Traitement d’images satellites avec Python

Première édition

Samuel Foucher

Philippe Apparicio

Yacine Bouroubi

Mickaël Germain

2025-01-13

Table des matières

# Préface

**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.

|  |
| --- |
| Important |
| **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.

**Maquette de la page couverture et identité graphique du livre :** .

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

**Révision linguistique :** Denise Latreille.

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

**Pour citer cet ouvrage :** Foucher S., Apparicio P., Bouroubi, Y. et M. Germain (2024). *Traitement d’images satellites avec Python*. Université de Sherbrooke, Département de géomatique appliquée. fabriqueREL. Licence CC BY-SA.



## 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](https://fr.wikipedia.org/wiki/Licence_publique_g%C3%A9n%C3%A9rale_GNU)). 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 — Remixer — Redistribuer**) et donc permet nécessairement la modification » ([***fabriqueREL***](https://fabriquerel.org/rel/)). La licence de ce livre, CC BY-SA ([figure 1](#fig-Licence)), 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 R avec [Quatro](https://quarto.org/). Quiconque peut ainsi télécharger gratuitement le code source du livre sur [github](https://github.com/SerieBoldR/MethodesAnalyseSpatiale) 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***](https://fabriquerel.org/rel/)).
* **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](#fig-Licence)), 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 (2024). *Traitement d’images satellites :* . 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 |

**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.

**Bloc *packages***

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

**Bloc objectif**

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

**Bloc notes**

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

**Bloc pour aller plus loin**

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

**Bloc astuce**

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

**Bloc attention**

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

**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>.

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](#fig-downloaffromgit)). 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](#sec-chap01)). Ce chapitre vous permettra de maîtriser la manipulation à bas niveau de différents types d’imagerie. Différents exemples et exercises 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](#sec-chap03)) et les transformations spatiales ([chapitre 5](#sec-chap04)).

**Partie 3. Classifications d’images** Cette troisième partie comprend deux chapitres : les classifications supervisées ([chapitre 6](#sec-chap05)) et non supervisées ([chapitre 7](#sec-chap06)).

**Partie 4. Données massives**. Cette quatrième et dernière partie comprend un seul chapitre qui est dédié aux plateformes de mégadonnes [chapitre 8](#sec-chap07), notammment 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***](https://fabriquerel.org/) (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](https://fabriquerel.org/rel/) (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 :

* Myriam Beaudet, bibliothécaire à l’Université de Sherbrooke.
* Marianne Dubé, coordonnatrice de la fabriqueREL, Université de Sherbrooke.
* Serge Piché, conseiller pédagogique, Université de Sherbrooke.
* Claude Potvin, conseiller en formation, Service de soutien à l’enseignement, Université Laval.

Nous remercions chaleureusement les personnes étudiantes des cours **à modifier plus tard** du [Baccalauréat en géomatique appliquée à l’environnement](https://www.usherbrooke.ca/admission/programme/271/baccalaureat-en-geomatiqueappliquee-a-lenvironnement/) et du [Microprogramme de 1er cycle en géomatique appliquée](https://www.usherbrooke.ca/admission/programme/429/microprogramme-de-1er-cycleen-geomatique-appliquee/) du [Département de géomatique appliquée](https://www.usherbrooke.ca/geomatique/) de l’[Université de Sherbrooke](https://www.usherbrooke.ca/) de la session d’été 2023 : à modifier plus tard.

Nous remercions aussi les membres du comité de révision pour leurs commentaires et suggestions très constructifs. Ce comité est composé de quatre personnes étudiantes du [Département de géomatique appliquée](https://www.usherbrooke.ca/geomatique/) de l’[Université de Sherbrooke](https://www.usherbrooke.ca/) :

* À compléter plus tard.
* À compléter plus tard.

Finalement, nous remercions Denise Latreille, réviseure linguistique et chargée de cours à l’Université Sherbrooke, pour la révision du manuel.

## Introduction aux images de télédétection

L’imagerie numérique a pris une place importante dans notre vie de tous les jours depuis une quinzaine d’année. 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 de tous les jours. On peut souligner au moins cinq caractéristiques principales:

1. Les images sont géoréférencées : Cela veut dire 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 3 bandes. Il n’est pas rare de trouver 4 bandes (Pléiade), 13 bandes (Sentinel-2, Landsat) et même 200 bandes pour des capteurs hyperspectraux.
4. Elles peuvent être calibrées : Les valeurs numérique 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](#sec-00-01)) ou chapitre par chapitre.

# À propos des auteurs

[**Samuel Foucher**](https://www.usherbrooke.ca/recherche/fr/specialistes/details/samuel.foucher) est professeur titulaire au [Département de géomatique appliquée](https://www.usherbrooke.ca/geomatique/) de l’[Université de Sherbrooke](https://www.usherbrooke.ca/). Il y enseigne aux [programmes de 1er et 2e cycles de géomatique](https://www.usherbrooke.ca/geomatique/etudes/programmes) 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](https://www.ciqss.org/)). 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.

[**Philippe Apparicio**](https://www.usherbrooke.ca/recherche/fr/specialistes/details/philippe.apparicio) est professeur titulaire au [Département de géomatique appliquée](https://www.usherbrooke.ca/geomatique/) de l’[Université de Sherbrooke](https://www.usherbrooke.ca/). Il y enseigne aux [programmes de 1er et 2e cycles de géomatique](https://www.usherbrooke.ca/geomatique/etudes/programmes) 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](https://www.ciqss.org/)). 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**](https://www.usherbrooke.ca/geomatique/departement/personnel/personnel-enseignant/mickael-germain) est professeur titulaire au [Département de géomatique appliquée](https://www.usherbrooke.ca/geomatique/) de l’[Université de Sherbrooke](https://www.usherbrooke.ca/). Il y enseigne aux [programmes de 1er et 2e cycles de géomatique](https://www.usherbrooke.ca/geomatique/etudes/programmes) 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](https://www.ciqss.org/)). 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.

[**Yacine Bouroubi**](https://www.usherbrooke.ca/geomatique/departement/personnel/personnel-enseignant/yacine-bouroubi) est professeur titulaire au [Département de géomatique appliquée](https://www.usherbrooke.ca/geomatique/) de l’[Université de Sherbrooke](https://www.usherbrooke.ca/). Il y enseigne aux [programmes de 1er et 2e cycles de géomatique](https://www.usherbrooke.ca/geomatique/etudes/programmes) 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](https://www.ciqss.org/)). 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.

# 1. Introduction au langage Python

Dans ce chapitre, nous allons présenter 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 une fin en soit ici. Python est pour nous principalement un outil de ‘scriptage’ et de manipulation de la donnée.

