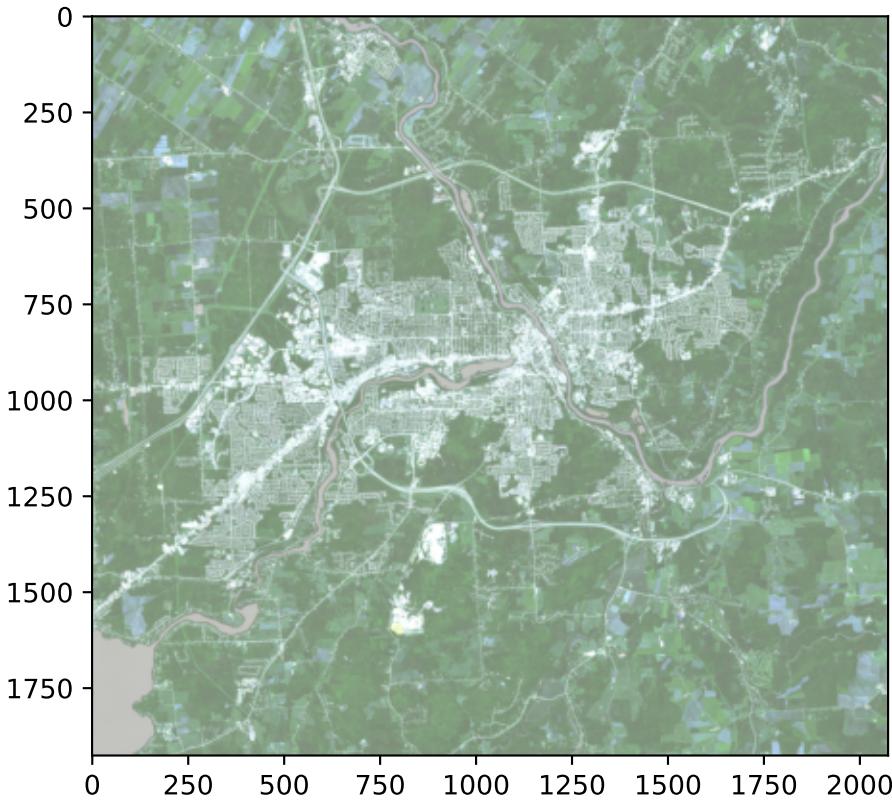
Pour l'affichage des images, la fonction `imshow` permet d'afficher une matrice 2D à une dimension en format *float* ou une matrice RVB avec 3 bandes. Il est important que les dimensions de la matrice soient dans l'ordre hauteur, largeur et bande.

```
1 import matplotlib.pyplot as plt
2 fig, ax= plt.subplots(figsize=(6, 5))
3 plt.imshow(img_rgnir[0].data)
4 plt.show()
```



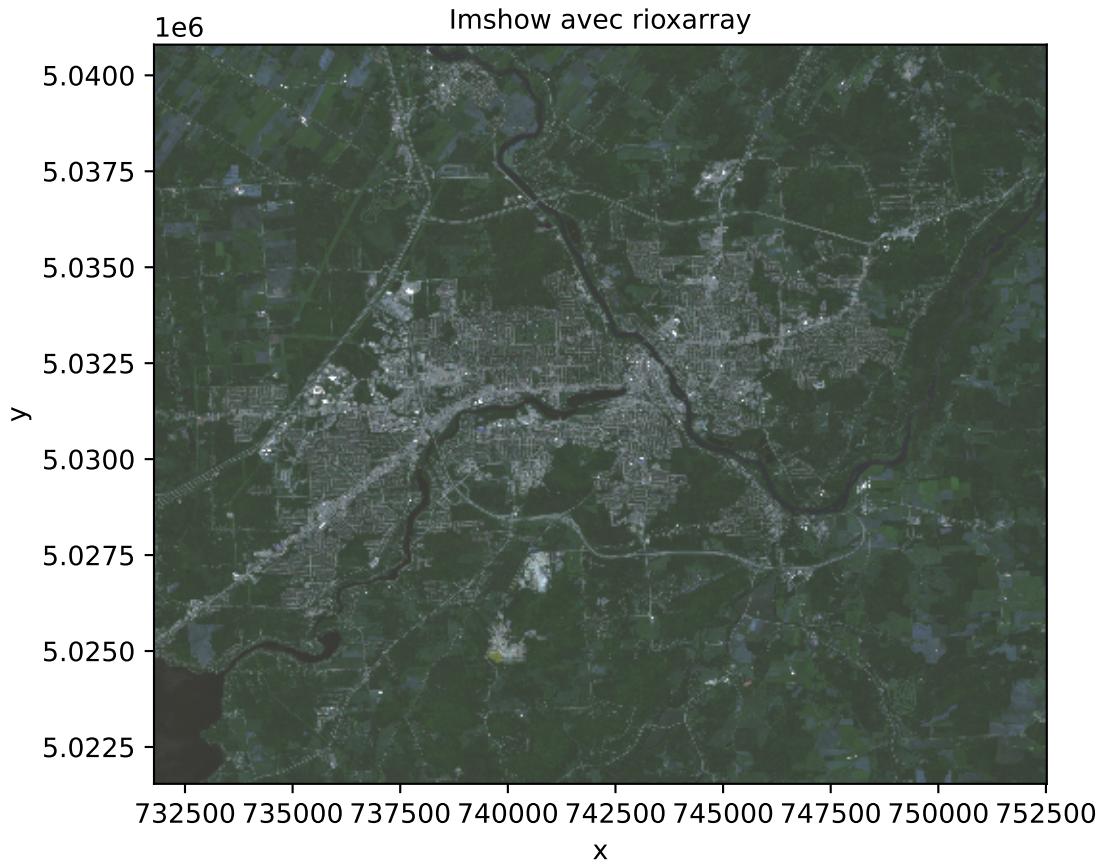
Pour un affichage à trois bandes, les valeurs seront ramenées sur une échelle de 0 à 1, il est donc nécessaire de normaliser les valeurs avant l'affichage:

```
1 import matplotlib.pyplot as plt
2 fig, ax= plt.subplots(figsize=(6, 5))
3 plt.imshow(img_rgnir.data.transpose(1,2,0)/2500.0)
4 plt.show()
```



On remarquera les valeurs des axes `x` et `y` avec une origine en haut à gauche. Ceci est un référentiel purement matriciel (lignes et colonnes); autrement dit, il n'y a pas ici de géoréférence. Pour pallier à cette limitation, les librairies `rasterio` et `xarray` proposent une extension de la fonction `imshow` permettant d'afficher les coordonnées cartographiques ainsi qu'un contrôle la dynamique de l'image:

```
1 import rioxarray as rxr
2 fig, ax= plt.subplots(figsize=(6, 5))
3 img_rgnir.sel(band=[1,2,3]).plot.imshow(vmin=86, vmax=5000)
4 ax.set_title('imshow avec rioxarray')
5 plt.show()
```



### 3.3 Réhaussements visuels

Le réhaussement visuel d'une image vise principalement à améliorer la qualité visuelle d'une image en améliorant le contraste, la dynamique ou la texture d'une image. De manière générale, ce réhaussement ne modifie pas la donnée d'origine mais il est appliquée dynamiquement à l'affichage pour des fins d'inspection visuelle. Le réhaussement nécessite généralement une connaissance des caractéristiques statistiques d'une image. Ces statistiques sont ensuite exploitées pour appliquer diverses transformations linéaires ou non linéaires.

#### 3.3.1 Statistiques d'une image

On peut considérer un ensemble de statistique pour chacune des bandes d'une image:

- valeurs minimales et maximales
- valeurs moyennes,
- Quartiles (1er quartile, médiane et 3ième quartile), quantiles et percentiles.
- écart-type, et coefficients d'asymétrie (*skewness*) et d'aplatissement (*kurtosis*)

Ces statistiques doivent être calculées pour chaque bande d'une image multispectrale.

### 3 Réhaussement et visualisation d'images

En ligne de commande, **gdalinfo** permet d'interroger rapidement un fichier image pour connaitre ces statistiques univariées de base:

```
!gdalinfo -stats landsat7.tif
```

```
Driver: GTiff/GeoTIFF
Files: landsat7.tif
      landsat7.tif.aux.xml
Size is 2181, 1917
Coordinate System is:
PROJCS["WGS 84 / Pseudo-Mercator",
  GEOGCS["WGS 84",
    DATUM["WGS_1984",
      SPHEROID["WGS 84",6378137,298.257223563,
        AUTHORITY["EPSG","7030"]],
      AUTHORITY["EPSG","6326"]],
    PRIMEM["Greenwich",0,
      AUTHORITY["EPSG","8901"]],
    UNIT["degree",0.0174532925199433,
      AUTHORITY["EPSG","9122"]],
    AUTHORITY["EPSG","4326"]],
  PROJECTION["Mercator_1SP"],
  PARAMETER["central_meridian",0],
  PARAMETER["scale_factor",1],
  PARAMETER["false_easting",0],
  PARAMETER["false_northing",0],
  UNIT["metre",1,
    AUTHORITY["EPSG","9001"]],
  AXIS["X",EAST],
  AXIS["Y",NORTH],
  EXTENSION["+PROJ4","+proj=merc +a=6378137 +b=6378137 +lat_ts=0.0 +lon_0=0.0 +x_0=0.0 +y_0=0 +k=1.0 +units=m +nadgrids=@null +wktext +no_defs"],
  AUTHORITY["EPSG","3857"]]
Origin = (-13651650.0000000000000000,4576290.0000000000000000)
Pixel Size = (30.00000000000000,-30.00000000000000)
Metadata:
  AREA_OR_POINT=Area
  OVR_RESAMPLING_ALG=NEAREST
  TIFFTAG_RESOLUTIONUNIT=1 (unitless)
  TIFFTAG_XRESOLUTION=1
  TIFFTAG_YRESOLUTION=1
Image Structure Metadata:
  COMPRESSION=DEFLATE
  INTERLEAVE=PIXEL
Corner Coordinates:
Upper Left  (-13651650.000, 4576290.000) (122d38' 5.49"W, 37d58'40.08"N)
Lower Left   (-13651650.000, 4518780.000) (122d38' 5.49"W, 37d34'10.00"N)
Upper Right  (-13586220.000, 4576290.000) (122d 2'49.53"W, 37d58'40.08"N)
Lower Right   (-13586220.000, 4518780.000) (122d 2'49.53"W, 37d34'10.00"N)
Center       (-13618935.000, 4547535.000) (122d20'27.51"W, 37d46'26.05"N)
Band 1 Block=512x512 Type=Byte, ColorInterp=Red
  Min=19.000 Max=233.000
  Minimum=19.000, Maximum=233.000, Mean=98.433, StdDev=21.164
  NoData Value=0
  Overviews: 1091x959, 546x480
  Metadata:
    STATISTICS_MAXIMUM=233
    STATISTICS_MEAN=98.433096940153
    STATISTICS_MINIMUM=19
    STATISTICS_STDDEV=21.164021026458
Band 2 Block=512x512 Type=Byte, ColorInterp=Green
  Min=19.000 Max=178.000
  Minimum=19.000, Maximum=178.000, Mean=55.068, StdDev=22.204
  NoData Value=0
  Overviews: 1091x959, 546x480
  Metadata:
    STATISTICS_MAXIMUM=178
    STATISTICS_MEAN=55.067787534804
    STATISTICS_MINIMUM=19
    STATISTICS_STDDEV=22.203571974581
Band 3 Block=512x512 Type=Byte, ColorInterp=Blue
  Min=19.000 Max=187.000
  Minimum=19.000, Maximum=187.000, Mean=43.341, StdDev=20.330
  NoData Value=0
  Overviews: 1091x959, 546x480
  Metadata:
    STATISTICS_MAXIMUM=187
    STATISTICS_MEAN=43.340507443056
    STATISTICS_MINIMUM=19
    STATISTICS_STDDEV=20.32987736339
```

Les librairies de base comme **rasterio** et **xarray** produisent facilement un sommaire des statistiques de base avec la fonction **stats**:

```

1 import rasterio as rio
2 import numpy as np
3 with rio.open('landsat7.tif') as src:
4     stats= src.stats()
5     print(stats)

```

La librairie `xarray` donne accès à des fonctionnalités plus sophistiquées comme le calcul des quantiles:

```

1 import rioxarray as riox
2 with riox.open_rasterio('landsat7.tif', masked= True) as src:
3     print(src)
4 quantiles = src.quantile(dim=['x','y'], q=[.025,.25,.5,.75,.975])
5 quantiles

```

```

<xarray.DataArray (band: 3, y: 1917, x: 2181)> Size: 50MB
[12542931 values with dtype=float32]
Coordinates:
 * band      (band) int64 24B 1 2 3
 * x         (x) float64 17kB -1.365e+07 -1.365e+07 ... -1.359e+07
 * y         (y) float64 15kB 4.576e+06 4.576e+06 ... 4.519e+06 4.519e+06
spatial_ref int64 8B 0
Attributes:
 AREA_OR_POINT: Area
 OVR_RESAMPLING_ALG: NEAREST
 TIFFTAG_RESOLUTIONUNIT: 1 (unitless)
 TIFFTAG_XRESOLUTION: 1
 TIFFTAG_YRESOLUTION: 1
 STATISTICS_MAXIMUM: 233
 STATISTICS_MEAN: 98.433096940153
 STATISTICS_MINIMUM: 19
 STATISTICS_STDDEV: 21.164021026458
 scale_factor: 1.0
 add_offset: 0.0

<xarray.DataArray (quantile: 5, band: 3)> Size: 120B
array([[ 54.,   19.,   19.],
       [ 85.,   38.,   27.],
       [ 99.,   54.,   38.],
       [111.,   69.,   57.],
       [140.,  102.,   89.]])
Coordinates:
 * band      (band) int64 24B 1 2 3
 * quantile (quantile) float64 40B 0.025 0.25 0.5 0.75 0.975

```

### 3.3.1.1 Calcul de l'histogramme

Le calcul d'un histogramme pour une image (une bande) permet d'avoir une vue plus détaillée de la répartition des valeurs radiométriques. Le calcul d'un histogramme nécessite minimalement de faire le choix du nombre de barre (*bins* ou de la largeur). Un *bin* est un intervalle de valeurs pour lequel on peut calculer le nombre de valeurs observées dans l'image. La fonction de base pour ce type de calcul est la fonction `numpy.histogram()`:

```

1 import numpy as np
2 array = np.random.randint(0,10,100) # 100 valeurs aléatoires entre 0 et 10
3 hist, bin_limites = np.histogram(array, density=True)
4 print('valeurs :',hist)
5 print(';limites :',bin_limites)

```

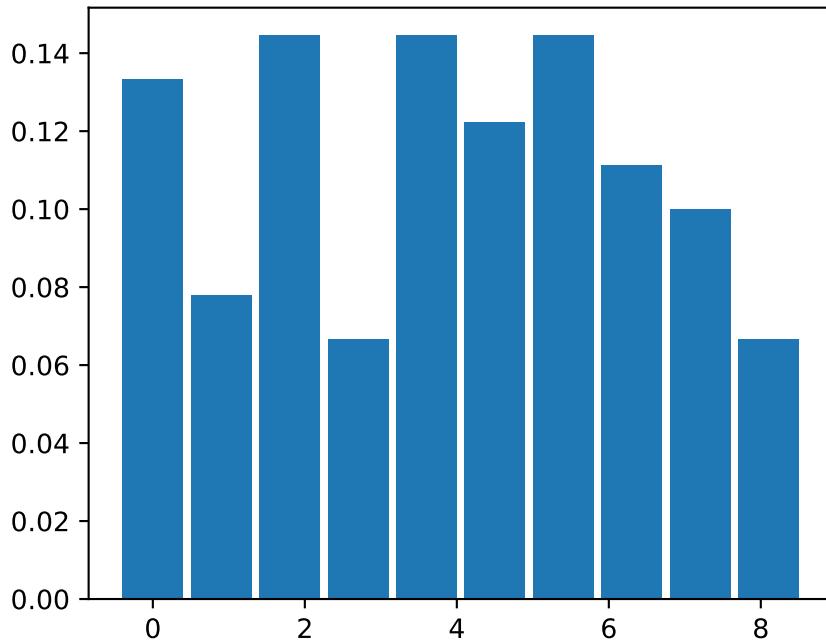
```

valeurs : [0.13333333 0.07777778 0.14444444 0.06666667 0.14444444 0.12222222
0.14444444 0.1111111 0.1 0.06666667]
;limites : [0. 0.9 1.8 2.7 3.6 4.5 5.4 6.3 7.2 8.1 9. ]

```

Le calcul se fait avec 10 intervalles par défaut.

```
1 fig, ax= plt.subplots(figsize=(5, 4))
2 plt.bar(bin_limites[:-1],hist)
3 plt.show()
```

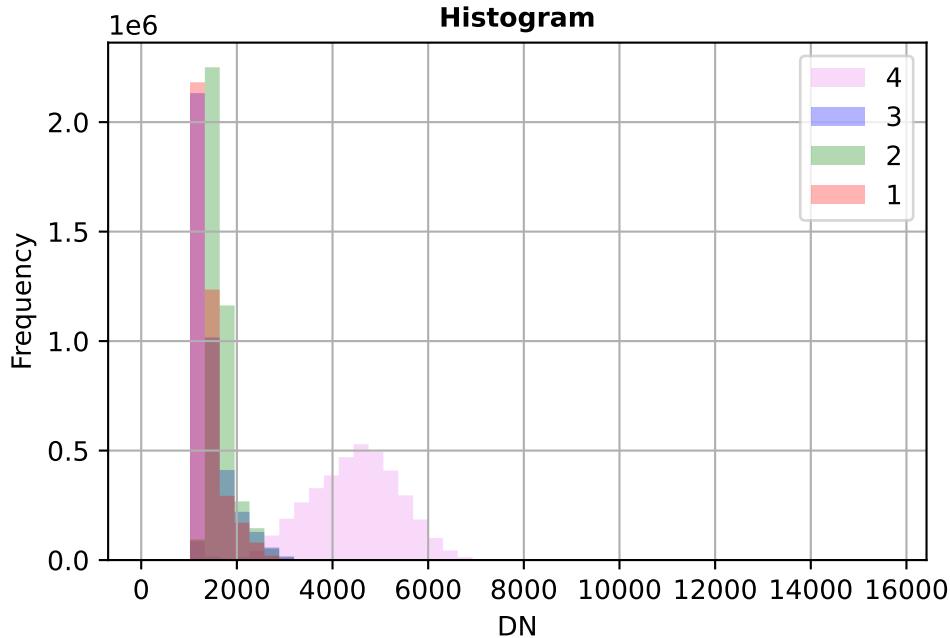


Pour des besoins de visualisation, le calcul des valeurs extrêmes de l'histogramme peut aussi se faire via les quantiles comme discutés auparavant.

### Visualisation des histogrammes

La librairie **rasterio** est probablement l'outil le plus simples pour visualiser rapidement des histogrammes sur une image multi-spectrale:

```
1 import rasterio as rio
2 from rasterio.plot import show_hist
3 with rio.open('RGBNIR_of_S2A.tif') as src:
4     show_hist(src, bins=50, lw=0.0, stacked=False, alpha=0.3,histtype='stepfilled', title="Histogram")
```



### 3.3.2 Réhaussements linéaires

Le réhaussement linéaire (*linear stretch*) d'une image est la forme la plus simple de réhaussement, elle consiste à 1) optimiser les valeurs des pixels d'une image afin de maximiser la dynamique disponibles à l'affichage, ou 2) à changer le format de stockage des valeurs (de 8 bits à 16 bits):

$$\text{nouvelle valeur d'un pixel} = \frac{\text{valeur d'un pixel} - min_0}{max_0 - min_0} \times (max_1 - min_1) + min_1 \quad (3.1)$$

Par cette opération, on passe de la dynamique de départ ( $max_0 - min_0$ ) vers la dynamique cible ( $max_1 - min_1$ ). Bien que cette opération semble triviale, il est important d'être conscient des trois contraintes suivantes:

1. **Faire attention à la dynamique cible**, ainsi, pour sauvegarder une image en format 8 bit, on utilisera alors  $max_1 = 255$  et  $min_1 = 0$ .
2. **Préservation de la valeur de no data** : il faut faire attention à la valeur  $min_1$  dans le cas d'une valeur présente pour *no\_data*. Par exemple, si *no\_data*=0 alors il faut s'assurer que  $min_1 > 0$ .
3. **Précision du calcul** : si possible réaliser la division ci-dessus en format *float*

#### 3.3.2.1 Cas des histogrammes asymétriques

Dans certains cas, la distribution de valeurs est très asymétrique et présente une longue queue avec des valeurs extrêmes élevées (à droite ou à gauche de l'histogramme). Le cas des images SAR est particulièrement représentatif de ce type de données. En effet, celles-ci peuvent présenter une distribution de valeurs de type exponentiel. Il est alors préférable d'utiliser des **percentiles** au préalable afin d'explorer la forme de l'histogramme et la distribution des valeurs:

```

1 NO_DATA_FLOAT= -999.0
2 # on prend tous les pixels de la première bande
3 values = img_SAR[0].values.flatten().astype(float)
4 # on exclut les valeurs invalides
5 values = values[~np.isnan(values)]
6 # on exclut le no data
7 values = values[values!=NO_DATA_FLOAT]
8 # calcul des percentiles
9 percentiles_position= (0,0.1,1,2,50,98,99,99.9,100)
10 percentiles= dict(zip(percentiles_position, np.percentile(values, percentiles_position)))
11 print(percentiles)

```

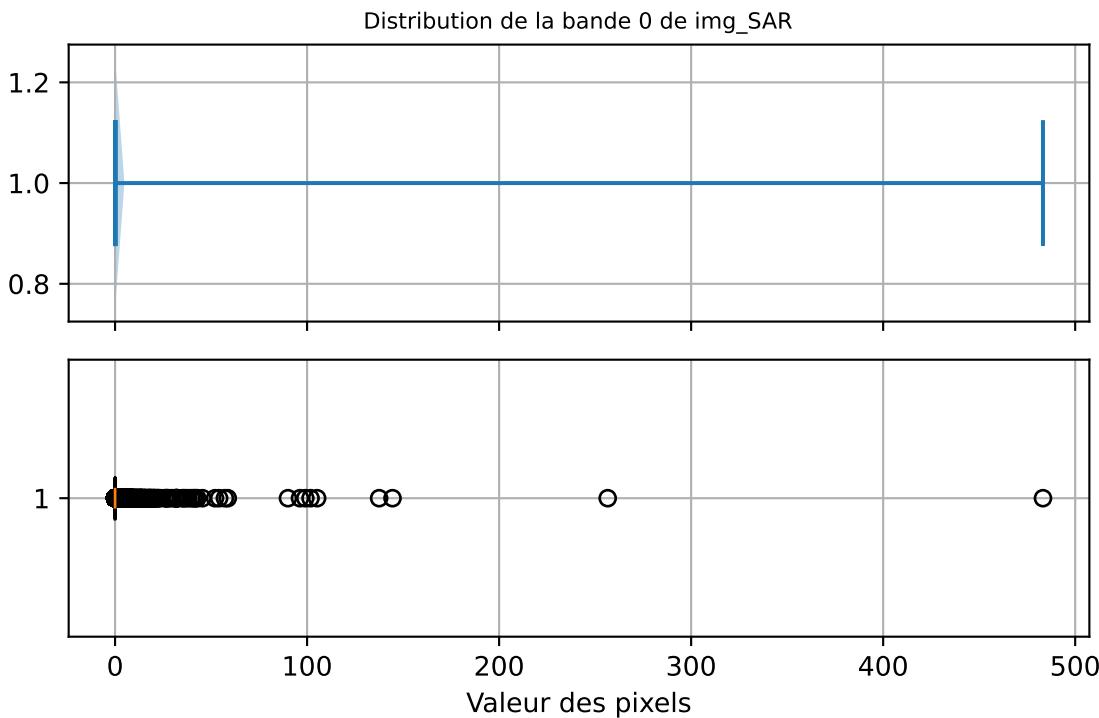
```
{0: np.float64(8.172580237442162e-06), 0.1: np.float64(1.588739885482937e-05), 1: np.float64(8.657756850880105e-05), 2:
→ np.float64(0.0001884606552214325), 50: np.float64(0.012372820172458877), 98: np.float64(0.1719470709562302), 99:
→ np.float64(0.27963151514529694), 99.9: np.float64(1.5235805057287233), 100: np.float64(483.223876953125)}
```

On constate que la valeur médiane (**0.012**) est très faible, ce qui signifie que 50% des valeurs sont inférieures à cette valeur alors que la valeur maximale (**483**) est 10 000 fois plus élevée! Une manière de visualiser cette distribution de valeurs est d'utiliser `boxplot` et `violinplot` de la librairie `matplotlib`:

```

1 fig, ax = plt.subplots(nrows=2, ncols=1, figsize=(6, 4), sharex=True)
2 ax[0].set_title('Distribution de la bande 0 de img_SAR', fontsize='small')
3 ax[0].grid(True)
4 ax[0].violinplot(values, orientation  ='horizontal',
5                   quantiles =(0.01,0.02,0.50,0.98,0.99),
6                   showmeans=False,
7                   showmedians=True)
8 ax[1].set_xlabel('Valeur des pixels')
9 ax[1].grid(True)
10 bplot = ax[1].boxplot(values, notch = True, orientation  ='horizontal')
11 plt.tight_layout()
12 plt.show()

```



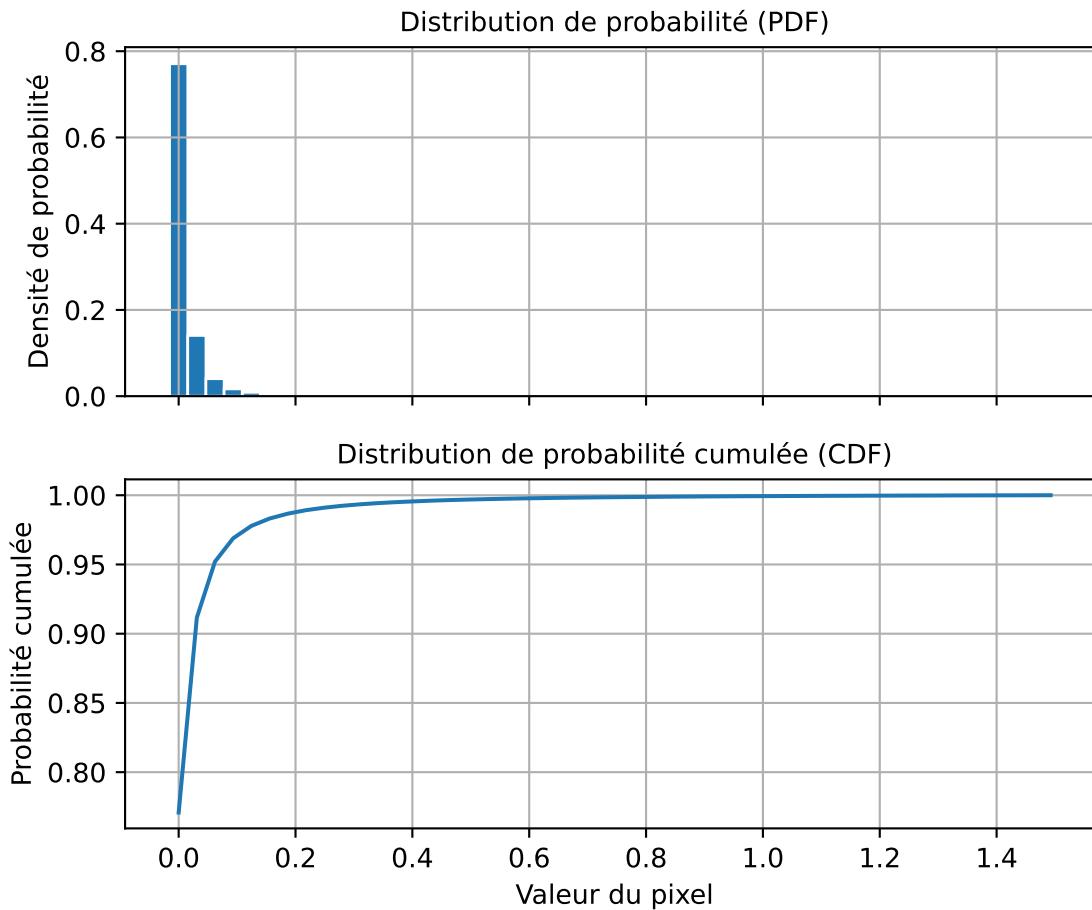
Afin de visualiser correctement l'histogramme, il faut se limiter à un intervalle de valeurs plus réduit. Dans le code ci-dessous, on impose à la fonction `np.histogram` de compter les valeurs de pixels dans des intervalles de valeurs fixés par la fonction `np.linspace(percentiles[0.1],percentiles[99.9], 50)` où `percentiles[0.1]` et `percentiles[99.9]` sont les 0.1% et 99.9% percentiles respectivement:

```

1 hist, bin_edges = np.histogram(values,
2                                 bins=np.linspace(percentiles[0.1],
3                                                 percentiles[99.9], 50),
4                                                 density=True)
5
6 fig, ax = plt.subplots(nrows=2, ncols=1, figsize=(6, 5), sharex=True)
7 ax[0].bar(bin_edges[:-1],
8            hist*(bin_edges[1]-bin_edges[0]),
9            width= (bin_edges[1]-bin_edges[0]),
10           edgecolor= 'w')
11 ax[0].set_title("Distribution de probabilité (PDF)")
12 ax[0].set_ylabel("Densité de probabilité")
13 ax[0].grid(True)
14
15 ax[1].plot(bin_edges[:-1],
16            hist.cumsum()*(bin_edges[1]-bin_edges[0]))
17 ax[1].set_title("Distribution de probabilité cumulée (CDF)")
18 ax[1].set_xlabel("Valeur du pixel")
19 ax[1].set_ylabel("Probabilité cumulée")
20 ax[1].grid(True)

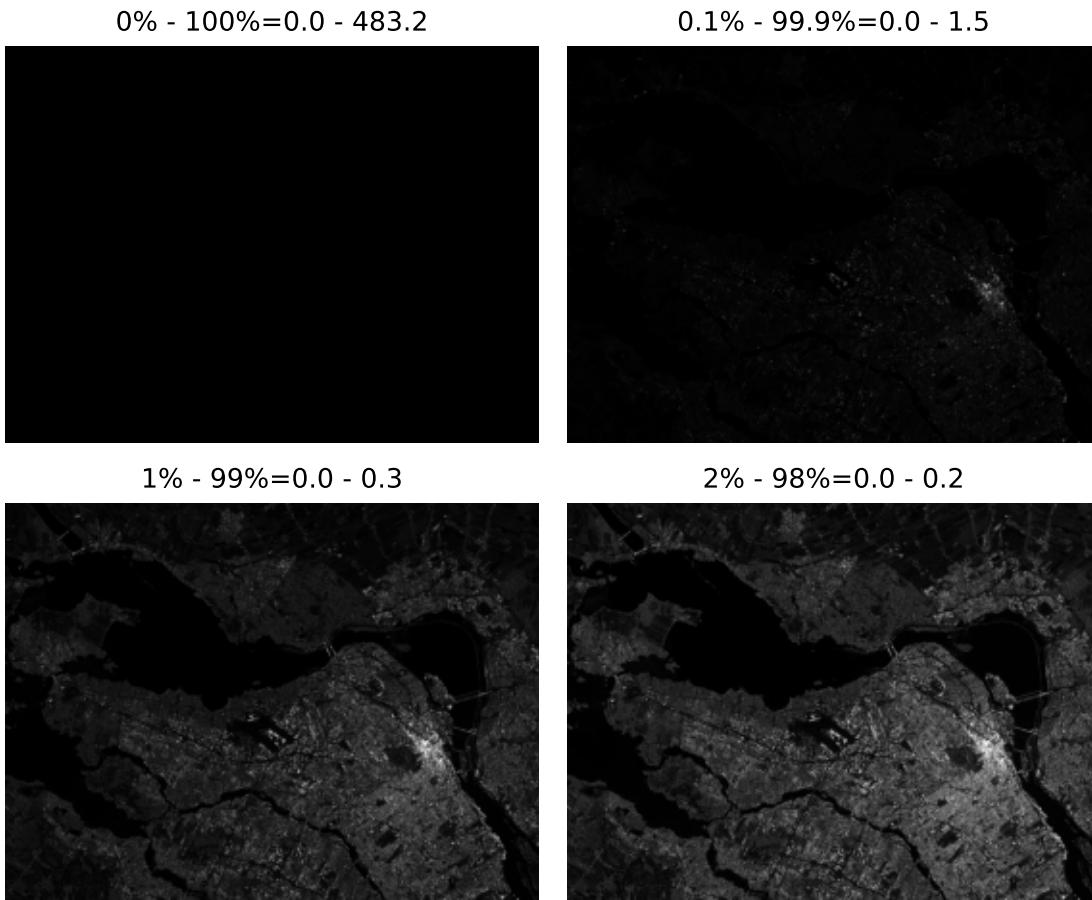
```

```
21 plt.tight_layout()
22 plt.show()
```



