Introduction à l'utilisation de Google Earth Engine

Utilisation d'une plateforme dans le cloud pour une meilleure observation de la Terre

Formation donnée à l'Université de Parakou du 02 au 05 juin 2025 dans le cadre du programme OBSYDYA financé par l'Union Européenne

Donnée par Jonathan Peereman

Version du 3 juin 2025 !

Table des matières

Travaux utilisés pour réaliser ce manuel et autres liens utiles

Cette formation se base sur des données produites dans le cadre du projet OBSYDYA réunissant des institutions du Bénin (Université de Parakou, INRAB), de France (CIRAD, IRD), et de Belgique (Université de Liège). Les explications se basent sur mon expérience avec Google Earth Engine ainsi que sur des tutoriels très bien détaillés produits par le site et le livre *Cloud-Based Remote Sensing with Google Earth Engine* par J.A. Cardille (https://www.eefabook.org/), les présentations de Noel Gorelick, le GEARS Lab (https://www.gears-lab.com/) et la plateforme Spatial Thoughts (https://spatialthoughts.com/). Ces tutoriels ont été traduits et légèrement adaptés.

Il existe de nombreuses ressources en ligne pour mieux comprendre les capacités de Google Earth Engine :

Tutoriels Google Earth Engine | https://developers.google.com/earth-engine/tutorials/tutorials

Contact : jonathan.peereman@uliege.be

Introduction

Ce manuel reprend l'essentiel de la formation, cependant il peut y avoir quelques différences.

Qu'est-ce que Google Earth Engine

Google Earth Engine (GEE) est une plateforme d'analyse spatiale dans le cloud qui permet de visualiser et manipuler de nombreux jeux de données gratuitement. A travers cette plateforme, il est possible d'accéder à de larges quantités d'information, de les modifier, et d'en extraire des informations sans avoir besoin d'un ordinateur puissant : seule une connexion internet stable est nécessaire.

Les données disponibles sur GEE sont produites par de nombreuses institutions et équipes de recherche et couvrent de nombreux domaines comme des indices économiques, des délinéations d'espaces protégés, des paramètres de la végétation, ou des images radar. Plus de 900 jeux de données publics sont disponibles, et environ 100 jeux de données sont ajoutés par an d'après Michael DeWitt et Katie Friis (GeoForGood, 2022). Il y avait déjà 90 petabytes de données en 2024. Beaucoup sont mis à jour régulièrement, quand d'autres ne sont produit qu'une seule fois (par exemple, carte d'occupation du sol en 2020). Bien que GEE ait de nombreux jeux de données couvrant l'ensemble du globe, d'autres jeux de données sont focalisés sur certaines régions. Il est ainsi important de bien lire la documentation de chaque jeu de données qui correspond aux critères demandés.

Cette formation se base sur l'interface web de Google Earth Engine qui est accessible à l'adresse https://code.earthengine.google.com/ et demande de programmer en Javascript. Cependant, il existe d'autres moyens d'y accéder comme les librairies python ou R qui passent par l'API de GEE. GEE permet aussi de collaborer et de partager des scripts pour la production de chaînes de traitement, ceci se fait au sein de projets communs.

Les serveurs GEE traitent les données selon les instructions fournies par l'utilisateur pour produire des valeurs numériques / textes, ainsi que des cartes, tableaux, figures, ou données vectorielles. Bien que ces calculs soient souvent très rapides, ils se font en arrière-plan et il n'est donc pas nécessaire de rester connecter à GEE pendant que le serveur travaille. Si les instructions sont erronées ou pas claires, des messages sont renvoyés à l'utilisateur ce qui permet de corriger le code. Il est important de noter que si beaucoup de calculs sont requis, il est souvent nécessaire de partitionner ces traitements en plusieurs étapes. De plus, si beaucoup de données sont extraites / exportées, GEE les traitera en plusieurs étapes qui se dérouleront itérativement.

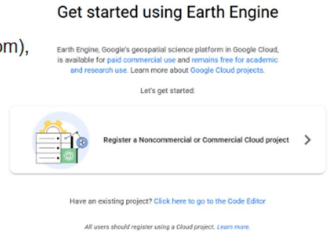
Finalement, GEE permet d'exporter des données sous diverses formes : tableaux, graphiques, cartes raster, shapefiles, ou dans des apps qui peuvent être interactives. L'export de fichiers se fait principalement à travers Google Drive, l'utilisateur doit avoir suffisamment d'espace disponible pour cela (la version gratuite de Google offre 10 Gb d'espace).

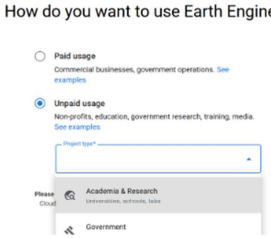
S'inscrire

L'inscription à Google Earth Engine est gratuite et simple en suivant les 10 étapes suivantes :

S'inscrire sur Google Earth Engine

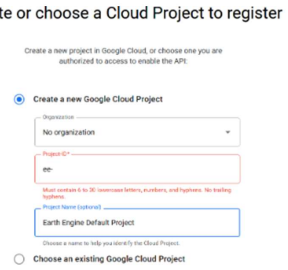
1. Avoir un compte Google (une addresse @gmail.com), sinon en créer un.
2. Sur https://code.earthengine.google.com/register, entrer l'adresse du compte Google et confirmer.
3. L'image à droite s'affiche, cliquer sur "**Register a Noncommercial or Commercial Cloud project**"

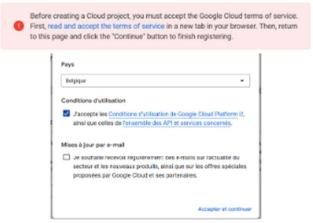




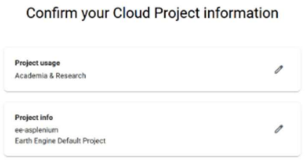
4. Sélectionner "**Unpaid usage**" avec "**Academia & Research**" comme Project type. Confirmer.

5. Un Cloud Project doit être créé (première option)
6. Choisir "**No organization**", choisir un Project ID qui correspondra au nom de votre compte et qui ne peut pas être modifié plus tard.
   Par exemple "ee-upetudiant456". Cet ID est a priori public.
   Le Project Name est optionel. Confirmer.





7. Il est possible que cette fenêtre en rouge s'affiche en bas de l'écran. Il vous est demandé d'accepter les terms of service de Google : cliquer sur le lien bleu pour ouvrir une seconde page. Ne pas fermer la page d'inscription.
8. Sur la page qui s'est ouverte, sélectionner "**J'accepte les Conditions…**" et confirmer.
9. Revenir sur la première page et confirmer à nouveau.



10. Vérifiez que vos informations soient correctes et confirmer.

Votre compte GEE est créé !

L'interface

L'interface du site GEE a quatre panneaux (Figure 1) :

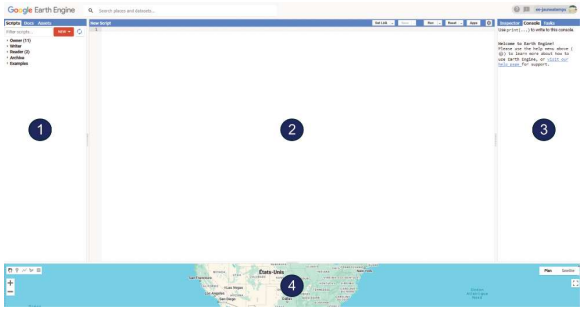


Figure 1 Capture d'écran de l'éditeur de code (code.earthengine.com) avec quatre volets : 1) scripts, documentation des fonctions, et données ; 2) éditeur ; 3) inspecteur (pour la carte), console, et tâches (importation ou exportation de données) ; 4) la carte et l'outil de création de géométries.

1. Tout à gauche, ce panneau possède trois onglets. L'onglet Scripts permet de gérer les différents scripts associés au compte Google. L'onglet Docs reprend la documentation des fonctions GEE. L'onglet Assets reprend toutes les données externes qui ont été importées par l'utilisateur sur son compte. Ce ne sont pas nécessairement les données importées par le code ouvert. Les scripts et les assets peuvent être partagés et organisés dans des dossiers.
2. L'éditeur de code, au centre, est l'endroit où le code est écrit et modifié. Il est associé à plusieurs boutons : Get Link permet de partager le code, Save sauvegarde le code, Run lance la lecture du code (Ctrl+Enter comme raccourci), Reset nettoie le script, et Apps permet d'aller vers les applications développées par l'utilisateur. Le code ne se sauvegarde pas automatiquement, il faut sauvegarder régulièrement !
3. Les différents résultats sont visibles à droite sous trois onglets : Inspector permet d'utiliser la souris et de cliquer sur divers endroits de la carte du volet (4) pour afficher la valeur des pixels. L'onglet Console retourne les erreurs et les données qui sont imprimées par l'utilisateur avec la fonction print(). L'onglet Tasks reprend les imports et exports de données demandés à GEE. Les problèmes au niveau de ces tâches y sont aussi repris.

4. La carte est affichée en bas et permet de visualiser les différentes données ainsi que de créer des points, lignes, et polygones qui peuvent être utilisés pour l'analyse.
   Les outils de la carte sont les suivant :

Déplacer la carte | Créer un point | Créer une ligne
Créer un polygone irrégulier | Créer un rectangle

Créer des polygones, points, lignes sur la carte va modifier le panneau de l'éditeur de code (2, dans la Figure 1) à travers l'ajout d'*Imports* dans un petit volet en haut de ce panneau (Figure 2).