Python, créé par [Guido van Rossum](https://en.wikipedia.org/wiki/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](https://wiki.python.org/moin/PythonDistributions) 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](https://www.python.org/downloads/) : C’est la distribution “vanille” officielle, comme la recette originale de Python. C’est le choix idéal pour la compatibilité et la conformité aux standards.
* [Anaconda](https://www.anaconda.com/download) : Pensez-y comme à un sundae tout garni. Il vient avec de nombreuses bibliothèques scientifiques préinstallées, idéal pour l’analyse de données et le machine learning.
* [Miniconda](https://docs.anaconda.com/miniconda/miniconda-install/) : est une distribution légère de Python qui vous permet d’ajouter les librairies au besoin.
* PyPy : C’est comme 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 de vos besoins, comme choisir entre une glace simple ou un 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](https://www.jetbrains.com/pycharm/) : C’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. Cepednant, cet outil peut être assez gourmand en mémoire et en CPU.
* [Visual Studio Code](https://code.visualstudio.com/) : Gratuit, léger mais puissant, il est personnalisable avec des extensions pour Python.
* [Spyder](https://www.spyder-ide.org/) : Logiciel libre et gratuit, orienté vers les applications scientifiques.
* [Jupyter Notebooks](https://jupyter.org/) : 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 appropriées pour des grands projets et le débogage.
* Sublime Text : C’est comme un stylo élégant et rapide. Léger et rapide, il est apprécié pour sa simplicité et sa vitesse. Le choix dépend de vos besoins, que vous soyez débutant ou développeur chevronné. L’important est de trouver l’éditeur qui vous convient le mieux pour coder confortablement.

## 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 librairies. Voici quelques bonnes pratiques à suivre :

1. **N’installez par la toute dernière version de Python** : installez toujours une version ou deux qui précède [la dernière version](https://www.python.org/downloads/). 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. Il est recommandé d’installer 1 ou 2 version antérieure, par exemple si 3.13 est la version plus récente, installer plutôt la version 3.11.
2. **N’utilisez pas de version obsolète de Python** : cela peut sembler contradictoire avec le point 1 mais c’est l’excès inverse. Si vous utilisez une version trop ancienne alors toutes vos librairies vont cessez d’évoluer et peuvent devenir obsolète.
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 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](https://www.anaconda.com/download) ou [Miniconda](https://docs.anaconda.com/miniconda/miniconda-install/), dans ce cas la commande conda est utilisée pour créer un environnement test avec Python 3.10:

conda env -n test python=3.10  
conda activate test

1. **Option 2** : vous utilisez [CPython](https://www.python.org/downloads/)

conda env -n test python=3.10  
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](https://colab.google/). Si vous souhaitez faire fonctionner ces notebook localement, vous pouvez installer un environnement local avec un serveur [Jupyter](https://jupyterlab.readthedocs.io/en/stable/getting_started/starting.html). Il suffit de suivre les étapes suivantes: 1. Installer WSL2 sous [Windows](https://learn.microsoft.com/en-us/windows/wsl/install) 2. Installer [vscode](https://code.visualstudio.com/docs/setup/windows) 3. Installer [Miniconda](https://docs.anaconda.com/miniconda/install/#quick-command-line-install) 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 TraitementImagesPythonVol1. 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: La librairie NumPy est le fondement de nombreuses librairies scientifiques (d’après (Harris 2020)). |

## 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 (mutables), 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 (immutables).
* Permettent les doublons.
* Souvent utilisé 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 chcune 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

**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 :
  + tmap est certainement le meilleur *package* pour la cartographie.
  + ggplot2 pour construire des graphiques.

## 1.7 Quiz de révision du chapitre

## 1.8 Cahier de révision (notebook)

# 2. Importation et manipulation de données spatiales

## 2.1 :rocket: Préambule

Assurez-vous de lire ce préambule avant d’exécutez le reste du notebook. ### :dart: 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:

### 2.1.1 Librairies

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

* [SciPy -](https://scipy.org/)
* [NumPy -](https://numpy.org/)
* [opencv-python · PyPI](https://pypi.org/project/opencv-python/)
* [scikit-image](https://scikit-image.org/)
* [Rasterio](https://rasterio.readthedocs.io/en/stable/)
* [Xarray](https://docs.xarray.dev/en/stable/)
* [rioxarray](https://corteva.github.io/rioxarray/stable/index.html)

Dans l’environnement Google Colab, seul rioxarray et gdal doivent être installé:

!apt-get update  
!apt-get install gdal-bin libgdal-dev  
!pip install -q rioxarray

Vérifier les importations:

import numpy as np  
import rioxarray as rxr  
from scipy import signal  
import xarray as xr  
import xrscipy  
import matplotlib.pyplot as plt

### 2.1.2 Données

Nous allons utilisés ces images dans ce chapitre:

!wget https://github.com/sfoucher/TraitementImagesPythonVol1/raw/refs/heads/main/data/chapitre01/subset\_RGBNIR\_of\_S2A\_MSIL2A\_20240625T153941\_N0510\_R011\_T18TYR\_20240625T221903.tif -O RGBNIR\_of\_S2A.tif  
!wget https://github.com/sfoucher/opengeos-data/raw/refs/heads/main/raster/landsat7.tif -O landsat7.tif  
!wget https://github.com/sfoucher/opengeos-data/raw/refs/heads/main/images/berkeley.jpg -O berkeley.jpg  
!wget https://raw.githubusercontent.com/sfoucher/TraitementImagesPythonVol1/refs/heads/main/images/modis-aqua.PNG -O modis-aqua.PNG

Vérifiez que vous êtes capable de les lire :

with rxr.open\_rasterio('berkeley.jpg', mask\_and\_scale= True) as img\_rgb:  
 print(img\_rgb)  
with rxr.open\_rasterio('RGBNIR\_of\_S2A.tif', mask\_and\_scale= True) as img\_rgbnir:  
 print(img\_rgbnir)

## 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 programme Python. L’imagerie satellite présente certains défis notamment en raison de la taille parfois très importante des images. Il existe maintenant certaines librairies, comme 🔖[Xarray](https://docs.xarray.dev/en/stable/), qui on cherchées à optimiser la lecture et l’écriture de grandes images. Il est donc conseiller de toujours garder un oeil sur l’espace mémoire occupé par les variables Python reliées aux images. La librairie principale en géomatique qui va nous permettre d’importer (et d’exporter) de l’imagerie est la librairie [GDAL](https://gdal.org) 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](https://numpy.org/) (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. **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.