Au niveau de l'affichage avec `matplotlib`, la dynamique peut être contrôlée directement avec les paramètres `vmin` et `vmax` comme ceci:

```
1 fig, ax = plt.subplots(nrows=2, ncols=2, figsize=(6, 5), sharex=True, sharey=True)
2 [a.axis('off') for a in ax.flatten()]
3 ax[0,0].imshow(img_SAR[0].values, vmin=percentiles[0], vmax=percentiles[100])
4 ax[0,0].set_title(f"0% - 100%={percentiles[0]:2.1f} - {percentiles[100]:2.1f}")
5 ax[0,1].imshow(img_SAR[0].values, vmin=percentiles[0.1], vmax=percentiles[99.9])
6 ax[0,1].set_title(f"0.1% - 99.9%={percentiles[0.1]:2.1f} - {percentiles[99.9]:2.1f}")
7 ax[1,0].imshow(img_SAR[0].values, vmin=percentiles[1], vmax=percentiles[99])
8 ax[1,0].set_title(f"1% - 99%={percentiles[1]:2.1f} - {percentiles[99]:2.1f}")
9 ax[1,1].imshow(img_SAR[0].values, vmin=percentiles[2], vmax=percentiles[98])
10 ax[1,1].set_title(f"2% - 98%={percentiles[2]:2.1f} - {percentiles[98]:2.1f}")
11 plt.tight_layout()
```



### 3.3.3 Réhaussements non linéaires

#### 3.3.3.1 Réhaussement par fonctions

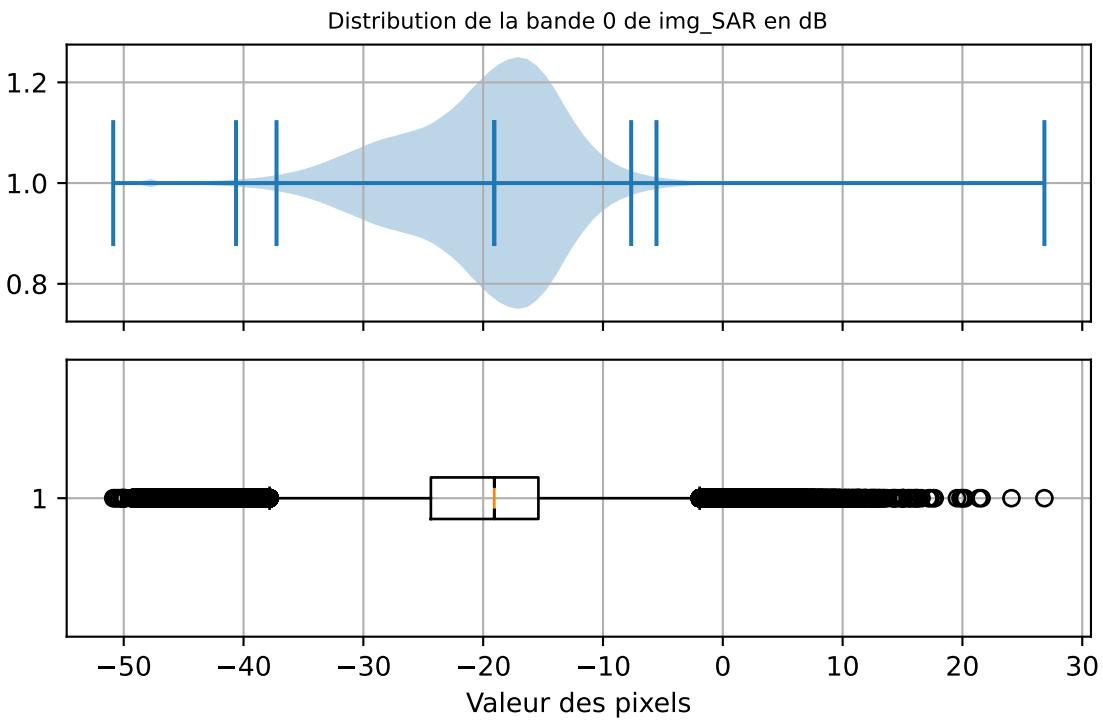
Le réhaussement par fonction consiste à appliquer une fonction non linéaire afin de modifier la dynamique de l'image. Par exemple, pour une image radar, une transformation populaire est d'afficher les valeurs de rétrodiffusion en décibel (dB) avec la fonction `log10()`.

```

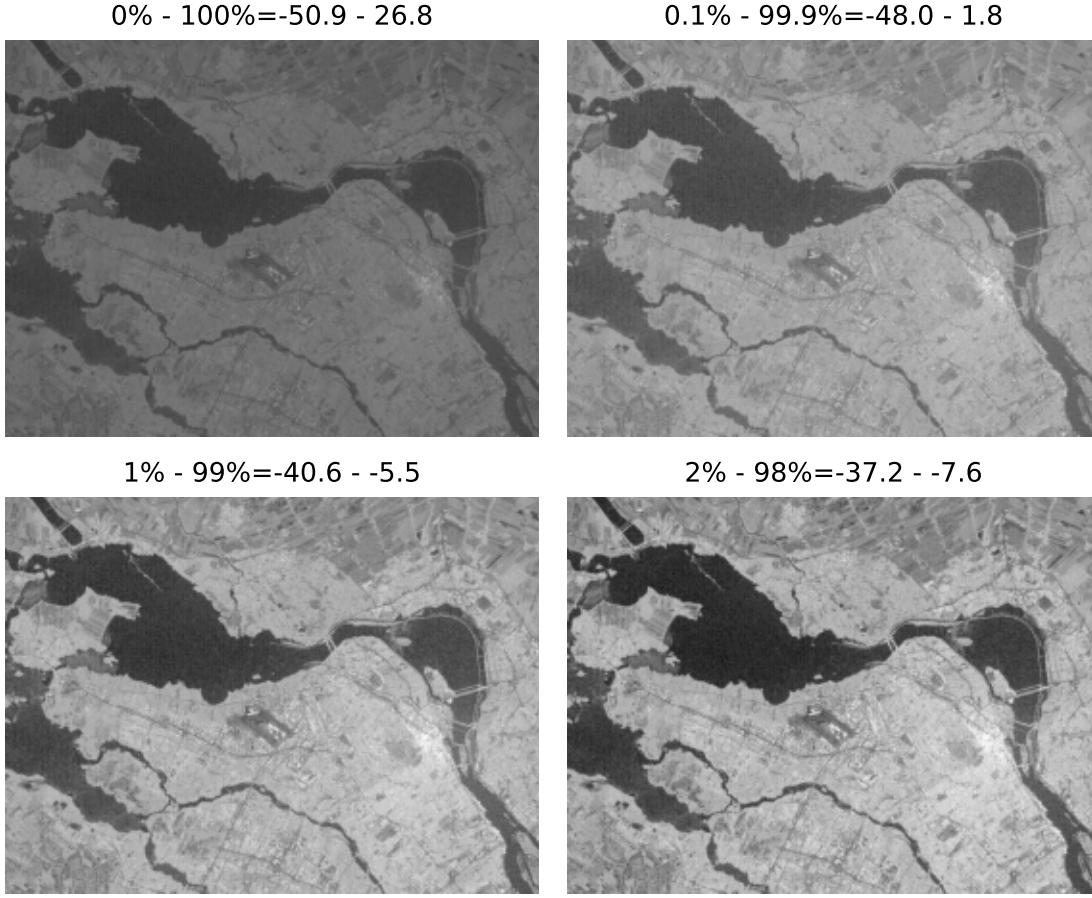
1 percentiles_position= (0,0.1,1,2,50,98,99,99.9,100)
2 values= 10*np.log10(img_SAR[0]).data
3 percentiles_db= dict(zip(percentiles_position, np.percentile(values, percentiles_position)))
4 print(percentiles_db)
```

```
(0: np.float64(-50.87641143798828), 0.1: np.float64(-47.98947440338135), 1: np.float64(-40.625951309204105), 2: np.float64(-37.24779266357422),
 ← 50: np.float64(-19.075313568115234), 98: np.float64(-7.6460519409179675), 99: np.float64(-5.534139237403945), 99.9:
 ← np.float64(1.8286541925668764), 100: np.float64(26.84148406982422))
```

Les boites à moustache (*boxplots*) ont une bien meilleure distribution qui est en effet très proche d'une distribution normale gaussienne:



On obtient ainsi les images suivantes:



### 3.3.3.2 Égalisation d'histogramme

L'égalisation d'histogramme consiste à modifier les valeurs des pixels d'une image source afin que la distribution cumulée des valeurs (CDF) devienne similaire à celle d'une image cible. La CDF (*Cumulative Distribution Function*) est simplement la somme cumulée des valeurs de l'histogramme:

$$CDF_{source}(i) = \frac{1}{K} \sum_{j=0}^{j \leq i} hist_{source}(j)$$

avec  $K$  choisi de façon à ce que la dernière valeur soit égale à 1 ( $CDF_{source}(i_{max}) = 1$ ). De la même manière,  $CDF_{cible}$  est la CDF d'une image cible. La formule générale pour l'égalisation d'histogramme est la suivante:

$$j = CDF_{cible}^{-1}(CDF_{source}(i))$$

On peut choisir  $CDF_{cible}$  comme correspondant à une image où chaque valeur de pixel est équiprobable (d'où le terme *égalisation*), ce qui veut dire  $hist_{cible}(j) = 1/L$  avec  $L$  égale au nombre de valeurs possibles dans l'image (par exemple  $L = 256$ ).

$$j = L \times CDF_{source}(i)$$

On peut appliquer cette procédure sur l'image SAR en dB de la façon suivante:

```

1 values= np.sort(np.log10(img_SAR[0].data.flatten()))
2 cdf_x= np.linspace(values[0], values[-1], 1000)
3 cdf_source= np.interp(cdf_x, values, np.arange(len(values))/len(values)*255)
4 values_eq=np.interp(np.log10(img_SAR[0].data), cdf_x, cdf_source).astype('uint8')
5 plt.imshow(values_eq)
6 plt.axis('off')

```



### 3.3.3.3 Palettes de couleur

Les palettes de couleurs sont appliquées dynamiquement à l'affichage sur une image à une seule bande. La librairie `matplotlib` contient un nombre considérable de [palettes](#).

```

1 from matplotlib import colormaps
2 list(colormaps)

```

Voici quelques exemples ci-dessous, les valeurs de l'image doivent être normalisées entre 0 et 1 ou entre 0 et 255 sinon les paramètres `vmin` et `vmax` doivent être spécifiés. On peut observer comment ces palettes révèlent les détails de l'image malgré une image originale très sombre.

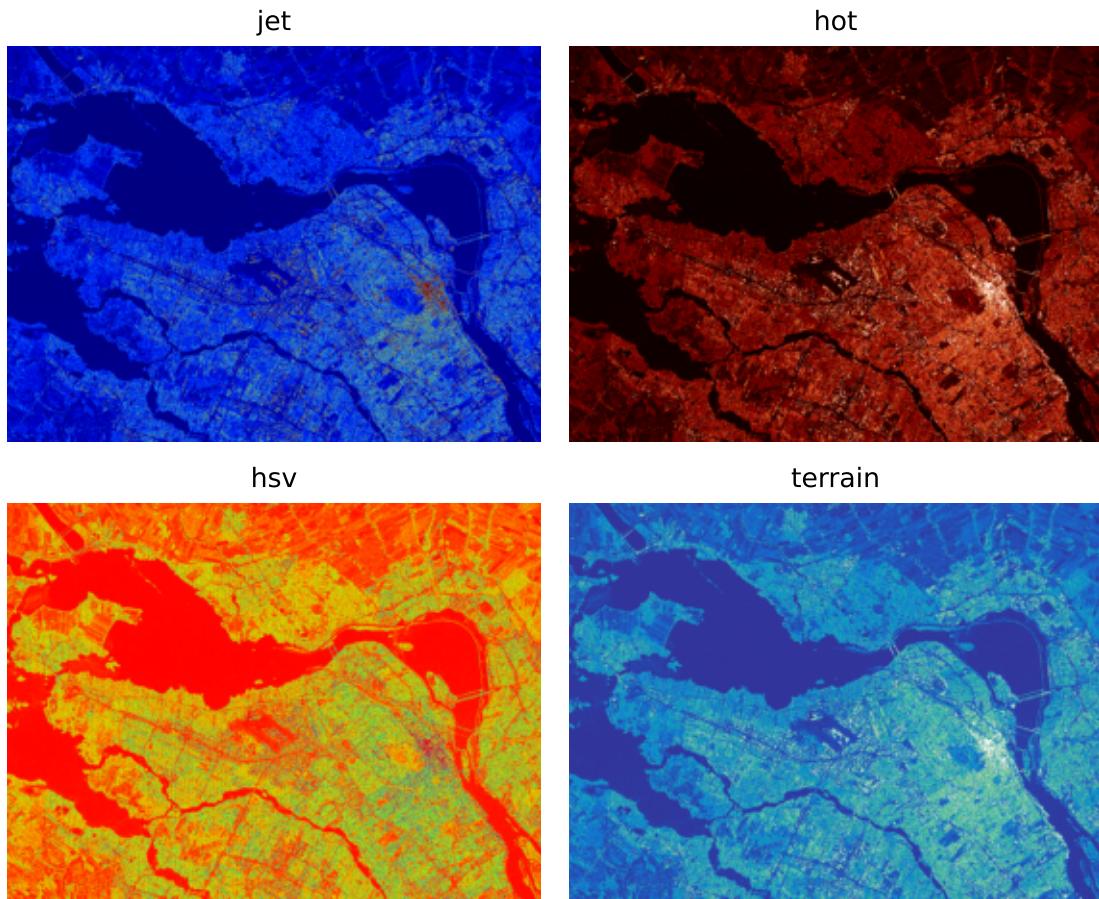
```

1 fig, ax = plt.subplots(nrows=2, ncols=2, figsize=(6, 5), sharex=True, sharey=True)
2 [a.axis('off') for a in ax.flatten()]
3 ax[0,0].imshow(img_SAR[0].data, vmin=percentiles[2], vmax=percentiles[98], cmap='jet')
4 ax[0,0].set_title(f"jet")
5 ax[0,1].imshow(img_SAR[0].data, vmin=percentiles[2], vmax=percentiles[98], cmap='hot')
6 ax[0,1].set_title(f"hot")
7 ax[1,0].imshow(img_SAR[0].data, vmin=percentiles[2], vmax=percentiles[98], cmap='hsv')
8 ax[1,0].set_title(f"hsv")
9 ax[1,1].imshow(img_SAR[0].data, vmin=percentiles[2], vmax=percentiles[98], cmap='terrain')

```

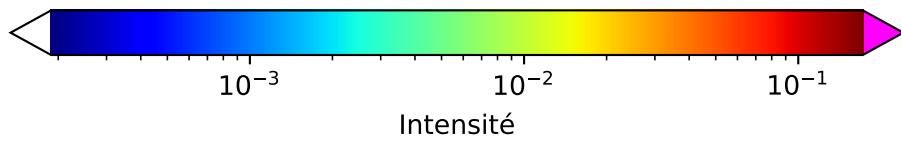
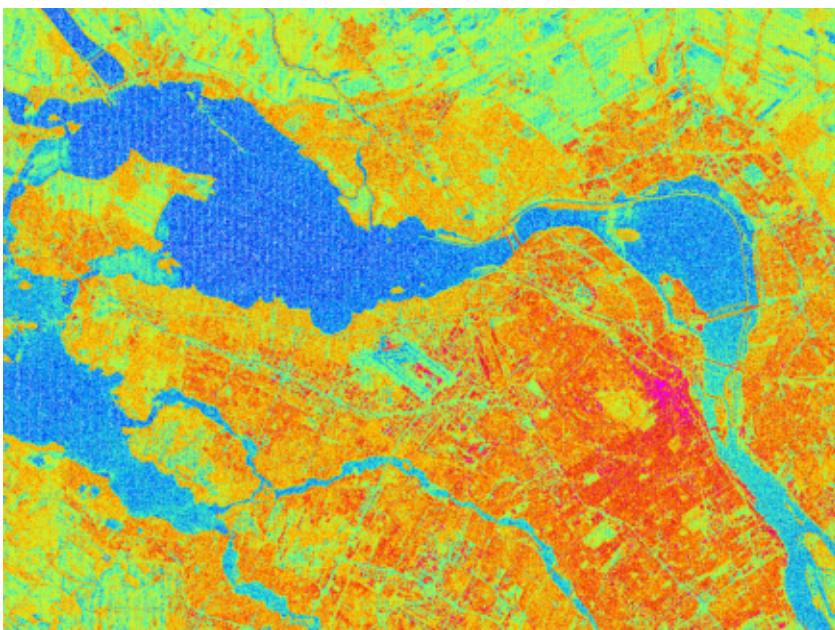
### 3 Réhaussement et visualisation d'images

```
10 ax[1,1].set_title(f"terrain")
11 plt.tight_layout()
```



Il peut être utile d'ajouter une barre de couleurs afin d'indiquer la correspondance entre les couleurs et les valeurs numériques:

```
1 import matplotlib as mpl
2 fig, ax = plt.subplots(figsize=(6, 6))
3 cmap= mpl.colormaps.get_cmap('jet').with_extremes(under='white', over='magenta')
4 h=plt.imshow(img_SAR[0].data, norm=mpl.colors.LogNorm(vmin=percentiles[2], vmax=percentiles[98]),
5             cmap=cmap)
6 fig.colorbar(h, ax=ax, orientation='horizontal', label="Intensité", extend='both')
7 ax.axis('off')
```



### 3.3.4 Composés colorés

Le système visuel humain est sensible seulement à la partie visible du spectre électromagnétique qui compose les couleurs de l'arc-en-ciel du bleu au rouge. L'ensemble des couleurs du spectre visible peut être obtenu à partir du mélange de trois couleurs primaires (rouge, vert et bleu). Ce système de décomposition à trois couleurs est à la base de la plupart des systèmes de visualisation ou de représentation de l'information de couleur. Si on prend le cas des images Sentinel-2, 12 bandes sont disponibles, plusieurs composés couleurs sont donc possibles (voir le site de [Copernicus](#)). Voici quelques exemples possibles, chaque composé mettant en valeur des propriétés différentes de la surface.

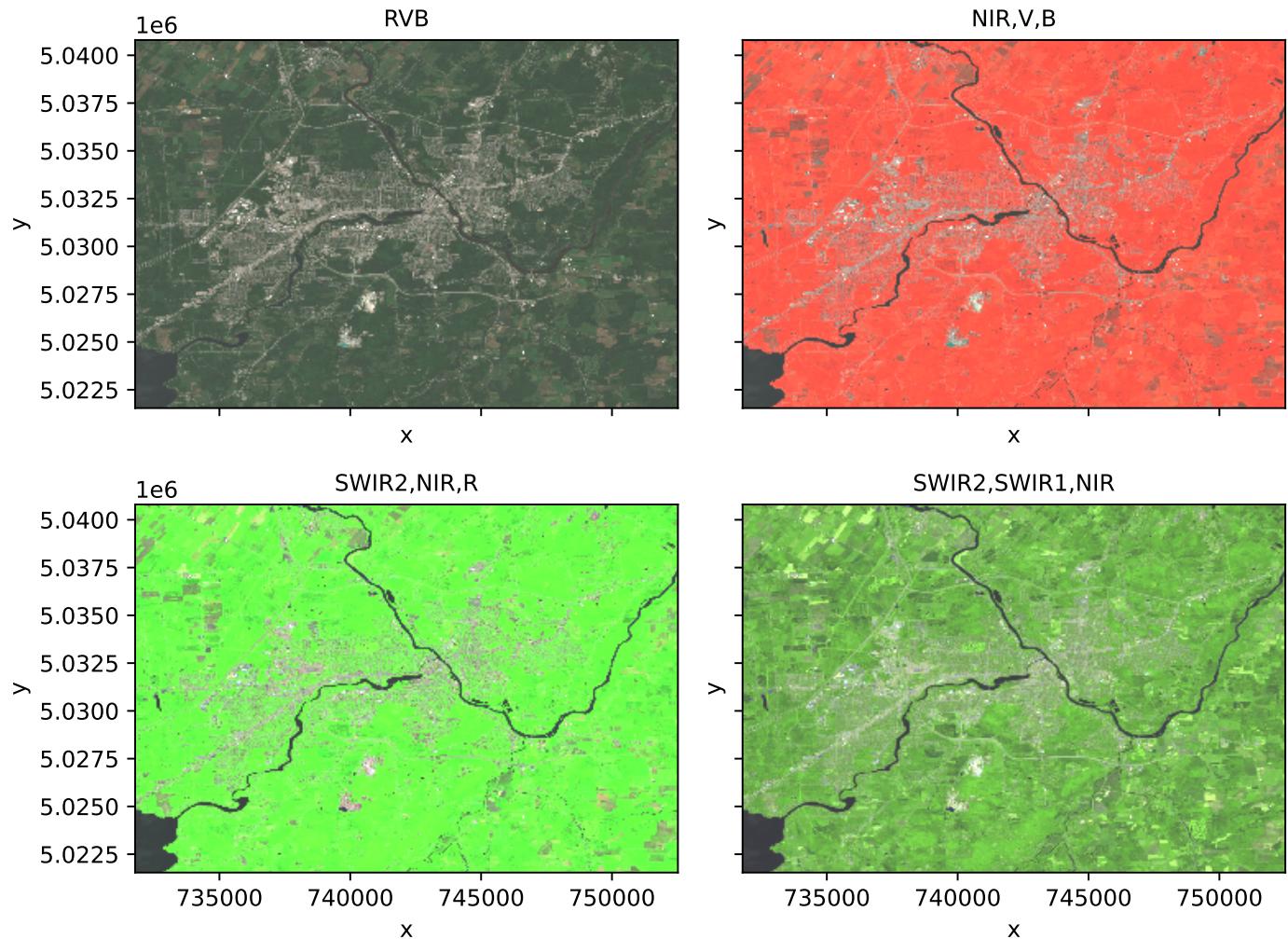
```

1 import rioxarray as rxr
2 fig, ax= plt.subplots(nrows=2, ncols= 2, figsize=(8, 6), sharex=True, sharey=True)
3 img_s2.sel(band=[4,3,2]).plot.imshow(vmin=86, vmax=4000, ax=ax[0,0])
4 ax[0,0].set_title('RVB')
5 img_s2.sel(band=[8,3,2]).plot.imshow(vmin=86, vmax=4000, ax=ax[0,1])
6 ax[0,1].set_title('NIR,V,B')
7 img_s2.sel(band=[12,8,4]).plot.imshow(vmin=86, vmax=4000, ax=ax[1,0])
8 ax[1,0].set_title('SWIR2,NIR,R')
9 img_s2.sel(band=[12,11,4]).plot.imshow(vmin=86, vmax=4000, ax=ax[1,1])

```

### 3 Réhaussement et visualisation d'images

```
10 ax[1,1].set_title('SWIR2,SWIR1,NIR')
11 plt.tight_layout()
12 plt.show()
```



## **Partie 2. Transformations des données satellitaires**

# 4 Transformations spectrales

## 4.1 Préambule

Assurez-vous de lire ce préambule avant d'exécuter le reste du notebook.

### 4.1.1 Objectifs

Dans ce chapitre, nous abordons l'exploitation de la dimension spectrale des images satellites. Ce chapitre est aussi disponible sous la forme d'un notebook Python:



#### Objectif

##### Objectifs d'apprentissage visés dans ce chapitre

À la fin de ce chapitre, vous devriez être en mesure de :

- comprendre le principe des indices spectraux;
- calculer différents indices avec spyndex;
- analyser le gain en information des indices;

### 4.1.2 Librairies

Les librairies qui vont être explorées dans ce chapitre sont les suivantes:

- SciPy
- NumPy
- spyndex
- Rasterio
- Xarray
- rioxarray

Dans l'environnement Google Colab, seul **rioxarray** doit être installés

```
1 %%capture
2 !pip install -qU matplotlib rioxarray xrscipy scikit-image pyarrow spyndex
```

Vérifiez les importations:

```

1 import numpy as np
2 import rioxarray as rxr
3 from scipy import signal
4 import xarray as xr
5 import xrscipy
6 import matplotlib.pyplot as plt
7 import spyndex
8 import rasterio as rio

```

### 4.1.3 Images utilisées

Nous utilisons les images suivantes dans ce chapitre:

```

1 %%capture
2 import gdown
3
4 gdown.download([
    'https://drive.google.com/uc?export=download&confirm=pbef&id=1a6Ypg0g10y4AJt9XWKWfnR12NW1XhNg_',
    'output= 'RGBNIR_of_S2A.tif')
5 gdown.download([
    'https://drive.google.com/uc?export=download&confirm=pbef&id=1a603L_ab0fU7h94K22At8qtBuLMGErwo',
    'output= 'sentinel2.tif')
6 gdown.download([
    'https://drive.google.com/uc?export=download&confirm=pbef&id=1_zwCLN-x7XJcNHJCH6Z8upEdUXtVtvsl',
    'output= 'berkeley.jpg')
7 gdown.download([
    'https://drive.google.com/uc?export=download&confirm=pbef&id=1dM6IVqjba6GHwTLmI7CpX8GP2z5txUq6',
    'output= 'SAR.tif')
8 gdown.download([
    'https://drive.google.com/uc?export=download&confirm=pbef&id=1aAq7crc_LoaLC3kG3HkQ6Fv5JfG0mswg',
    'output= 'carte.tif')
9 gdown.download([
    'https://drive.google.com/uc?export=download&confirm=pbef&id=1iCZNYTv0qEZRzPhe22nPdpV4Ks7NsY3b',
    'output= 'ASCIIdata_splib07b_rsSentinel2.zip')
10 !unzip -q ASCIIdata_splib07b_rsSentinel2.zip

```

Vérifiez que vous êtes capable de les lire :

```

1 with rxr.open_rasterio('berkeley.jpg', mask_and_scale= True) as img_rgb:
2     print(img_rgb)
3 with rxr.open_rasterio('RGBNIR_of_S2A.tif', mask_and_scale= True) as img_rgnbir:
4     print(img_rgnbir)
5 with rxr.open_rasterio('sentinel2.tif', mask_and_scale= True) as img_s2:
6     print(img_s2)

```

```
7 with rxr.open_rasterio('carte.tif', mask_and_scale= True) as img_carte:
8     print(img_carte)
```

## 4.2 Qu'est ce que l'information spectrale?

L'information spectrale touche à l'exploitation de la dimension spectrale des images (c.à.d le long des bandes spectrales de l'image). La taille de cette dimension spectrale dépend du type de capteurs considéré. Un capteur à très haute résolution spectrale par exemple aura très peu de bandes (4 ou 5). Un capteur multispectral pourra contenir une quinzaine de bande. À l'autre extrême, on trouvera les capteurs hyperspectraux qui peuvent contenir des centaines de bandes spectrales.