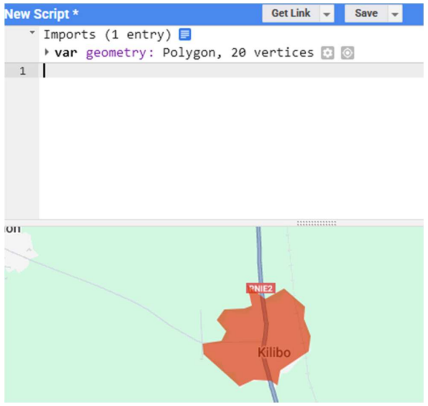


Figure 2 L'éditeur de code change lors de la création d'une géométrie avec l'ajout d'une section *Imports* où il est possible de renommer les géométries créées ou de les supprimer.

Ici la nouvelle forme est appelée *geometry* mais peut être renommée en cliquant sur son nom ou sur l'icône qui permet aussi de changer sa couleur, son type, ou de la supprimer. Il est aussi possible de supprimer la forme en survolant son nom dans ce volet, icône apparaît alors à gauche. Le dernier icone, , permet de zoomer sur la forme.

Préparer scripts (codes) et dossiers

Les scripts créés sont sauvegardés et accessibles dans l'onglet Scripts du panneau de gauche. Ils sont soit détenus par l'utilisateur (catégorie Owner), soit partagés avec l'utilisateur en mode écriture (Writer) ou lecture (Reader). Il est souhaitable de créer différents dossiers dans la catégorie Owner pour organiser ses scripts. Le bouton NEW offre ainsi la possibilité de créer des Repository ou Folder, les Repository étant le plus haut niveau et sont les contenants qui peuvent être partagés avec d'autres utilisateurs. Tous les dossiers contenus par ce repository sont dès lors partagés. Les scripts sont aussi créés à partir du bouton NEW, en choisissant l'option File.

Programmer en JavaScript

La programmation dans GEE en Javascript ne diffère pas fortement de la programmation Javascript classique, mais de nombreuses fonctions Earth Engine (commençant par *ee.*) sont disponibles et permettent de créer des objets du côté serveur. De cette façon tout le traitement des données est réalisé du côté serveur et non du côté client (ordinateur de l'utilisateur). Il est important de considérer le fait que certains codes entraînent des échanges serveur – client qui ralentissent fortement l'exécution du code. Donc, un code qui tourne uniquement du côté serveur sera préféré.

Vocabulaire

Variable : éléments du code, qui utilisent une partie de la mémoire (serveur, desktop)

- Affecter : une valeur est affectée à une variable
- Expression : ensemble de variable(s) et opérateur(s) qui renvoient vers une valeur
- Instruction : L'ensemble des expressions qui se termine par ";"
- Mot-clé : Mot reservé (*reserved word*) qui ne peut pas être utilisé pour une variable
- Objet : type de variable avec des propriétés (valeur, méthode). La documentation sur les fonctions de GEE est d'ailleurs segmentée en fonction du type d'objet considéré.
- Caractère d'échappements : //, /* */, texte qui n'est pas interprété

Créer des variables

Pour créer une variable, il est nécessaire de commencer une nouvelle ligne avec le mot « var ». Ci-dessous, la variable « b » est créée et est égale à 234.

```
var b = 234; // la variable c est créée : un nombre
var a = ee.Number(234); // variable a : objet de type nombre
var b = ee.Number(b); // converti le nombre c en objet de type
nombre
var site_1_nom = 'Foret classee de Ndali'
var site_1_nom = ee.String('Foret classee de Ndali');
print(site_1_nom);
```

La valeur des variables peut être affichée avec `print()`, cette fonction prend un argument qui est la variable dont le contenu doit être imprimé et un texte qui est facultatif. Ces deux arguments sont séparés par une virgule, et contrairement à d'autres fonctions, leur ordre n'est pas important.

```
print('la valeur de a :', a);
```

Attention à ne pas nommer une variable avec un nom contenant des espaces, caractères accentués, ou qui est un mot-clé ou le nom d'une fonction déjà existante (par exemple « print », « **var** »).

Calculs simples

Les différentes variables peuvent être utilisées pour effectuer des calculs tels que :

```
var c = a.add(b); // la variable c est créée
print(c);
```

Il n'est pas nécessaire de créer une variable à chaque étape. Par exemple :

```
print(c.subtract(a)) // soustrait a à c
```

Ainsi, de nombreuses fonctions peuvent être ajoutées pour former une instruction « complexe » :

```
print(c.subtract(a).divide(ee.Number(3).multiply(a)))
```

Les erreurs de syntaxe sont fréquentes et souvent indiquées dans l'éditeur de code par le symbole à côté de la ligne où se trouve l'erreur. Les erreurs sont montrées dans l'onglet Console lorsque le code est interprété. L'interprétation du code est arrêtée lorsqu'une erreur est détectée.

Listes et arrays

Les listes et arrays sont deux types d'objet permettant de stocker de nombreuses données sur GEE. La création de ces deux types d'objet est similaire (avec l'utilisation des crochets « [] », tel que :

```
var a = ee.Number(10);
var b = ee.Number(20);
var c = ee.Number(530);
var liste = ee.List([a, b, b, a, c]);
var array = ee.Array([a, b, b, a, c]);
print(liste); print(array);
```

Il est ainsi possible de créer une liste de listes ou un array d'arrays :

```
var liste = ee.List([[a, b, b, a, c], [a, c, c, c, a]]);
var array = ee.Array([[a, b, b, a, c], [a, c, c, c, a]]);
```

Quand une liste peut contenir des listes de différentes tailles (différent nombre d'éléments), un array ne peut contenir que des arrays de même taille. De plus, un array ne peut contenir que des valeurs numériques alors qu'une liste peut contenir du texte.

Les listes sont dynamiques et peuvent être facilement manipulées pour ajouter, modifier, retirer ou consulter des éléments tel qu'avec la fonction `add()` pour ajouter un élément ou `contains()` pour vérifier qu'une liste contient un élément :

```
var liste = ee.List([a, b, b, a, c]);
print(liste.contains(a));
print(liste.add('d'));
```

Les arrays permettent de stocker de nombreuses valeurs numériques et d'effectuer des calculs complexes (voir chapitres suivants) impliquant des matrices. La fonction `gte()` (*greater than or equal*) compare deux arrays pour identifier les éléments qui sont plus grands ou égaux aux éléments à la même position dans le deuxième array (0 si faux, 1 si vrai), comme :


```
var array1 = ee.Array([[a, b, b, a, c], [a, c, c, c, a]]);
var array_gte = array1.gte([[1, 2, 3, 11, 40], [10, 15, 45, 0,
9]]);
print(array_gte);
```

Sauvegarder le code

Le bouton  permet de sauvegarder le code qui se retrouvera ainsi dans l'onglet Scripts du volet de gauche. La sauvegarde automatique du code n'existe pas sur GEE, il faut penser à fréquemment sauvegarder. Dans l'onglet Scripts, il suffit de survoler le nom du script avec la souris pour afficher trois boutons  qui permettent respectivement de consulter les différentes versions du code, de le renommer, et de le supprimer.

Les données spatiales dans GEE

Image et ImageCollection

Les objets de type Image correspondent aux rasters traditionnels de type geotiff. En plus des multiples bandes que peut posséder une Image, cet objet possède aussi des propriétés (*properties*) qui stockent des valeurs générales pour l'objet et peuvent être assimilées aux données qui sont en général stockées dans les fichiers metadata accompagnant les rasters habituels (par exemple, date d'acquisition, pourcentage de couverture nuageuse).

GEE permet de regrouper de multiples Images dans un objet de type ImageCollection. Toutes ces images vont ainsi posséder les mêmes types de bandes et propriétés sans devoir partager la même étendue spatiale. Il est ainsi possible de filtrer une ImageCollection sur la base de ses propriétés ou de la région d'intérêt.

Feature et FeatureCollection

Un objet de type Feature correspond à un vecteur et les attributs qui sont associés au vecteur ont pour équivalent les propriétés de l'objet Feature. De plus, un Feature possède des informations concernant sa géométrie (*geometry*) dont son type (point, ligne, polygone et leurs équivalents multiples) et ses coordonnées.

De multiples Features peuvent être stockés en une FeatureCollection. Cette collection contiendra alors plusieurs Features avec des propriétés aux valeurs différentes. Comme pour une ImageCollection, il est possible de filtrer une FeatureCollection à partir de la valeur des propriétés et de l'étendue spatiale des Features.

Il est important de noter que les fonctions s'appliquant aux Image, Feature, ImageCollection, et FeatureCollection ne sont pas les mêmes.

Sources des données sur GEE

Il y a quatre principales sources de données spatiales sur GEE qui sont soit des Image, ImageCollection, Feature, ou FeatureCollection. Le catalogue Earth Engine

(https://developers.google.com/earth-engine/datasets/catalog) comprend de nombreux jeux de données fournis par diverses institutions et certains sont mis à jour quotidiennement. Il existe aussi le catalogue de la communauté Earth Engine qui n'est pas géré par Google et qui est consultable à partir d'autres sites comme https://gee-community-catalog.org/. Ces données sont en général le fruit d'études publiées dans des journaux scientifiques. Tous ces jeux de données sont gratuitement accessibles mais peuvent être associés à des restrictions comme le besoin de les citer correctement si elles sont utilisées dans un rapport ou article. Enfin, la troisième source de données sont les Assets qui sont uploadé à partir de l'onglet Assets du volet de gauche. Ces données peuvent être privées ou partagées, mais sont limitées en espace de stockage par compte GEE (250Gb et 50000 assets au maximum, en mai 2025).

Importer des données externes

Importer des données se fait directement depuis l'interface principale. Il est préférable de d'abord créer un dossier en allant dans l'onglet Assets (volet de gauche), et choisir l'option *Folder* à partir du bouton *New*. La création d'une ImageCollection suit les mêmes étapes. Les données sont ensuite importées à partir du bouton *New* et il faut sélectionner le type de donnée appropriée.