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

### 2.2.1 Formats des images

Il existe maintenant 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](https://gdal.org/en/latest/drivers/raster/index.html) pour une liste complète).

On peut distinguer deux grandes familles de format: 1. Les formats de type **RVB** issus de l’imagerie numérique grand publique comme 🔖[JPEG](https://gdal.org/en/latest/drivers/raster/jpeg.html#raster-jpeg), [png](https://gdal.org/en/latest/drivers/raster/png.html#raster-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 bit). 2. **Les géo-formats** issus des domaines scientifiques ou techniques comme GeoTIFF, HDF5, etc. qui peuvent inclure plus que trois bandes et des dynamiques plus élevées (16 bit ou même float).

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

**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, ces formats 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 librairies Rasterio, [PIL](https://he-arc.github.io/livre-python/pillow/index.html) ou [OpenCV](https://docs.opencv.org/4.10.0/index.html).

##### 2.2.1.1.1 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.

|  |
| --- |
| Bloc de code 2.1: Lecture d’une image en format PNG avec PIL  from PIL import Image img = Image.open('modis-aqua.PNG') img |

##### 2.2.1.1.2 Lecture avec la librairie OpenCV

La librairie [OpenCV](https://docs.opencv.org/4.10.0/index.html) est aussi très populaire en vision par ordinateur. La fonction imread donne directement un objet de type NumPy en sortie.

|  |
| --- |
| Bloc de code 2.2: Lecture d’une image en format PNG avec OpenCV  import cv2 img = cv2.imread('modis-aqua.PNG') img |

##### 2.2.1.1.3 Lecture avec la librairie RasterIO

Rien ne nous empêche de lire une image de format RVB avec [RasterIO](https://rasterio.readthedocs.io/en/stable/) comme décrit dans ([bloc 2.3](#lst-lecturerasterioPNG)). Vous noterez cependant les avertissements concernant l’absence de géoréférence pour ce type d’image.

|  |
| --- |
| Bloc de code 2.3: Lecture d’une image en format PNG avec OpenCV  import rasterio img= rasterio.open('modis-aqua.PNG') img |

#### 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](https://www.earthdata.nasa.gov/esdis/esco/standards-and-practices/geotiff) 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](https://gdal.org) (*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.

##### 2.2.1.2.1 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](http://cogeo.org/)), 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 le cloud.

### 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](https://rasterio.readthedocs.io/en/stable/cli.html#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.

|  |
| --- |
| Bloc de code 2.4: Collecte d’information sur une image avec gdal  !gdalinfo RGBNIR\_of\_S2A.tif |

Le plus simple est d’utiliser la fonction rio info:

|  |
| --- |
| Bloc de code 2.5: Collecte d’information sur une image avec rasterio  !rio info RGBNIR\_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](https://numpy.org/). Cette librairie contient un nombre très important de fonctionnalités couvrant l’algèbre linéaires, les statistiques, etc. et constitue la fondation de nombreuses librairies (voir ([figure 2.1](#fig-naturenumpy1)))

|  |
| --- |
| Figure 2.1: La librairie NumPy est le fondement de nombreuses librairies scientifiques (d’après (Harris 2020)). |

### 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 ([bloc 2.6](#lst-numpyshape)), le résultat nous informe que la matrice a 3 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](#tbl-numpytype)) (voir aussi 🔖[Data types — NumPy v2.1 Manual](https://numpy.org/doc/stable/user/basics.types.html)).

|  |
| --- |
| Bloc de code 2.6: Lecture d’une image en format PNG avec OpenCV  import cv2 img = cv2.imread('modis-aqua.PNG') print('Nombre de dimensions: ',img.ndim) print('Dimensions de la matrice: ',img.shape) print('Type de la donnée: ',img.dtype) |

|  |
| --- |
| Tableau 2.1: Type de données de NumPy  from IPython.display import Markdown from tabulate import tabulate table = [["uint8", "char", 8, 0, 255],  ["int8", "signed char", 8, -127, +128],  ["uint16", "unsigned short", 16, 0, -32768, +32767],  ["int16", "short", 16, 0, 655355]] Markdown(tabulate(table, headers=["dtype", "Nom", "Taille (bits)", "Min", "Max"], tablefmt="pipe")) |

**Les différents types de données en dans NumPy**

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

### 2.3.3 Découpage et indexation de la matrice

L’indexation et le découpage 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](#fig-naturenumpy2)). 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](https://docs.python.org/fr/3/tutorial/introduction.html#lists) en Python. Les indices commencent à 0 et se termine à la taille-1 de l’axe considéré.

|  |
| --- |
| 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.

import cv2  
img = cv2.imread('modis-aqua.PNG')  
img\_col = img[:,1,:]  
print('Nombre de dimensions: ',img\_col.ndim)  
print('Dimensions de la matrice: ',img\_col.shape)

**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 Exemple 1: calcul d’un rapport de bande

#### 2.3.3.2 Exemple 2: application d’un filtrage spatial

### 2.3.4 Mosaïquage, masquage et découpage

#### 2.3.4.1 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 raison 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 dûs 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.

### 2.3.5 Changement de projection cartographique

### 2.3.6 Recalage d’images et co-registration

## 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](https://docs.xarray.dev/en/latest/getting-started-guide/why-xarray.html) 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 aux 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.



netcdf, xarray, GRIB.

Données météos, exemple avec SWOT.

## 2.5 Importation de données vectorielles

### 2.5.1 Importation d’un fichier *shapefile*

### 2.5.2 Importation d’une couche dans un *GeoPackage*

### 2.5.3 Importation d’une couche dans une *geodatabase* d’ESRI

### 2.5.4 Importation d’un fichier *shapefile*

## 2.6 Manipulation de données vectorielles

### 2.6.1 Requêtes attributaires

## 2.7 Quiz de révision du chapitre

## 2.8 Exercices de révision

# 3. Réhaussement et visualisation d’images

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

## 3.1 :rocket: Préambule

### 3.1.1 :dart: 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:

### 3.1.2 Librairies

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

* [SciPy -](https://scipy.org/)
* [NumPy -](https://numpy.org/)
* [opencv-python · PyPI](https://pypi.org/project/opencv-python/)
* [scikit-image](https://scikit-image.org/)
* [Rasterio](https://rasterio.readthedocs.io/en/stable/)
* [Xarray](https://docs.xarray.dev/en/stable/)
* [rioxarray](https://corteva.github.io/rioxarray/stable/index.html)

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

%%capture  
!apt-get update  
!apt-get install gdal-bin libgdal-dev  
!pip install -q rioxarray  
!pip install -qU "geemap[workshop]"