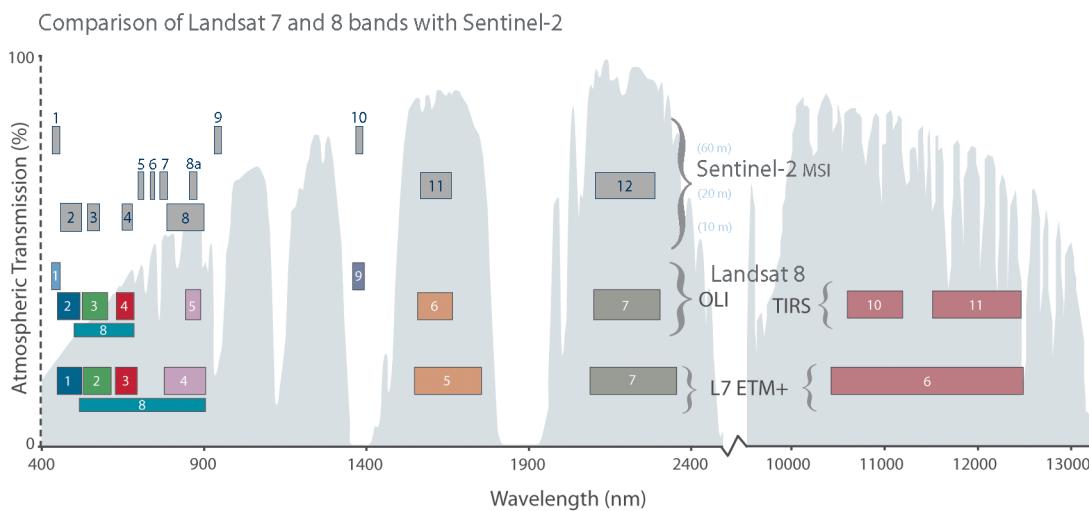
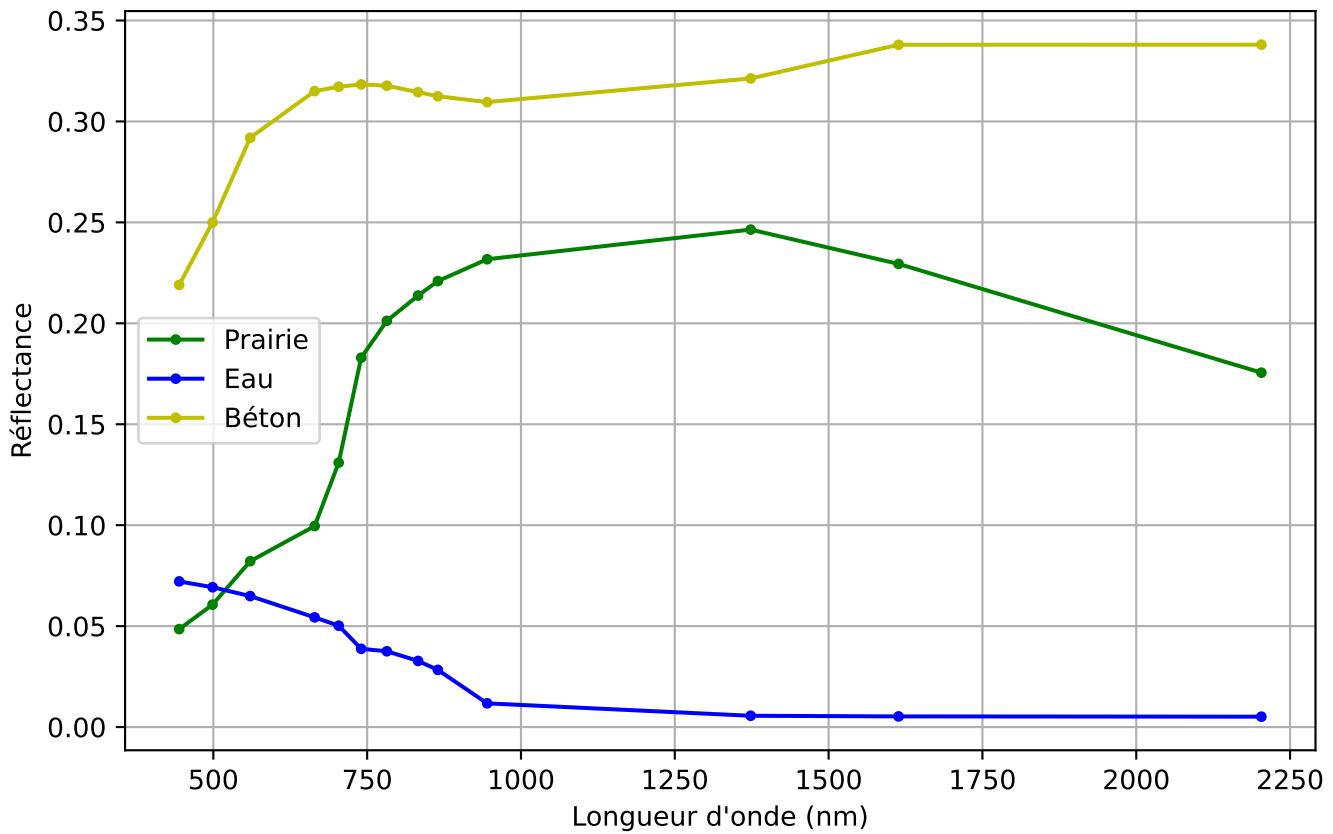


FIGURE 4.1 – Positions des bandes spectrales pour quelques capteurs ([source](#))

Pour une surface donnée, la forme des valeurs le long de l'axe spectrale caractérise le type de matériau observé ainsi que son état. On parle souvent alors de signature spectrale. On peut voir celle-ci comme une généralisation de la couleur d'un matériau au delà des bandes visibles du spectre. L'exploitation de ces signatures spectrales est probablement un des principes les plus importants en télédétection qui le distingue de la vision par ordinateur. L'[USGS](#) maintient une base de données spectrales acquises en laboratoire (Kokaly et Klein 2017). On peut observer sur la figure ci-dessous comment la forme et l'amplitude de trois signatures différentes peut changer en fonction du type de surface.

```
Text(0, 0.5, 'Réflectance')
```

Exemples de signatures spectrales pour trois surfaces différentes pour les bandes spectrales de Sentinel-2



### 4.3 Indices spectraux

Il existe une vaste littérature sur les indices spectraux, le choix d'un indice plutôt qu'un autre dépend fortement de l'application visée, nous allons simplement couvrir les principes de base ici. Le principe d'un indice spectral consiste à mettre en valeur certaines caractéristiques saillantes du spectre comme des pentes, des gradients, etc.

La librairie Python [Awesome Spectral Indices](#) maintient une liste de plus de 200 indices spectraux (radar et optiques). La liste complète est affichable avec la commande suivante:

```
1 spyndex.indices
```

```
SpectralIndices(['AFRI1600', 'AFRI2100', 'ANDWI', 'ARI', 'ARI2', 'ARVI', 'ATSAVI', 'AVI', 'AWEInsh', 'AWEish', 'BAI', 'BAIS2', 'BCC',
← 'BI', 'BITM', 'BIXS', 'BLFEI', 'BNDVI', 'BRBA', 'BWDRVI', 'BaI', 'CCI', 'CIG', 'CIRE', 'CRI550', 'CRI700', 'CSI', 'CSIT', 'CVI', 'DBI',
← 'DBSI', 'DPDD', 'DSI', 'DSWI1', 'DSWI1', 'DSWI2', 'DSWI3', 'DSWI4', 'DSWI5', 'DVI', 'DVIPplus', 'DpRVIIH', 'DpRVIVV', 'EBBI', 'EBI', 'EMBI', 'ENDVI',
← 'EVI', 'EV12', 'EViv', 'EXG', 'ExGR', 'ExR', 'FAI', 'FCVI', 'GARI', 'GBNDVI', 'GCC', 'GDVI', 'GEMI', 'GLI', 'GM1', 'GM2', 'GNDVI', 'GOSAVI',
← 'GRNDVI', 'GRVI', 'GSAVI', 'GVMI', 'IAVI', 'IBI', 'IKAW', 'IPVI', 'IRECI', 'LSWI', 'MBI', 'MBWI', 'MCARI', 'MCARI2', 'MCARI705',
← 'MCARIOSAVI', 'MCARIOSAVI705', 'MGRVI', 'MIRBI', 'MLSWI26', 'MLSWI27', 'MNDVI', 'MNDWI', 'MNLI', 'MRBVI', 'MSAVI', 'MSI', 'MSR', 'MSR705',
← 'MTCI', 'MTVI1', 'MTVI2', 'MuWIR', 'NBAI', 'NBLI', 'NBLIOLI', 'NBR', 'NBR2', 'NBRSWIR', 'NBRT1', 'NBRT2', 'NBRT3', 'NBRplus', 'NBSIMS',
← 'NBUI', 'ND705', 'NDBI', 'NDBai', 'NDCI', 'NDDI', 'NDGI', 'NDGlai', 'NDII', 'NDISib', 'NDISig', 'NDISimndwi', 'NDISIndwi', 'NDISIr', 'NDMI',
← 'NDPI', 'NDPoli', 'NDPoni', 'NDREI', 'NDST', 'NDST', 'NDSTW', 'NDSTW', 'NDSWIR', 'NDSATI', 'NDSoI', 'NDTI', 'NDVI', 'NDVI705', 'NDVIMNDWI',
← 'NDVIT', 'NDWI', 'NDWIns', 'NDYI', 'NGRDI', 'NHFD', 'NIRV', 'NIRvH2', 'NIRvP', 'NLI', 'NMDI', 'NRFig', 'NRFir', 'NSDS', 'NSDS11', 'NSDS12',
← 'NSDS13', 'NSTv1', 'NSTv2', 'NW1', 'NormG', 'NormNR', 'NormR', 'OCVI', 'OSAVI', 'OSI', 'PI', 'PISI', 'PSRI', 'QpRVI', 'RCC', 'RDVI', 'REDSI',
← 'RENDVI', 'RFDI', 'RGBVI', 'RGRI', 'RI', 'RI4XS', 'RNDVI', 'RVI', 'S2REP', 'S2WI', 'S3', 'SARVI', 'SAVI', 'SAV12', 'SAVIT', 'SEVI', 'SI',
← 'SICI', 'SLAVI', 'SR', 'SR2', 'SR3', 'SR555', 'SR705', 'SWI', 'SWM', 'SeLI', 'TCARI', 'TCARIOSAVI', 'TCARIOSAVI705', 'TCI', 'TDVI', 'TGI',
← 'TRRV1', 'TSAVI', 'TTVI', 'TVI', 'TWI', 'TrIVI', 'UI', 'VARI', 'VAR1700', 'VDDPI', 'VHVVD', 'VHVVR', 'VI6T', 'VI700', 'VIBI', 'VIG',
← 'VVVHD', 'VVVHR', 'VVVHS', 'VgNIRBI', 'VnNIRBI', 'WDRVI', 'WDVI', 'WI1', 'WI2', 'WI2015', 'WRI', 'bNIRV', 'KEVI', 'KIPVI', 'kNDVI', 'kRVI',
← 'KVARI', 'mND705', 'mSR705', 'sNIRvLSWI', 'sNIRvNDPI', 'sNIRvNDVILSWIP', 'sNIRvNDVILSWIS', 'sNIRvSWIR'])
```

## 4 Transformations spectrales

Le détail d'un indice particulier, par exemple le 'NDVI', est aussi affichable:

```
1 spyndex.indices["NDVI"]
```

```
SpectralIndex(NDVI: Normalized Difference Vegetation Index)
  * Application Domain: vegetation
  * Bands/Parameters: ['N', 'R']
  * Formula: (N-R)/(N+R)
  * Reference: https://ntrs.nasa.gov/citations/19740022614
```

`spyndex` pré-suppose une nomenclature prédéfinie des **bandes**, on peut voir la correspondance sur le tableau ci-dessous:

```
1 spyndex.bands
```

```
Bands(['A', 'B', 'G', 'G1', 'N', 'N2', 'R', 'RE1', 'RE2', 'RE3', 'S1', 'S2', 'T', 'T1', 'T2', 'WV', 'Y'])
```

TABLEAU 4.1 – Noms des bandes Sentinel-2

Index	Noms	Spyndex	Noms
1	B01	A	Aérosol
2	B02	B	Bleu
3	B03	G	Vert
4	B04	R	Rouge
5	B05	RE1	Red edge 1
6	B06	RE1	Red edge 2
7	B07	RE2	Red edge 3
8	B08	N	Proche-infrarouge 1
9	B08A	N2	Proche-infrarouge 2
10	B09	-	Vapeur d'eau
11	B11	S1	Infra-rouge onde courte 1
12	B12	S2	Infra-rouge onde courte 1

Deux options sont possibles, on peut soit renommer les noms des bandes avec `xarray` ou "mapper" les noms vers les noms appropriés. Regardons les dimensions de notre jeux de données:

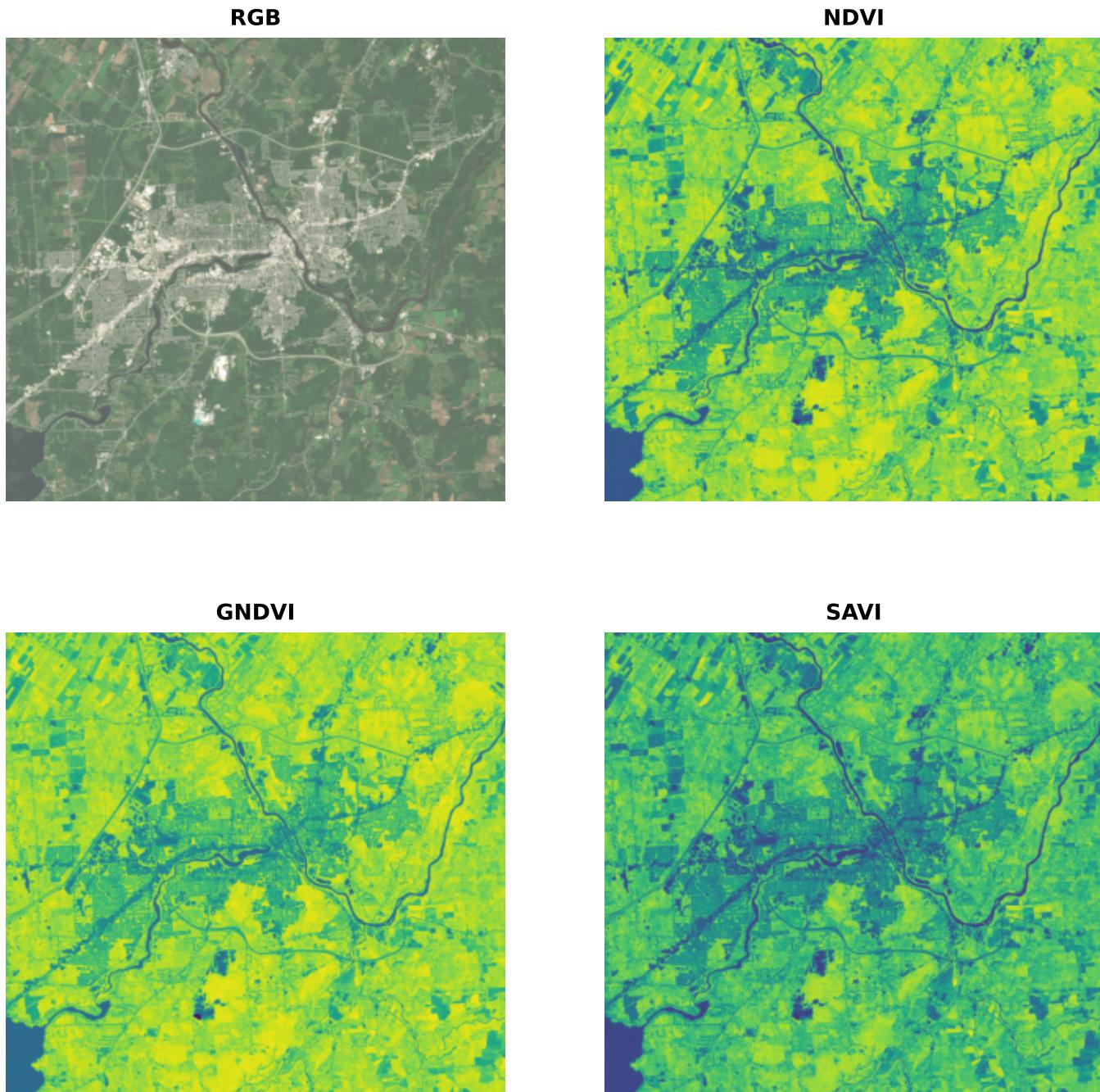
```
1 img_s2.dims
```

```
('band', 'y', 'x')
```

On peut simplement changer les index (`coords`) de la dimension `band`:

```
1 img_s2 = img_s2.sel(band = list(range(1,13))).assign_coords({'band':["A", "B", "G", "R", "RE1", "RE2",
  ↴ "RE3", "N", "N2", "WV", "S1", "S2"]})
2 img_s2=img_s2/10000 # normalisation en réflectance
```

```
1 from rasterio import plot
2 idx = spyndex.computeIndex(
3     index = ["NDVI","GNDVI","SAVI"],
4     params = {
5         "N": img_s2.sel(band = "N"),
6         "R": img_s2.sel(band = "R"),
7         "G": img_s2.sel(band = "G"),
8         "L": 0.5
9     }
10 )
11
12 # Plot the indices (and the RGB image for comparison)
13 fig, ax = plt.subplots(2,2,figsize = (9,9))
14 [a.axis('off') for a in ax.flatten()]
15 plot.show(img_s2.sel(band = ["R","G","B"]).data / 0.3,ax = ax[0,0],title = "RGB")
16 plot.show(idx.sel(index = "NDVI"),ax = ax[0,1],title = "NDVI")
17 plot.show(idx.sel(index = "GNDVI"),ax = ax[1,0],title = "GNDVI")
18 plot.show(idx.sel(index = "SAVI"),ax = ax[1,1],title = "SAVI")
```



On peut vérifier l'utilité des indices en vérifiant leur séparabilité pour certaines classes d'intérêts. Nous reprenons ici l'exemple de la section [section 6.2.3](#) pour vérifier l'utilité des indices **NDVI**, **NDWI** et **NDBI**:

```

1 import pandas as pd
2 import seaborn as sns
3
4 # On sélectionne trois classes
5 class_selected= [1,3,9]

```

```

6 df= pd.concat([gdf[gdf['class'] ==c] for c in class_selected], ignore_index=True)
7 idx["Land Cover"] = [nom_classes[l] for l in df["class"].tolist()]
8 # Compute the desired spectral indices
9 idx = spyndex.computeIndex(
10     index = ["NDVI","NDWI","NDBI"],
11     params = {
12         "N": df["SR_B8"],
13         "R": df["SR_B4"],
14         "G": df["SR_B3"],
15         "S1": df["SR_B11"]
16     }
17 )
18
19 colors= [couleurs_classes[c] for c in class_selected]
20 # Plot a pairplot to check the indices behaviour
21 plt.figure(figsize = (15,15))
22 g = sns.PairGrid(idx,hue = "Land Cover",palette = sns.color_palette(colors))
23 g.map_lower(sns.scatterplot)
24 g.map_upper(sns.kdeplot,fill = True,alpha = .5)
25 g.map_diag(sns.kdeplot,fill = True)
26 g.add_legend()
27 plt.show()

```

#### 4 Transformations spectrales

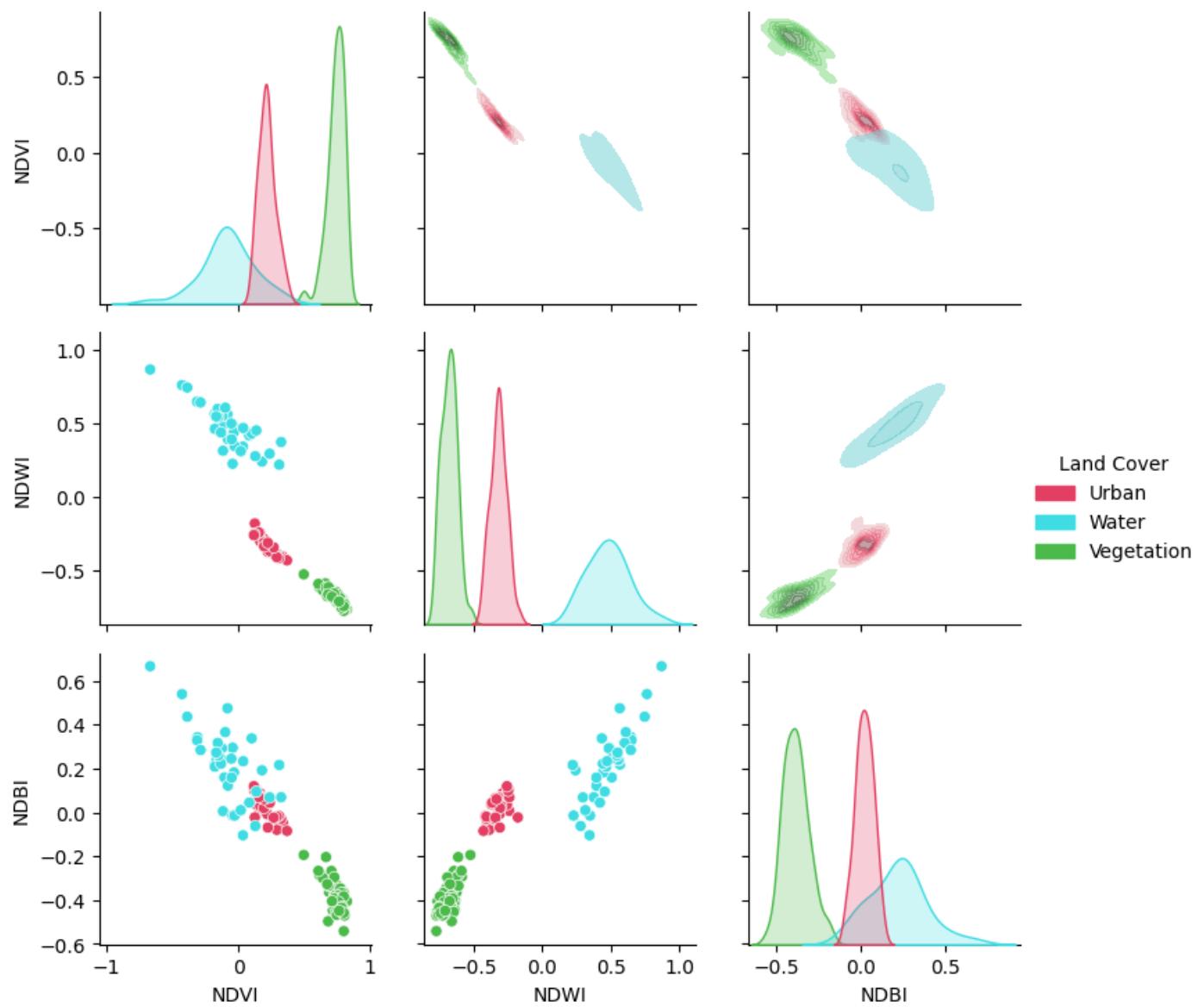


FIGURE 4.2 – Visualisation des points d'une image Sentinel-2 pour trois classes

# 5 Transformations spatiales

## 5.1 Préambule

Assurez-vous de lire ce préambule avant d'exécuter le reste du notebook.

### 5.1.1 Objectifs

Dans ce chapitre, nous abordons quelques techniques de traitement d'images dans le domaine spatial uniquement. Ce chapitre est aussi disponible sous la forme d'un notebook Python sur Google Colab:



#### Objectif

##### Objectifs d'apprentissage visés dans ce chapitre

À la fin de ce chapitre, vous devriez être en mesure de :

- comprendre le principe de la décomposition de Fourier;
- comprendre le principe de la convolution;
- appliquer un filtrage local à l'aide d'une fenêtre;
- segmenter une image en super-pixels et calculer leurs propriétés

### 5.1.2 Librairies

Les librairies utilisées dans ce chapitre sont les suivantes:

- SciPy
- NumPy
- opencv-python · PyPI
- scikit-image
- Rasterio
- Xarray
- rioxarray

Dans l'environnement Google Colab, seul `rioxarray` doit être installé:

```
1 %%capture
2 !pip install -qU matplotlib rioxarray xrscipy scikit-image
```

Vérifiez les importations:

```

1 import numpy as np
2 import numpy.fft
3 import rioxarray as rxr
4 from scipy import signal, ndimage
5 import xarray as xr
6 import xrscipy
7 import matplotlib.pyplot as plt
8 from skimage import data, measure, graph, segmentation, color
9 from skimage.color import rgb2gray
10 from skimage.segmentation import slic, mark_boundaries
11 import pandas as pd

```

### 5.1.3 Images utilisées

Nous utilisons les images suivantes dans ce chapitre:

```

1 %%capture
2 import gdown
3
4 gdown.download(
    'https://drive.google.com/uc?export=download&confirm=pbef&id=1a6Ypg0g1Oy4AJt9XWkWfnR12NW1XhNg_',
    output= 'RGBNIR_of_S2A.tif')
5 gdown.download(
    'https://drive.google.com/uc?export=download&confirm=pbef&id=1a4PQ68Ru8zBphbQ22j0sgJ4D2quw-Wo6',
    output= 'landsat7.tif')
6 gdown.download(
    'https://drive.google.com/uc?export=download&confirm=pbef&id=1_zwCLN-x7XJcNHJCH6Z8upEdUxtVtvS1',
    output= 'berkeley.jpg')
7 gdown.download(
    'https://drive.google.com/uc?export=download&confirm=pbef&id=1dM6IVqjba6GHwTLmI7CpX8GP2z5txUq6',
    output= 'SAR.tif')

```

Vérifiez que vous êtes capable de les lire :

```

1 with rxr.open_rasterio('berkeley.jpg', mask_and_scale= True) as img_rgb:
2     print(img_rgb)
3 with rxr.open_rasterio('RGBNIR_of_S2A.tif', mask_and_scale= True) as img_rgbnir:
4     print(img_rgbnir)
5 with rxr.open_rasterio('SAR.tif', mask_and_scale= True) as img_SAR:
6     print(img_SAR)

```

## 5.2 Analyse fréquentielle

L'analyse fréquentielle, issue du traitement du signal, permet d'avoir un autre point de vue sur les données à partir de ses composantes harmoniques. La modification de ces composantes de Fourier modifie l'ensemble de l'image et permet de corriger des problèmes systématiques comme des artefacts ou du bruit de capteur. Bien que ce domaine soit un peu éloigné de la télédétection, les images issues des capteurs sont toutes sujettes à des étapes de traitement du signal et il faut donc en connaître les grands principes afin de pouvoir comprendre certains enjeux lors des traitements.

### 5.2.1 La transformée de Fourier

La transformée de Fourier permet de transformer une image dans un espace fréquentiel. Cette transformée est complètement réversible. Dans le cas des images numériques, on parle de **2D-DFT** (*2D-Discrete Fourier Transform*) qui est un algorithme optimisé pour le calcul fréquentiel (Cooley et Tukey 1965). La *1D-DFT* peu s'écrire simplement comme une projection sur une série d'exponentielles complexes:

$$X[k] = \sum_{n=0 \dots N-1} x[n] \times \exp(-j \times 2\pi \times k \times n/N) \quad (5.1)$$

La transformée inverse prend une forme similaire:

$$x[k] = \frac{1}{N} \sum_{n=0 \dots N-1} X[n] \times \exp(j \times 2\pi \times k \times n/N) \quad (5.2)$$

Le signal d'origine est donc reconstruit à partir d'une somme de sinusoides complexes  $\exp(j2\pi \frac{k}{N}n)$  de fréquence  $k/N$ . Noter qu'à partir de  $k = N/2$ , les sinusoides se répètent à un signe près et forme un miroir des composantes, la convention est alors de mettre ces composantes dans une espace négatif  $[-N/2, \dots, -1]$ .

Dans le cas d'un simple signal périodique à une dimension avec une fréquence de  $4/16$  (donc 4 périodes sur 16) on obtient deux pics de fréquence à la position de 4 cycles observés sur  $N = 16$  observations. Les puissances de Fourier sont affichées dans un espace fréquentiel en cycles par unité d'espacement de l'échantillon (avec zéro au début) variant entre -1 et +1. Par exemple, si l'espacement des échantillons est en secondes, l'unité de fréquence est cycles/seconde (ou Hz). Dans le cas de  $N$  échantillons, le pic sera observé à la fréquence  $+/- 4/16 = 0.25$  cycles/secondes. La fréquence d'échantillonnage  $F_s$  du signal a aussi beaucoup d'importance aussi et doit être au moins à deux fois la plus haute fréquence observée (ici  $F_s > 0.5$ ) sinon un phénomène de repliement appelé aliasing sera observé.

```

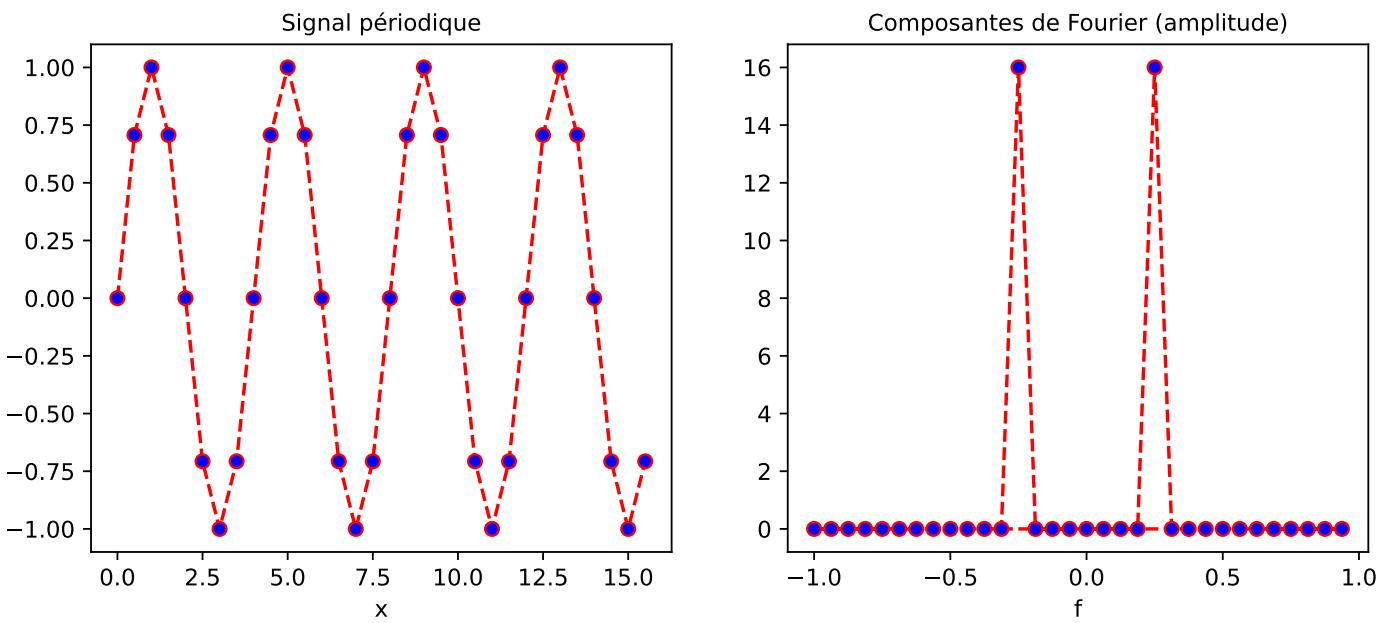
1 import math
2 Fs= 2.0
3 Ts= 1/Fs
4 N= 16
5 arr = xr.DataArray(np.sin(2*math.pi*np.arange(0,N,Ts)*4/16),
6                     dims='x', coords={'x': np.arange(0,N,Ts)})
7 fourier = np.fft.fft(arr)
8 freq = np.fft.fftfreq(fourier.size, d=Ts)

```

```

9 fourier = xr.DataArray(fourier,
10                         dims=('f'), coords={'f': freq})
11
12 fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(10, 4))
13 plt.subplot(1, 2, 1)
14 arr.plot.line(color='red', linestyle='dashed', marker='o', markerfacecolor='blue')
15 axes[0].set_title("Signal périodique")
16 plt.subplot(1, 2, 2)
17 np.abs(fourier).plot.line(color='red', linestyle='dashed', marker='o', markerfacecolor='blue')
18 axes[1].set_title("Composantes de Fourier (amplitude)")
19 plt.show()

```



### 5.2.2 Filtrage fréquentiel

Un filtrage fréquentiel consiste à modifier le spectre de Fourier afin d'éliminer ou de réduire certaines composantes fréquentielles. On distingue habituellement trois catégories de filtres fréquentiels:

1. Les filtres passe-bas qui ne préservent que les basses fréquences pour, par exemple, lisser une image.
2. Les filtres passe-hauts qui ne préservent que les hautes fréquences pour ne préserver que les détails.
3. Les filtres passe-bandes qui vont préserver les fréquences dans une bande de fréquence particulière.