Pour une Image, choisir « GeoTIFF [...] ». Dans la nouvelle fenêtre, le bouton *Select* permet de sélectionner les fichiers à importer. Ensuite, le champ *Asset ID* permet de renseigner le nom du dossier (ou de l'ImageCollection) vers lequel les fichiers seront importés ainsi que le nom de l'Image qui sera créée à partir de ces fichiers. Par exemple, pour un dossier nommé « Bagou » et une Image que l'on souhaite nommer « carteOS2025 », il faudra écrire « Bagou/carteOS2025 » après le début de l'*AssetID* qui est déjà complété par GEE (« [...]/assets/ »). D'autres valeurs peuvent être ajoutées mais ceci est facultatif. Des propriétés (*properties*) de l'Image peuvent être indiquées en utilisant les boutons *Add start time*, *Add end time*, et *Add property*. Les deux premiers boutons permettent d'ajouter une information concernant la date de l'Image. Enfin, les *Advanced options* permettent de modifier le paramètre de production de la pyramide d'images (*Pyramiding policy* avec moyenne, mode, minimum, maximum, et sample). Le paramètre Mode doit être utilisé dans le cas de données catégorielles comme une carte d'occupation du sol, et le mode Sample est recommandé pour des données comme une bande QA. Le champ *Masking mode* renseigne sur l'utilisation de la valeur NA ou d'une bande des fichiers importés pour masquer certains pixels. Enfin, le bouton *Upload* lance le processus d'importation des fichiers.

Les *Advanced options* sont différentes pour les fichiers vectoriels. Le champ *Character encoding* est important à vérifier si les attributs des vecteurs possèdent des caractères accentués, hors de l'alphabet latin, ou spéciaux. La valeur *Maximum error* est la précision en mètre à préserver lors du changement de projection. Enfin, il faut choisir l'option *Split large geometries* si les vecteurs importés possèdent de nombreux points. Dans ce cas les vecteurs seront découpés en plusieurs vecteurs pour faciliter les calculs.

Enfin, des tableaux peuvent être importés au format csv. GEE se chargera de la projection à partir du nom des colonnes renseignant la longitude et la latitude (voir *X column* et *Y column* dans *Advanced options*) et le CRS.

Il ne faut pas immédiatement fermer la fenêtre principale une fois que le bouton *Upload* a été cliqué. Le déroulement de l'importation doit être vérifié à partir de l'onglet *Tasks* (volet de droite). L'importation se déroule en deux étapes : les données sont d'abord envoyées sur le serveur de GEE (il ne faut ainsi pas perdre la connexion internet), puis elles sont traitées par GEE (*Ingest*, cette étape se fait à distance donc il n'est pas nécessaire de rester connecté). Si une erreur survient, elle sera renseignée dans cet onglet. Une fois que les données sont importées (symbole ✓), elles sont visibles dans l'onglet *Assets*, mais il est souvent nécessaire de rafraîchir le contenu de l'onglet en utilisant le bouton en haut de l'onglet.

Des conversions de fichiers sont souvent nécessaires car tous les formats ne sont pas acceptés par GEE.

Utiliser des données

Les données du catalogue Earth Engine sont facilement ajoutées à un script. Sur la page principale du catalogue (https://developers.google.com/earth-engine/datasets/catalog), recherchez les données WorldClim en entrant « Worldclim » dans le champ de recherche *Filter list of datasets*. Choisir le jeu de données WorldClim BIO Variables V1. La page du jeu de données renseigne de nombreuses informations comme la période sur laquelle se basent ces données (*Dataset Availability*), le fournisseur de données (*Dataset Provider*), la localisation du jeu de donnée (*Earth Engine Snippet*) ainsi que de nombreux mots-clés. Plusieurs onglets sont consultables comme la description des données (*Description*), les bandes (*Bands*), les restrictions quant à l'utilisation de ces données (*Terms of Use*), et des informations sur les références à citer (*Citations*). L'onglet *Bands* est important à consulter car il indique la résolution spatiale (*Pixel Size*), et les informations sur chaque bande y compris le facteur de conversion, *Scale*, qui est la valeur qui doit être utilisée pour obtenir la valeur correcte de chaque pixel. Enfin, plus bas, un bloc de code permet d'avoir un premier aperçu des données une fois collé et lu dans l'éditeur de code.

```
var dataset = ee.Image('WORLDCLIM/V1/BIO');
```

Cette instruction crée un nouvel objet nommé « dataset » de type Image en important les données.

```
var annualMeanTemperature = dataset.select('bio01').multiply(0.1);
```

L'Image « dataset » est ici traitée pour d'abord sélectionner la bande « bio01 » (correspondant à la température moyenne selon l'onglet *Bands*) qui est multipliée par 0.1 (le facteur de conversion). Deux étapes sont ainsi contenues dans cette instruction pour produire la nouvelle variable « annualMeanTemperature ».

```
print(annualMeanTemperature);
var couleurs = {min: -23, max: 30, palette: ['blue', 'yellow',
'red'],};
Map.addLayer(annualMeanTemperature, couleurs, 'Temperature Moyenne
annuelle');
```

La variable « couleurs » est créée et contient des valeurs pour la dernière instruction `Map.addLayer()`. Cette variable est en fait un dictionnaire pour lequel des valeurs sont

associées à différents arguments : -23 pour min, 30 pour max, et une liste de couleurs pour palette. Cette liste contient des couleurs selon un gradient choisi (du bleu au rouge en passant par le jaune), et est obligatoirement en anglais car GEE peut interpréter ces mots pour y associer une couleur sur la carte. L'instruction `Map.addLayer()` ajoute ainsi une couche sur la carte (volet du bas) en utilisant les arguments suivants : l'Image « annualMeanTemperature », les instructions de la variable couleurs, et le nom de la couche sera « Temperature Moyenne annuelle ». L'outil inspecteur (*Inspector*) du volet de droite peut être activé en cliquant sur l'onglet puis en cliquant sur un pixel de la carte.

Les fonctions GEE prennent souvent plusieurs arguments, certains étant facultatifs car ils ont une valeur par défaut. Comme indiqué sur la Figure 3, la fonction `Map.addLayer()` prend 5 arguments au maximum dont un est obligatoire (l'objet à afficher sur la carte). Comme indiqué dans la colonne *Type*, cet objet peut être une ImageCollection, FeatureCollection, Feature, Image ou RawMapId. Dans le script, les arguments doivent être écrits dans le même ordre qu'indiqué sur l'image pour que GEE puisse comprendre correctement l'instruction.

Usage	Returns
`Map.addLayer(eeObject, visParams, name, shown, opacity)`	ui.Map.Layer

Argument	Type	Details
`eeObject`	Collection|Feature|Image|RawMapId	The object to add to the map.
`visParams`	*FeatureVisualizationParameters| ImageVisualizationParameters, optional*	*The visualization parameters. For Images and ImageCollection, see ee.data.getMapId for valid parameters. For Features and FeatureCollections, the only supported key is "color", as a CSS 3.0 color string or a hex string in "RRGGBB" format. Ignored when eeObject is a map ID.*
`name`	*String, optional*	*The name of the layer. Defaults to "Layer N".*
`shown`	*Boolean, optional*	*A flag indicating whether the layer should be on by default.*
`opacity`	*Number, optional*	*The layer's opacity represented as a number between 0 and 1. Defaults to 1.*

Figure 3 Aperçu de la documentation pour la fonction `Map.addLayer()`.

Alternativement, le nom de chaque argument peut être écrit à l'intérieur de crochets ({}) comme :

```
Map.addLayer({eeObject: annualMeanTemperature,
visParams: couleurs,
name: 'Temperature moyenne annuelle',
opacity: 0.5});
```

Ceci permet de ne pas avoir à écrire la valeur de tous les arguments dans l'ordre.

Les étapes sont similaires pour importer une FeatureCollection. Sur la page d'accueil du catalogue (https://developers.google.com/earth-engine/datasets/catalog), rechercher « protected area » et ouvrir le jeu de donnée WDPA : World Database on Protected Areas (polygons). Les informations sont du même type que pour les Image et ImageCollection, cependant l'onglet *Bands* est remplacé par un onglet *Table Schema* qui contient les informations sur les différents attributs associés aux Feature (*Name*, *Type* et *Description*).

De même, chercher « FAO GAUL » permet d'afficher plusieurs jeux de données. Parmi ceux-ci, le jeu de données FAO GAUL : Global Administrative Unit Layers 2015, Country Boundaries

(frontières) qui est une FeatureCollection est importé et nommé « pays ». La seconde instruction utilise la fonction `filter()` et la fonction `ee.Filter.eq()` pour filtrer la FeatureCollection « pays » et sélectionner les Feature dont la propriété « ADM0_NAME » a la valeur « Benin ». Cette nouvelle variable est appelée « benin ». Il existe de nombreux types de filtres comme égal (`ee.Filter.eq`), plus grand ou égal (`ee.Filter.gte`), plus grand (`ee.Filter.gt`), ou différent de (`ee.Filter.not`).

```
var pays = ee.FeatureCollection('FAO/GAUL/2015/level0');
var benin = pays.filter(ee.Filter.eq('ADM0_NAME', 'Benin'));
```

La fonction `Map.centerObject()` modifie la carte du volet du bas en recentrant sur l'objet « benin » avec un zoom de 4 (la valeur doit être comprise entre 0 et 24). La dernière instruction affiche l'objet « benin » sans couleur particulière et cette couche est nommée « Benin ».

```
print(benin);
Map.centerObject(benin, 4);
Map.addLayer(benin, {}, 'Benin');
```

Il est aussi possible d'importer les jeux de données directement depuis la page contenant l'éditeur de code. Il suffit alors d'entrer le nom du jeu de données dans le champ de recherche qui se situe en haut de l'éditeur de code. Cependant, il est moins facile de lire toutes les informations à partir de cette fenêtre en comparaison avec le catalogue.

La documentation (apidocs)

La documentation sur les fonctions Earth Engine est importante et sera souvent consultée. Elle est accessible depuis l'onglet *Docs* (volet de gauche) de l'éditeur de code. Les fonctions sont organisées par catégorie ou objet sur lequel elles s'appliquent. Cependant, plus d'informations seront disponibles (y compris des exemples) en passant par la page https://developers.google.com/earth-engine/apidocs/ (barre de gauche > Client Libraries).