Vérifier les importations:

import numpy as np  
import rioxarray as rxr  
from scipy import signal  
import xarray as xr  
import xrscipy  
import matplotlib.pyplot as plt

### 3.1.3 Données

Nous allons utilisez les images suivantes dans ce chapitre:

%%capture  
!wget https://github.com/sfoucher/TraitementImagesPythonVol1/raw/refs/heads/main/data/chapitre01/subset\_RGBNIR\_of\_S2A\_MSIL2A\_20240625T153941\_N0510\_R011\_T18TYR\_20240625T221903.tif -O RGBNIR\_of\_S2A.tif  
!wget https://github.com/sfoucher/opengeos-data/raw/refs/heads/main/raster/landsat7.tif -O landsat7.tif  
!wget https://github.com/sfoucher/opengeos-data/raw/refs/heads/main/images/berkeley.jpg -O berkeley.jpg  
!wget https://github.com/sfoucher/TraitementImagesPythonVol1/raw/refs/heads/main/data/chapitre01/subset\_0\_of\_S1A\_split\_NR\_Cal\_Deb\_ML\_Spk\_SRGR.tif -O SAR.tif

Vérifiez que vous êtes capable de les lire :

with rxr.open\_rasterio('berkeley.jpg', mask\_and\_scale= True) as img\_rgb:  
 print(img\_rgb)  
with rxr.open\_rasterio('RGBNIR\_of\_S2A.tif', mask\_and\_scale= True) as img\_rgbnir:  
 print(img\_rgbnir)  
with rxr.open\_rasterio('subset\_0\_of\_S1A\_split\_NR\_Cal\_Deb\_ML\_Spk\_SRGR.tif', mask\_and\_scale= True) as img\_SAR:  
 print(img\_SAR)

## 3.2 Réhaussements visuels

Le but du 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 est plutôt appliquée dynamiquement à l’affichage pour des fins d’inspection visuelle.

### 3.2.1 Statistiques d’une image

On peut considérer un ensemble de statistique globales pour chacune des bandes d’une image: - valeurs minimales et maximales - valeurs moyennes, médianes et quantiles - écart-types, skewness et kurtosis Ces statistiques doivent être calculées pour chaque bande d’une image multispectrale.

En ligne de commande, gdalinfo permet d’interroger rapidement un fichier image pour connaitre les statistiques de base:

|  |
| --- |
| Bloc de code 3.1: Statistiques d’une image avec gdal  !gdalinfo -stats landsat7.tif |

Les librairies de base comme xarray et numpy peuvent facilement produire des statistiques comme avec la fonction [stats](https://rasterio.readthedocs.io/en/stable/api/rasterio.io.html#rasterio.io.BufferedDatasetWriter.stats):

import rasterio as rio  
import numpy as np  
with rio.open('landsat7.tif') as src:  
 stats= src.stats()  
 print(stats)

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

import rioxarray as riox  
with riox.open\_rasterio('landsat7.tif', masked= True) as src:  
 print(src)  
quantiles = src.quantile(dim=['x','y'], q=[.025,.25,.5,.75,.975])  
quantiles

#### 3.2.1.1 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 d’une valeur du nombre de *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():

import numpy as np  
array = np.random.randint(0,10,100) # 100 valeurs aléatoires entre 0 et 10  
hist, bin\_limites = np.histogram(array, density=True)  
print('valeurs :',hist)  
print(';imites :',bin\_limites)

Le calcul se fait avec 10 intervalles par défaut.

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

##### 3.2.1.1.1 Visualisation des histogrammes

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

import rasterio as rio  
from rasterio.plot import show\_hist  
with rio.open('RGBNIR\_of\_S2A.tif') as src:  
 show\_hist(src, bins=50, lw=0.0, stacked=False, alpha=0.3,histtype='stepfilled', title="Histogram")

### 3.2.2 Réhaussements linéaires

Le réhaussement linéaire d’une image est la forme la plus simple de réhaussement, elle consiste 1) à optimiser les valeurs des pixels d’une image afin de maximiser la dynamique disponibles à l’affichage, ou 2) changer le format de stockage des valeurs (e.g. de 8 bit à 16 bit):

Par cette opération, on passe de la dynamique de départ () vers la dynamique cible (). 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 et . 2. **Préservation de la valeur de no data** : il faut faire attention à la valeur dans le cas d’une valeur présente pour *no\_data*. Par exemple, si *no\_data=0* alors il faut s’assurer que . 3. **Précision du calcul** : si possible réaliser la division ci-dessus en format *float*

### 3.2.3 Réhaussements non linéaires

Calcul d’histogrammes, étirement, égalisation, styling

### 3.2.4 Composés couleurs

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. On peut trouver des variantes comme le système HSV (*Hue-Saturation-Value*) utilisé en encodage de données vidéos.

## 3.3 Visualisation

### 3.3.1 Visualisation en Python

Il faut d’entrée mentionner que Python n’est pas vraiment fait pour visualiser de la donnée de grande taille, le niveau d’interactivité est aussi plus limité. Néanmoins, il est possible de visualiser de petites images avec la librairie Matplotlib.

### 3.3.2 Outils de visualisation

Il existe plusieurs outils gratuits de visualisation d’une image satellite, on peut mentionner les deux principaux: - QGIS - ESA Snap

### 3.3.3 Visualisation sur le Web

Une des meilleures pratiques pour visualiser une image de grande taille est d’utiliser un service de type Web Mapping Service (WMS). Cependant, type de service nécessite une architecture client-serveur qui est plus complexe à mettre en place.

Google Earth Engine offre des moyens de visualiser de la donnée locale: 🔖 *Working with Local Geospatial Data* — via [17. Geemap — Introduction to GIS Programming](https://geog-312.gishub.org/book/geospatial/geemap.html#working-with-local-geospatial-data)

🔖 \_\_ — via [data/raster at main · opengeos/data](https://github.com/opengeos/data/tree/main/raster)

### 3.3.4 Visualisation 3D

drapper une image satellite sur un DEM

## 3.4 Quiz de révision du chapitre

## 3.5 Exercices de révision

# 4. Transformations spectrales

## 4.1 :rocket: Préambule

Assurez-vous de lire ce préambule avant d’exécutez le reste du notebook. ### :dart: 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:

### 4.1.1 Librairies

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

* [SciPy -](https://scipy.org/)
* [NumPy -](https://numpy.org/)
* [opencv-python · PyPI](https://pypi.org/project/opencv-python/)
* [scikit-image](https://scikit-image.org/)
* [Rasterio](https://rasterio.readthedocs.io/en/stable/)
* [Xarray](https://docs.xarray.dev/en/stable/)
* [rioxarray](https://corteva.github.io/rioxarray/stable/index.html)