La librairie **Scipy** contient différents filtres fréquentiels. Notez, qu'un filtrage fréquentiel est une simple multiplication de la réponse du filtre  $F[k]$  par les composantes fréquentielles du signal à filtrer  $X[k]$ :

$$X_f[k] = F[k] \times X[k] \quad (5.3)$$

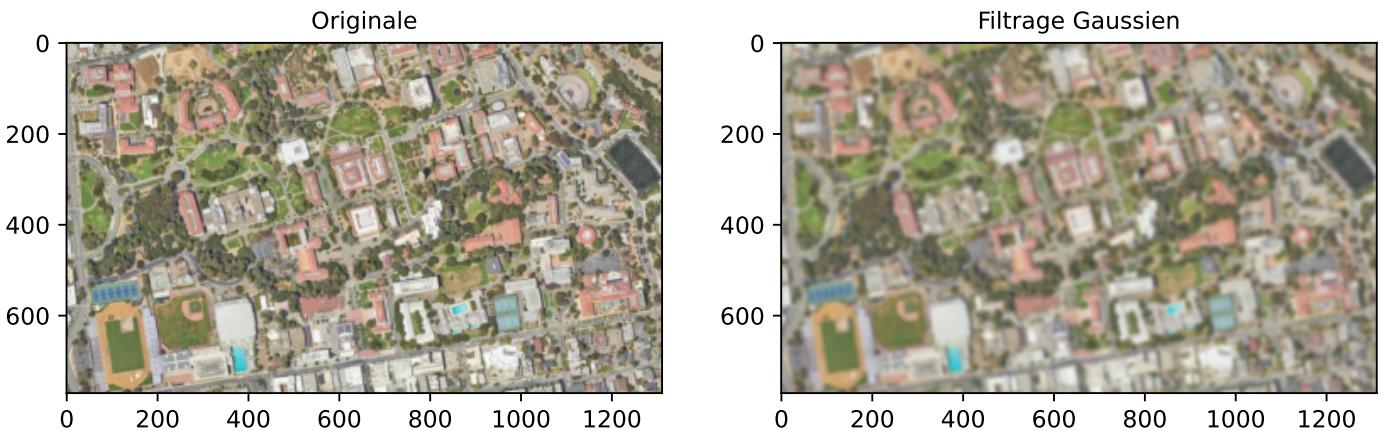
À noter que cette multiplication dans l'espace de Fourier est équivalente à une opération de convolution dans l'espace originale du signal  $x$ :

$$x_f = IDFT^{-1}[F] * x \quad (5.4)$$

```

1 fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 4))
2 input_ = numpy.fft.fft2(img_rgb.to_numpy())
3 result = [ndimage.gaussian_filter(input_[b], sigma=4) for b in range(3)] # on filtre chaque bande avec
4     ↳ un filtre Gaussien
5 result = numpy.fft.ifft2(result)
6 ax1.imshow(img_rgb.to_numpy().transpose(1, 2, 0).astype('uint8'))
7 ax1.set_title('Originale')
8 ax2.imshow(result.real.transpose(1, 2, 0).astype('uint8')) # La partie imaginaire n'est pas utile ici
9 ax2.set_title('Filtrage Gaussien')
9 plt.show()

```



### 5.2.3 L'aliasing

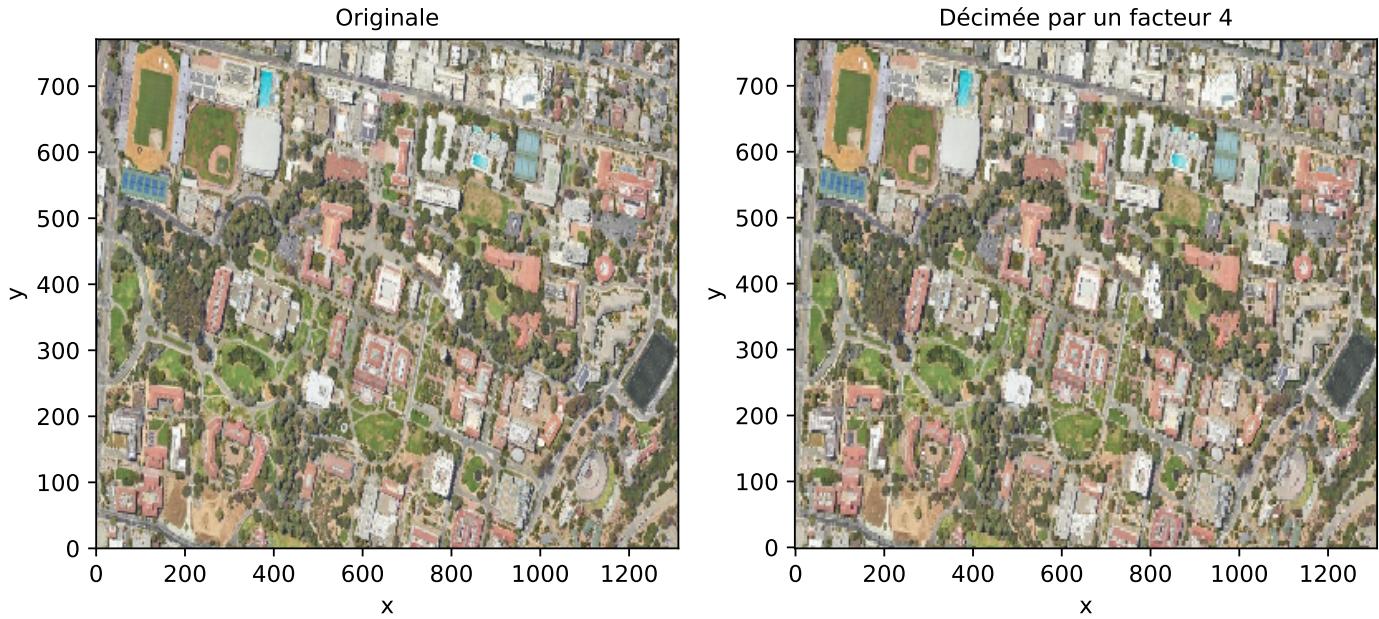
L'aliasing est un problème fréquent en traitement du signal. Il résulte d'une fréquence d'échantillonnage trop faible par rapport au contenu fréquentiel du signal. Cela peut se produire lorsque vous sous-échantillonner fortement une image avec un facteur de décimation (par exemple un pixel sur deux). En prenant un pixel sur deux, on réduit la fréquence d'échantillonnage d'un facteur 2 ce qui réduit le contenu fréquentiel de l'image et donc les fréquences maximales de l'image. L'image présente alors un aspect faussement texturée avec beaucoup de hautes fréquences:

```

1 fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(10, 4))
2 plt.subplot(1, 2, 1)
3 img_rgb.astype('int').plot.imshow(rgb="band")
4 axes[0].set_title("Originale")
5 plt.subplot(1, 2, 2)
6 img_rgb[:, ::4, ::4].astype('int').plot.imshow(rgb="band")

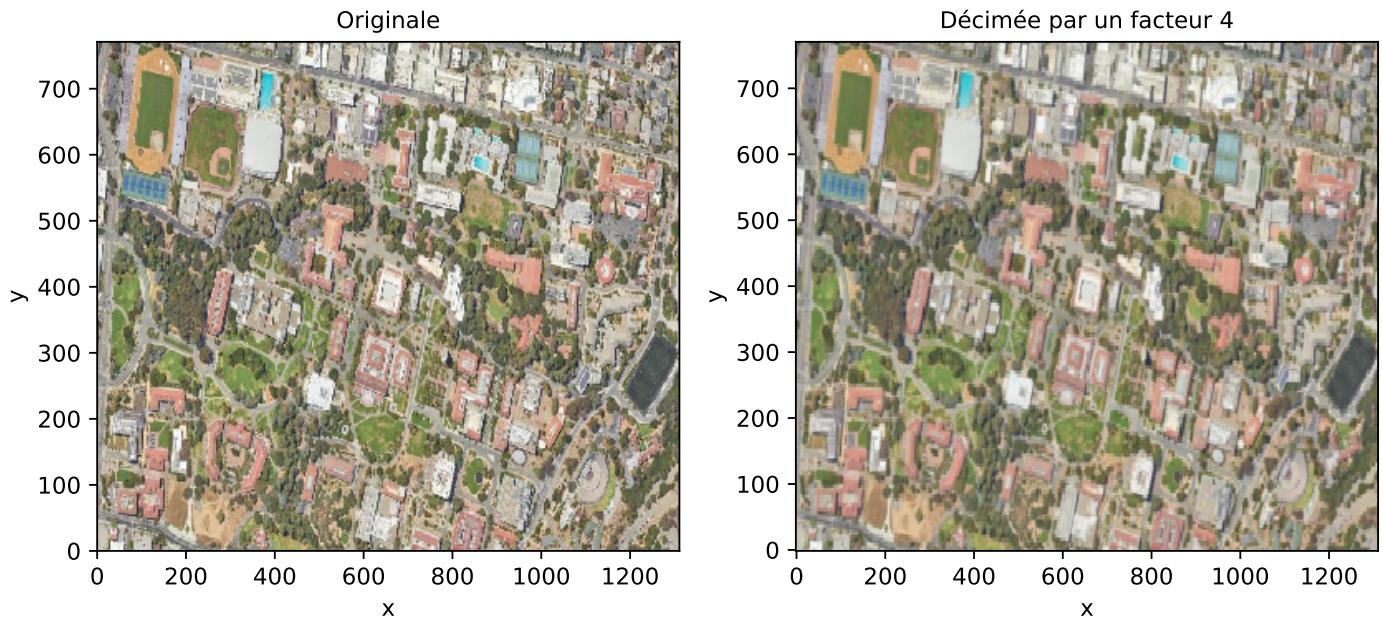
```

```
7 axes[1].set_title("Décimée par un facteur 4")
8 plt.show()
```



Une façon de réduire le contenu fréquentiel est de filtrer par un filtre passe-bas pour réduire les hautes fréquences par exemple avec un filtre Gaussien:

```
1 from scipy.ndimage import gaussian_filter
2 q= 4
3 sigma= q*1.1774/math.pi
4 arr = xr.DataArray(gaussian_filter(img_rgb.to_numpy(), sigma= (0,sigma,sigma)), dims=('band','y", "x"),
5 → coords= {'x': img_rgb.coords['x'], 'y': img_rgb.coords['y'], 'spatial_ref': 0})
6
7 fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(10, 4))
8 plt.subplot(1, 2, 1)
9 img_rgb.astype('int').plot.imshow(rgb="band")
10 axes[0].set_title("Originale")
11 plt.subplot(1, 2, 2)
12 arr[:,::q,::q].astype('int').plot.imshow(rgb="band")
13 axes[1].set_title("Décimée par un facteur 4")
14 plt.show()
```



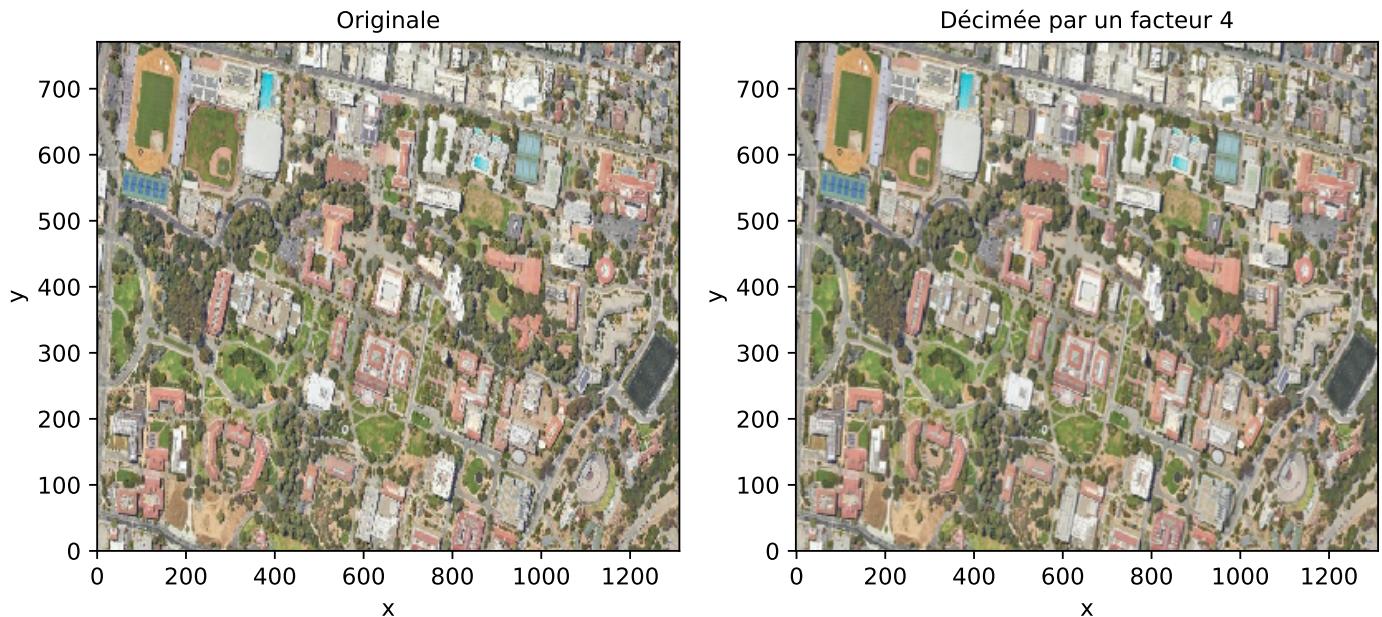
La fonction `decimate` dans `scipy.signal` réalise l'opération de décimation (*downsampling*) en une seule étape:

```

1 import xrscipy.signal as dsp
2
3 fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(10, 4))
4 plt.subplot(1, 2, 1)
5 img_rgb.astype('int').plot.imshow(rgb="band")
6 axes[0].set_title("Originale")
7 plt.subplot(1, 2, 2)
8 dsp.decimate(img_rgb, q=4, dim='x').astype('int').plot.imshow(rgb="band")
9 axes[1].set_title("Décimée par un facteur 4")

Text(0.5, 1.0, 'Décimée par un facteur 4')

```



### 5.3 Filtrage d'image

Le filtrage d'image a plusieurs objectifs en télédétection:

1. La réduction du bruit afin d'améliorer la résolution radiométrique et améliorer la lisibilité de l'image.
2. Le réhaussement de l'image afin d'améliorer le contraste ou faire ressortir les contours.
3. La production de nouvelles caractéristiques, c.-à.-d dérivées de nouvelles images mettant en valeur certaines informations dans l'image comme la texture, les contours, etc.

Il existe de nombreuses méthodes de filtrage dans la littérature qui sont habituellement regroupées en quatre catégories:

1. Le filtrage peut-être global ou local, c.-à.-d qu'il prend en compte soit toute l'image pour filtrer (ex: filtrage par Fourier), soit uniquement avec une fenêtre ou un voisinage local.
2. La fonction de filtrage peut-être linéaire ou non linéaire.
3. La fonction de filtrage peut être stationnaire ou adaptative.
4. Le filtrage peut-être mono-échelle ou multi-échelle.

La librairie **Scipy** ([Multidimensional image processing \(scipy.ndimage\)](#)) contient une panoplie complète de filtres.

### 5.3.1 Filtrage linéaire stationnaire

Un filtrage linéaire stationnaire consiste à appliquer une même pondération locale des valeurs des pixels dans une fenêtre glissante. La taille de cette fenêtre est généralement un chiffre impair (3,5, etc.) afin de définir une position centrale et une fenêtre symétrique. La valeur calculée à partir de tous les pixels dans la fenêtre est alors attribuée au pixel central.

Le filtre le plus simple est certainement le filtre moyen qui consiste à appliquer le même poids uniforme dans la fenêtre glissante. Par exemple pour un filtre 5x5:

$$F = \frac{1}{25} \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} \quad (5.5)$$

En python, on dispose des fonctions `rolling` et `sliding_window` définis dans la librairie `numpy`. Par exemple pour le cas du filtre moyen, on construit une nouvelle vue de l'image avec deux nouvelles dimensions `x_win` et `y_win`:

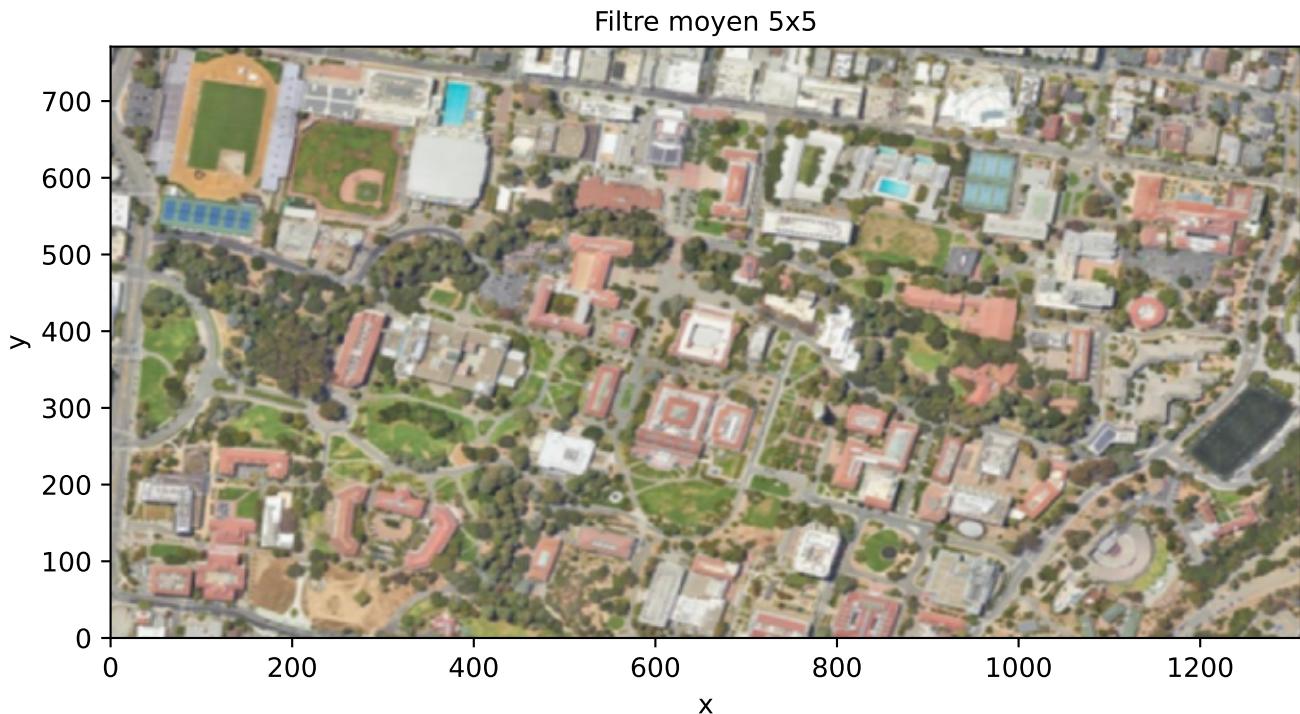
```
1 rolling_win = img_rgb.rolling(x=5, y=5, min_periods= 3, center= True).construct(x="x_win", y="y_win",
2   ↵ keep_attrs= True)
2 print(rolling_win[0,0,1,...])
3 print(rolling_win.shape)
```

```
<xarray.DataArray (x_win: 5, y_win: 5)> Size: 100B
array([[ nan,  nan,  nan,  nan,  nan],
       [ nan,  nan, 209., 210., 209.],
       [ nan,  nan, 213., 214., 212.],
       [ nan,  nan, 213., 212., 210.],
       [ nan,  nan, 210., 209., 206.]], dtype=float32)
Coordinates:
  band      int64 8B 1
  x        float64 8B 1.5
  y        float64 8B 0.5
  spatial_ref int64 8B 0
Dimensions without coordinates: x_win, y_win
(3, 771, 1311, 5, 5)
```

L'avantage de cette approche est qu'il n'y a pas d'utilisation inutile de la mémoire. Noter les `nan` sur les bords de l'image car la fenêtre déborde sur les bordures de l'image. Par la suite un opérateur de moyenne peut être appliqué sur les axes `x_win` et `y_win` correspondant aux fenêtres glissantes.

```
1 filtre_moyen= rolling_win.mean(dim= ['x_win', 'y_win'], skipna= True)
2 fig, ax = plt.subplots(nrows=1, ncols=1, figsize=(8, 4))
3 filtre_moyen.astype('int').plot.imshow(rgb="band")
4 ax.set_title("Filtre moyen 5x5")
```

```
Text(0.5, 1.0, 'Filtre moyen 5x5')
```



Lorsque la taille  $W$  de la fenêtre devient trop grande, il est préférable d'utiliser une convolution dans le domaine fréquentielle. La fonction `fftconvolve` de la librairie `scipy.signal` offre cette possibilité:

```

1 kernel = np.outer(signal.windows.gaussian(70, 8),
2                     signal.windows.gaussian(70, 8))
3 blurred = signal.fftconvolve(img_rgb, kernel, mode='same')
```

### 5.3.1.1 Filtrage par convolution

La façon la plus efficace d'appliquer un filtre linéaire est d'appliquer une convolution. La convolution est généralement très efficace car elle est peut être calculée dans le domaine fréquentiel. Prenons l'exemple du filtre de Scharr (Jahne et S. 1999), qui permet de détecter les contours horizontaux et verticaux:

$$F = \begin{bmatrix} -3 - 3j & 0 - 10j & +3 - 3j \\ -10 + 0j & 0 + 0j & +10 + 0j \\ -3 + 3j & 0 + 10j & +3 + 3j \end{bmatrix} \quad (5.6)$$

Remarquez l'utilisation de chiffres complexes afin de passer deux filtres différents sur la partie réelle et imaginaire.

```

1 scharr = np.array([[ -3-3j,  0-10j,   +3 -3j],
2                   [-10+0j,  0+ 0j,   +10 +0j],
3                   [ -3+3j,  0+10j,   +3 +3j]]) # Gx + j*Gy
```

```

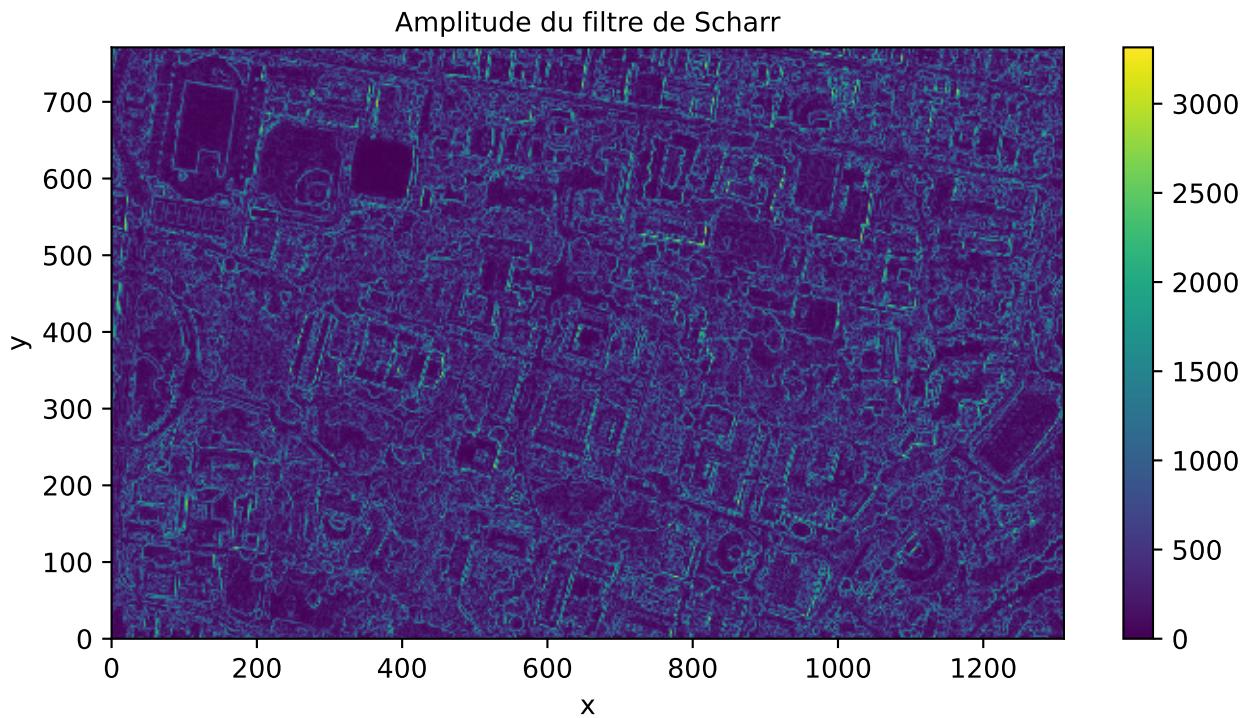
4 print(img_rgb.isel(band=0).shape)
5 grad = signal.convolve2d(img_rgb.isel(band=0), scharr, boundary='symm', mode='same')
6 # on reconstruit un xarray à partir du résultat:
7 arr = xr.DataArray(np.abs(grad), dims=("y", "x"), coords= {'x': img_rgb.coords['x'], 'y':
8   → img_rgb.coords['y'], 'spatial_ref': 0})
9 print(arr)
10 fig, ax = plt.subplots(nrows=1, ncols=1, figsize=(8, 4))
11 arr.plot.imshow()
12 ax.set_title("Amplitude du filtre de Scharr")

```

```

(771, 1311)
<xarray.DataArray (y: 771, x: 1311)> Size: 8MB
array([[ 65.96969001,  58.85575588,  54.91812087, ..., 1474.      ,
       1037.01205393, 389.99487176], ...,
       [ 61.07372594, 39.8246155 , 89.18520057, ..., 1763.79647352,
        864.92543031, 270.20362692], ...,
       [ 98.48857802, 112.44554237, 168.10710871, ..., 2110.61365484,
        870.36658943, 204.40156555], ...,
       [ 143.17821063, 597.00753764, 2479.42977315, ..., 216.00925906,
        248.33847869, 200.89798406], ...,
       [ 106.07544485, 393.67245268, 2188.78824924, ..., 124.96399481,
        159.90622252, 346.34087255], ...,
       [ 41.59326869, 229.05894438, 1845.1216762 , ..., 175.16278143,
        33.37663854, 414.3911196 ]])
Coordinates:
* x           (x) float64 10kB 0.5 1.5 2.5 ... 1.308e+03 1.31e+03 1.31e+03
* y           (y) float64 6kB 0.5 1.5 2.5 3.5 4.5 ... 767.5 768.5 769.5 770.5
  spatial_ref  int64 8B 0
Text(0.5, 1.0, 'Amplitude du filtre de Scharr')

```



## 5.4 Gestion des bordures

L'application de filtres à l'intérieur de fenêtres glissantes implique de gérer les bords de l'image, car la fenêtre de traitement va nécessairement déborder de quelques pixels en dehors de l'image (généralement la moitié de la fenêtre déborde). On peut soit décider d'ignorer les valeurs en dehors de l'image en imposant une valeur `nan`, soit prolonger l'image de quelques lignes et colonnes avec des valeurs miroirs ou constantes.

### 5.4.0.1 Filtrage par une couche convolutionnelle

#### Installation de Pytorch

Cette section nécessite la librairie Pytorch avec un GPU et ne fonctionnera que sur Colab. On peut quand même installer une version locale CPU de pytorch: `pip install -qU torch==2.4.0+cpu`

Une couche convolutionnelle est simplement un ensemble de filtres appliqués sur la donnée d'entrée. Ce type de filtrage est à la base des réseaux dits convolutionnels qui seront abordés dans le tome 2. On peut ici imposer les mêmes filtres de gradient dans la couche convolutionnelle :

```

1 import torch
2 import torch.nn as nn
3 import numpy as np
4 import matplotlib.pyplot as plt
5 normalized_img= torch.tensor(img_rgb.to_numpy())
6 nchannels= normalized_img.size()[0] # nombre de canaux de l'image
7
8 # On forme une couche convolutionnelle
9 conv_layer = nn.Conv2d(in_channels= nchannels, out_channels=2, kernel_size=3, padding=1, stride=1,
   → dilation= 1)
10
11 # Filtre de Sobel
12 sobel_x = np.array([[-3, 0, 3], [-10, 0, 10], [-3, 0, 3]])
13 sobel_y = np.array([[-3, -10, -3], [0, 0, 0], [3, 10, 3]])
14 # Le filtre (kernel) est formé de deux filtres
15 kernel = np.stack([sobel_x, sobel_y])
16 kernel = kernel.reshape(2, 1, 3, 3)
17 # On répète le filtre pour chaque bande
18 kernel = np.tile(kernel, (1,nchannels,1,1))
19 print(kernel.shape)
20 kernel = torch.as_tensor(kernel,dtype=torch.float32)
21 conv_layer.weight = nn.Parameter(kernel)
22 conv_layer.bias = nn.Parameter(torch.zeros(2,))
23
24 input= normalized_img.unsqueeze(0) # il faut ajouter une dimension pour le nombre d'échantillons
25 print(input.shape)
26 # Visualize the filters
27 fig, axs = plt.subplots(1, 2, figsize=(8, 5))
28 for i in range(2):

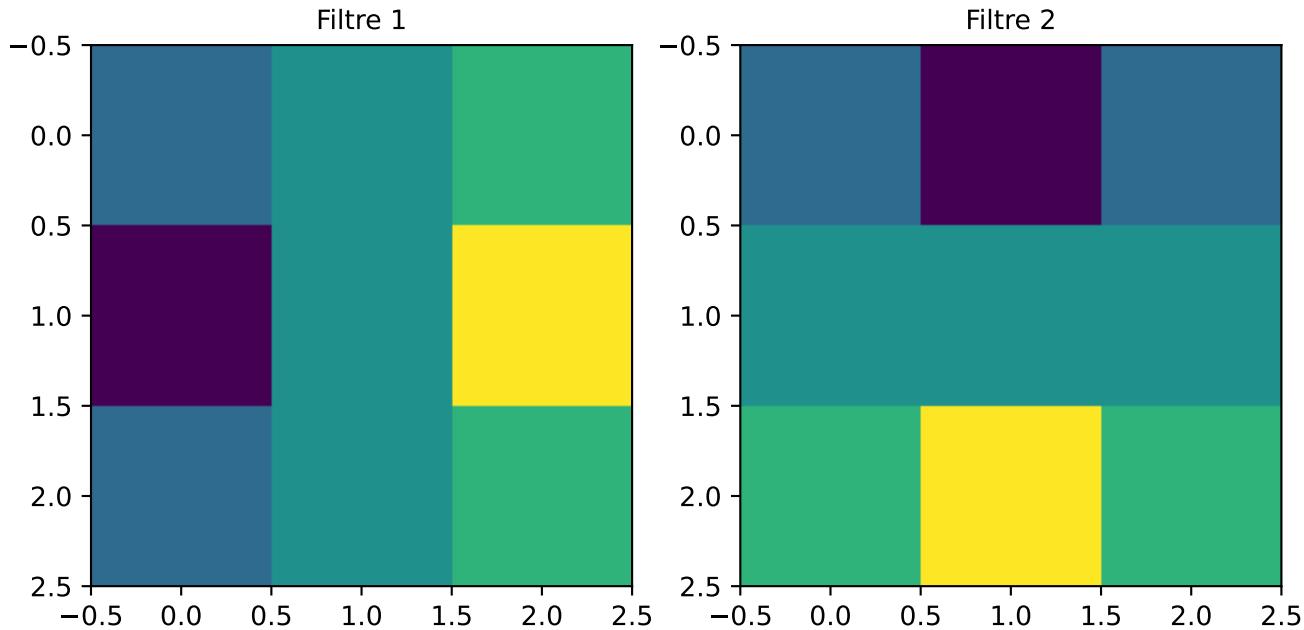
```

```

29     axs[i].imshow(conv_layer.weight.data.numpy()[i, 0])
30     axs[i].set_title(f'Filtre {i+1}')
31 plt.show()

```

(2, 3, 3, 3)  
torch.Size([1, 3, 771, 1311])



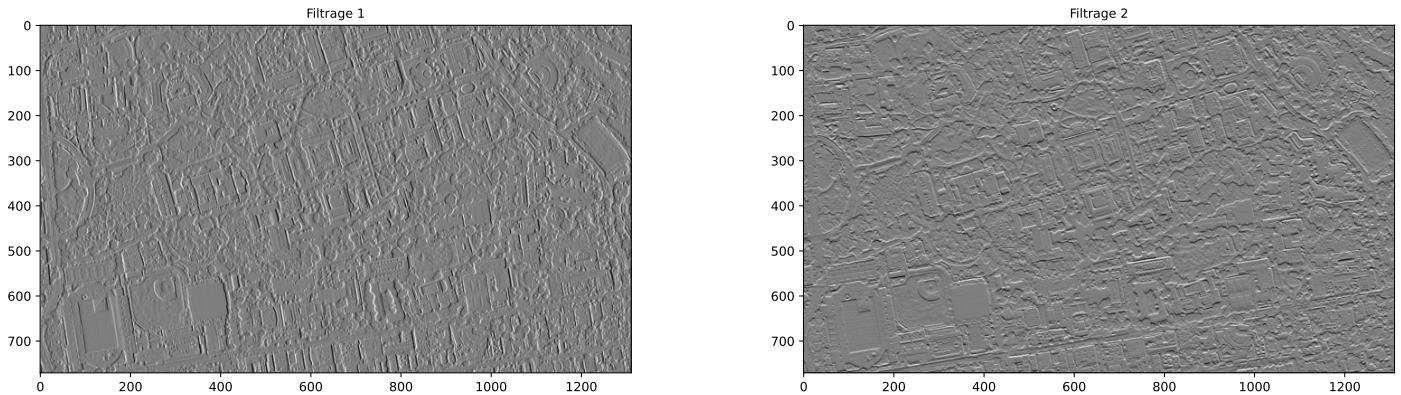
Le résultat est alors calculé sur GPU (si disponible):

```

1 import torch
2 import matplotlib.pyplot as plt
3
4 output = conv_layer(input)
5 print(f'Image (BxCxHxW): {input.shape}')
6 print(f'Sortie (BxFxHxW): {output.shape}')
7
8 fig, axs = plt.subplots(1, 2, figsize=(20, 5))
9 for i in range(2):
10     axs[i].imshow(output.detach().data.numpy()[0,i], vmin=-5000, vmax=5000, cmap='gray')
11     axs[i].set_title(f'Filtrage {i+1}')
12 plt.show()

```

Image (BxCxHxW): torch.Size([1, 3, 771, 1311])  
Sortie (BxFxHxW): torch.Size([1, 2, 771, 1311])



#### 5.4.1 Filtrage adaptatif

Les filtrages adaptatifs consistent à appliquer un traitement en fonction du contenu local d'une image. Le filtre n'est alors plus stationnaire et sa réponse peut varier en fonction du contenu local. Ce type de filtre est très utilisé pour filtrer les images SAR (Synthetic Aperture Radar) qui sont dégradées par un bruit multiplicatif que l'on appelle *speckle*. On peut voir un exemple d'une image Sentinel-1 (bande HH) sur la région de Montréal, remarquez que l'image est affichée en dB en appliquant la fonction `log10`.