Pratique

Importer et visualiser une ImageCollection dans une région donnée

Cette section décrit les étapes pour produire une image la température moyenne du sol dans la région de l'Atacora entre janvier et février 2023. La fonction `print()` est régulièrement utilisée afin de visualiser les changements de l'ImageCollection.

Chercher le jeu de donnée MODIS Terra Land Surface Temperature and Emissivity (8 jours, 1 km de résolutions), MOD11A2.061, dans le catalogue Earth Engine puis l'importer tel que :

```
var MOD11A2 = ee.ImageCollection('MODIS/061/MOD11A2');
print(MOD11A2)
```

MOD11A2 contient de nombreuses Image correspondant la température du sol mesurée tous les 8 jours depuis 2000 avec une résolution d'1 kilomètre. La région d'intérêt est l'Atacora qui peut être importée depuis le jeu de donnée FAO GAUL Level 1 comme :

```
var FAOlvl1 = ee.FeatureCollection('FAO/GAUL/2015/level1');
var atacora = FAOlvl1.filter(ee.Filter.eq('ADM1_NAME',
'Atakora'));
print(atacora)
```

Le jeu de données MOD11A2 est filtré spatialement et temporellement avec les fonctions `filterBounds()` et `filter()` comme schématisé sur la Figure 4 :

```
var MOD11A2 = ee.ImageCollection('MODIS/061/MOD11A2')
   .filterBounds(atacora)
   .filter(ee.Filter.date('2023-01-01', '2023-03-01'));
print(MOD11A2)
print(MOD11A2.size()); // la taille de l'ImageCollection
```

Ici seules les Image comprises dans les Features de la variable « atacora » et prises entre le 1er Janvier et le 1er Mars 2023 sont considérées. L'ImageCollection obtenue contient 8 Images de chacune 12 bandes (dont « LST_Day_1km », la température du sol). Les bandes n'ont pas toutes le même type d'information car certaines ont des valeurs de type int16 quand d'autres sont de type int8 (des nombres entiers). Elles ont toutes la même projection (SR-ORG:6974). Les propriétés incluent la date de l'Image (« system:index » et « system:time_start »). La fonction `size()` permet d'obtenir le nombre d'éléments dans une collection (ici, 8).

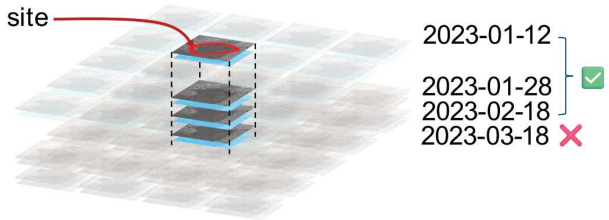


Figure 4 Exemple de l'utilisation des filtres pour sélectionner des Images dans une ImageCollection.

Pour sélectionner la bande correspondant à la température du sol :

```
var MOD11A2 = MOD11A2.select('LST_Day_1km');
print(MOD11A2); // chaque Image a une seule bande
```

Comme nous cherchons la température moyenne sur la période janvier-février, les observations sur cette période (les 8 Images) doivent être moyennées. La fonction `reduce()` permet d'appliquer un *reducer* sur l'ensemble des observations pour chaque pixel. Ainsi, ici, la moyenne des 8 Images est calculée pour chaque pixel dans l'ImageCollection et la fonction

`reduce()` produit alors une Image avec une bande correspondant à la moyenne des bandes LST_Day_1km des 8 Images en utilisant le *reducer* `ee.Reducer.mean()`.

```
var MOD11A2moyenne = MOD11A2.reduce(ee.Reducer.mean());
print(MOD11A2moyenne);
```

Avant de visualiser, il faut appliquer le facteur de conversion (*Scale*) correspondant à la bande LST_Day_1km pour l'ImageCollection MODIS/061/MOD11A2. Ce facteur est de 0,02 pour produire une valeur en K, ainsi :

```
var MOD11A2moyenne_C = MOD11A2moyenne
   .multiply(0.02)
   .subtract(273.15); // conversion en degres C
```

Enfin, pour visualiser les valeurs entre 25 et 45 (°C) :

```
var couleur = {
   min: 25, max: 45,
   palette: ['blue', 'yellow', 'red']
};
Map.centerObject(atacora, 8);
Map.addLayer(MOD11A2moyenne_C, couleur, 'T_moyenne_JanFev');
Map.addLayer(atacora);
```

Exporter des données

GEE permet d'exporter de nombreux types de données : rasters, tableaux, shapefiles, figures, vidéos. Ces exports sont soit internes (un nouvel Asset) ou externes (dans le compte Google Drive).

Exporter un raster

L'export d'un raster est nécessairement basé sur une Image (et non une ImageCollection). L'export vers le compte Google Drive est tel que :

```
Export.image.toDrive({
   image: MOD11A2moyenne_C,
   description: 'temp_moyenne_site_1',
   folder: 'cours_GEE_UP',
   region: atacora,
   scale: 1000,
   crs: 'EPSG:4326',
   fileFormat: 'GeoTIFF',
});
```

La fonction `Export.image.toDrive()` prend plusieurs arguments qui sont le nom de l'Image à exporter, une description de cette tâche, le nom du dossier Google Drive (*folder*) qui

est automatiquement créé s'il n'existe pas déjà, la région d'intérêt, la résolution spatiale (*scale*), la projection (*crs*), et un format de fichier (*fileFormat*). Une fois que l'instruction est interprétée par le serveur, la tâche correspondant à l'export est ajoutée à l'onglet *Tasks* dans le volet de droite où il suffit de cliquer sur le bouton *Run* pour lancer la tâche et éventuellement modifier quelques paramètres. Il est possible de lancer de nombreuses tâches en même temps, mais GEE ne les traitera pas en parallèle : GEE préférera en traiter un petit nombre puis traiter les suivantes une fois que les précédentes tâches sont terminées. Pour annuler une tâche, cliquer sur la tâche à annuler et cliquer sur le bouton *Cancel*.

Les rasters exportés peuvent être très grands, utiliser une petite région d'intérêt (*region*) ou convertir le type de donnée de l'Image (voir fonctions comme `float()`, `double()`) permettent de réduire la taille de l'export. La fonction `clip()` permet de découper une Image à partir d'un objet Feature comme :

```
Map.addLayer(MOD11A2moyenne_C.clip(atacora), couleur, 'T_moyenne_
clip');
```

Exporter un tableau

Cette section décrit les étapes pour exporter la température et précipitation moyennes pour avril et mai entre 1990 et 2020 pour les différentes régions du Bénin. Les limites administratives sont issues d'une FeatureCollection et les données sur le climat sont sous forme d'Image ou d'ImageCollection.

```
var BeninLevel1 = ee.FeatureCollection('FAO/GAUL/2015/level1')
.filter(ee.Filter.eq('ADM0_NAME', 'Benin'));
var climat = ee.ImageCollection('ECMWF/ERA5_LAND/MONTHLY_AGGR')
   .select('temperature_2m', 'total_precipitation_sum')
   .filter(ee.Filter.calendarRange(1990, 2020, 'year'))
   .filter(ee.Filter.calendarRange(4, 5, 'month'));
```

Le filtre `ee.Filter.calendarRange()` permet de filtrer des mois en particulier sur une période de plusieurs années comme ci-dessus. Il faut ainsi bien préciser le type (« year », « month », etc. en anglais).

```
var climat_moyenne = climat.mean(); // produit image
print(climat_moyenne) // image à deux bandes
```

La fonction `mean()` s'applique à chaque bande de l'ImageCollection « climat » pour produire une Image à deux bandes où chaque bande est la moyenne sur l'ImageCollection (Figure 5). Ici, la fonction `mean()` fonctionne de la même façon que les fonctions `ee.Reducer` qui incluent aussi `ee.Reducer.mean()`.

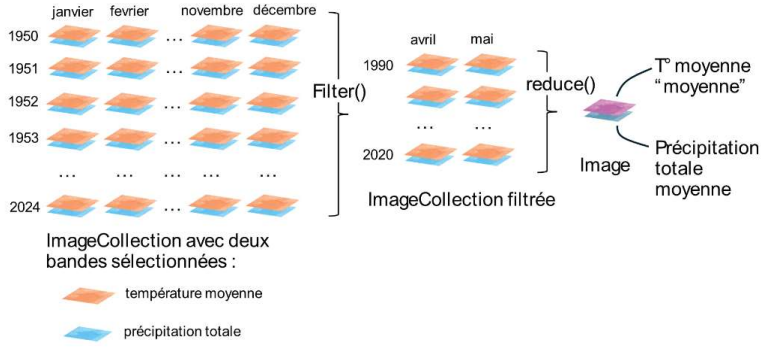


Figure 5 Exemple de l'utilisation de la fonction `Filter()` et de la fonction `reducer()` pour obtenir la température et la précipitation totale moyenne.