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

%%capture  
!apt-get update  
!apt-get install gdal-bin libgdal-dev  
!pip install -q rioxarray  
!pip install -qU "geemap[workshop]"

Vérifier les importations:

import numpy as np  
import rioxarray as rxr  
from scipy import signal  
import xarray as xr  
import xrscipy  
import matplotlib.pyplot as plt

### 4.1.2 Images utilisées

Nous allons utilisez les images suivantes dans ce chapitre:

%%capture  
!wget https://github.com/sfoucher/TraitementImagesPythonVol1/raw/refs/heads/main/data/chapitre01/subset\_RGBNIR\_of\_S2A\_MSIL2A\_20240625T153941\_N0510\_R011\_T18TYR\_20240625T221903.tif -O RGBNIR\_of\_S2A.tif  
!wget https://github.com/sfoucher/opengeos-data/raw/refs/heads/main/raster/landsat7.tif -O landsat7.tif  
!wget https://github.com/sfoucher/opengeos-data/raw/refs/heads/main/images/berkeley.jpg -O berkeley.jpg  
!wget https://github.com/sfoucher/TraitementImagesPythonVol1/raw/refs/heads/main/data/chapitre01/subset\_1\_of\_S2A\_MSIL2A\_20240625T153941\_N0510\_R011\_T18TYR\_20240625T221903\_resampled.tif -O sentinel2.tif

Vérifiez que vous êtes capable de les lire :

with rxr.open\_rasterio('berkeley.jpg', mask\_and\_scale= True) as img\_rgb:  
 print(img\_rgb)  
with rxr.open\_rasterio('RGBNIR\_of\_S2A.tif', mask\_and\_scale= True) as img\_rgbnir:  
 print(img\_rgbnir)  
with rxr.open\_rasterio('sentinel2.tif', mask\_and\_scale= True) as img\_s2:  
 print(img\_s2)

## 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 spatiale 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.

## 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 du spectre comme des pentes, des gradients, etc.

## 4.4 Réduction de dimension

### 4.4.1 Analyses en composantes principales

## 4.5 Exercices de révision

# 5. Transformations spatiales

## 5.1 :rocket: Préambule

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

### 5.1.1 :dart: 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:

### 5.1.2 Librairies

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

* [SciPy -](https://scipy.org/)
* [NumPy -](https://numpy.org/)
* [opencv-python · PyPI](https://pypi.org/project/opencv-python/)
* [scikit-image](https://scikit-image.org/)
* [Rasterio](https://rasterio.readthedocs.io/en/stable/)
* [Xarray](https://docs.xarray.dev/en/stable/)
* [rioxarray](https://corteva.github.io/rioxarray/stable/index.html)

Dans l’environnement Google Colab, seul rioxarray doit être installés:

%%capture  
!pip install -qU matplotlib rioxarray xrscipy

Vérifier les importations:

import numpy as np  
import rioxarray as rxr  
from scipy import signal  
import xarray as xr  
import xrscipy  
import matplotlib.pyplot as plt

### 5.1.3 Images utilisées

Nous allons utilisez les images suivantes dans ce chapitre:

%%capture  
!wget https://github.com/sfoucher/TraitementImagesPythonVol1/raw/refs/heads/main/data/chapitre01/subset\_RGBNIR\_of\_S2A\_MSIL2A\_20240625T153941\_N0510\_R011\_T18TYR\_20240625T221903.tif -O RGBNIR\_of\_S2A.tif  
!wget https://github.com/sfoucher/opengeos-data/raw/refs/heads/main/raster/landsat7.tif -O landsat7.tif  
!wget https://github.com/sfoucher/opengeos-data/raw/refs/heads/main/images/berkeley.jpg -O berkeley.jpg  
!wget https://github.com/sfoucher/TraitementImagesPythonVol1/raw/refs/heads/main/data/chapitre01/subset\_0\_of\_S1A\_split\_NR\_Cal\_Deb\_ML\_Spk\_SRGR.tif -O SAR.tif

Vérifiez que vous êtes capable de les lire :

with rxr.open\_rasterio('berkeley.jpg', mask\_and\_scale= True) as img\_rgb:  
 print(img\_rgb)  
with rxr.open\_rasterio('RGBNIR\_of\_S2A.tif', mask\_and\_scale= True) as img\_rgbnir:  
 print(img\_rgbnir)  
with rxr.open\_rasterio('SAR.tif', mask\_and\_scale= True) as img\_SAR:  
 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 modifications 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 fourniment par les capteurs sont tous sujets 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 fréquentielle. Cette transformée est complètement reversible. 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:

La transformée inverse prend une forme similaire:

Le signal d’origine est donc reconstruit à partir d’une somme de sinusoïde complexe de fréquence . Noter qu’à partir de , les sinusoïdes se répètent à un signe près et forme un miroir des composantes, la convention est lors de mettre ces composantes dans une espace négatif .

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 observations. Les puissances de Fourier sont affichés 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 cycles/secondes. La fréquence d’échantillonnage du signal a aussi beaucoup d’importance aussi et doit être au moins a deux fois la plus haute fréquence observée (ici ) sinon un phénomène de repliement appelé aliasing sera observé.

import math  
Fs= 2.0  
Ts= 1/Fs  
N= 16  
arr = xr.DataArray(np.sin(2\*math.pi\*np.arange(0,N,Ts)\*4/16),  
 dims=('x'), coords={'x': np.arange(0,N,Ts)})  
fourier = np.fft.fft(arr)  
freq = np.fft.fftfreq(fourier.size, d=Ts)  
fourier = xr.DataArray(fourier,  
 dims=('f'), coords={'f': freq})  
  
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(10, 4))  
plt.subplot(1, 2, 1)  
arr.plot.line(color='red', linestyle='dashed', marker='o', markerfacecolor='blue')  
axes[0].set\_title("Signal périodique")  
plt.subplot(1, 2, 2)  
np.abs(fourier).plot.line(color='red', linestyle='dashed', marker='o', markerfacecolor='blue')  
axes[1].set\_title("Composantes de Fourier (amplitude)")  
plt.show()

### 5.2.2 Filtrage fréquentielle

Un filtrage fréquentielle consiste à modifier le spectre de Fourier afin d’éliminer ou de réduire certaines composantes fréquentielles. On peut distinguer trois grandes catégories de filtres fréquentielles:

1. Les filtres passe-bas qui ne préservent que les basses fréquences pour, par exemple, lisser une image.
2. Les filtres passe-haut 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 bandes particulières.