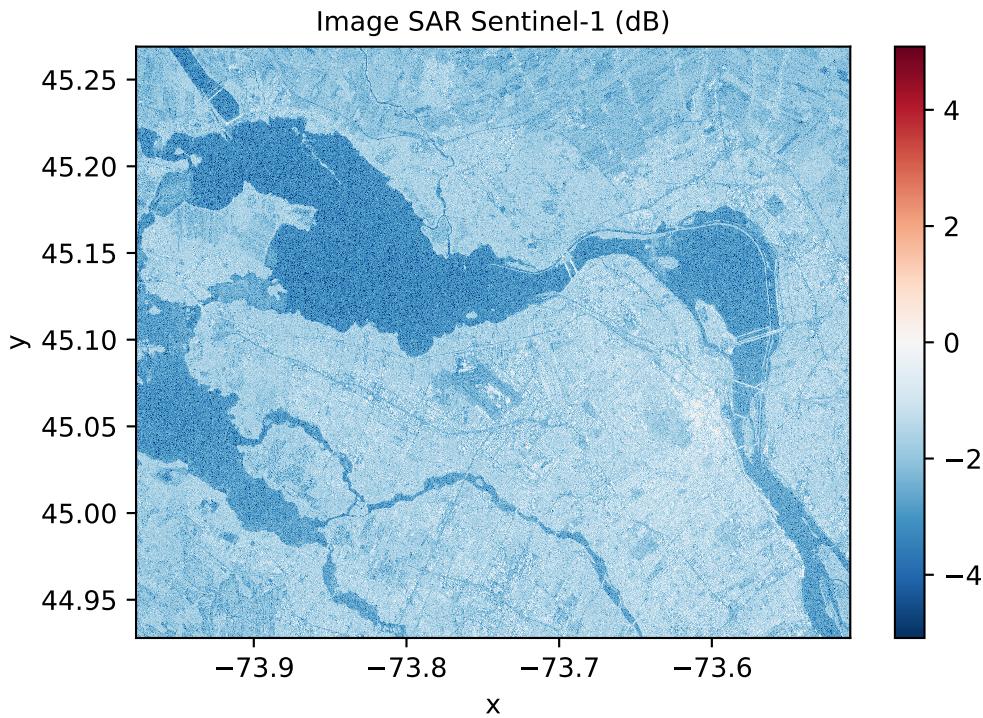
```

1 print(img_SAR.rio.resolution())
2 print(img_SAR.rio.crs)
3 fig, axs = plt.subplots(1, 1, figsize=(6, 4))
4 xr.ufuncs.log10(img_SAR.sel(band=1).drop("band")).plot()
5 axs.set_title("Image SAR Sentinel-1 (dB)")

(0.00029254428869762705, -0.000287092818453516
EPSG:4326

Text(0.5, 1.0, 'Image SAR Sentinel-1 (dB)')

```



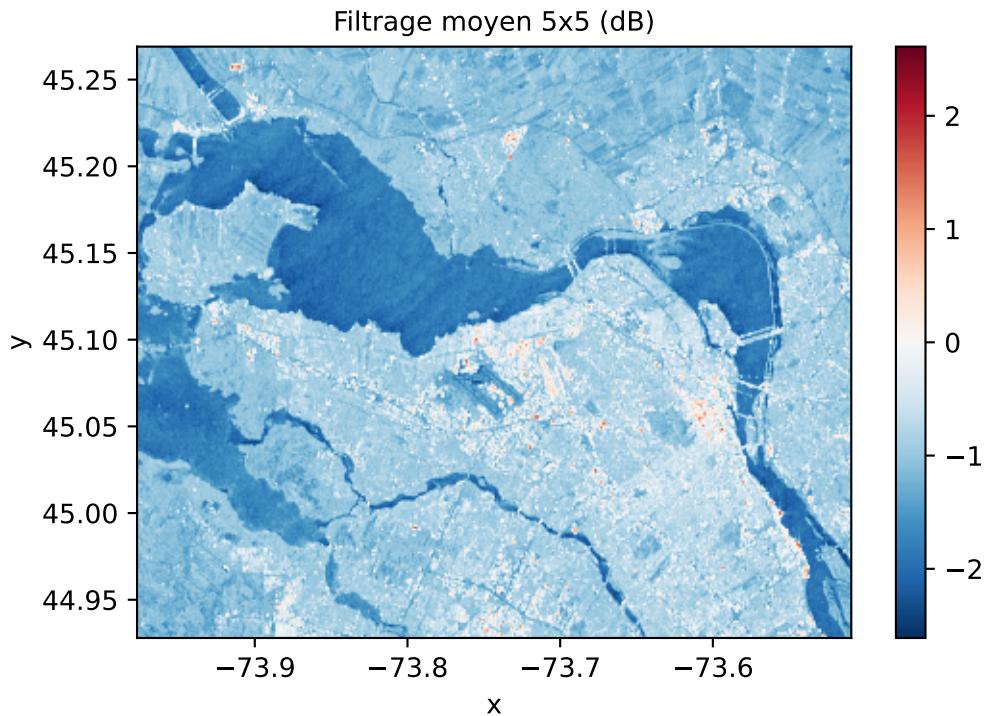
Un des filtres les plus simples pour réduire le bruit est d'appliquer un filtre moyen, par exemple un  $5 \times 5$  ci-dessous:

```

1 rolling_win = img_SAR.sel(band=2).rolling(x=5, y=5, min_periods= 3, center= True).construct(x="x_win",
2   → y="y_win", keep_attrs= True)
2 filtre_moyen= rolling_win.mean(dim= ['x_win', 'y_win'], skipna= True)
3 fig, axs = plt.subplots(1, 1, figsize=(6, 4))
4 xr.ufuncs.log10(filtre_moyen).plot.imshow()
5 axs.set_title("Filtrage moyen 5x5 (dB)")

Text(0.5, 1.0, 'Filtrage moyen 5x5 (dB)')

```



Au lieu d'appliquer un filtre moyen de manière indiscriminée, le filtre de Lee (Lee 1986) applique une pondération en fonction du contenu local de l'image  $I$  dans sa forme la plus simple :

$$I_F = I_M + K \times (I - I_M)$$

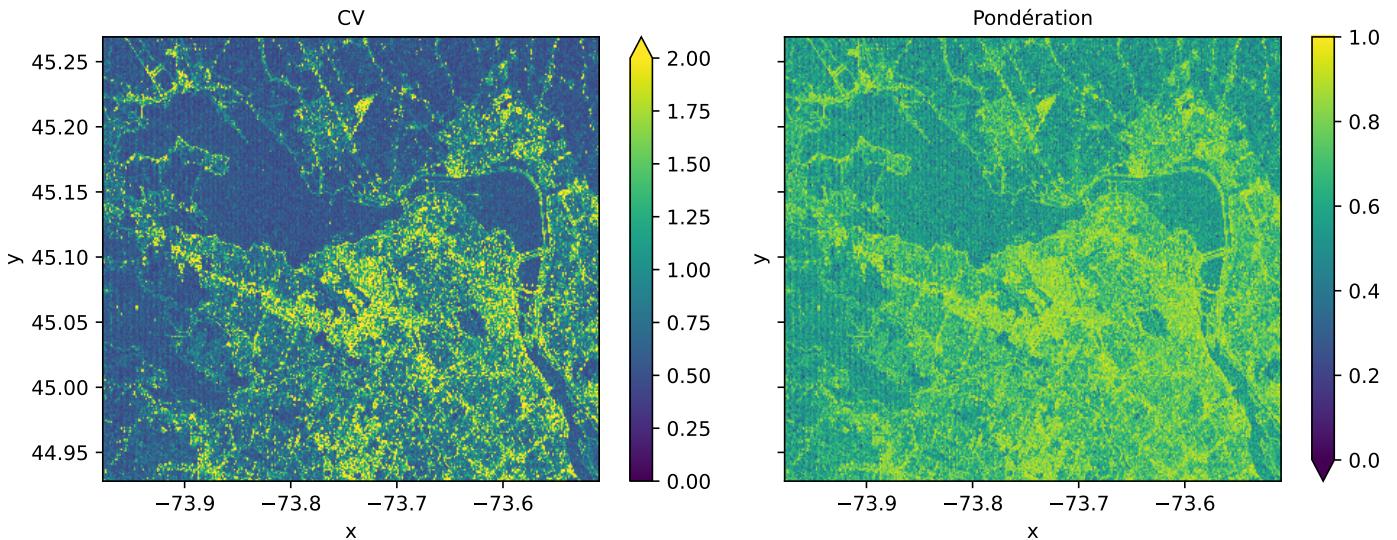
$$K = \frac{\sigma_I^2}{\sigma_I^2 + \sigma_{bruit}^2} \quad (5.7)$$

De la sorte, si la variance locale est élevée  $K$  s'approche de 1 préservant ainsi les détails de l'image  $I$  sinon l'image moyenne  $I_M$  est appliquée.

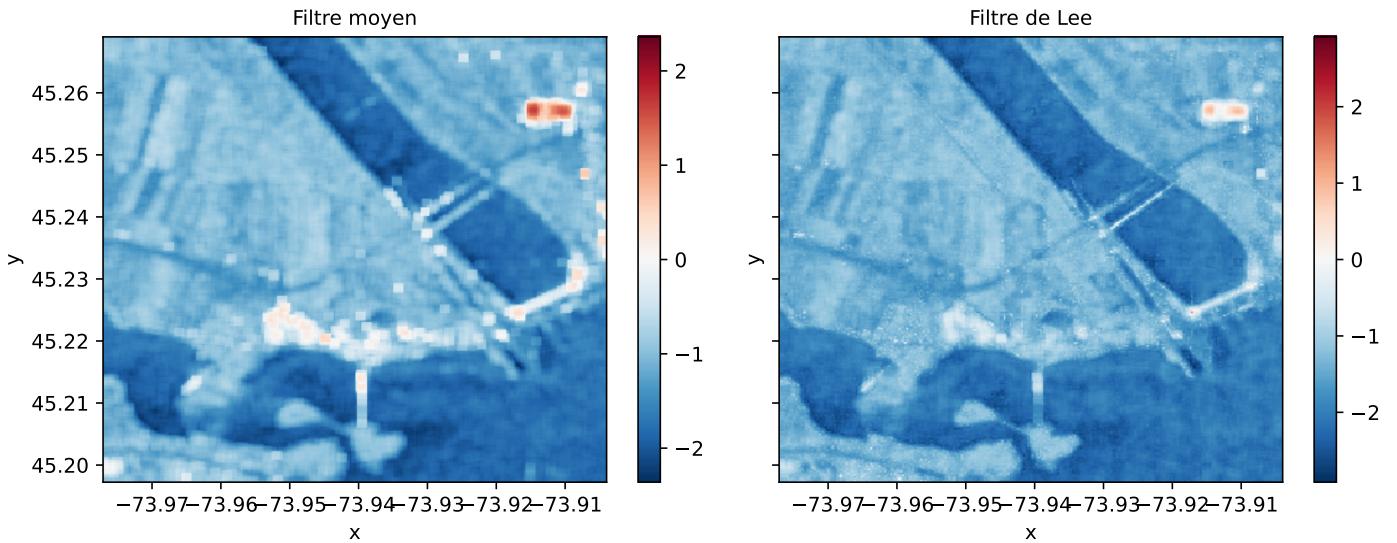
```

1 rolling_win = img_SAR.sel(band=2).rolling(x=5, y=5, min_periods= 3, center= True).construct(x="x_win",
2   ↵ y="y_win", keep_attrs= True)
3 filtre_moyen= rolling_win.mean(dim= ['x_win', 'y_win'], skipna= True)
4 ecart_type= rolling_win.std(dim= ['x_win', 'y_win'], skipna= True)
5 cv= ecart_type/filtre_moyen
6 ponderation = (cv - 0.25) / cv
7
8 fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(10, 4), sharex=True, sharey=True)
9 plt.subplot(1, 2, 1)
10 cv.plot.imshow( vmin=0, vmax=2)
11 axes[0].set_title("CV")
12 plt.subplot(1, 2, 2)
13 ponderation.plot.imshow( vmin=0, vmax=1)
14 axes[1].set_title("Pondération")
15 plt.tight_layout()

```



On zoomant sur l'image, on voit clairement que les détails de l'image sont mieux préservés :



## 5.5 Segmentation

La segmentation d'image consiste à séparer une image en régions homogènes spatialement connexes (segments) où les valeurs sont uniformes selon un certain critère (couleurs, texture, etc.). Une image présente généralement beaucoup de pixels redondants, l'intérêt de ce type de méthode est essentiellement de réduire la quantité de pixels nécessaire. En télédétection, on parle souvent d'approche objet. En vision par ordinateur, on parle parfois de super-pixel. Il existe de nombreuses méthodes de segmentation, la librairie `sickit-image` rend disponible plusieurs implémentations sur des images RVB ([Comparison of segmentation and superpixel algorithms — skimage 0.25.0 documentation](#)).

### 5.5.1 Super-pixel

Ce type de méthode cherche à former des régions homogènes et compactes dans l'image (Achanta et Süstrunk 2012). Une des méthodes les plus simples est la méthode SLIC (*Simple Linear Iterative Clustering*), elle combine un regroupement de type k-moyennes avec une distance hybride qui prend en compte les différences de couleur entre pixels mais aussi leur distance par rapport centre du super-pixel:

1. Décomposer l'image en  $N$  régions régulières de taille  $S \times S$
2. Initialiser les centres  $C_k$  de chaque segment  $k$
3. Rechercher les pixels ayant la distance la plus petite dans une région  $2S \times 2S$ :

$$D_{SLIC} = d_{couleur} + \frac{m}{S} d_{xy}$$

4. Mettre à jour les centre  $C_k$  de chaque segment  $k$  et réitérer à l'étape 3.

Les régions évoluent rapidement avec les itérations, plus le poids  $m$  est élevé, plus la forme du super-pixel est contrainte et ne suivra pas vraiment le contenu de l'image:

```

1 img = img_rgb.to_numpy().astype('uint8').transpose(1,2,0)
2
3 segments_slic1 = slic(img, n_segments=250, compactness=10, sigma=1, start_label=1, max_num_iter=1)
4 segments_slic2 = slic(img, n_segments=250, compactness=10, sigma=1, start_label=1, max_num_iter=2)
5 segments_slic100 = slic(img, n_segments=250, compactness=100, sigma=1, start_label=1, max_num_iter=10)
6 segments_slic100b = slic(img, n_segments=250, compactness=10, sigma=1, start_label=1, max_num_iter=10)
7
8 print(f'SLIC nombre de segments: {len(np.unique(segments_slic1))}')
9
10 fig, ax = plt.subplots(2, 2, figsize=(10, 6), sharex=True, sharey=True)
11
12 ax[0, 0].imshow(mark_boundaries(img, segments_slic1))
13 ax[0, 0].set_title("Initialisation")
14 ax[0, 1].imshow(mark_boundaries(img, segments_slic2))
15 ax[0, 1].set_title('2 itérations')
16 ax[1, 0].imshow(mark_boundaries(img, segments_slic100))
17 ax[1, 0].set_title('10 itérations avec m=100')
18 ax[1, 1].imshow(mark_boundaries(img, segments_slic100b))
19 ax[1, 1].set_title('10 itérations avec m=10')
20
21 for a in ax.ravel():
22     a.set_axis_off()
23
24 plt.tight_layout()
25 plt.show()
```

SLIC nombre de segments: 240



Le nombre de segments initial est probablement le paramètre le plus important. Une manière de l'estimer est d'évaluer l'échelle moyenne des segments homogènes dans l'image à analyser. On observe ci-dessous l'impact de passer d'une échelle  $40 \times 40$  à  $20 \times 20$ . En prenant la moyenne de chaque segment, on constate que l'échelle  $40 \times 40$  génère des segments trop grands mélangeant plusieurs classes.

```

1 from skimage import color, segmentation
2 n_regions = int((img.shape[0] * img.shape[1])/(40*40))
3 print('Nb segments: ',n_regions)
4 segments_slic_40 = slic(img, n_segments=n_regions, compactness=10, sigma=1, start_label=1,
5   ↪ max_num_iter=10)
6 print(f'SLIC nombre de segments: {len(np.unique(segments_slic_40))}')
7 out = color.label2rgb(segments_slic_40, img, kind='avg', bg_label=0)
8 out_40 = segmentation.mark_boundaries(out, segments_slic_40, (0, 0, 0))

9 n_regions = int((img.shape[0] * img.shape[1])/(20*20))
10 print('Nb segments: ',n_regions)
11 segments_slic_20 = slic(img, n_segments=n_regions, compactness=10, sigma=1, start_label=1,
12   ↪ max_num_iter=10)
13 print(f'SLIC nombre de segments: {len(np.unique(segments_slic_20))}')
14 out = color.label2rgb(segments_slic_20, img, kind='avg', bg_label=0)
15 out_20 = segmentation.mark_boundaries(out, segments_slic_20, (0, 0, 0))

```

## 5 Transformations spatiales

```
16 fig, ax = plt.subplots(2, 1, figsize=(6, 8), sharex=True, sharey=True)
17
18 ax[0].imshow(out_40)
19 ax[0].set_title("Initialisation avec 631 segments")
20 ax[1].imshow(out_20)
21 ax[1].set_title('Initialisation avec 2526 segments')
22 for a in ax.ravel():
23     a.set_axis_off()
24 plt.tight_layout()
25 plt.show()
```

```
Nb segments: 631
SLIC nombre de segments: 459
Nb segments: 2526
SLIC nombre de segments: 2201
```

Initialisation avec 631 segments



Initialisation avec 2526 segments



### 5.5.2 Fusion des segments par graphe de proximité

Une segmentation peut produire beaucoup trop de segments. On parle alors de sur-segmentation. Ceci est recherché dans certains cas pour permettre de bien capturer les détails fins de l'image. Cependant, afin de réduire

le nombre de segments, un post-traitement possible est de fusionner les segments similaires selon certaines règles ou distances. Un graphe d'adjacence de régions (figure 5.1) est formé à partir des segments connectés où chaque noeud représente un segment et un lien de proximité (Jaworek-Korjakowska (2018)). À partir de ce graphe, on peut fusionner les noeuds similaires à partir de leur distance radiométrique.

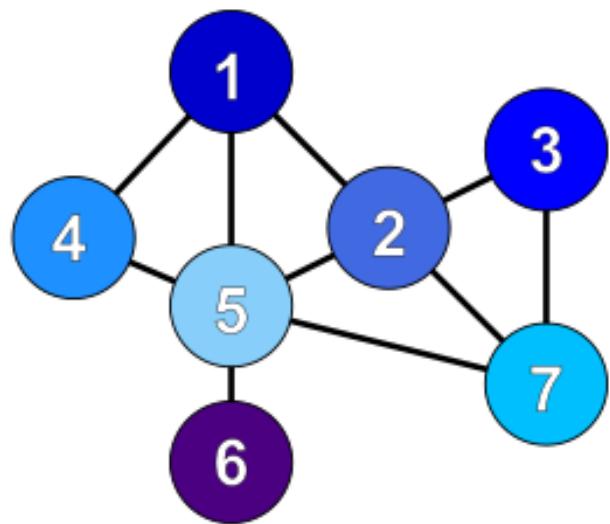
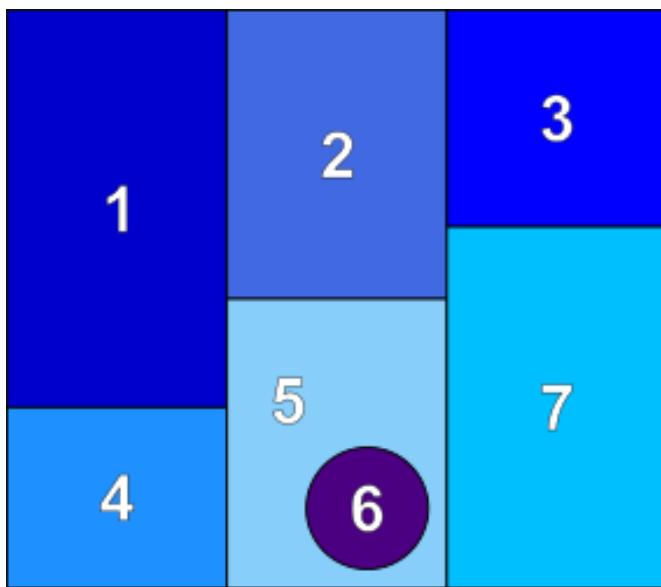


FIGURE 5.1 – Graphe d'adjacence de régions, d'après (Jaworek-Korjakowska (2018)). Chaque noeud est un segment, un lien est formé uniquement si les segments se touchent (par exemple le segment 6 ne touche que la région 5). La fonction `graph.rag_mean_color` produit un graphe à partir d'une segmentation et de l'image originale. Chaque noeud tient la couleur de chaque segment dans un attribut appelé '`mean color`'.

```

1 def _weight_mean_color(graph, src, dst, n):
2     """Fonction pour gérer la fusion des nœuds en recalculant la couleur moyenne.
3     La méthode suppose que la couleur moyenne de `dst` est déjà calculée.
4     """
5     diff = graph.nodes[dst]['mean color'] - graph.nodes[n]['mean color']
6     diff = np.linalg.norm(diff)
7     #print(diff)
8     return {'weight': diff}
9
10
11 def merge_mean_color(graph, src, dst):
12     """Fonction appelée avant la fusion de deux nœuds d'un graphe de distance de couleur moyenne.
13     Cette méthode calcule la couleur moyenne de `dst`.
14     """
15     graph.nodes[dst]['total color'] += graph.nodes[src]['total color']
16     graph.nodes[dst]['pixel count'] += graph.nodes[src]['pixel count']
17     graph.nodes[dst]['mean color'] = (
18         graph.nodes[dst]['total color'] / graph.nodes[dst]['pixel count']
19     )

```

```

20 g = graph.rag_mean_color(img, segments_slic_20)
21 print('Nombre de segments:', len(g))
22 labels2 = graph.merge_hierarchical(
23     segments_slic_20,
24     g,
25     thresh=20,
26     rag_copy=False,
27     in_place_merge=True,
28     merge_func=merge_mean_color,
29     weight_func=_weight_mean_color,
30 )
31 print('Nombre de segments:', len(g))
32
33 out1 = color.label2rgb(segments_slic_20, img, kind='avg', bg_label=0)
34 out1 = segmentation.mark_boundaries(out1, segments_slic_20, (0, 0, 0))
35 out2 = color.label2rgb(labels2, img, kind='avg', bg_label=0)
36 out2 = segmentation.mark_boundaries(out2, labels2, (0, 0, 0))
37
38 fig, ax = plt.subplots(nrows=2, sharex=True, sharey=True, figsize=(6, 8))
39
40 ax[0].imshow(out1)
41 ax[0].set_title("Avant fusion")
42 ax[1].imshow(out2)
43 ax[1].set_title("Après fusion")
44 for a in ax:
45     a.axis('off')
46
47 plt.tight_layout()

```

Nombre de segments: 2201  
 Nombre de segments: 1187

Avant fusion



Après fusion



### 5.5.3 Approche objet

L'approche objet consiste à traiter chaque segment comme un objet avec un ensemble de propriétés. La librairie `skimage` offre la possibilité d'enrichir chaque segment avec des propriétés et de former un tableau:

```

1 properties = ['label', 'area', 'centroid', 'num_pixels', 'intensity_mean', 'intensity_std']
2
3 table=  measure.regionprops_table(labels2, intensity_image= img_rgb.to_numpy().transpose(1,2,0),
4   → properties=properties)
5
6 table = pd.DataFrame(table)
7 table.head(10)

```

	label	area	centroid-0	centroid-1	num_pixels	intensity_mean-0	intensity_mean-1	intensity_mean-2
0	1	641.0	15.466459	69.489860	641	136.730103	132.851791	117.126366
1	2	480.0	10.997917	92.614583	480	201.208328	198.262497	188.483337
2	3	712.0	16.683989	114.776685	712	185.349716	183.113770	170.994385
3	4	1803.0	31.974487	139.379368	1803	117.897392	108.367722	97.769829
4	5	448.0	5.004464	166.542411	448	183.511154	181.276779	167.720978
5	6	459.0	9.934641	191.668845	459	133.557739	133.821350	129.697174
6	7	355.0	5.160563	222.895775	355	148.574646	148.802811	142.580276
7	8	334.0	4.904192	255.904192	334	125.973053	121.197601	108.973053
8	9	1481.0	32.279541	292.865631	1481	204.102631	172.359894	137.501007
9	10	445.0	8.013483	308.053933	445	145.373032	138.182022	121.402245

Ce tableau pourra être exploiter pour une tâche de classification par la suite (on parle alors de classification objet).

## **Partie 3. Classifications d'images**

# 6 Classifications d'images supervisées

## 6.1 Préambule

Assurez-vous de lire ce préambule avant d'exécuter le reste du notebook.

### 6.1.1 Objectifs

Dans ce chapitre, nous ferons une introduction générale à l'apprentissage automatique et abordons quelques techniques fondamentales. La librairie centrale utilisée dans ce chapitre sera `scikit-learn`. Ce chapitre est aussi disponible sous la forme d'un notebook Python sur Google Colab:



#### Objectif

##### Objectifs d'apprentissage visés dans ce chapitre

À la fin de ce chapitre, vous devriez être en mesure de :

- comprendre les principes de l'apprentissage automatique supervisé;
- mettre en place un pipeline d'entraînement;
- savoir comment évaluer les résultats d'un classificateur;
- visualiser les frontières de décision;
- mettre en place des techniques de classifications comme K-NN et les arbres de décision;

### 6.1.2 Librairies

Les librairies utilisées dans ce chapitre sont les suivantes :

- `SciPy`
- `NumPy`
- `opencv-python` · `PyPI`
- `scikit-image`
- `Rasterio`
- `xarray`
- `rioxarray`
- `geopandas`
- `scikit-learn`

Dans l'environnement Google Colab, seul `rioxarray` et `xrscipy` sont installés:

```
1 %%capture
2 !pip install -qU matplotlib rioxarray xrscipy
```

Vérifiez les importations nécessaires en premier:

```
1 import numpy as np
2 import rioxarray as rxr
3 from scipy import signal
4 import xarray as xr
5 import rasterio
6 import xrscipy
7 import matplotlib.pyplot as plt
8 from matplotlib.colors import ListedColormap
9 import geopandas
10 from shapely.geometry import Point
11 import pandas as pd
12 from numba import jit
13 from sklearn.neighbors import KNeighborsClassifier
14 from sklearn.model_selection import train_test_split
15 from sklearn.pipeline import Pipeline
16 from sklearn.metrics import confusion_matrix, classification_report, ConfusionMatrixDisplay
17 from sklearn.preprocessing import StandardScaler
18 from sklearn.inspection import DecisionBoundaryDisplay
19 from sklearn.discriminant_analysis import LinearDiscriminantAnalysis, QuadraticDiscriminantAnalysis
20 from sklearn.datasets import make_blobs, make_classification, make_gaussian_quantiles
```

### 6.1.3 Images utilisées

Nous utilisons les images suivantes dans ce chapitre:

```
1 %%capture
2 import gdown
3
4 gdown.download([
    'https://drive.google.com/uc?export=download&confirm=pbef&id=1a6Ypg0g10y4AJt9XWKWfnR12NW1XhNg_',
    'output= "RGBNIR_of_S2A.tif"')
5 gdown.download([
    'https://drive.google.com/uc?export=download&confirm=pbef&id=1a4PQ68Ru8zBphbQ22j0sgJ4D2quw-Wo6',
    'output= "landsat7.tif"')
6 gdown.download([
    'https://drive.google.com/uc?export=download&confirm=pbef&id=1_zwCLN-x7XJcNHJCH6Z8upEdUXtVtvS1',
    'output= "berkeley.jpg"')
7 gdown.download([
    'https://drive.google.com/uc?export=download&confirm=pbef&id=1dM6IVqjba6GHwTLmI7CpX8GP2z5txUq6',
    'output= "SAR.tif"'])
```

```

8 gdown.download(
    ↪ 'https://drive.google.com/uc?export=download&confirm=pbef&id=1aAq7crc_LoaLC3kG3HkQ6Fv5JfG0mswg',
    ↪ output= 'carte.tif')

```

Vérifiez que vous êtes capable de les lire :

```

1 with rxr.open_rasterio('berkeley.jpg', mask_and_scale= True) as img_rgb:
2     print(img_rgb)
3 with rxr.open_rasterio('RGBNIR_of_S2A.tif', mask_and_scale= True) as img_rgnir:
4     print(img_rgnir)
5 with rxr.open_rasterio('SAR.tif', mask_and_scale= True) as img_SAR:
6     print(img_SAR)
7 with rxr.open_rasterio('carte.tif', mask_and_scale= True) as img_carte:
8     print(img_carte)

```

## 6.2 Principes généraux

Une classification supervisée ou dirigée consiste à attribuer une étiquette (une classe) de manière automatique à chaque point d'un jeu de données. Cette classification peut se faire à l'aide d'une cascade de règles pré-établies (arbre de décision) ou à l'aide de techniques d'apprentissage automatique (*machine learning*). L'utilisation de règles pré-établies atteint vite une limite car ces règles doivent être fournies manuellement par un expert. Ainsi, l'avantage de l'apprentissage automatique est que les règles de décision sont dérivées automatiquement du jeu de données via une phase dite d'entraînement. On parle souvent de solutions générées par les données (*Data Driven Solutions*). Cet ensemble de règles est souvent appelé **modèle**. On visualise souvent ces règles sous la forme de *frontières de décisions* dans l'espace des données. Cependant, un des défis majeurs de ce type de technique est d'être capable de produire des règles qui soient généralisables au-delà du jeu d'entraînement.