La dernière étape de manipulation des données est de calculer la température moyenne et la précipitation moyenne à l'échelle de chaque région du Bénin. Le nom de chaque région peut être affiché à partir de la fonction `aggregate_array()` qui collecte toutes les valeurs d'une propriété au sein d'une collection.

```
print(BeninLevel1.aggregate_array('ADM1_NAME'));
```

A partir de ceci, il est possible d'extraire les statistiques avec la fonction `reduceRegion()`, tel que :

```
var Atakora = BeninLevel1.filter(ee.Filter.eq('ADM1_NAME',
'Atakora')).first();
var stats = climat_moyenne.reduceRegion({
   reducer: ee.Reducer.mean(),
   geometry: Atakora.geometry(),
   scale: 11000,
   crs: 'EPSG:4326'
});
print(stats)
```

Cependant, comme ce sont les statistiques pour chaque région du Bénin qui sont demandées, c'est la fonction `reduceRegions()` qui est requise car elle s'applique à des FeatureCollection.

```
var stats_benin = climat_moyenne.reduceRegions({
   reducer: ee.Reducer.mean(),
   collection: BeninLevel1,
   scale: 11000,
   crs: 'EPSG:4326'
});
```

```
print(stats_benin)
```

Ici, la variable « stats_benin » possède des Features correspondants aux régions du Bénin et chaque Feature a maintenant deux propriétés supplémentaires décrivant la température et la précipitation moyennes. L'extraction de ces données au format tableau (csv parmi plusieurs formats possibles) se fait à partir de la fonction `Export.table.toDrive()` où l'argument *selectors* indique les propriétés qui doivent être extraites : le nom des propriétés deviendra le nom des colonnes du tableau.

```
Export.table.toDrive({
   collection: stats_benin,
   description: 'temperatureprecipitation19902020Benin',
   folder: 'cours_GEE_UP',
   fileFormat: 'CSV',
   selectors: ['ADM1_NAME', 'temperature_2m',
'total_precipitation_sum']
});
```

Créer une fonction, utiliser l'agrégation spatiale et exporter un vecteur

Bien qu'il existe déjà de nombreuses fonctions dans GEE, il est souvent nécessaire d'en créer de nouvelles adaptées aux besoins de l'étude. La création d'une fonction est simple et va se baser sur une valeur pour en retourner une autre (*return*). Une fonction se produit de la façon suivante :

```
var mafonction = function(parametre){
   var nouvellevaleur = parametre.add(15);
   return nouvellevaleur
};
```

Cette fonction est très simple. Les fonctions peuvent contenir de nombreuses étapes et manipulations de données. La fonction s'applique ainsi :

```
var test = mafonction(ee.Number(14));
print(test)
```

Dans de nombreux cas, les fonctions sont utilisées sur plusieurs éléments d'un objet comme les éléments d'une liste, les Features ou Images d'une collection. Ceci se fait en utilisant la fonction `map()` qui est l'alternative aux boucles classiques basées sur `for i`. Ici, une boucle est utilisée pour retirer les décimales des valeurs de la température et précipitation dans la FeatureCollection « stats_benin ».

```
var funcor = function(feature){
   var temp = ee.Number(feature.get('temperature_2m'));
   var prec = ee.Number(feature.get('total_precipitation_sum'));
   var tempcor =
temp.subtract(273.15).multiply(100).round().divide(100)
   var preccor =
prec.multiply(1000).multiply(100).round().divide(100)
   return feature.set('temperature', tempcor).set('precipitation',
preccor)
};
var stats_benin_corr = stats_benin.map(funcor)
print(stats_benin_corr);
```

Ici, la fonction `funcor()` est produite pour corriger les valeurs de température et précipitation (« temperature_2m » et « total_precipitation_sum ») qui sont d'abord extraites du Feature considéré (avec la fonction `get()`). La température est convertie en °C depuis °K, puis elle est transformée pour qu'il ne reste que deux décimales. Cette transformation nécessite de passer par une étape intermédiaire (en multipliant par 100) où le reste des décimales est enlevé à l'aide d'un arrondi (fonction `round()`). La précipitation est corrigée en passant d'abord la valeur en mètre à une valeur en mm, puis en arrondissant de la même manière que pour la température. La fin de la fonction, *return*, retourne le Feature avec deux nouvelles propriétés ajoutées avec la fonction `set()` qui s'appellent « temperature » et « precipitation ». Cette fonction est ensuite appliquée à la FeatureCollection « stats_benin » à l'aide de la fonction `map()`.

Indices de végétation et filtre simple

Cette section décrit les étapes pour calculer la valeur moyenne du NDVI dans les aires cultivées du site d'Awanla pour le mois de février 2023 à partir de données Sentinel-2. La méthode est la suivante (Figure 6) :

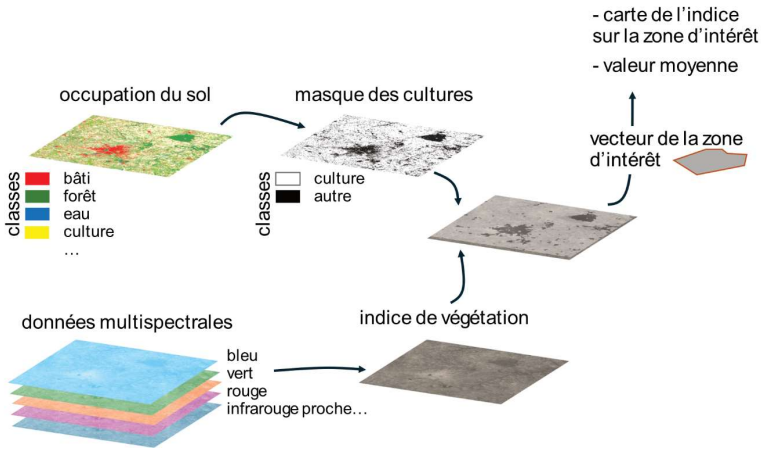


Figure 6 Etapes pour extraire la valeur moyenne d'un indice de végétation dans les espaces cultivés.

Tout d'abord, il faut importer la carte d'occupation du sol pour le site d'Awanla développée par OBSYDYA pour 2023. Cette carte n'est pas disponible dans le catalogue Earth Engine, il s'agit d'un asset importé dans le cadre de la formation.

```
var carteOS = ee.Image('projects/ee-
jauneatemps/assets/pastoralism/OBSYDYA_OS_2023_L1_Awanla')
Map.centerObject(carteOS)
Map.addLayer(carteOS)
```

Une palette de couleurs est ajoutée pour distinguer les classes :

```
var couleursOS = ['eebc1d', 'f1e5ac', 'f531c1', 'bb270d',
'4cbb17', '40e0d0', '355e3b', '008000', '8a9a5b', 'ff0000',
'fff8dc', '0000ff']; // liste de couleurs en hexadécimal
Map.addLayer(carteOS, {min: 11, max: 61, palette: couleursOS},
'carteOS', false); // false indique que la carte ne doit pas etre
affichee par defaut
```

Dans cette carte d'occupation du sol, les valeurs 11 à 14 représentent des cultures non arborées, et toutes les autres classes ont une valeur > 14. Comme l'indice de végétation est appliqué aux cultures, il faut masquer le reste de l'occupation du sol. Ce masque se fait simplement à partir de la fonction `lte()`, *less than or equal* (moins que ou egal), qui converti toutes les valeurs égales ou en deçà du seuil (ici, 14) en valeur 1 et le reste en valeur 0. La valeur 0 est la valeur qui sera utilisée pour masquer les régions qui ne sont pas d'intérêt.

```
var carteOSmask = carteOS.lte(14);
Map.addLayer(carteOSmask, {}, 'mask');
```


Si les critères sont plus complexes, il est possible d'utiliser les fonctions `or()` et `and()` qui permettent de combiner plusieurs seuils ou conditions. Les deux exemples suivants ne sont pas utilisés dans la suite de cette section :

```
var mask = carteOS.eq(31).or(carteOS.eq(21));
var mask = carteOS.gte(21).and(carteOS.lte(42));
```

La prochaine étape est de produire un indice de végétation à partir d'une image Sentinel-2. Ceci se fait à partir du jeu de données « S2_HARMONIZED » pour la période de février 2023. Comme de nombreuses images sont nuageuses dans cette région et à cette période, il est recommandé de filtrer le jeu de données à l'aide de la propriété de l'Image appelée « CLOUDY_PIXEL_PERCENTAGE » qui est associée à chaque Image de la collection Sentinel-2. Une valeur de 20% de pixels nuageux au maximum a été définie. De plus, il est nécessaire de filtrer l'ImageCollection pour ne garder que les Image qui recouvrent le site d'Awanla. Pour cela, nous utilisons la fonction `filterBounds()` avec un polygone appelé « geometry » créé manuellement sur le site d'Awanla. Enfin, la fonction `maskS2clouds()` doit être copiée depuis la page Earth Engine du jeu de données « S2_HARMONIZED » et appliquée à l'ImageCollection avec `map()`.

```
function maskS2clouds(image) {
  var qa = image.select('QA60');

  // Bits 10 and 11 are clouds and cirrus, respectively.
  var cloudBitMask = 1 << 10;
  var cirrusBitMask = 1 << 11;

  // Both flags should be set to zero, indicating clear
  //conditions.
  var mask = qa.bitwiseAnd(cloudBitMask).eq(0)
      .and(qa.bitwiseAnd(cirrusBitMask).eq(0));

  return image.updateMask(mask).divide(10000);
}

var sentinel2 = ee.ImageCollection('COPERNICUS/S2_HARMONIZED')
  .filterDate('2023-02-01', '2023-03-31')
  .filter(ee.Filter.lt('CLOUDY_PIXEL_PERCENTAGE', 20))
  .filterBounds(geometry)
  .map(maskS2clouds);
```

La production d'un indice de végétation comme le NDVI (normalisé) se fait simplement dans GEE car il existe la fonction `normalizedDifference()` qui calcule l'indice à partir du nom de deux bandes de l'Image. Ici, cette fonction est intégrée à une plus grande fonction qui sera appliquée à chaque Image. La bande créée est renommée « NDVI » grâce à la fonction `rename()` et elle est ajoutée à l'Image avec la fonction `addBands()`.

```
var bandeNDVI = function(image) {
   var ndvi = image.normalizedDifference(['B8', 'B4'])
   .rename('NDVI');
   return image.addBands(ndvi);
};
var sentinel2_ndvi = sentinel2.map(bandeNDVI).select('NDVI');
```

L'ImageCollection « sentinel2_ndvi » possède ainsi plusieurs Image avec une seule bande (« NDVI »). La fonction `max()` est appliquée à l'ImageCollection pour produire une Image où chaque pixel aura la valeur maximale possible à travers les multiples Image de l'ImageCollection.

```
var sentinel2_ndvi_max = sentinel2_ndvi.max();
Map.addLayer(sentinel2_ndvi_max, {min: 0, max: 0.5,
   palette: ['orange', 'yellow', 'green']},
   'sentinel2_ndvi_max', false);
```

Ici, les règles pour la palette de couleurs sont directement intégrées à l'instruction `Map.addLayer()` sans passer par la création d'une variable. Le maximum est défini à un NDVI de 0.5 car les valeurs sont généralement basses.