La librairie Scipy contient différents filtres fréquentielles. Notez, qu’un filtrage fréquentielle est une simple multiplication de la réponse du filtre par les composantes fréquentielles du signal à filtrer :

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

from scipy import ndimage  
import numpy.fft  
  
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 4))  
input\_ = numpy.fft.fft2(img\_rgb.to\_numpy())   
result = [ndimage.fourier\_gaussian(input\_[b], sigma=4) for b in range(3)] # on filtre chaque bande avec un filtre Gaussien  
result = numpy.fft.ifft2(result)  
ax1.imshow(img\_rgb.to\_numpy().transpose(1, 2, 0).astype('uint8'))  
ax1.set\_title('Originale')  
ax2.imshow(result.real.transpose(1, 2, 0).astype('uint8')) # La partie imaginaire n'est pas utile ici  
ax2.set\_title('Filtrage Gaussien')  
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 fréquentielle du signal. Ceci peut se produire lorsque vous sous-échantillonner fortement une image avec un facteur de décimation (par exemple 1 pixel sur 2). En prenant un pixel sur 2, on réduit la fréquence d’échantillonnage d’un facteur 2 ce qui nous impose de réduire le contenu fréquentielle de l’image et donc les fréquences maximales de l’image. L’image présente alors un aspect faussement texturée avec beaucoup de haute fréquences:

fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(10, 4))  
plt.subplot(1, 2, 1)  
img\_rgb.astype('int').plot.imshow(rgb="band")  
axes[0].set\_title("Originale")  
plt.subplot(1, 2, 2)  
img\_rgb[:,::4,::4].astype('int').plot.imshow(rgb="band")  
axes[1].set\_title("Décimée par un facteur 4")  
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:

from scipy.ndimage import gaussian\_filter  
  
q= 4  
sigma= q\*1.1774/math.pi  
arr = xr.DataArray(gaussian\_filter(img\_rgb.to\_numpy(), sigma= (0,sigma,sigma)), dims=('band',"y", "x"), coords= {'x': img\_rgb.coords['x'], 'y': img\_rgb.coords['y'], 'spatial\_ref': 0})  
  
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(10, 4))  
plt.subplot(1, 2, 1)  
img\_rgb.astype('int').plot.imshow(rgb="band")  
axes[0].set\_title("Originale")  
plt.subplot(1, 2, 2)  
arr[:,::q,::q].astype('int').plot.imshow(rgb="band")  
axes[1].set\_title("Décimée par un facteur 4")  
plt.show()

import xrscipy.signal as dsp  
  
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(10, 4))  
plt.subplot(1, 2, 1)  
img\_rgb.astype('int').plot.imshow(rgb="band")  
axes[0].set\_title("Originale")  
plt.subplot(1, 2, 2)  
dsp.decimate(img\_rgb, q=4, dim='x').astype('int').plot.imshow(rgb="band")  
axes[1].set\_title("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 dériver 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, on peut rassembler ces filtres en quatre grandes catégories:

1. Le filtrage peut-être global ou local, c.à.d prendre en compte toute l’image pour filtrer (ex: filtrage par Fourier) ou seulement localement 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-échelles

La librairie Scipy ([Multidimensional image processing (scipy.ndimage)](https://docs.scipy.org/doc/scipy/reference/ndimage.html)) 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 impaire (3,5, etc.) afin de définir une position centrale et une fenêtre symétrique.

|  |
| --- |
| Note |
| Mettre une figure ici |

Le filtre le plus simple est certainement le filtre moyen qui consiste à appliquer le même poids uniforme dans la fenêtre glissante.

En python, on dispose des fonctions rolling et sliding\_window définis dans la librairie numpy. Par exemple pour le cas du filtre moyen on peut construire une nouvelle vue de l’image avec deux nouvelles dimensions x\_win et y\_win:

import rioxarray as rxr  
rolling\_win = img\_rgb.rolling(x=5, y=5, min\_periods= 3, center= True).construct(x="x\_win", y="y\_win", keep\_attrs= True)  
print(rolling\_win[0,0,1,...])  
print(rolling\_win.shape)

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 moyenne peut être appliqué.

filtre\_moyen= rolling\_win.mean(dim= ['x\_win', 'y\_win'], skipna= True)  
fig, ax = plt.subplots(nrows=1, ncols=1, figsize=(8, 4))  
filtre\_moyen.astype('int').plot.imshow(rgb="band")  
ax.set\_title("Filtre moyen 5x5")

|  |
| --- |
| Note |
| Filtre de Sobel, filtre Prewitt |

#### 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 fréquentielle. Prenons l’exemple du filtre de Scharr (Jahne et S. 1999), ce filtre permet de détecter les contours horizontaux et verticaux:

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

scharr = np.array([[ -3-3j, 0-10j, +3 -3j],  
 [-10+0j, 0+ 0j, +10 +0j],  
 [ -3+3j, 0+10j, +3 +3j]]) # Gx + j\*Gy  
print(img\_rgb.isel(band=0).shape)  
grad = signal.convolve2d(img\_rgb.isel(band=0), scharr, boundary='symm', mode='same')  
# on reconstruit un xarray à partir du résultat:  
arr = xr.DataArray(np.abs(grad), dims=("y", "x"), coords= {'x': img\_rgb.coords['x'], 'y': img\_rgb.coords['y'], 'spatial\_ref': 0})  
print(arr)  
fig, ax = plt.subplots(nrows=1, ncols=1, figsize=(8, 4))  
arr.plot.imshow()  
ax.set\_title("Amplitude du filtre de Scharr")

##### 5.3.1.1.1 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, prolonger l’image de quelques lignes et colonnes avec des valeurs mirroirs ou constantes.