Les classifications supervisées ou dirigées presupposent donc que nous avons à disposition **un jeu d'entraînement** déjà étiqueté. Celui-ci va nous permettre de construire un modèle. Afin que ce modèle soit représentatif et robuste, il nous faut assez de données d'entraînement. Les algorithmes d'apprentissage automatique sont très nombreux et plus ou moins complexes pouvant produire des frontières de décision très complexes et non linéaires.

### 6.2.1 Comportement d'un modèle

Cet exemple tiré de **scikit-learn** illustre les problèmes d'ajustement insuffisant ou **sous-apprentissage** (*underfitting*) et d'ajustement excessif ou **sur-apprentissage** (*overfitting*) et montre comment nous pouvons utiliser la régression linéaire avec un modèle polynomiale pour approximer des fonctions non linéaires. La figure 6.1 montre la fonction que nous voulons approximer, qui est une partie de la fonction cosinus (couleur orange). En outre, les échantillons de la fonction réelle et les approximations de différents modèles sont affichés en bleu. Les modèles ont des caractéristiques polynomiales de différents degrés. Nous pouvons constater qu'une fonction linéaire (polynôme de degré 1) n'est pas suffisante pour s'adapter aux échantillons d'apprentissage. C'est ce qu'on appelle un sous-ajustement (*underfitting*) qui produit un biais systématique quels que soient les points d'entraînement. Un polynôme de degré 4 se rapproche presque parfaitement de la fonction réelle. Cependant, pour des degrés plus élevés, le modèle s'adaptera trop aux données d'apprentissage, c'est-à-dire

qu'il apprendra le bruit des données d'apprentissage. Nous évaluons quantitativement le sur-apprentissage et le sous-apprentissage à l'aide de la validation croisée. Nous calculons l'erreur quadratique moyenne (EQM) sur l'ensemble de validation. Plus elle est élevée, moins le modèle est susceptible de se généraliser correctement à partir des données d'apprentissage.

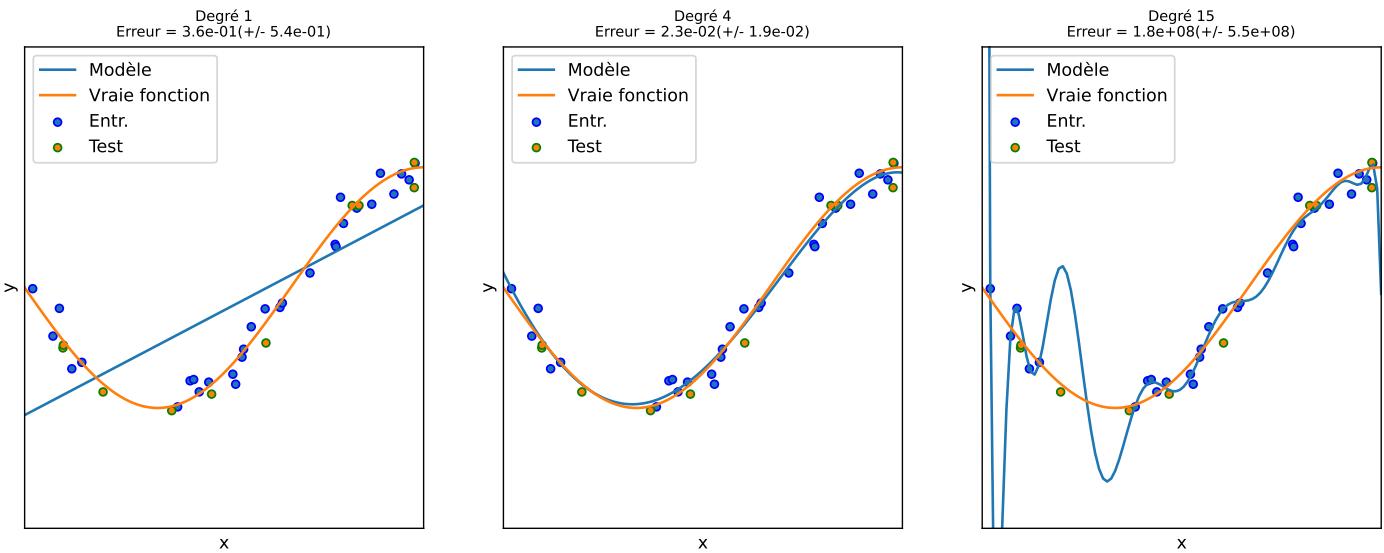


FIGURE 6.1 – Exemples de sur et sous-apprentissage.

On constate aussi que sans les échantillons de validation, nous serions incapables de déterminer la situation de sur-apprentissage, l'erreur sur les points d'entraînement seuls étant excellente pour un degré 15.

### 6.2.2 Pipeline

La construction d'un modèle implique généralement toujours les mêmes étapes illustrées sur la figure 6.2:

1. La préparation des données implique parfois un pré-traitement afin de normaliser les données.
2. Partage des données en trois groupes: entraînement, validation et test.
3. L'apprentissage du modèle sur l'ensemble d'entraînement. Cet apprentissage nécessite de déterminer les valeurs des hyper-paramètres du modèle par l'usager.
4. La validation du modèle sur l'ensemble de validation. Cette étape vise à vérifier que les hyper-paramètres du modèle sont adéquats.
5. Enfin le test du modèle sur un ensemble de données indépendant.

### 6.2.3 Construction d'un ensemble d'entraînement

Les données d'entraînement permettent de construire un modèle. Elles peuvent prendre des formes très variées mais on peut voir cela sous la forme d'un tableau  $N \times D$ :

1. La taille  $N$  du jeu de données.

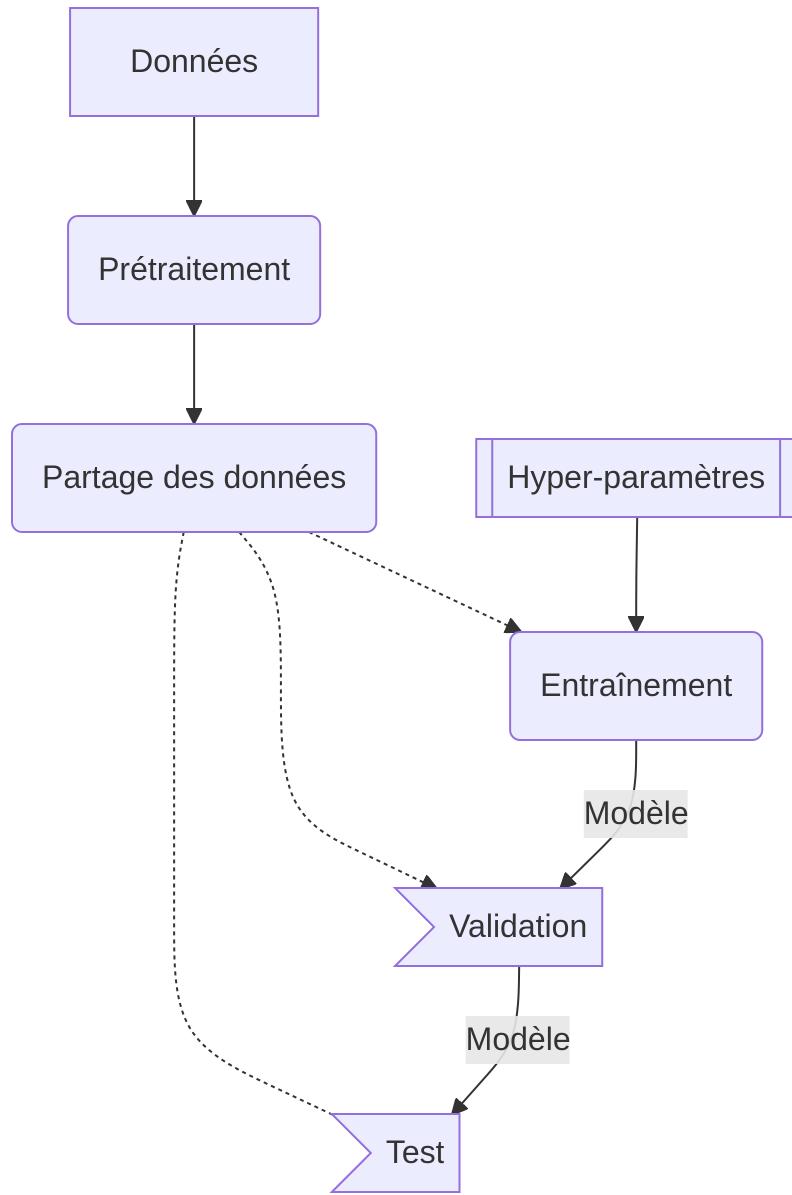


FIGURE 6.2 – Étapes standards dans un entraînement.

2. Chaque entrée définit un échantillon ou un point dans un espace à plusieurs dimensions.
3. Chaque échantillon est décrit par  $D$  dimensions ou caractéristiques (*features*).

Une façon simple de construire un ensemble d'entraînement est d'échantillonner un produit existant. Nous utilisons une carte d'occupation des sols qui contient 12 classes différentes.

```

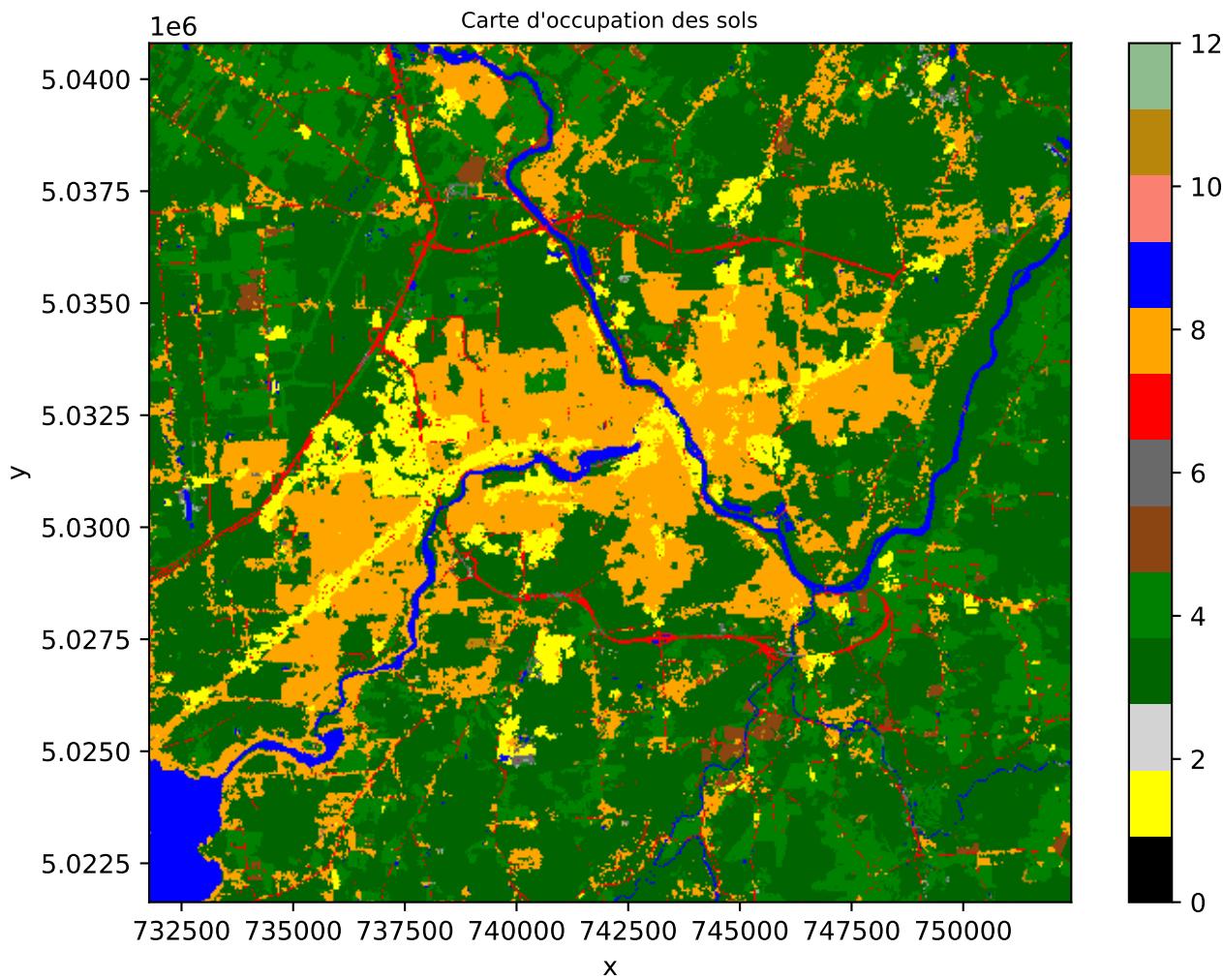
1 couleurs_classes= {'NoData': 'black', 'Commercial': 'yellow', 'Nuages': 'lightgrey',
2           'Foret': 'darkgreen', 'Faible_végétation': 'green', 'Sol_nu': 'saddlebrown',
3           'Roche': 'dimgray', 'Route': 'red', 'Urbain': 'orange', 'Eau': 'blue', 'Tourbe':
4           ↪ 'salmon', 'Végétation épars': 'darkgoldenrod', 'Roche avec végétation':
5           ↪ 'darkseagreen'}
```

On peut visualiser la carte de la façon suivante :

```

1 import matplotlib.pyplot as plt
2 import rioxarray as rxr
3 cmap_classes = ListedColormap(couleurs_classes)
4
5 fig, ax = plt.subplots(nrows=1, ncols=1, figsize=(8, 6))
6 img_carte.squeeze().plot.imshow(cmap=cmap_classes, vmin=0, vmax=12)
7 ax.set_title("Carte d'occupation des sols", fontsize="small")
```

Text(0.5, 1.0, "Carte d'occupation des sols")



On peut facilement calculer la fréquence d'occurrences des 12 classes dans l'image à l'aide de `numpy`:

```
1 img_carte= img_carte.squeeze() # nécessaire pour ignorer la dimension du canal
2 compte_classe = np.unique(img_carte.data, return_counts=True)
3 print(compte_classe)
```

```
(array([ 1.,  3.,  4.,  5.,  6.,  7.,  8.,  9., 11., 12., nan],
      dtype=float32), array([ 193558, 2104777, 670158, 29523, 14624, 94751, 750046,
      123671, 9079, 4327, 10]))
```

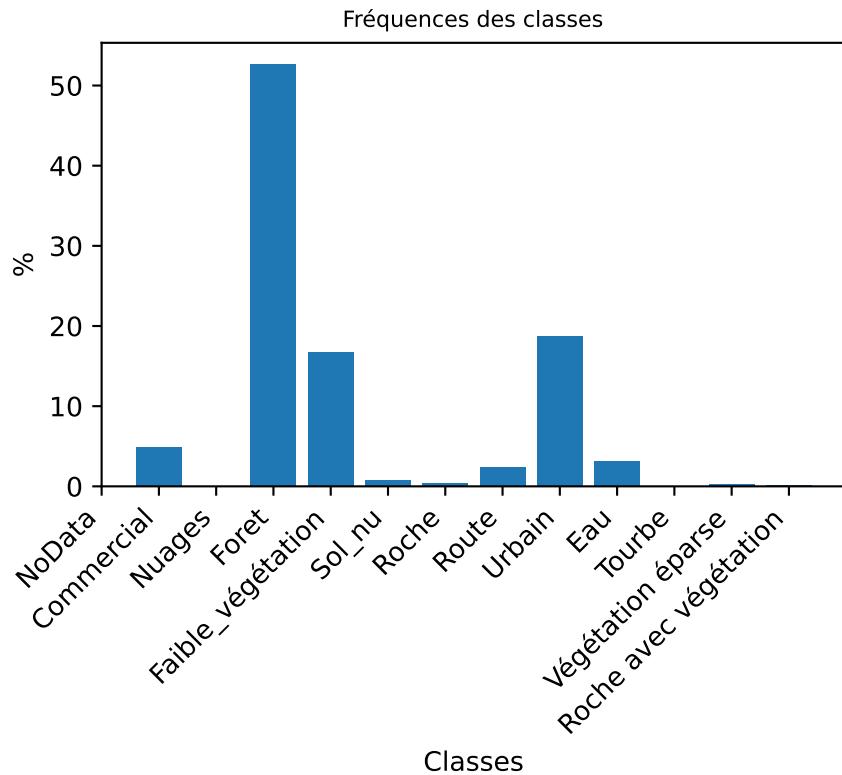
La fréquence d'apparition de chaque classe varie grandement, on parle alors d'un **ensemble déséquilibré**. Ceci est très commun dans la plupart des ensembles d'entraînement, puisque les classes ont très rarement la même fréquence. Par exemple, à la lecture du graphique en barres verticales, on constate que la classe forêt est très présente contrairement à plusieurs autres classes (notamment, tourbe, végétation éparses, roche, sol nu et nuages).

```
1 valeurs, comptes = compte_classe
2
```

```

3 # Create the histogram
4 plt.figure(figsize=(5, 3))
5 plt.bar(valeurs, comptes/comptes.sum()*100)
6 plt.xlabel("Classes")
7 plt.ylabel("%")
8 plt.title("Fréquences des classes", fontsize="small")
9 plt.xticks(range(len(nom_classes)), nom_classes, rotation=45, ha='right')
10 plt.show()

```



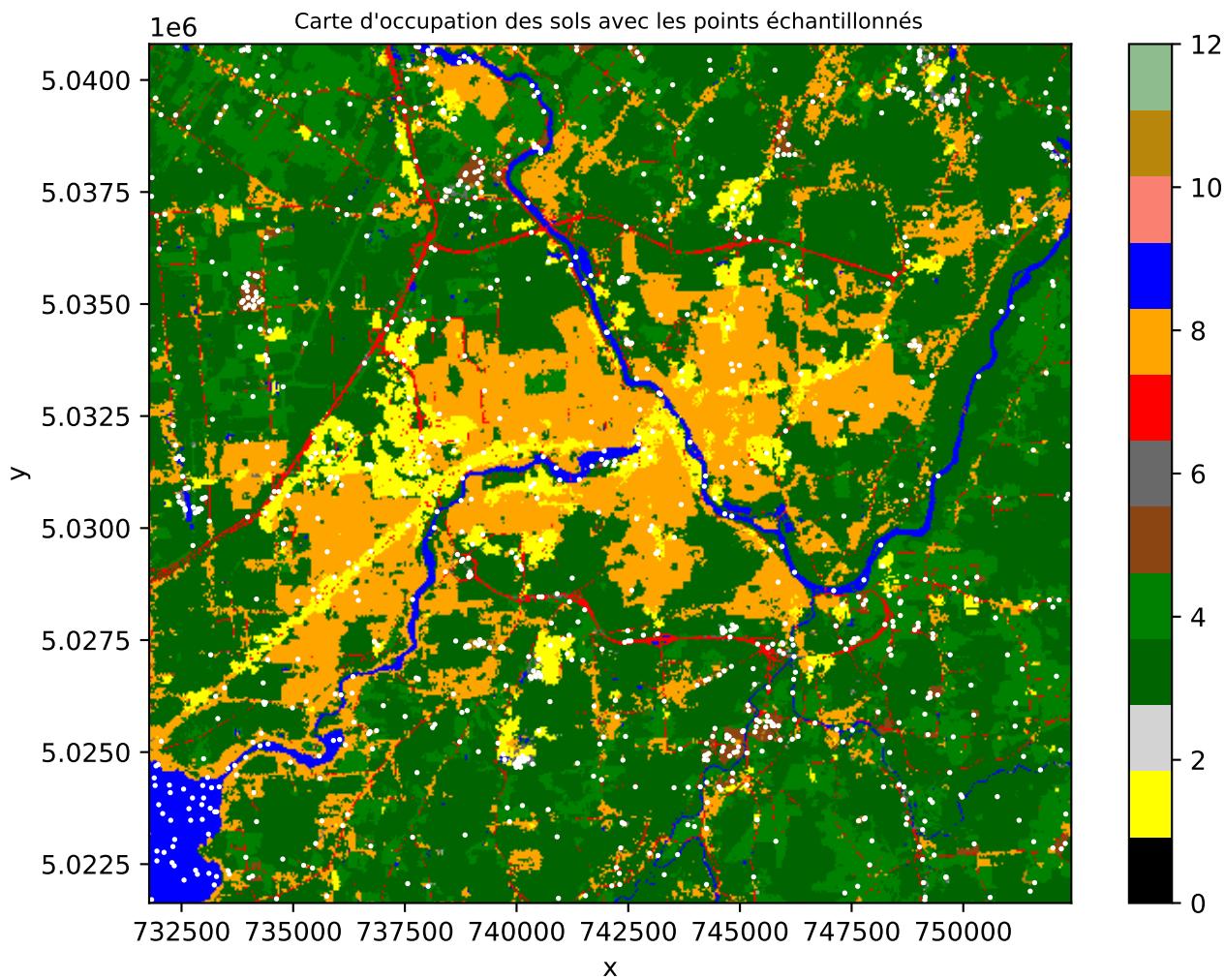
On peut échantillonner aléatoirement 100 points pour chaque classe:

```

1 img_carte= img_carte.squeeze()
2 class_counts = np.unique(img_carte.data, return_counts=True)
3
4 # Liste vide des points échantillonnés
5 sampled_points = []
6 class_labels= [] # contient les étiquettes des classes
7 for class_label in range(1,13): # pour chacune des 12 classes
8     # On cherche tous les pixels pour cette étiquette
9     class_pixels = np.argwhere(img_carte.data == class_label)
10
11    # On se limite à 100 pixels par classe
12    n_samples = min(100, len(class_pixels))

```

```
13
14 # On les choisit les positions aléatoirement
15 np.random.seed(0) # ceci permet de répliquer le tirage aléatoire
16 sampled_indices = np.random.choice(len(class_pixels), n_samples, replace=False)
17
18 # On prends les positions en lignes, colonnes
19 sampled_pixels = class_pixels[sampled_indices]
20
21 # On ajoute les points à la liste
22 sampled_points.extend(sampled_pixels)
23 class_labels.extend(np.array([class_label]*n_samples)[:,np.newaxis])
24
25 # Conversion en NumPy array
26 sampled_points = np.array(sampled_points)
27 class_labels = np.array(class_labels)
28 # On peut naviguer les points à l'aide de la géoréférence
29 transformer = rasterio.transform.AffineTransformer(img_carte.rio.transform())
30 transform_sampled_points= transformer.xy(sampled_points[:,0], sampled_points[:,1])
31
32 fig, ax = plt.subplots(nrows=1, ncols=1, figsize=(8, 6))
33 img_carte.squeeze().plot.imshow(cmap=cmap_classes, vmin=0, vmax=12)
34 ax.scatter(transform_sampled_points[0], transform_sampled_points[1], c='w', s=1) # Plot sampled points
35 ax.set_title("Carte d'occupation des sols avec les points échantillonés", fontsize="small")
36 plt.show()
```



Une fois les points sélectionnés, on ajoute les valeurs des bandes provenant d'une image satellite. Pour cela, on utilise la méthode `sample()` de `rasterio`. Éventuellement, la librairie `geopandas` permet de gérer les données d'entraînement sous la forme d'un tableau transportant aussi l'information de géoréférence. Afin de pouvoir classifier ces points, on ajoute les valeurs radiométriques provenant de l'image Sentinel-2 à 4 bandes `RGBNIR_of_S2A.tif`. Ces valeurs seront stockées dans la colonne `value` sous la forme d'un vecteur en format `string` :

```

1 points = [Point(xy) for xy in zip(transform_sampled_points[0], transform_sampled_points[1])]
2 gdf = geopandas.GeoDataFrame(range(1,len(points)+1), geometry=points, crs=img_carte.rio.crs)
3 coord_list = [(x, y) for x, y in zip(gdf["geometry"].x, gdf["geometry"].y)]
4 with rasterio.open('RGBNIR_of_S2A.tif') as src:
5     gdf["value"] = [x for x in src.sample(coord_list)]
6 gdf['class']= class_labels
7 gdf.to_csv('sampling_points.csv') # sauvegarde sous forme d'un format csv
8 gdf.head()

```

	0	geometry	value	class
0	1	POINT (740369.77 5032078.683)	[1894, 1994, 2112, 2318]	1
1	2	POINT (737542.924 5031770.119)	[1440, 1650, 1449, 5021]	1
2	3	POINT (736726.722 5031411.786)	[1666, 1972, 1819, 3437]	1
3	4	POINT (736816.305 5027470.128)	[1858, 2078, 2190, 2436]	1
4	5	POINT (736746.629 5031362.018)	[2194, 2304, 2268, 3075]	1

## 6.3 Analyse préliminaire des données

Une bonne pratique avant d'appliquer une technique d'apprentissage automatique est de regarder les caractéristiques de vos données:

1. Le nombre de dimensions (*features*).
2. Certaines dimensions sont informatives (discriminantes) et d'autres ne le sont pas.
3. Le nombre classes.
4. Le nombre de modes (*clusters*) par classes.
5. Le nombre d'échantillons par classe.
6. La forme des groupes.
7. La séparabilité des classes ou des groupes.

Une manière d'évaluer la séparabilité de vos classes est d'appliquer des modèles Gaussiens sur chacune des classes. Le modèle Gaussien multivarié suppose que les données sont distribuées comme un nuage de points symétrique et unimodale. La distribution d'un point  $x$  appartenant à la classe  $i$  est la suivante:

$$P(x|Classe = i) = \frac{1}{(2\pi)^{D/2}|\Sigma_i|^{1/2}} \exp\left(-\frac{1}{2}(x - m_i)^t \Sigma_k^{-1} (x - m_i)\right)$$

La méthode **QuadraticDiscriminantAnalysis** permet de calculer les paramètres des Gaussiennes multivariées pour chacune des classes.

On peut calculer une distance entre deux nuages Gaussiens avec la distance dites de Jeffries-Matusita (JM) basée sur la distance de Bhattacharyya  $B$  (Jensen 2016):

$$JM_{ij} = 2(1 - e^{-B_{ij}})$$

$$B_{ij} = \frac{1}{8}(m_i - m_j)^t \left(\frac{\Sigma_i + \Sigma_j}{2}\right)^{-1} (m_i - m_j) + \frac{1}{2} \ln \left(\frac{|(\Sigma_i + \Sigma_j)/2|}{|\Sigma_i|^{1/2} |\Sigma_j|^{1/2}}\right)$$

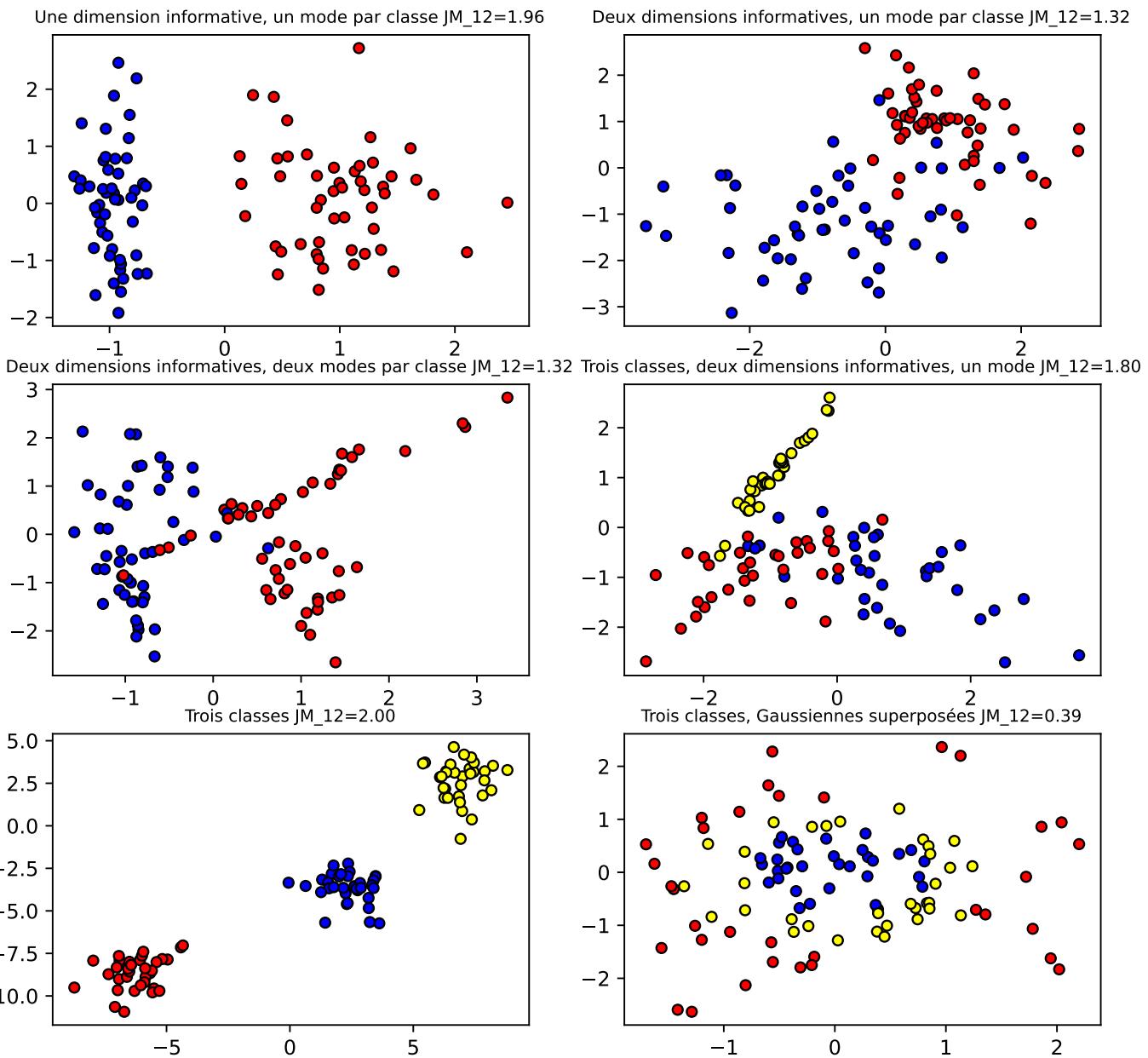
Cette distance présuppose que chaque classe  $i$  est décrite par son centre  $m_i$  et de sa dispersion dans l'espace à  $D$  dimensions mesurée par la matrice de covariance  $\Sigma_i$ . On peut en faire facilement une fonction Python à l'aide de **numpy**:

```

1 def bhattacharyya_distance(m1, s1, m2, s2):
2     # Calcul de la covariance moyenne
3     s = (s1 + s2) / 2
4
5     # Calcul du premier terme (différence des moyennes)
6     m_diff = m1 - m2
7     term1 = np.dot(np.dot(m_diff.T, np.linalg.inv(s)), m_diff) / 8
8
9     # Calcul du second terme (différence de covariances)
10    term2 = 0.5 * np.log(np.linalg.det(s)) / np.sqrt(np.linalg.det(s1) * np.linalg.det(s2)))
11
12    return term1 + term2
13
14 def jeffries_matusita_distance(m1, s1, m2, s2):
15     B = bhattacharyya_distance(m1, s1, m2, s2)
16     return 2 * (1 - np.exp(-B))