La dernière étape est de mesurer le NDVI pour les cultures. Le masque créé précédemment (« carteOSmask ») est utilisé pour masquer les zones non cultivées sur l'Image du NDVI maximal avec la fonction `updateMask()`. Les pixels seront masqués où le masque prend une valeur de 0.

```
var sentinel2_ndvi_max_masque =
sentinel2_ndvi_max.updateMask(carteOSmask);
Map.addLayer(sentinel2_ndvi_max_masque, {min: 0, max: 0.5,
   palette: ['orange', 'yellow', 'green']},
   'sentinel2_ndvi_max_masque', false);
```

Enfin, le NDVI moyen dans les espaces cultivés est mesuré à l'aide de la fonction `reduceRegion()`. Cependant, une délimitation spatiale doit être fournie à l'argument *geometry*. Le contour du site d'Awanla est produit à partir de la fonction `geometry()` et de l'Image « carteOS ».

```
var zoneinteret = carteOS.geometry();
Map.addLayer(zoneinteret, {}, 'limites');

var NDVIcultures = sentinel2_ndvi_max_masque.reduceRegion({
   reducer: ee.Reducer.mean(),
   geometry: zoneinteret,
   scale: 10,
   crs: 'EPSG:4326',
   maxPixels: 1e13
```



```
});
print(NDVIcultures)
```

L'argument *maxPixels* doit être spécifié avec une valeur suffisamment grande car la carte d'occupation possède de nombreux pixels. Ici, 1e13 = 10^{13}.

Algorithmes GEE et topographie

GEE contient plusieurs fonctions qui s'appliquent aux données topographiques (modèle numérique de terrain, MNT). Ces fonctions permettent de calculer la pente moyenne, l'aspect (entre 0 et 359°), et de produire des images de type *hillshade* (estompage de pente).

```
var mnt = ee.Image('CGIAR/SRTM90_V4');
var topographie = ee.Algorithms.Terrain(mnt);
print(topographie);
Map.addLayer(topographie.select('slope'),
   {min: 0, max: 90, palette: ['white', 'black']});
Map.addLayer(topographie.select('aspect'),
   {min: 0, max: 360, palette: ['red', 'orange', 'yellow',
'green', 'blue']});
Map.addLayer(topographie.select('hillshade'));
```

La fonction `ee.Algorithms.Terrain()` prend une Image avec l'élévation du sol et produit trois bandes avec la pente moyenne, l'aspect, et l'*hillshade*. Il existe des fonctions dédiées à ces trois informations pour produire les mêmes données : `ee.Terrain.aspect()`, `ee.Terrain.slope()`, et `ee.Terrain.hillshade()`.

Images composites

La création d'images composites permet de produire des images sans nuages à partir d'une multitude d'images avec certains nuages mais qui ne seront pas localisés aux mêmes endroits. Ainsi, il est possible d'obtenir une vue sans nuage pour chaque pixel.

```
var benin = ee.FeatureCollection('FAO/GAUL/2015/level0')
   .filter(ee.Filter.eq('ADM0_NAME', 'Benin'));

var S2_2024 = ee.ImageCollection('COPERNICUS/S2_HARMONIZED')
   .filterDate('2024-10-01', '2024-12-01')
   .filter(ee.Filter.lt('CLOUDY_PIXEL_PERCENTAGE', 20))
   .map(maskS2clouds);

var composite = S2_2024.median().clip(benin);

var RVBVis = {
   min: 0.0,
```

```
    max: 0.3,
    bands: ['B4', 'B3', 'B2'],
};

Map.centerObject(benin, 7);
Map.addLayer(composite, RVBVis, 'VraieCouleur');
```

Les deux premières étapes sont d'importer les frontières du Bénin à partir du jeu de données FAO GAUL et d'importer les données Sentinel-2 pour les mois d'Octobre-Novembre 2024. Comme dans la section précédente, la collection est filtrée pour ne garder que les Images avec un couverture nuageuse en deçà de 20% et la fonction `maskS2clouds()` (non collée dans cet exemple) est appliquée avec `map()`. L'image composite est créée en prenant la médiane (fonction `median()`) des valeurs de chaque pixel dans l'ImageCollection « S2_2024 » puis en la découpant avec les frontières du Bénin. L'Image produite « composite » est affichée en utilisant les bandes B4, B3, et B2 (rouge, vert, bleu de Sentinel-2).

Détection des eaux de surface

Cette section décrit les étapes pour détecter les eaux de surface à l'aide d'un indice basé sur des données multispectrales (Sentinel-2) et d'un seuil. L'indice utilisé est le Normalized Difference Wetness Index (NDWI) qui est basé sur les bandes correspondant au vert et au proche infrarouge.

$$NDWI = \frac{vert - proche\ infrarouge}{vert + proche\ infrarouge}$$

En effet, l'eau a une réflectance significativement plus élevée aux longueurs d'onde correspondant au vert qu'à celles correspondant aux proches infrarouges (Figure 7). Ceci n'est pas le cas pour la réflectance du sol ou de la végétation saine. Cependant, comme les eaux de surface contiennent d'autres éléments (suspension, végétation flottante, ...), le seuil pour différencier l'eau du reste de l'occupation du sol doit être ajusté en fonction du site d'étude.

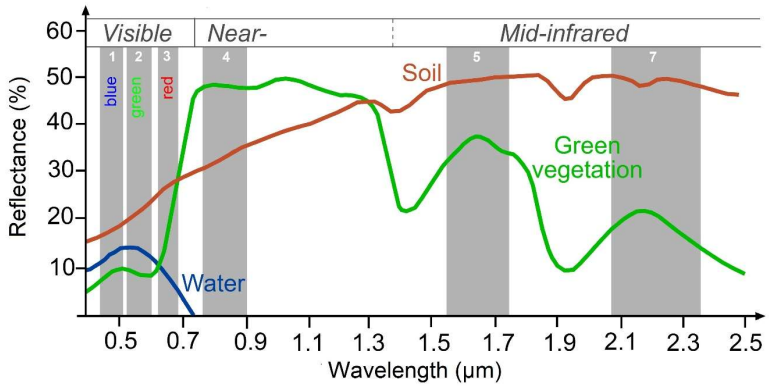


Figure 7 Spectres de réflectance de l'eau, du sol, et de la végétation verte dans le visible, le proche infrarouge, et l'infrarouge moyen.

```
var S2 = ee.ImageCollection('COPERNICUS/S2_HARMONIZED')
    .filterDate('2023-01-01', '2023-03-31')
    .filter(ee.Filter.lt('CLOUDY_PIXEL_PERCENTAGE', 20))
    .map(maskS2clouds);

var NDWIfon = function(image) {
    var ndwi =
image.normalizedDifference(['B3','B8']).rename('NDWI');
    return image.addBands(ndwi);
};
var S2ndwi = S2.map(NDWIfon).select('NDWI');
var S2ndwimax = S2ndwi.max();
```

Comme dans les sections précédentes, le jeu de données « S2_HARMONIZED » est importé et filtré pour sélectionner les observations de janvier à mars 2023 avec une couverture nuageuse en deçà de 20%, puis la fonction `maskS2clouds()` est utilisée (voir section Indices de végétation et filtre simple). Ensuite, une fonction est créée pour calculer l'indice NDWI à partir des bandes correspondantes de Sentinel-2. Cette fonction est appliquée à l'ImageCollection « S2 », puis cette ImageCollection est convertie en une Image « S2ndwimax » pour obtenir la valeur maximale du NDWI par pixel sur la période d'étude.

La seconde étape est de définir un seuil, ici -0.1, et de l'utiliser pour masquer les valeurs du NDWI en deçà de -0.1 :

```
var seuil = ee.Number(-0.1);
var masque = S2ndwimax.gte(seuil);
var S2ndwimax_m = S2ndwimax.updateMask(masque);
```

L'Image est ajoutée à la carte et cette carte est centrée sur la ville de Sori. La fonction `Map.setCenter()` prend trois arguments : la longitude, la latitude, et le zoom (15).

```
Map.addLayer(S2ndwimax_m);
Map.setCenter(2.78527, 10.71729, 15); // ville de Sori
```

La carte produite comprend de nombreux faux positifs. Ces faux positifs sont caractérisés par leurs petites tailles et sont souvent associés à des bâtiments. Ainsi, une manière de retirer ces faux positifs est d'enlever les petits groupes de pixels à l'aide d'un seuil (par exemple, les groupes comprenant moins de 5 pixels sont enlevés). Il existe deux façons d'enlever ces pixels : la première passe par la fonction `connectedPixelCount()` qui compte le nombre de pixels dans un patch avec le paramètre *eightConnected* qui définit si on compte les pixels qui se touchent par un coin ; la deuxième façon passe par la fonction `reduceToVectors()` qui vectorise l'Image et qui compte automatiquement le nombre de pixels par Feature produit (la propriété créée s'appelle *count*). La première façon est la suivante :

```
var S2ndwimax_m_pxlgrp = S2ndwimax_m.int()
   .connectedPixelCount(100, true)
   .reproject({
   crs: 'EPSG:4326',
   scale: 10
});

Map.addLayer(S2ndwimax_m_pxlgrp, {}, 'connectedPixelCount');
var masque_grp = S2ndwimax_m_pxlgrp.gte(5); // masque groups < 5
pixels
var S2ndwimax_m_g = S2ndwimax_m.updateMask(masque_grp);
Map.addLayer(S2ndwimax_m_g);
```

Ici, la fonction `int()` converti d'abord les valeurs du NDWI en nombre entier car c'est un prérequis pour la fonction `connectedPixelCount()`. Cette fonction prend l'argument *maxSize* = 100 car le seuil utilisé ne requiert pas de compter au-delà de 100 pixels, et l'argument *eightConnected* est définit comme *true* car on souhaite compter les pixels se touchant par un coin. Ensuite, la fonction `reproject()` permet de garder la résolution de 10m de l'Image NDWI « S2ndwimax_m » et de ne pas être affecté par le niveau de zoom dans la carte (volet du bas). En effet, la fonction `connectedPixelCount()` est affectée par le niveau de zoom, et peut retourner des valeurs erronées si la résolution de l'Image n'est pas précisée avec `reproject()`. Enfin, un masque est créé pour masquer les groupes de pixels comprenant moins de 5 pixels.