#### 5.3.1.2 Filtrage par une couche convolutionnelle

|  |
| --- |
| Important |
| Cette section nécessite la librairie Pytorch avec un GPU et ne fonctionnera que sur Colab. |

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:

import torch  
import torch.nn as nn  
import numpy as np  
import matplotlib.pyplot as plt  
normalized\_img= torch.tensor(img\_rgb.to\_numpy())  
nchannels= normalized\_img.size()[0] # nombre de canaux de l'image  
  
# Define a conv2d layer  
conv\_layer = nn.Conv2d(in\_channels= nchannels, out\_channels=2, kernel\_size=3, padding=1, stride=1, dilation= 1)  
  
# Filtre de Sobel  
sobel\_x = np.array([[-3, 0, 3], [-10, 0, 10], [-3, 0, 3]])  
sobel\_y = np.array([[-3, -10, -3], [0, 0, 0], [3, 10, 3]])  
  
kernel = np.stack([sobel\_x, sobel\_y])  
kernel = kernel.reshape(2, 1, 3, 3)  
  
kernel = np.tile(kernel,(1,nchannels,1,1))  
print(kernel.shape)  
kernel = torch.as\_tensor(kernel,dtype=torch.float32)  
conv\_layer.weight = nn.Parameter(kernel)  
conv\_layer.bias = nn.Parameter(torch.zeros(2,))  
  
input= normalized\_img.unsqueeze(0) # il faut ajouter une dimension pour le nombre d'échantillons  
print(input.shape)  
# Visualize the filters  
fig, axs = plt.subplots(1, 2, figsize=(8, 5))  
for i in range(2):  
 axs[i].imshow(conv\_layer.weight.data.numpy()[i, 0])  
 axs[i].set\_title(f'Filtre {i+1}')  
plt.show()

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

import torch  
import matplotlib.pyplot as plt  
  
output = conv\_layer(input)  
print(f'Image (BxCxHxW): {input.shape}')  
print(f'Sortie (BxFxHxW): {output.shape}')  
  
fig, axs = plt.subplots(1, 2, figsize=(20, 5))  
for i in range(2):  
 axs[i].imshow(output.detach().data.numpy()[0,i], vmin=-5000, vmax=5000, cmap= 'gray')  
 axs[i].set\_title(f'Filtrage {i+1}')  
plt.show()

### 5.3.2 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, remarquée que l’image est affichée en dB en appliquant la fonction log10.

print(img\_SAR.rio.resolution())  
print(img\_SAR.rio.crs)  
fig, axs = plt.subplots(1, 1, figsize=(6, 4))  
xr.ufuncs.log10(img\_SAR.sel(band=1).drop("band")).plot()  
axs.set\_title("Image SAR Sentinel-1 (dB)")

Un des filtres les plus simples pour réduire le bruit est d’appliquer un filtre moyenne, par exemple un ci dessous:

rolling\_win = img\_SAR.sel(band=2).rolling(x=5, y=5, min\_periods= 3, center= True).construct(x="x\_win", y="y\_win", keep\_attrs= True)  
filtre\_moyen= rolling\_win.mean(dim= ['x\_win', 'y\_win'], skipna= True)  
fig, axs = plt.subplots(1, 1, figsize=(6, 4))  
xr.ufuncs.log10(filtre\_moyen).plot.imshow()  
axs.set\_title("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 dans sa forme la plus simple:

Ainsi si la variance locale est élevée s’approche de préservant ainsi les détails de l’image sinon l’image moyenne est appliquée.

rolling\_win = img\_SAR.sel(band=2).rolling(x=5, y=5, min\_periods= 3, center= True).construct(x="x\_win", y="y\_win", keep\_attrs= True)  
filtre\_moyen= rolling\_win.mean(dim= ['x\_win', 'y\_win'], skipna= True)  
ecart\_type= rolling\_win.std(dim= ['x\_win', 'y\_win'], skipna= True)  
cv= ecart\_type/filtre\_moyen  
ponderation = (cv - 0.25) / cv  
  
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(12, 4))  
plt.subplot(1, 2, 1)  
cv.plot.imshow( vmin=0, vmax=2)  
axes[0].set\_title("CV")  
plt.subplot(1, 2, 2)  
ponderation.plot.imshow( vmin=0, vmax=1)   
axes[1].set\_title("Pondération")

On zoomant sur l’image on peut clairement voir que les détails de l’image sont mieux préservés:

## 5.4 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 pxiels 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](https://scikit-image.org/docs/stable/auto_examples/segmentation/plot_segmentations.html#sphx-glr-auto-examples-segmentation-plot-segmentations-py)).

## 5.5 Vectorisation et rasterisation

## 5.6 Analyse de terrain

### 5.6.1 Élévation

### 5.6.2 Pente

### 5.6.3 Ombrage

### 5.6.4 Visibilité

## 5.7 Quiz de révision du chapitre

## 5.8 Exercices de révision

# 6. Classifications d’images supervisées

## 6.1 :rocket: Préambule

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

### 6.1.1 :dart: 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:

### 6.1.2 Librairies

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

* [SciPy -](https://scipy.org/)
* [NumPy -](https://numpy.org/)
* [opencv-python · PyPI](https://pypi.org/project/opencv-python/)
* [scikit-image](https://scikit-image.org/)
* [Rasterio](https://rasterio.readthedocs.io/en/stable/)
* [Xarray](https://docs.xarray.dev/en/stable/)
* [rioxarray](https://corteva.github.io/rioxarray/stable/index.html)

Dans l’environnement Google Colab, seul rioxarray doit être installés:

%%capture  
!pip install -qU matplotlib rioxarray xrscipy

Vérifier les importations:

import numpy as np  
import rioxarray as rxr  
from scipy import signal  
import xarray as xr  
import xrscipy  
import matplotlib.pyplot as plt

### 6.1.3 Images utilisées

Nous allons utilisez les images suivantes dans ce chapitre:

%%capture  
!wget https://github.com/sfoucher/TraitementImagesPythonVol1/raw/refs/heads/main/data/chapitre01/subset\_RGBNIR\_of\_S2A\_MSIL2A\_20240625T153941\_N0510\_R011\_T18TYR\_20240625T221903.tif -O RGBNIR\_of\_S2A.tif  
!wget https://github.com/sfoucher/opengeos-data/raw/refs/heads/main/raster/landsat7.tif -O landsat7.tif  
!wget https://github.com/sfoucher/opengeos-data/raw/refs/heads/main/images/berkeley.jpg -O berkeley.jpg  
!wget https://github.com/sfoucher/TraitementImagesPythonVol1/raw/refs/heads/main/data/chapitre01/subset\_0\_of\_S1A\_split\_NR\_Cal\_Deb\_ML\_Spk\_SRGR.tif -O SAR.tif

Vérifiez que vous êtes capable de les lire :

with rxr.open\_rasterio('berkeley.jpg', mask\_and\_scale= True) as img\_rgb:  
 print(img\_rgb)  
with rxr.open\_rasterio('RGBNIR\_of\_S2A.tif', mask\_and\_scale= True) as img\_rgbnir:  
 print(img\_rgbnir)  
with rxr.open\_rasterio('SAR.tif', mask\_and\_scale= True) as img\_SAR:  
 print(img\_SAR)