```

La figure ci-dessous illustre différentes situations avec des données simulées ainsi que les distances JM correspondantes :



On forme notre ensemble d'entraînement à partir du fichier `csv` de la section [6.2.3](#).

```

1 df= pd.read_csv('sampling_points.csv')
2 # Extraire la colonne 'value'.
3 # 'value' est une chaîne de caractères représentation d'une liste de nombres.
4 # Nous devons la convertir en données numériques réelles.
5 X = df['value'].apply(lambda x: np.fromstring(x[1:-1], dtype=float, sep=' ')).to_list()
6
7 # on obtient une liste de numpy array qu'il faut convertir en un numpy array 2D
8 X= np.array([row.tolist() for row in X])
9 idx= X.sum(axis=-1)>0 # on exclut certains points sans valeurs

```

```

10 X= X[idx,...]
11 y = df['class'].to_numpy()
12 y= y[idx]
13 class_labels = np.unique(y).tolist() # on cherche à savoir combien de classes uniques
14 n_classes = len(class_labels)
15 if max(class_labels) > n_classes: # il se peut que certaines classes soit absentes
16     y_new= []
17     for i,l in enumerate(class_labels):
18         y_new.extend([i]*sum(y==l))
19     y_new = np.array(y_new)
20
21 couleurs_classes2= [couleurs_classes[c] for c in np.unique(y).tolist()] # couleurs des classes
22 nom_classes2= [nom_classes[c] for c in np.unique(y).tolist()]
23 cmap_classes2 = ListedColormap(couleurs_classes2)

```

On peut faire une analyse de séparabilité sur notre ensemble d'entraînement de 10 classes. On obtient un tableau symétrique de 10x10 valeurs. On observe des valeurs inférieures à 1, indiquant des séparabilités faibles entre ces classes sous l'hypothèse du modèle Gaussien:

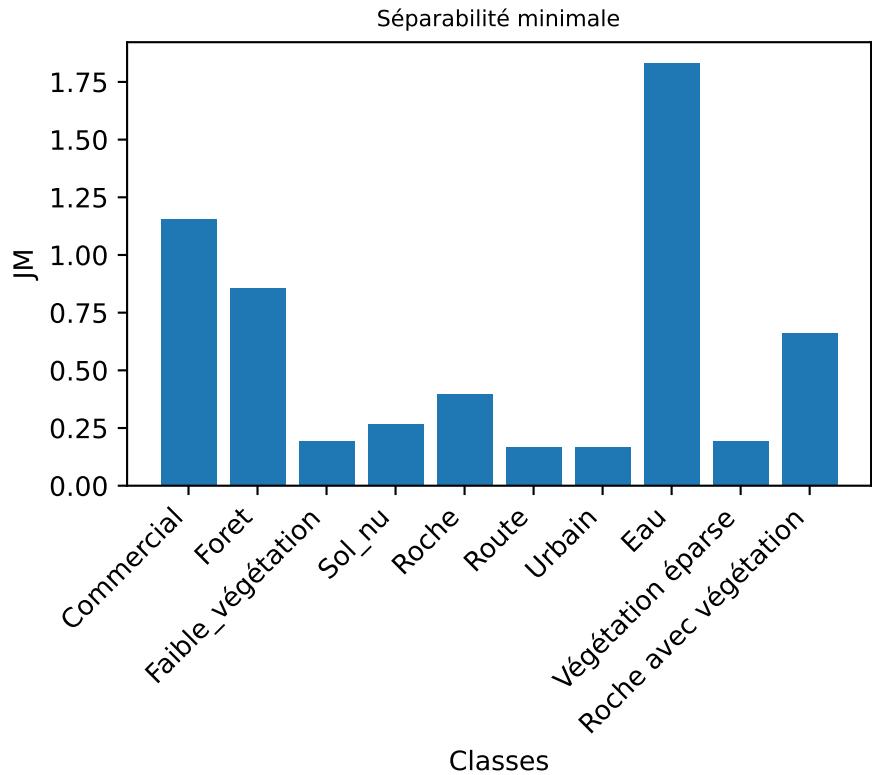
```

1 qda= QuadraticDiscriminantAnalysis(store_covariance=True)
2 qda.fit(X, y_new) # calcul des paramètres des distributions Gaussiennes
3 JM= []
4 classes= np.unique(y_new).tolist() # étiquettes uniques des classes
5 for cl1 in classes:
6     for cl2 in classes:
7         JM.append(jeffries_matusita_distance(qda.means_[cl1], qda.covariance_[cl1], qda.means_[cl2],
8                                         qda.covariance_[cl2]))
9
9 JM= np.array(JM).reshape(len(classes),len(classes))
10 JM= pd.DataFrame(JM, index=classes, columns=classes)
11 JM.head(10)

```

	0	1	2	3	4	5	6	7	8	9
0	0.000000	1.931891	1.809590	1.761760	1.156486	1.326107	1.319344	1.830671	1.873676	1.700417
1	1.931891	0.000000	1.082978	0.918865	1.788737	1.527192	1.331400	1.901749	0.854802	1.133180
2	1.809590	1.082978	0.000000	0.266647	1.428062	1.255001	1.198888	1.947302	0.193032	0.782982
3	1.761760	0.918865	0.266647	0.000000	1.413401	1.219793	1.127950	1.929637	0.377379	0.840250
4	1.156486	1.788737	1.428062	1.413401	0.000000	0.397103	0.596618	1.956182	1.517926	1.036828
5	1.326107	1.527192	1.255001	1.219793	0.397103	0.000000	0.167221	1.976696	1.248383	0.660213
6	1.319344	1.331400	1.198888	1.127950	0.596618	0.167221	0.000000	1.956804	1.207618	0.660589
7	1.830671	1.901749	1.947302	1.929637	1.956182	1.976696	1.956804	0.000000	1.966022	1.886064
8	1.873676	0.854802	0.193032	0.377379	1.517926	1.248383	1.207618	1.966022	0.000000	0.741273
9	1.700417	1.133180	0.782982	0.840250	1.036828	0.660213	0.660589	1.886064	0.741273	0.000000

Afin d'évaluer chaque classe, on peut calculer la séparabilité minimale, ce qui nous permet de constater que la classe eau a le maximum de séparabilité avec les autres classes.



## 6.4 Mesures de performance d'une méthode de classification

Lorsque que l'on cherche à établir la performance d'un modèle, il convient de mesurer la performance du classificateur utilisé. Il existe de nombreuses mesures de performance qui sont toutes dérivées de la matrice de confusion. Cette matrice compare les étiquettes provenant de l'annotation (la vérité terrain) et les étiquettes prédites par un modèle. On peut définir  $C(i, j)$  comme étant le nombre de prédictions dont la vérité terrain indique la classe  $i$  qui sont prédites dans la classe  $j$ . La fonction `confusion_matrix` permet de faire ce calcul, voici un exemple très simple:

```

1 y_true = ["cat", "ant", "cat", "cat", "ant", "bird", "bird"]
2 y_pred = ["ant", "ant", "cat", "cat", "ant", "cat", "bird"]
3 confusion_matrix(y_true, y_pred, labels=["ant", "bird", "cat"])

```

```

array([[2, 0, 0],
       [0, 1, 1],
       [1, 0, 2]])

```

La fonction `classification_report` permet de générer quelques métriques:

```

1 y_true = ["cat", "ant", "cat", "cat", "ant", "bird", "bird"]
2 y_pred = ["ant", "ant", "cat", "cat", "ant", "cat", "bird"]
3 print(classification_report(y_true, y_pred, target_names=["ant", "bird", "cat"]))

```

	precision	recall	f1-score	support
ant	0.67	1.00	0.80	2
bird	1.00	0.50	0.67	2
cat	0.67	0.67	0.67	3
accuracy			0.71	7
macro avg	0.78	0.72	0.71	7
weighted avg	0.76	0.71	0.70	7

Le rappel (*recall*) pour une classe donnée est la proportion de la vérité terrain qui a été correctement identifiée et est sensible aux confusions entre classes (erreurs d'omission). Les valeurs de rappels correspondent à une normalisation de la matrice de confusion par rapport aux lignes.

$$Recall_i = C_{ii} / \sum_j C_{ij}$$

Une faible valeur de rappel signifie que le classificateur confond facilement la classe concernée avec d'autres classes.

La précision est la portion des prédictions qui ont été bien classifiées et est sensible aux fausses alarmes (erreurs de commission). Les valeurs de précision correspondent à une normalisation de la matrice de confusion par rapport aux colonnes.

$$Precision_i = C_{ii} / \sum_i C_{ij}$$

Une faible valeur de précision signifie que le classificateur trouve facilement la classe concernée dans d'autres classes.

Le **f1-score** calcul une moyenne des deux métriques précédentes:

$$\text{f1-score}_i = 2 \frac{Recall_i \times Precision_i}{Recall_i + Precision_i}$$

## 6.5 Méthodes non paramétriques

Les méthodes non paramétriques ne font pas d'hypothèses particulières sur les données. Un des inconvénients de ces modèles est que le nombre de paramètres du modèle augmente avec la taille des données.

### 6.5.1 Méthode des parallélépipèdes

La méthode du parallélépipède est probablement la plus simple et consiste à délimiter directement le domaine des points d'une classe par une boîte (un parallélépipède) à  $D$  dimensions. Les limites de ces parallélépipèdes forment alors des frontières de décision manuelles qui permettent d'attribuer une classe d'appartenance à un nouveau point. Un des avantages de cette technique est que si un point n'est dans aucun parallélépipède alors il est non classifié. Par contre, la construction de ces parallélépipèdes se complexifient grandement avec le nombre de bandes. À une dimension, deux paramètres, équivalents à un seuillage d'histogramme, sont suffisants. À deux dimensions, vous devez définir 4 segments par classe. Avec trois bandes, vous devez définir six plans par classes

et à D dimensions, D hyperplans à D-1 dimensions par classe. Le modèle ici est donc une suite de valeurs `min` et `max` pour chacune des bandes et des classes:

```

1 def parrallepiped_train(X_train, y_train):
2     classes= np.unique(y_train).tolist()
3     clf= []
4     for cl in classes:
5         data_cl= X_train[y_train == cl,...] # on cherche les données pour la classe courante
6
7         limits=[]
8         for b in range(data_cl.shape[1]):
9             limits.append([data_cl[:,b].min(), data_cl[:,b].max()]) # on calcul le min et max pour chaque
→   bande
10        clf.append(np.array(limits))
11    return clf
12 clf= parrallepiped_train(X, y_new)

```

La prédiction consiste à trouver pour chaque point la première limite qui est satisfaite. Notez qu'il n'y a aucun moyen de décider quelle est la meilleure classe si le point appartient à plusieurs classes.

```

1 @jit(nopython=True)
2 def parrallepiped_predict(clf, X_test):
3     y_pred= []
4     for data in X_test:
5         y_pred.append(np.nan)
6         for cl, limits in enumerate(clf):
7             inside= True
8             for b,limit in enumerate(limits):
9                 inside = inside and (data[b] >= limit[0]) & (data[b] <= limit[1])
10            if ~inside:
11                break
12            if inside:
13                y_pred[-1]=cl
14    return np.array(y_pred)

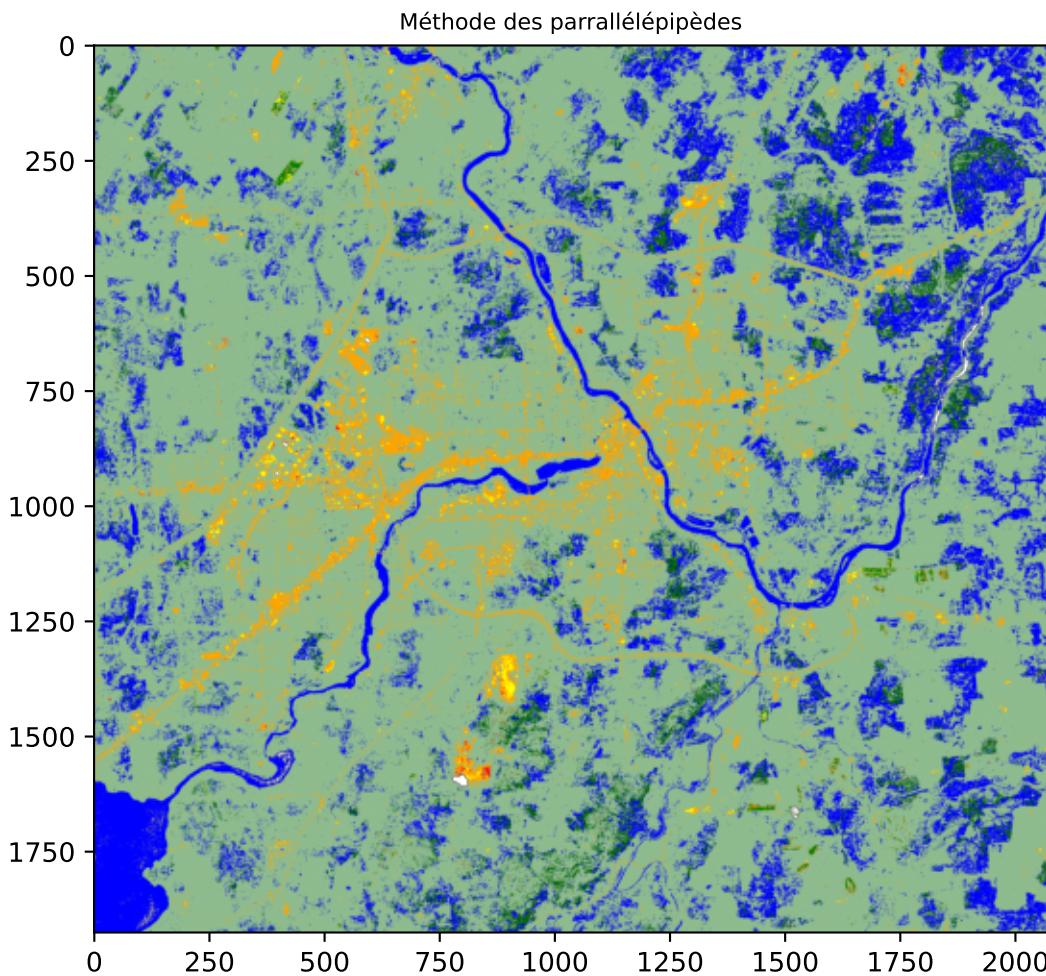
```

On peut appliquer ensuite le modèle sur l'image au complet. Les résultats sont assez mauvais, puisque seule la classe eau en bleu semble être bien classifiée.

```

1 data_image= img_rgnir.to_numpy().transpose(1,2,0).reshape(img_rgnir.shape[1]*img_rgnir.shape[2],4)
2 y_image= parrallepiped_predict(clf, data_image)
3 y_image= y_image.reshape(img_rgnir.shape[1],img_rgnir.shape[2])
4
5 fig, ax = plt.subplots(nrows=1, ncols=1, figsize=(8, 6))
6 plt.imshow(y_image, cmap=cmap_classes2)
7 ax.set_title("Méthode des parrallélépipèdes", fontsize="small")
8 plt.show()

```



On peut calculer quelques mesures de performance sur l'ensemble d'entraînement :

```

1 y_pred= parralleped_predict(clf, X)
2 nom_classes2= [nom_classes[c] for c in np.unique(y).tolist()]
3 print(classification_report(y_new, y_pred, target_names=nom_classes2, zero_division=np.nan))

```

	precision	recall	f1-score	support
Commercial	1.00	0.06	0.11	100
Forêt	1.00	0.09	0.17	100
Faible_végétation	1.00	0.02	0.04	100
Sol_nu	nan	0.00	0.00	100
Roche	0.00	0.00	0.00	100
Route	0.00	0.00	0.00	100
Urban	0.08	0.08	0.08	100
Eau	0.83	0.88	0.85	100
Végétation éparsse	1.00	0.01	0.02	100
Roche avec végétation	0.13	1.00	0.23	100
accuracy			0.21	1000
macro avg	0.56	0.21	0.15	1000
weighted avg	0.56	0.21	0.15	1000

### 6.5.1.1 La malédiction de la haute dimension

Augmenter le nombre de dimension ou de caractéristiques des données permet de résoudre des problèmes complexes comme la classification d'image. Cependant, cela amène beaucoup de contraintes sur le volume des données. Supposons que nous avons  $N$  points occupant un segment linéaire de taille  $d$ . La densité de points est  $N/d$ . Si nous augmentons le nombre de dimension  $D$ , la densité de points va diminuer exponentiellement en  $1/d^D$ . Par conséquent, pour garder une densité constante et donc une bonne estimation des parallélépipèdes, il nous faudrait augmenter le nombre de points en puissance de  $D$ . Ceci porte le nom de la malédiction de la dimensionnalité (*dimensionality curse*). En résumé, l'espace vide augmente plus rapidement que le nombre de données d'entraînement et l'espace des données devient de plus en plus parcimonieux (*sparse*). Pour contrecarrer ce problème, on peut sélectionner les meilleures caractéristiques ou appliquer une réduction de dimension comme une ACP (Analyse en composantes principales).

### 6.5.2 Plus proches voisins

La méthode des plus proches voisins (*K-Nearest-Neighbors*) est certainement la plus simple des méthodes pour classifier des données. Elle consiste à comparer une nouvelle donnée avec ses voisins les plus proches en fonction d'une simple distance Euclidienne. Si une majorité de ces  $K$  voisins appartiennent à une classe majoritaire alors cette classe est sélectionnée. Afin de permettre un vote majoritaire, on choisira un nombre impair pour la valeur de  $K$ . Malgré sa simplicité, cette technique peut devenir assez demandante en temps de calcul pour un nombre important de points et un nombre élevé de dimensions.

Reprendons l'ensemble d'entraînement formé à partir de notre image RGNBIR précédente :

```

1 df= pd.read_csv('sampling_points.csv')
2 # Extraire la colonne 'value'.
3 # 'value' est une chaîne de caractères comme représentation d'une liste de valeurs.
4 # Nous devons la convertir en données numériques réelles.
5 X = df['value'].apply(lambda x: np.fromstring(x[1:-1], dtype=float, sep=' ')).to_list()
6
7 # on obtient une liste de numpy array qu'il faut convertir en un numpy array 2D
8 X= np.array([row.tolist() for row in X])
9 idx= X.sum(axis=-1)>0 # il se peut qu'il y ait des valeurs erronées
10 X= X[idx,...]
11 y = df['class'].to_numpy()
12 y= y[idx]
13 class_labels = np.unique(y).tolist() # on cherche à savoir combien de classes uniques
14 n_classes = len(class_labels)
15 if max(class_labels) > n_classes: # il se peut que certaines classes soit absentes
16     y_new= []
17     for i,l in enumerate(class_labels):
18         y_new.extend([i]*sum(y==l))
19     y_new = np.array(y_new)
20 nom_classes2= [nom_classes[c] for c in np.unique(y).tolist()]

```

Il importe de préalablement centrer (moyenne = 0) et de réduire (variance = 1) les données avant d'appliquer la méthode K-NN; avec cette méthode de normalisation, on dit parfois que l'on blanchit les données. Puisque la variance de chaque dimension est égale à 1 (et donc l'inertie totale est égale au nombre de bandes), on s'assure qu'elle ait le même poids ait le même poids dans le calcul des distances entre points. Cette opération porte le nom de **StandardScaler** dans **scikit-learn**. On peut alors former un pipeline de traitement combinant les deux opérations :

```
1 clf = Pipeline(
2     steps=[("scaler", StandardScaler()), ("knn", KNeighborsClassifier(n_neighbors=1))]
3 )
```

Avant d'effectuer un entraînement, on met généralement une portion des données pour valider les performances :

```
1 X_train, X_test, y_train, y_test = train_test_split(X, y_new, test_size=0.2, random_state=0)
```

On peut visualiser les frontières de décision du K-NN pour différentes valeurs de  $K$  lorsque seulement deux bandes sont utilisées (Rouge et proche infra-rouge ici) :

```
Number of mislabeled points out of a total 200 points : 143
Number of mislabeled points out of a total 200 points : 141
Number of mislabeled points out of a total 200 points : 136
Number of mislabeled points out of a total 200 points : 130
```

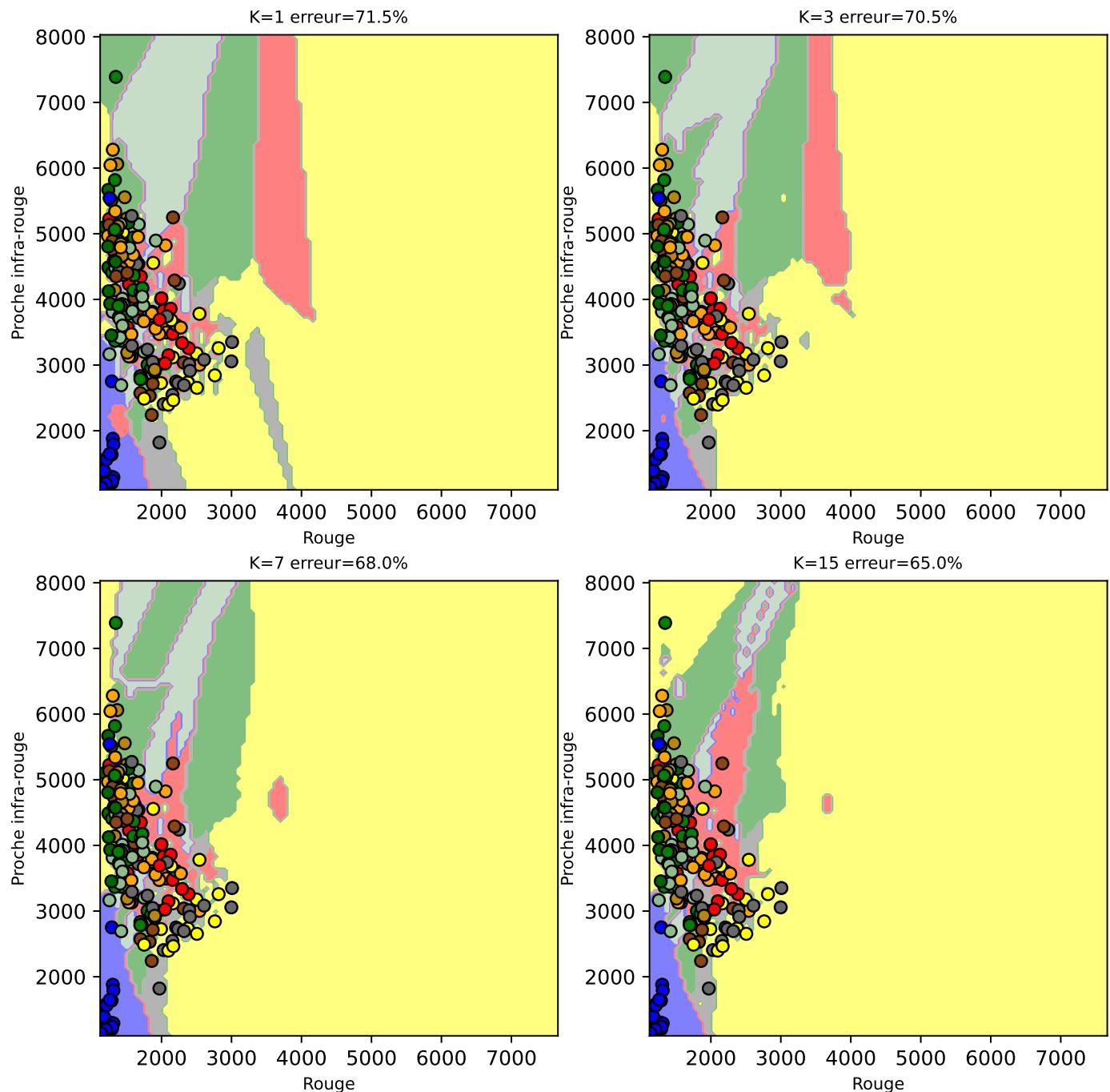


FIGURE 6.3 – Frontières de décision pour le classificateur K-NN

On peut voir comment les différentes frontières de décision se forment dans l'espace des bandes Rouge-NIR. L'augmentation de K rend ces frontières plus complexes et le calcul plus long.

```

1 clf.set_params(knn__weights='distance', knn__n_neighbors = 7).fit(X_train, y_train)
2 y_pred = clf.predict(X_test)

```

```

3 print("Nombre de points misclassified sur %d points : %d"
4   % (X_test.shape[0], (y_test != y_pred).sum()))

```

Nombre de points misclassified sur 200 points : 117

Le rapport de performance est le suivant :

```

1 nom_classes2= [nom_classes[c] for c in np.unique(y).tolist()]
2 print(classification_report(y_test, y_pred, target_names=nom_classes2, zero_division=np.nan))

```

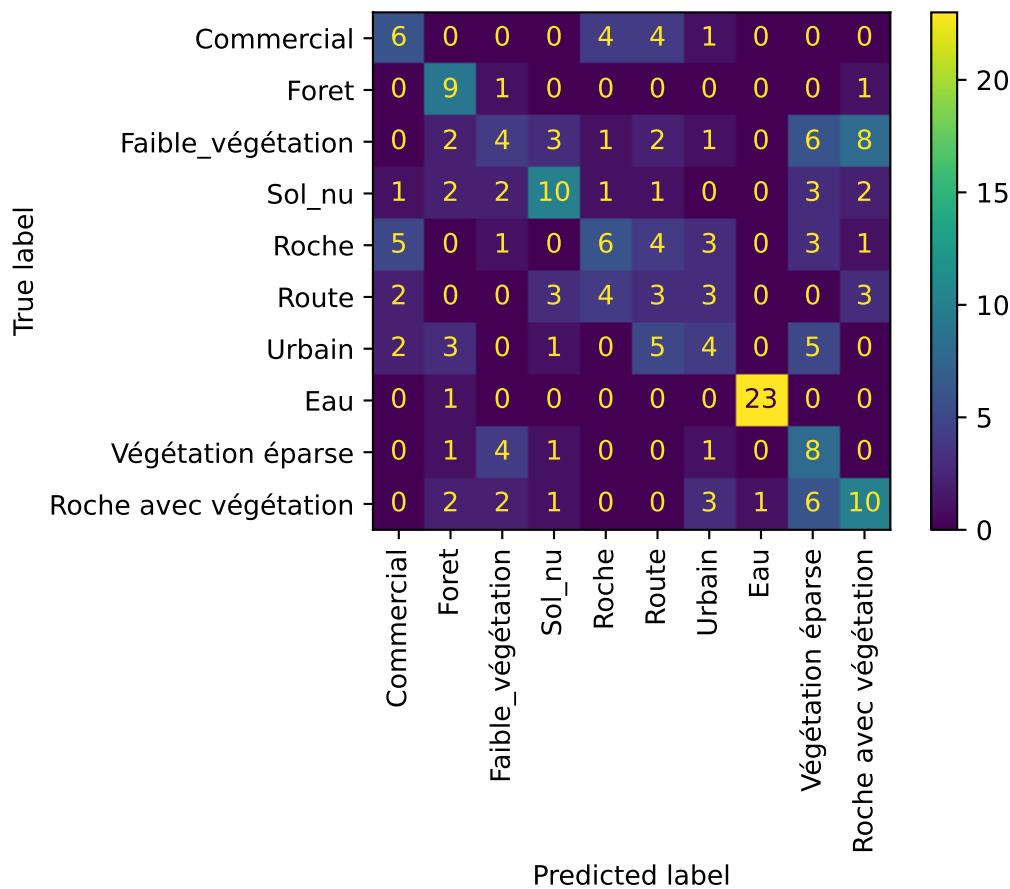
	precision	recall	f1-score	support
Commercial	0.38	0.40	0.39	15
Forêt	0.45	0.82	0.58	11
Faible_végétation	0.29	0.15	0.20	27
Sol_nu	0.53	0.45	0.49	22
Roche	0.38	0.26	0.31	23
Route	0.16	0.17	0.16	18
Urban	0.25	0.20	0.22	20
Eau	0.96	0.96	0.96	24
Végétation épars	0.26	0.53	0.35	15
Roche avec végétation	0.40	0.40	0.40	25
accuracy			0.41	200
macro avg	0.40	0.43	0.40	200
weighted avg	0.42	0.41	0.40	200

La matrice de confusion peut-être affichée de manière graphique :

```

1 disp= ConfusionMatrixDisplay.from_predictions(y_test, y_pred, display_labels=nom_classes2,
2   ↴ xticks_rotation='vertical')

```



L'application du modèle (la prédiction) peut se faire sur toute l'image en transposant l'image sous forme d'une matrice avec Largeur x Hauteur lignes et 4 colonnes :

```

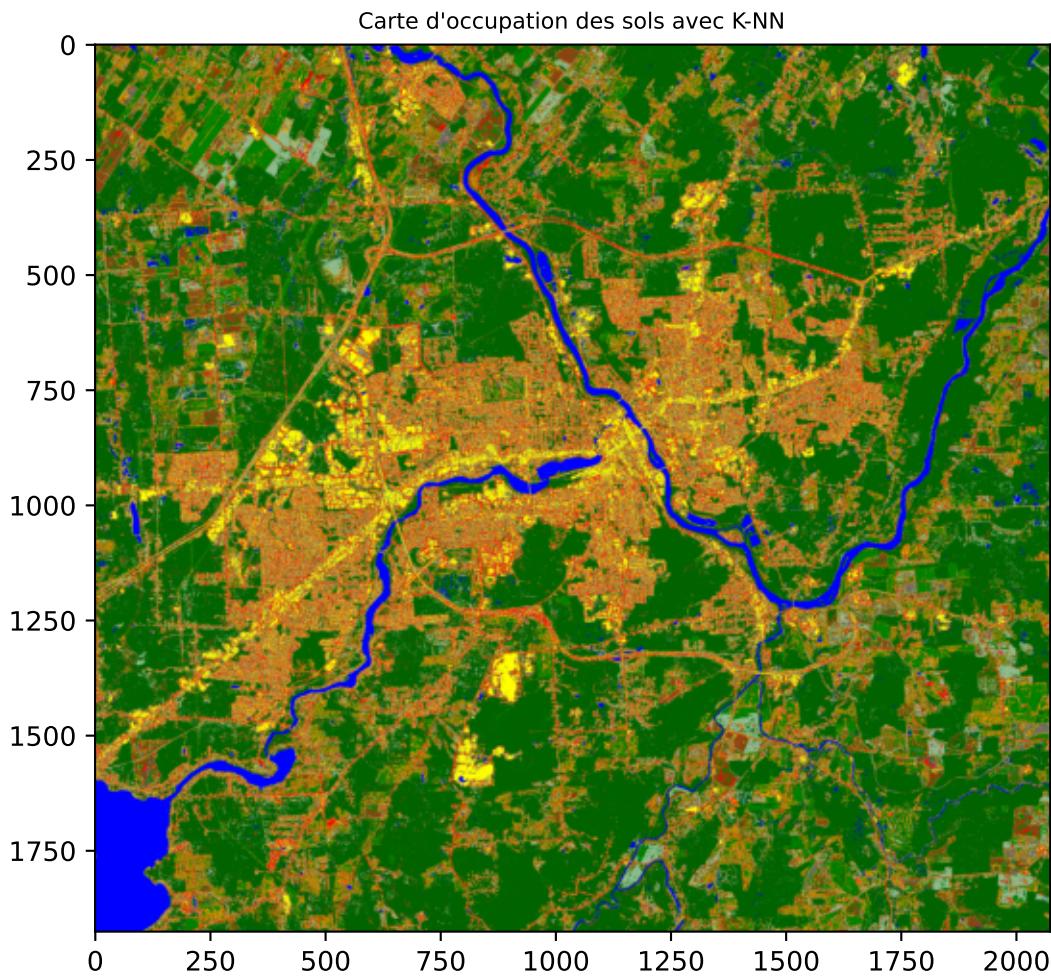
1 data_image= img_rgnir.to_numpy().transpose(1,2,0).reshape(img_rgnir.shape[1]*img_rgnir.shape[2],4)
2 y_classe= clf.predict(data_image)
3 y_classe= y_classe.reshape(img_rgnir.shape[1],img_rgnir.shape[2])