Une étape supplémentaire peut être ajoutée si on souhaite ne garder que les pixels avec au moins 3 observations sur la période qui a été définie lors du filtre de la collection Sentinel-2 (ici janvier-mars 2023). La fonction `count()` qui s'applique à une ImageCollection produit une Image où chaque pixel prend la valeur égale au nombre de pixels dans l'ImageCollection qui ne sont pas masqués (Figure 8) :

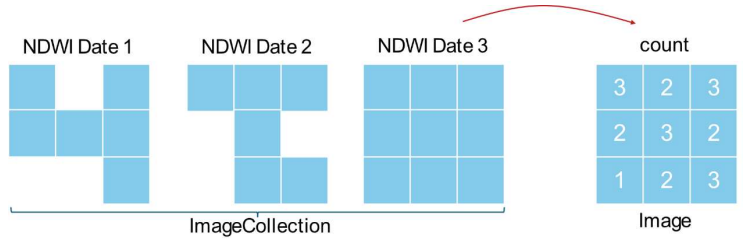


Figure 8 Exemple de l'application de la fonction `count()` sur une ImageCollection contenant trois images. La fonction count() compte le nombre de valeurs non nulles (bleu) pour chaque pixel et produit une image avec une bande correspondant au nombre total de valeurs non nulles.

```
var S2ndwicount = S2ndwi.count();
var masque_count = S2ndwicount.gte(3); // au moins 3 observations
var S2ndwimax_m_g_c = S2ndwimax_m_g.updateMask(masque_count);
Map.addLayer(S2ndwimax_m_g_c);
```

La méthode passant par la création d'un vecteur (FeatureCollection) est la suivante. Il faut d'abord créer un polygone « siteSori » pour délimiter la région d'intérêt. La FeatureCollection produite « S2ndwimax_m2 » est alors filtrée à partir de la propriété « count » en ne choisissant que les Feature avec « count » supérieur à 5 (`ee.Filter.gt()`, *greater than*).

```
var S2ndwimax_m2 = S2ndwimax_m.int().reduceToVectors({
   geometry: siteSori,
   crs: 'EPSG: 4326',
   scale: 10,
   geometryType: 'polygon',
   eightConnected: true,
   labelProperty: 'zone',
   bestEffort: true,
   maxPixels: 1e9
});
var S2ndwimax_m2vector = S2ndwimax_m2.filter(ee.Filter.gt('count',
5));
Map.addLayer(S2ndwimax_m2vector);
```

Classification

Classification non-supervisée

La classification non-supervisée d'une Image se base sur le regroupement de pixels au sein de classes selon leur proximité selon les différentes bandes de l'Image (un espace à *n* bandes-dimensions). Le nombre de classes est défini par l'utilisateur, ainsi que la taille de l'échantillon qui servira à produire les classes. L'exemple suivant se base sur les images Landsat. La carte d'occupation du sol pour Awanla développée par OBSYDYA est utilisée pour identifier une région d'intérêt. Le jeu de données Landsat 8 est utilisé ici, d'abord filtré spatialement et

temporellement pour n'inclure que des Image de la région d'intérêt prises entre janvier et mars 2023. La fonction `sort()` associée à la propriété « CLOUD_COVER » permet de ré-arranger les Image de l'ImageCollection selon la valeur de « CLOUD_COVER ». La fonction `first()` sélectionne la première Image de l'ImageCollection et retourne un objet de type Image, qui sera ici l'Image la moins nuageuse de la collection.

```
var carteOS = ee.Image('projects/ee-
jauneatemps/assets/pastoralism/OBSYDYA_OS_2023_L1_Awanla')
var site_etude = carteOS.geometry();
var landsat = ee.ImageCollection('LANDSAT/LC08/C02/T1_L2')
   .filterBounds(site_etude)
   .filterDate('2023-01-01', '2023-03-31')
   .sort('CLOUD_COVER')
   .first();
var landsat_vis = {
   bands: ['SR_B4', 'SR_B3', 'SR_B2'],
   min: 7000,
   max: 15000
};
Map.centerObject(site_etude, 10);
Map.addLayer(landsat, landsat_vis, 'Image L8');
Map.addLayer(site_etude)
```

L'échantillonnage se fait à partir de la fonction `sample()`, ici avec une résolution spatiale de 30 mètres (celle de Landsat 8), pour un échantillon de taille 1000 pixels :

```
var echantillon = landsat.sample({
   region: landsat.geometry(),
   scale: 30,
   numPixels: 1000,
   tileScale: 8
});
```

Cet échantillon est alors utilisé pour entraîner la classification à base de 5 classes à partir de la fonction `ee.Clusterer.wekaKMeans()` (méthode des k-moyennes) et de `train()`.

```
var clusterer = ee.Clusterer.wekaKMeans(5).train(echantillon);
```

A partir de cet entraînement, l'Image « landsat » est entièrement classifiée avec la fonction `cluster()`. La fonction `randomVisualizer()` permet d'assigner des couleurs automatiquement à chaque valeur de l'Image.

```
var Kclassifie = landsat.cluster(clusterer);
Map.addLayer(Kclassifie.randomVisualizer(), {}, 'K-means classifié
en 5 classes');
```


Classification supervisée

Contrairement à la classification non-supervisée, la classification supervisée se base sur un échantillonnage manuel des différents types d'occupation du sol (classe). Ainsi, en plus d'utiliser une Image comme dans la section précédente, l'utilisateur va devoir créer plusieurs sets d'échantillon (le nombre de set = le nombre de classes). Dans cet exemple, les sets sont créés dans GEE à partir de l'outil de création de points de la carte. Comme dans la section précédente, le contour de la région d'intérêt et une Image Landsat 8 sont importées :

```
var carteOS = ee.Image('projects/ee-
jauneatemps/assets/pastoralism/OBSYDYA_OS_2023_L1_Awanla')
var site_etude = carteOS.geometry();
var landsat = ee.ImageCollection('LANDSAT/LC08/C02/T1_L2')
   .filterBounds(site_etude)
   .filterDate('2023-01-01', '2023-03-31')
   .sort('CLOUD_COVER')
   .first();
var landsat_vis = {
   bands: ['SR_B4', 'SR_B3', 'SR_B2'],
   min: 7000,
   max: 15000
};
Map.centerObject(site_etude, 10);
Map.addLayer(landsat, landsat_vis, 'Image L8');
Map.addLayer(site_etude);
```

Ensuite, créer un nouvel objet géométrique à partir de l'outil . Cet objet doit être modifié avec le bouton : le nom (*Name*) doit être modifié pour qu'il corresponde à la classe (par exemple « foret »), et l'option *Import as* doit être modifiée pour qu'elle soit « FeatureCollection ». Ceci permet de créer une nouvelle propriété (*properties*) avec le bouton + Property. La valeur de *Property* est « classe », et sa valeur (*Value*) est 0 pour la première classe (ici, « foret »). Cet objet doit être ensuite modifié sur la carte à l'aide de l'outil pour rajouter des points où l'occupation du sol correspond à la classe. Ces étapes doivent être répétée en créant un nouvel objet géométrique avec le bouton + new layer pour chaque classe d'occupation du sol qui doit être labélisée : à chaque classe sera donnée une valeur différente (par exemple 0 pour forêts, 1 pour le bâti, etc). Après avoir créé les 5 classes d'intérêts : « foret », « bati », « eau », « vegetationbasse » et « solnu » (les noms des FeatureCollections), celles-ci sont regroupées dans une FeatureCollection avec la fonction `ee.FeatureCollection()`. Cependant, ceci produit une FeatureCollection comprenant 5 FeatureCollections, nous souhaitons la transformer en une simple FeatureCollection comprenant des Features. La fonction `flatten()` est utilisée à cette fin.

```
var entrainement = ee.FeatureCollection([foret, bati, eau,
vegetationbasse, solnu]).flatten();
print(entrainement);
```

Une liste doit être créée avec le nom des bandes à utiliser pour la classification. Le nom de ces bandes doit correspondre aux bandes disponibles dans l'Image « landsat ».

```
var bandes = ['SR_B1', 'SR_B2', 'SR_B3', 'SR_B4', 'SR_B5',
'SR_B6', 'SR_B7','ST_B10'];
```

A partir de cette liste, les bandes d'intérêt sont sélectionnées dans l'Image « landsat ». Ensuite, la fonction `sampleRegions()` échantillonne les valeurs de chacune des bandes à partir des points d'entraînement et les regroupe selon les classes définies ultérieurement (classe 0 pour forêt, etc).

```
var echantillon = landsat.select(bandes)
   .sampleRegions({
      collection: entrainement,
      properties: ['classe'],
      scale: 30
});
print(echantillon);
```

Cet échantillon sert à entraîner un *classifier* à partir de la fonction `train()`. Il existe de nombreux types de classifier sur GEE, ici c'est celui de la fonction `ee.Classifier.smileCart()` qui est utilisé. La fonction `train()` a besoin de *features*, soit l'échantillon, du nom de la propriété de classification, ici « classe », et des valeurs utilisées pour classifier (« bandes »).

```
var classifier = ee.Classifier.smileCart().train({
   features: echantillon,
   classProperty: 'classe',
   inputProperties: bandes
});
```