## 6.2 Classification d’images pixel par pixel

### 6.2.1 Parallélépipède

### 6.2.2 Méthodes paramétriques

### 6.2.3 Méthodes non paramétriques

### 6.2.4 SVEM, réseaux de neurones, forêts aléatoires

## 6.3 Segmentation d’images

### 6.3.1 Classification objet

### 6.3.2 Approches par arbre (BPT, etc.)

## 6.4 Quiz de révision du chapitre

## 6.5 Exercices de révision

# 7. Classifications d’images non supervisées

## 7.1 :rocket: Préambule

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

### 7.1.1 :dart: 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:

### 7.1.2 Librairies

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

* [SciPy -](https://scipy.org/)
* [NumPy -](https://numpy.org/)
* [opencv-python · PyPI](https://pypi.org/project/opencv-python/)
* [scikit-image](https://scikit-image.org/)
* [Rasterio](https://rasterio.readthedocs.io/en/stable/)
* [Xarray](https://docs.xarray.dev/en/stable/)
* [rioxarray](https://corteva.github.io/rioxarray/stable/index.html)

Dans l’environnement Google Colab, seul rioxarray doit être installés:

%%capture  
!pip install -qU matplotlib rioxarray xrscipy

Vérifier les importations:

import numpy as np  
import rioxarray as rxr  
from scipy import signal  
import xarray as xr  
import xrscipy  
import matplotlib.pyplot as plt

### 7.1.3 Images utilisées

Nous allons utilisez les images suivantes dans ce chapitre:

%%capture  
!wget https://github.com/sfoucher/TraitementImagesPythonVol1/raw/refs/heads/main/data/chapitre01/subset\_RGBNIR\_of\_S2A\_MSIL2A\_20240625T153941\_N0510\_R011\_T18TYR\_20240625T221903.tif -O RGBNIR\_of\_S2A.tif  
!wget https://github.com/sfoucher/opengeos-data/raw/refs/heads/main/raster/landsat7.tif -O landsat7.tif  
!wget https://github.com/sfoucher/opengeos-data/raw/refs/heads/main/images/berkeley.jpg -O berkeley.jpg  
!wget https://github.com/sfoucher/TraitementImagesPythonVol1/raw/refs/heads/main/data/chapitre01/subset\_0\_of\_S1A\_split\_NR\_Cal\_Deb\_ML\_Spk\_SRGR.tif -O SAR.tif

Vérifiez que vous êtes capable de les lire :

with rxr.open\_rasterio('berkeley.jpg', mask\_and\_scale= True) as img\_rgb:  
 print(img\_rgb)  
with rxr.open\_rasterio('RGBNIR\_of\_S2A.tif', mask\_and\_scale= True) as img\_rgbnir:  
 print(img\_rgbnir)  
with rxr.open\_rasterio('SAR.tif', mask\_and\_scale= True) as img\_SAR:  
 print(img\_SAR)

## 7.2 Classifications strictes

### 7.2.1 K-means

### 7.2.2 K-mediodes

### 7.2.3 Isodata

## 7.3 Classifications floues

## 7.4 C-Means

### 7.4.1 C-Means intégrant une dimension spatiale

## 7.5 Quiz de révision du chapitre

## 7.6 Exercices de révision

**Exercice 1.** À compléter

library(sf)  
library(terra)  
# ...  
# à compléter

Correction à la **?@sec-08051**.

**Exercice 2.** À compléter

library(sf)  
library(terra)  
# ...  
# à compléter

Correction à la **?@sec-08052**.

**Exercice 3.** À compléter

library(sf)  
library(terra)  
# ...  
# à compléter

Correction à la **?@sec-08053**.

# 8. Introduction aux plateformes de mégadonnées

## 8.1 :rocket: Préambule

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

### 8.1.1 :dart: 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:

### 8.1.2 Librairies

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

* [SciPy -](https://scipy.org/)
* [NumPy -](https://numpy.org/)
* [opencv-python · PyPI](https://pypi.org/project/opencv-python/)
* [scikit-image](https://scikit-image.org/)
* [Rasterio](https://rasterio.readthedocs.io/en/stable/)
* [Xarray](https://docs.xarray.dev/en/stable/)
* [rioxarray](https://corteva.github.io/rioxarray/stable/index.html)

Dans l’environnement Google Colab, seul rioxarray doit être installés:

%%capture  
!pip install -qU matplotlib rioxarray xrscipy

Vérifier les importations:

import numpy as np  
import rioxarray as rxr  
from scipy import signal  
import xarray as xr  
import xrscipy  
import matplotlib.pyplot as plt

### 8.1.3 Images utilisées

Nous allons utilisez les images suivantes dans ce chapitre:

%%capture  
!wget https://github.com/sfoucher/TraitementImagesPythonVol1/raw/refs/heads/main/data/chapitre01/subset\_RGBNIR\_of\_S2A\_MSIL2A\_20240625T153941\_N0510\_R011\_T18TYR\_20240625T221903.tif -O RGBNIR\_of\_S2A.tif  
!wget https://github.com/sfoucher/opengeos-data/raw/refs/heads/main/raster/landsat7.tif -O landsat7.tif  
!wget https://github.com/sfoucher/opengeos-data/raw/refs/heads/main/images/berkeley.jpg -O berkeley.jpg  
!wget https://github.com/sfoucher/TraitementImagesPythonVol1/raw/refs/heads/main/data/chapitre01/subset\_0\_of\_S1A\_split\_NR\_Cal\_Deb\_ML\_Spk\_SRGR.tif -O SAR.tif

Vérifiez que vous êtes capable de les lire :

with rxr.open\_rasterio('berkeley.jpg', mask\_and\_scale= True) as img\_rgb:  
 print(img\_rgb)  
with rxr.open\_rasterio('RGBNIR\_of\_S2A.tif', mask\_and\_scale= True) as img\_rgbnir:  
 print(img\_rgbnir)  
with rxr.open\_rasterio('SAR.tif', mask\_and\_scale= True) as img\_SAR:  
 print(img\_SAR)

## 8.2 Données massives

## 8.3 Manipulation de données satellitaires avec *Google Earth Engine*

## 8.4 Quiz de révision du chapitre

## 8.5 Exercices de révision

**Exercice 1.** À compléter

library(sf)  
library(terra)  
# ...  
# à compléter

Correction à la **?@sec-08071**.

**Exercice 2.** À compléter

library(sf)  
library(terra)  
# ...  
# à compléter

Correction à la **?@sec-08072**.

**Exercice 3.** À compléter

library(sf)  
library(terra)  
# ...  
# à compléter

Correction à la **?@sec-08073**.

# Bibliographie

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.

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/>.