```

```

1 fig, ax = plt.subplots(nrows=1, ncols=1, figsize=(8, 6))
2 plt.imshow(y_classe, cmap=cmap_classes2)
3 ax.set_title("Carte d'occupation des sols avec K-NN", fontsize="small")
4 plt.show()

```



### 6.5.3 Méthodes par arbre de décision

La méthode par arbre de décision consiste à construire une cascade de règles de décision sur chaque caractéristique du jeu de donnée (Breiman et C. Stone 1984). On pourra trouver plus de détails dans la documentation de [scikit-learn](#) ([Decision Trees](#)). Les arbres de décision ont tendance à surapprendre surtout si le nombre de dimensions est élevé. Il est donc conseillé d'avoir un bon ratio entre le nombre d'échantillons et le nombre de dimensions.

```
1 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

```
Number of mislabeled points out of a total 200 points : 167
Number of mislabeled points out of a total 200 points : 154
Number of mislabeled points out of a total 200 points : 143
Number of mislabeled points out of a total 200 points : 128
```

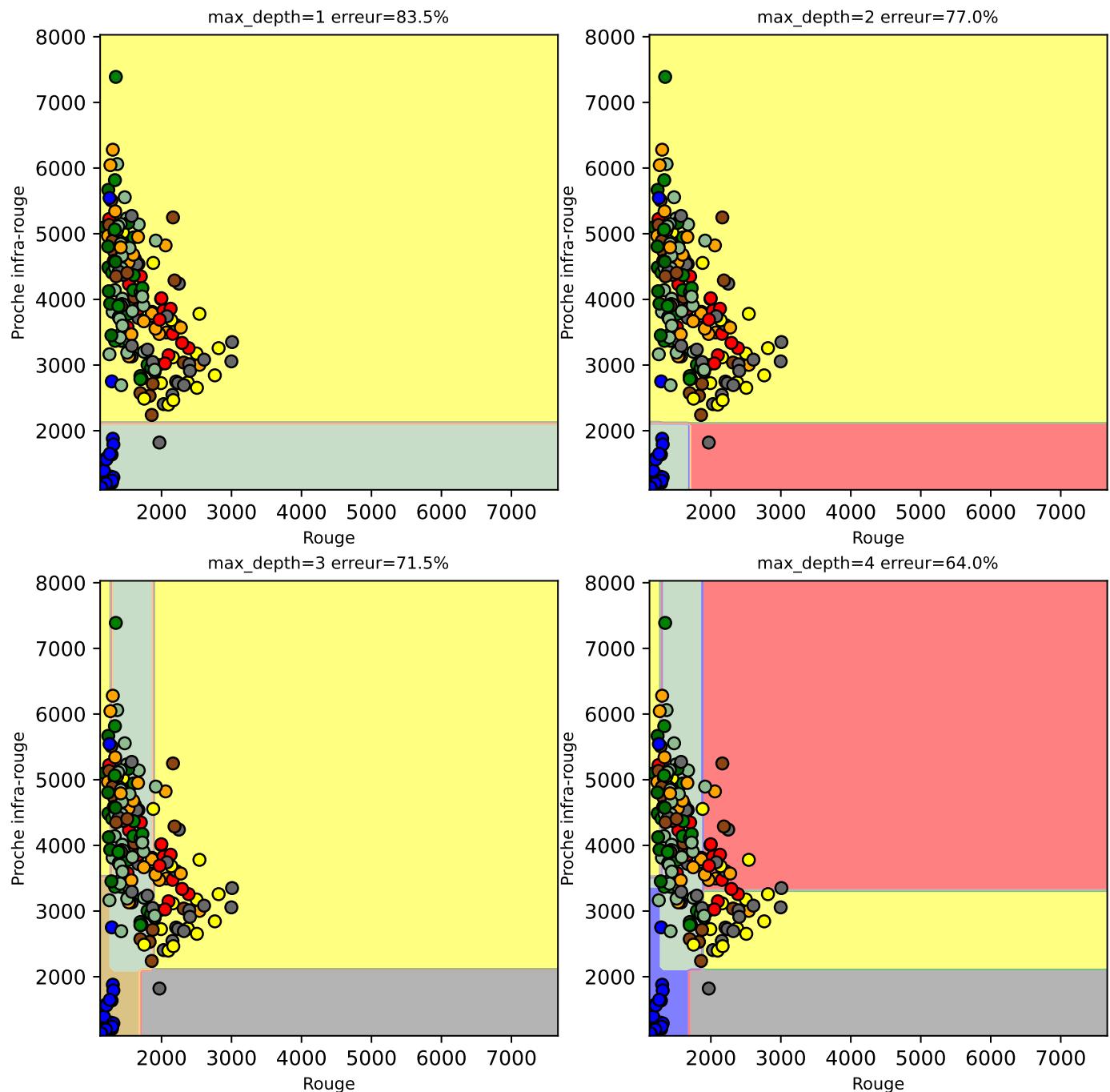


FIGURE 6.4 – Frontières de décision pour des arbres de décision de différente profondeur

On peut observer que les frontières de décision sont formées d'un ensemble de plans simple. Chaque plan étant issu d'une règle de décision formé d'un seuil sur chacune des dimensions. On entraîne un arbre de décision avec une profondeur maximale de 5:

```

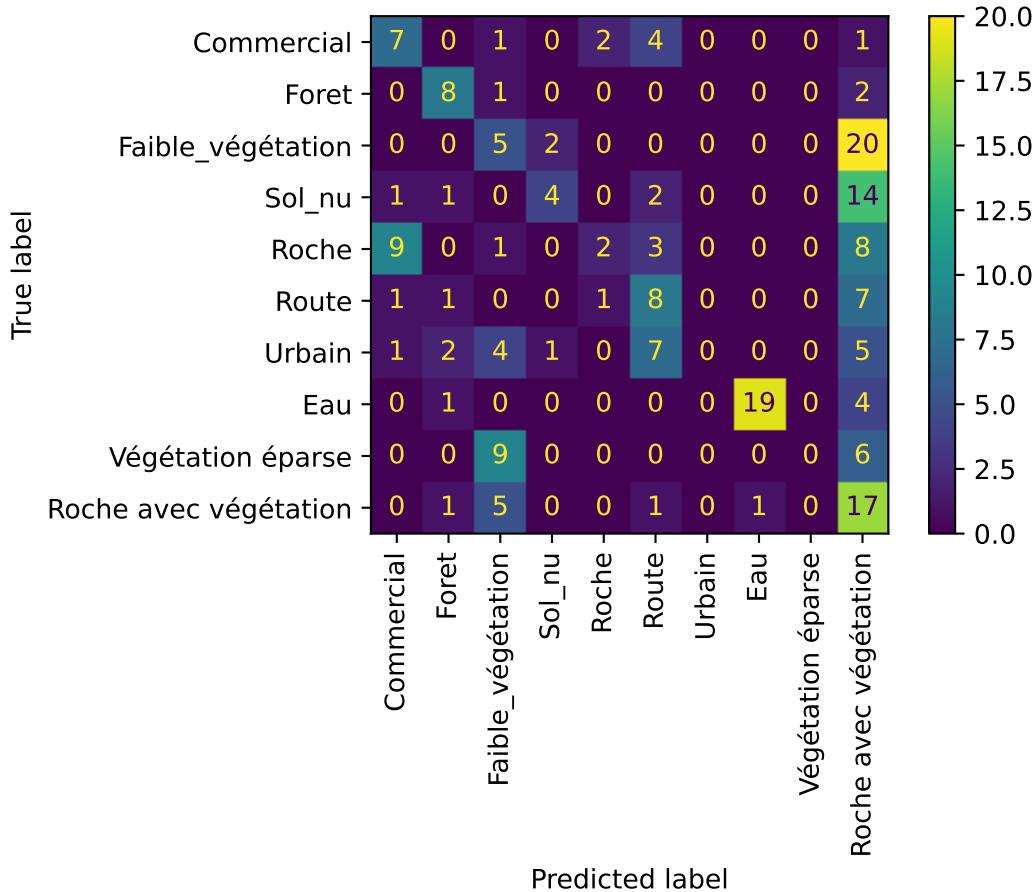
1 clf = tree.DecisionTreeClassifier(max_depth=5)
2 clf.fit(X_train, y_train)
3 y_pred = clf.predict(X_test)
4 print("Nombre de points misclassifiés sur %d points : %d"
5     % (X_test.shape[0], (y_test != y_pred).sum()))

```

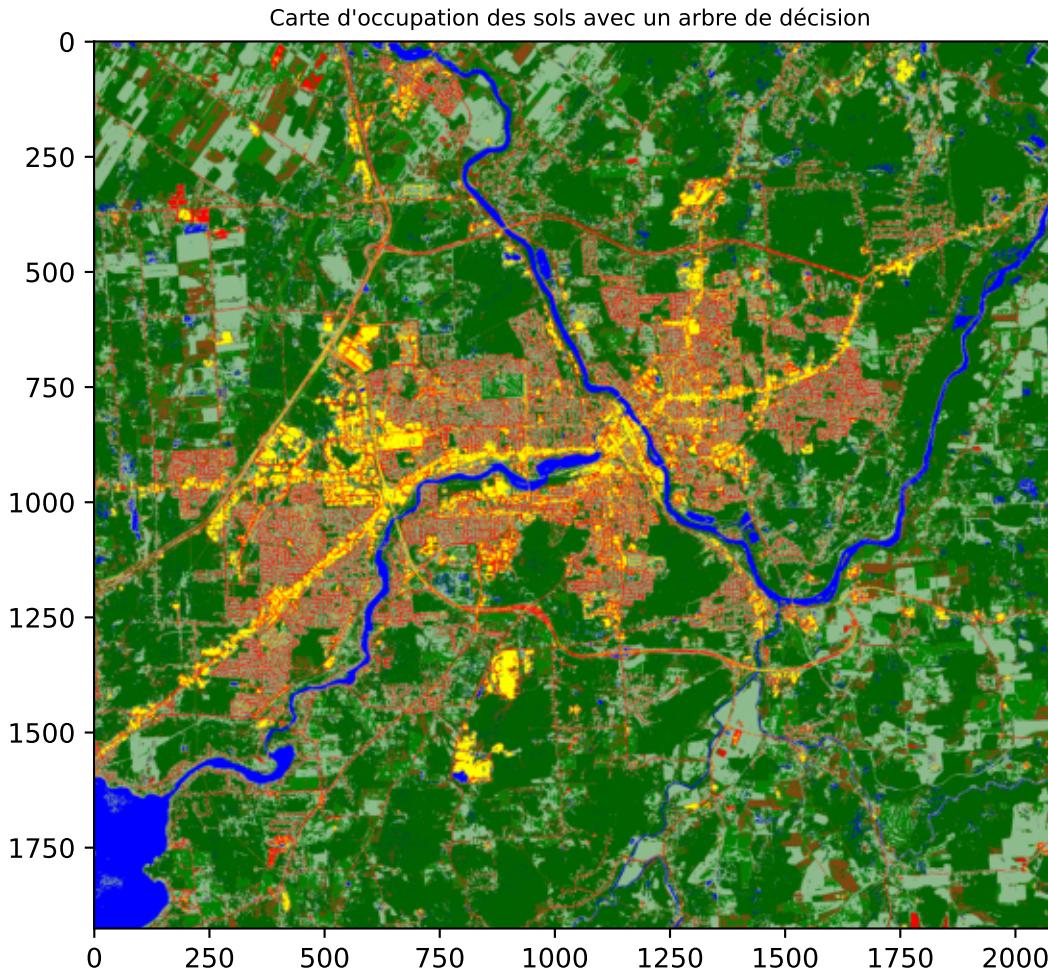
Nombre de points misclassifiés sur 200 points : 130

Le rapport de performance et la matrice de confusion:

	precision	recall	f1-score	support
Commercial	0.37	0.47	0.41	15
Foret	0.57	0.73	0.64	11
Faible_végétation	0.19	0.19	0.19	27
Sol_nu	0.57	0.18	0.28	22
Roche	0.40	0.09	0.14	23
Route	0.32	0.44	0.37	18
Urbain	nan	0.00	0.00	20
Eau	0.95	0.79	0.86	24
Végétation éparsse	nan	0.00	0.00	15
Roche avec végétation	0.20	0.68	0.31	25
accuracy			0.35	200
macro avg	0.45	0.36	0.32	200
weighted avg	0.44	0.35	0.31	200



L'application du modèle (la prédiction) peut se faire sur toute l'image en transposant l'image sous forme d'une matrice avec Largeur x Hauteur lignes et 4 colonnes:



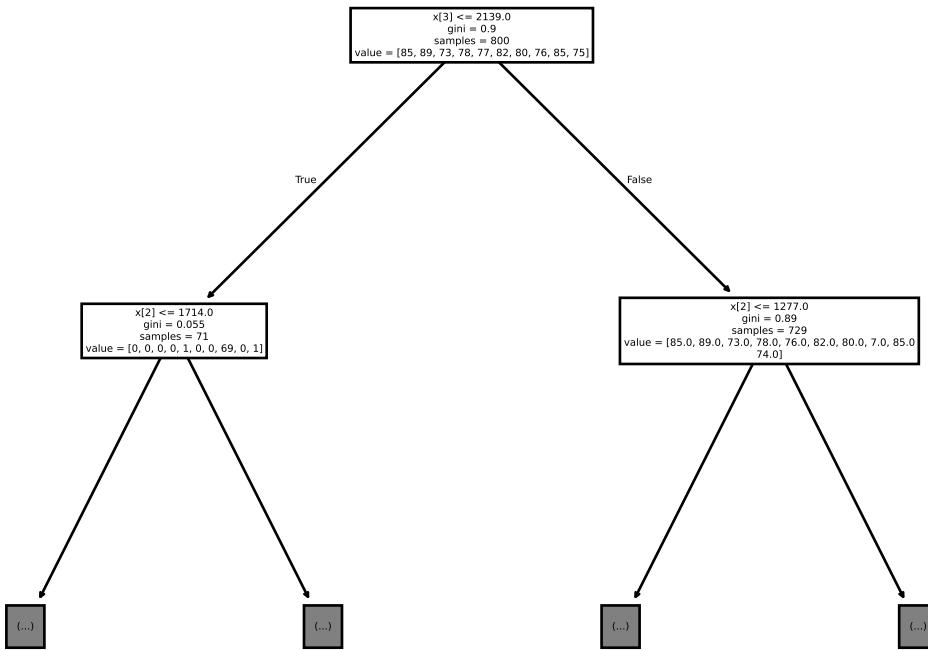
Il est possible de visualiser l'arbre mais cela contient beaucoup d'information

```

1 fig, ax = plt.subplots(nrows=1, ncols=1, figsize=(8, 6))
2 tree.plot_tree(clf, max_depth=1)

[Text(0.5, 0.8333333333333334, 'x[3] <= 2139.0\n gini = 0.9\n samples = 800\nvalue = [85, 89, 73, 78, 77, 82, 80, 76, 85, 75]'),
Text(0.25, 0.5, 'x[2] <= 1714.0\n gini = 0.055\n samples = 71\nvalue = [0, 0, 0, 1, 0, 0, 69, 0, 1]'),
Text(0.375, 0.6666666666666667, 'True '),
Text(0.125, 0.1666666666666666, '\n (...) \n'),
Text(0.375, 0.1666666666666666, '\n (...) \n'),
Text(0.75, 0.5, 'x[2] <= 1277.0\n gini = 0.89\n samples = 729\nvalue = [85.0, 89.0, 73.0, 78.0, 76.0, 82.0, 80.0, 7.0, 85.0\n74.0]'),
Text(0.625, 0.6666666666666667, ' False'),
Text(0.625, 0.1666666666666666, '\n (...) \n'),
Text(0.875, 0.1666666666666666, '\n (...) \n')]

```



## 6.6 Méthodes paramétriques

Les méthodes paramétriques se basent sur des modélisations statistiques des données pour permettre une classification. Contrairement aux méthodes non paramétriques, elles ont un nombre fixe de paramètres qui ne dépend pas de la taille du jeu de données. Par contre, des hypothèses sont faites a priori sur le comportement statistique des données. La classification consiste alors à trouver la classe la plus vraisemblable dont le modèle statistique décrit le mieux les valeurs observées. L'ensemble d'entraînement permettra alors de calculer les paramètres de chaque Gaussienne pour chacune des classes d'intérêt.

### 6.6.1 Méthode Bayésienne naïve

La méthode Bayésienne naïve Gaussienne consiste à poser des hypothèses simplificatrices sur les données, en particulier l'indépendance des données et des dimensions. Ceci permet un calcul plus simple.

```

1 from sklearn.naive_bayes import GaussianNB
2 gnb = GaussianNB()
3 y_pred = gnb.fit(X_train, y_train).predict(X_test)
  
```

```

4 print("Nombre de points erronés sur %d points : %d"
5     % (X_test.shape[0], (y_test != y_pred).sum()))

```

Nombre de points erronés sur 200 points : 131

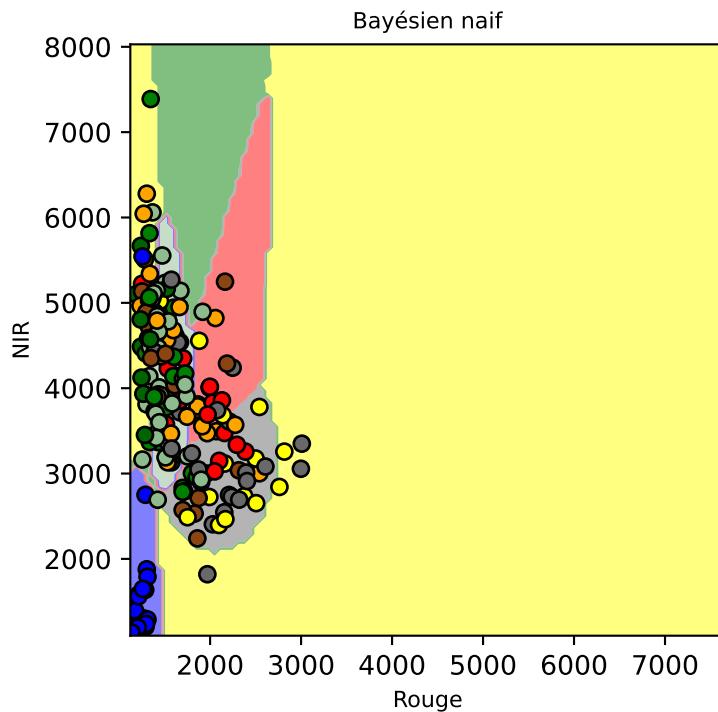


FIGURE 6.5 – Frontières de décision pour un classificateur Bayésien naïf

On observe que les frontières de décision sont beaucoup plus régulières que pour K-NN.

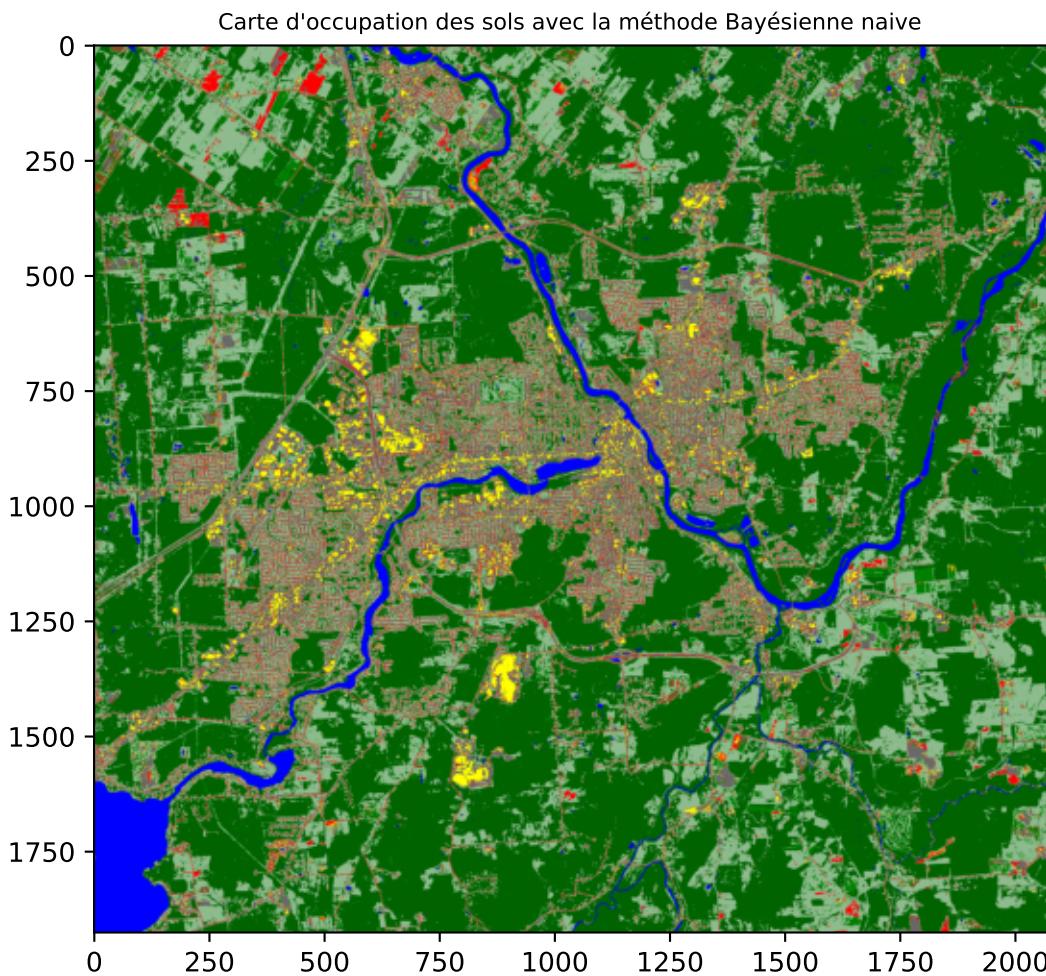
```

1 gnb.fit(X_train, y_train)
2 y_pred = gnb.predict(X_test)
3 print("Nombre de points misclassifiés sur %d points : %d"
4     % (X_test.shape[0], (y_test != y_pred).sum()))

```

Nombre de points misclassifiés sur 200 points : 131

De la même manière, la prédiction peut s'appliquer sur toute l'image:



### 6.6.2 Analyse discriminante quadratique (ADQ)

L'analyse discriminante quadratique peut-être vue comme une généralisation de l'approche Bayésienne naïve qui suppose des modèles Gaussiens indépendants pour chaque dimension et chaque point. Ici, on va considérer un modèle Gaussien multivarié.

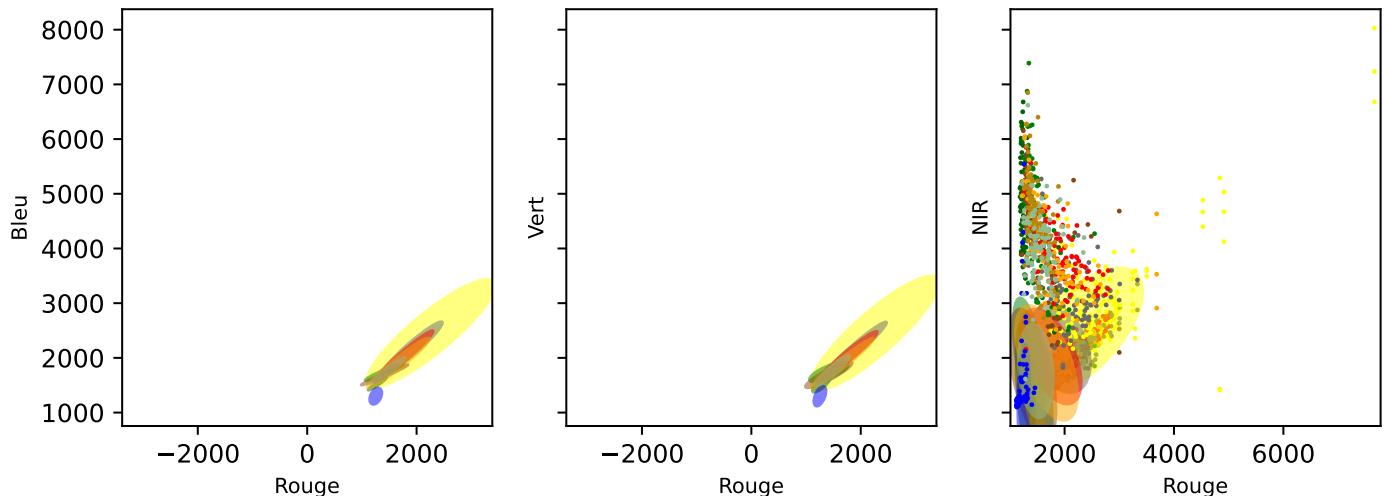
```

1 qda = QuadraticDiscriminantAnalysis(store_covariance=True)
2 qda.fit(X_train, y_train)
3 y_pred = qda.predict(X_test)
4 print("Nombre de points misclassifiés sur %d points : %d"
5     % (X_test.shape[0], (y_test != y_pred).sum()))

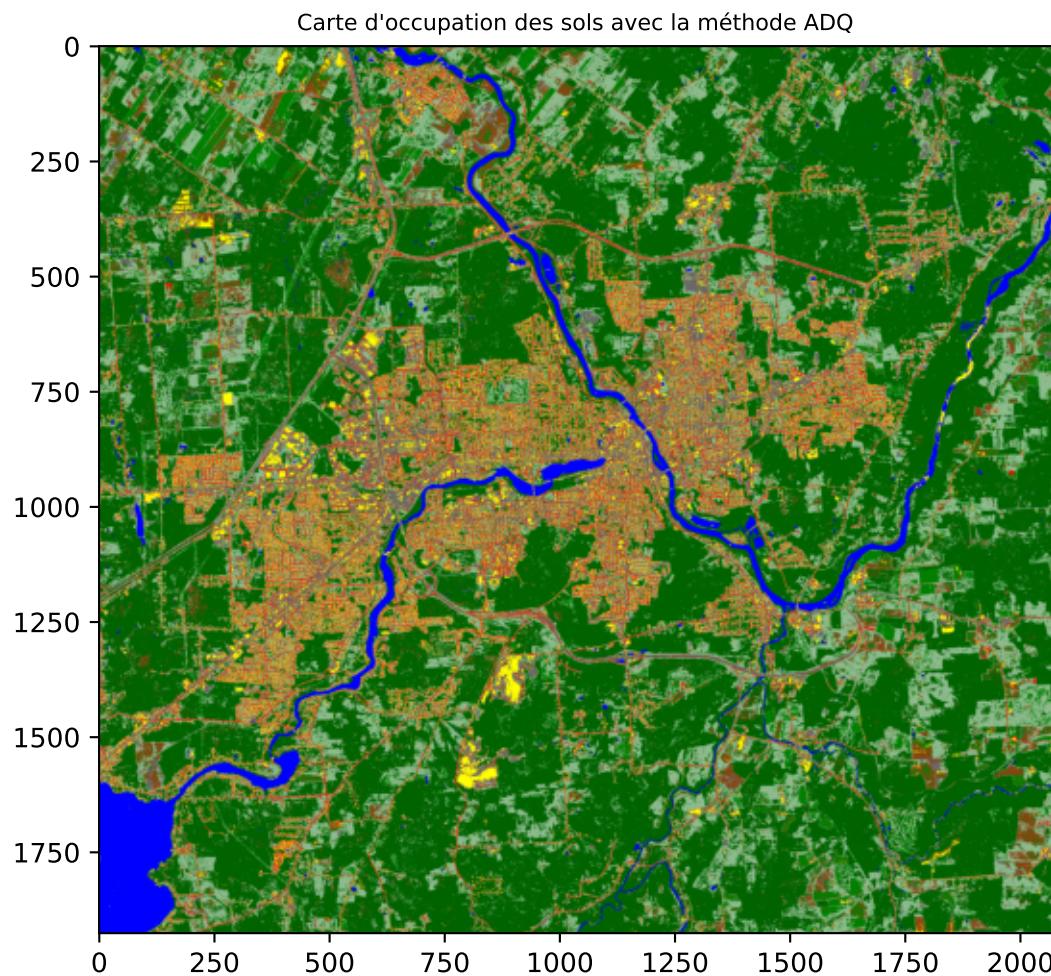
```

Nombre de points misclassifiés sur 200 points : 124

Les Gaussiennes multivariées peuvent être visualiser sous forme d'ellipses décrivant le domaine des valeurs de chaque classe:



De la même manière, la prédiction peut s'appliquer sur toute l'image:



# Bibliographie

- Achanta, Kevin Smith adhakrishna, Appu Shaji et Sabine Süsstrunk. 2012. « SLIC Superpixels Compared to State-of-the-art Superpixel Methods. » *TPAMI*: 636-643. <https://doi.org/10.1109/TPAMI.2012.120>.
- Breiman, Friedman, L. et C. C. Stone. 1984. *Classification and Regression Trees*. Wadsworth, Belmont, CA.
- Cooley, James W. et John W. Tukey. 1965. « An algorithm for the machine calculation of complex Fourier series. » *Math. Comput.*: 297-301. <https://web.stanford.edu/class/cme324/classics/cooley-tukey.pdf>.
- Harris, Millman, C. R. 2020. « Array programming with NumPy. » *Nature*: 357-362. <https://doi.org/10.1038/s41586-020-2649-2>.
- Hoyer, S. et J. Hamman. 2017. « xarray: N-D labeled Arrays and Datasets in Python. » *Journal of Open Research Software* 5 (1): 10. <https://doi.org/10.5334/jors.148>.
- Jahne, Scharr, B et Korkel S. 1999. *Principles of filter design*. Handbook of Computer Vision; Applications; Academic Press.
- Jaworek-Korjakowska, P., J.; Kłeczek. 2018. « Region Adjacency Graph Approach for Acral Melanocytic Lesion Segmentation. » *Applied Sciences* 8: 1430. [10.3390/app8091430](https://doi.org/10.3390/app8091430).
- Jensen, J. R. 2016. *Introductory digital image processing: A remote sensing perspective*. Pearson Education, Inc.
- Kokaly, Clark, R. F. et A. J. Klein. 2017. « USGS Spectral Library Version 7 Data: U.S. Geological Survey data release. » *Applied Sciences*. [10.5066/F7RR1WDJ](https://doi.org/10.5066/F7RR1WDJ).
- Lee, J. S. 1986. « Speckle suppression and analysis for synthetic aperture radar images. » *Opt. Eng.* 25 (5): 636-643. <https://doi.org/10.1117/12.7973877>.
- OGC. 2019. « OGC GeoTIFF Standard. » <https://docs.ogc.org/is/19-008r4/19-008r4.html/>.