Une fois entraîné, ce modèle est appliqué aux bandes de l'Image « landsat » pour produire une Image classifiée :

```
var classification = landsat.select(bandes).classify(classifier);
var classificationVis = {
   min: 0,
   max: 4,
   palette: ['green', 'red', 'blue', 'yellow', 'pink']
};
Map.addLayer(classification, classificationVis,
'classificationsupervCART');
```

Précision de la classification

La mesure de la précision d'une classification est importante afin de valider la carte produite. Elle se base sur plusieurs indices (Précision, Précision de type Producer, Précision de type User) à l'aide du compte de vrai/faux positifs et vrai/faux négatifs. Pour cela, il faut segmenter l'échantillon produit précédemment en un jeu de donnée d'entraînement et un jeu de données de test qui servira à valider la classification produite avec les données d'entraînement. Ici, le jeu de données d'entraînement reprend 80% des points créés pour l'échantillon de chaque classe, et le jeu de données de validation reprend 20% des points (le reste). Pour cela, chaque élément de « echantillon » reçoit une valeur aléatoire entre 0 et 1 pour une nouvelle propriété nommée « random » avec la fonction `randomColumn()`. Ensuite, ces éléments sont triés en deux groupes, « groupetrain » (entraînement) et « groupetest » (test, validation) selon la valeur associée à la propriété « random ». La fonction `lt()` sélectionne les éléments avec une valeur < 0.8 (*less than*) et la fonction `gte()` sélectionne les éléments avec une valeur ≥ 0.8.

```
var traintest = echantillon.randomColumn();
var groupetrain = traintest.filter(ee.Filter.lt('random', 0.8));
var groupetest = traintest.filter(ee.Filter.gte('random', 0.8));
print(groupetrain);
print(groupetest);
```

Comme précédemment, un *classifier* est entraîné, mais cette fois avec le set d'entraînement « groupetrain ».

```
var classifier2 = ee.Classifier.smileCart().train({
   features: groupetrain,
   classProperty: 'classe',
   inputProperties: bandes
});
```

Enfin, une matrice de confusion est produite. D'abord, le *classifier* produit à partir du set d'entraînement est utilisé pour classifier le set de validation « groupetest » et la fonction `errorMatrix()` permet de comparer les classes qui avaient été définies manuellement (« classe ») à celles prédites par le *classifier* (argument *predicted*, ici « classification »). Cette matrice est utilisée pour produire des valeurs décrivant la précision de la classification à l'aide des fonctions `accuracy()`, `producersAccuracy()`, `consumersAccuracy()`, et `kappa()`.

```
var matriceconfusion =
groupetest.classify(classifier2)
   .errorMatrix({
      actual: 'classe',
      predicted: 'classification'
});
print('Matrice de confusion :', matriceconfusion);
print('Précision :', matriceconfusion.accuracy());
print('Précision (prod) :', matriceconfusion.producersAccuracy());
print('Précision (user) :', matriceconfusion.consumersAccuracy());
print('Kappa :', matriceconfusion.kappa());
```

Jointures, jointures spatiales

Les jointures attributaires sont fréquemment utilisées sur GEE à partir des diverses fonctions `ee.Join()`. Ces jointures permettent de filtrer des collections selon leurs propriétés.

Jointure en fonction de la date

Dans l'exemple suivant, nous lions deux ImageCollections en fonction de la date d'acquisition des Image. L'ImageCollection MODISEVI est créée à partir du jeu de donnée MCD43A4_006_EVI qui est basé sur MODIS et contient une Image avec une bande EVI produite tous les jours. L'ImageCollection MODIS est créée à partir du jeu de données MCD43A2 qui a une résolution temporelle d'un jour et contient des informations concernant la qualité des produits Albedo de MODIS.

```
var MODISEVI = ee.ImageCollection('MODIS/MCD43A4_006_EVI')
   .filter(ee.Filter.date('2014-01-01', '2014-02-01'));
var MODIS = ee.ImageCollection('MODIS/006/MCD43A2')
   .filter(ee.Filter.date('2014-01-27', '2014-02-15'));
```

Un filtre est ensuite créé avec la fonction `ee.Filter.equals()`. Dans ce cas, seules les Images partageant la même valeur pour la propriété « system:index » seront gardées.

```
var filtre = ee.Filter.equals({
   leftField: 'system:index',
   rightField: 'system:index'
});
```

La fonction `ee.Join.simple()` est ensuite utilisée pour appliquer le filtre et ne garder que les Images de la première ImageCollection définie dans `apply()` (la collection primaire, ici MODISEVI) partageant les mêmes propriétés que la seconde ImageCollection définie dans `apply()` (ici, MODIS) comme schématisé dans la Figure 9. Ainsi, ici seules les Images produites à la fin du mois de janvier sont conservées.

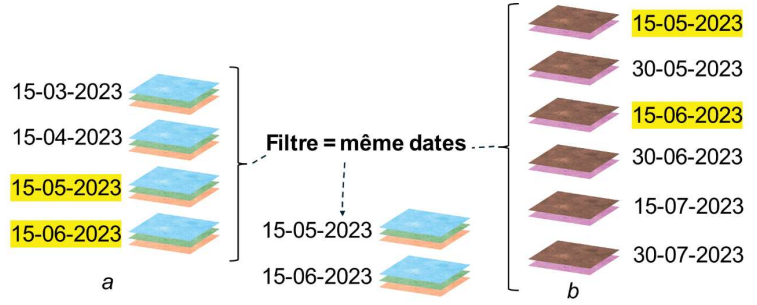


Figure 9 Exemple de l'utilisation d'un join simple (`ee.Join.simple()`) avec un filtre égal entre deux ImageCollections. Ici, seules les Images de la première ImageCollection qui partagent les mêmes dates que des Images de la seconde ImageCollection seront sélectionnées.

```
var joinsimple = ee.Join.simple().apply(MODISEVI, MODIS, filtre);
print(joinsimple);
```

La fonction `ee.Join.inner()` va combiner les deux ImageCollections selon le filtre utilisé et va produire une FeatureCollection avec des Images provenant des deux ImageCollections (ici, MODISEVI et MODIS ; Figure 10). En consultant les données imprimées par `print(joininner)`, il est clair que les propriétés de chaque Feature sont en fait les Images d'intérêt.

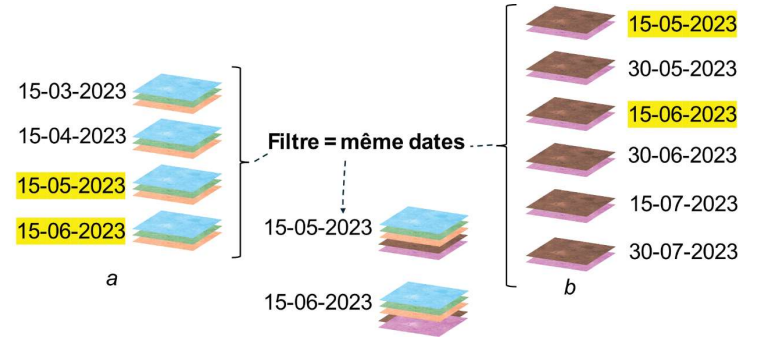


Figure 10 Exemple d'un join de type `ee.Join.inner` qui utilise le fitre égal pour sélectionner les Images de deux collections qui partagent la même date.

```
var joininner = ee.Join.inner().apply(MODISEVI, MODIS, filtre);
print(joininner);
```

Pour transformer cette FeatureCollection en ImageCollection, une fonction peut être utilisée :

```
var joininnerMODIS = joininner.map(function(feature) {
```

```
    return ee.Image.cat(feature.get('primary'),
feature.get('secondary'));
});
print(joininnerMODIS);
```

Ici, pour chaque Feature de la FeatureCollection, les propriétés sont utilisées pour retourner une Image combinant les bandes provenant de MODISEVI et MODIS. La fonction `ee.Image.cat()` combine plusieurs Image en une seule Image.

Jointure spatiale

Les jointures spatiales permettent de filtrer des FeatureCollections selon la localisation des Features.

Ici, nous importons deux jeux de données : la FeatureCollection « WDPApoints » reprend la localisation des aires protégées selon le jeu de données World Database on Protected Areas, et la FeatureCollection « Benin » reprend les limites administratives des départements du pays.

```
var WDPApoints = ee.FeatureCollection('WCMC/WDPA/current/points');
var Benin = ee.FeatureCollection('FAO/GAUL/2015/level1')
    .filter(ee.Filter.eq('ADM0_NAME', 'Benin'));
Map.addLayer(Benin);
Map.centerObject(Benin, 7);
```

Un filtre est créé pour filtrer les FeatureCollections selon leur localisation (propriété « .geo ») et leur intersection avec la fonction `ee.Filter.intersects()`.

```
var filtre = ee.Filter.intersects({
    leftField: '.geo',
    rightField: '.geo',
    maxError: 10
});
```

Nous utilisons la function `ee.Join.saveAll()` qui va stocker les Features de la collection secondaire dans la collection primaire s'ils touchent bien la collection primaire. Ces informations seront stockées sous le nom « pointsWDPA ».

```
var saveAllJoin = ee.Join.saveAll({
    matchesKey: 'pointsWDPA',
});
```

Cette fonction est utilisée avec le filtre créé précédemment. Ici, la FeatureCollection « Benin » est la collection primaire car elle est renseignée en premier dans la fonction `apply()`.

```
var WDPApointsauBenin = saveAllJoin.apply(Benin, WDPApoints,
filtre);
```



```
print(WDPApointsauBenin);
```

Si nous désirons avoir le résumé du nombre d'aires protégées par département, une fonction peut être créée pour obtenir le nombre de Features ajoutées avec le join précédent à la FeatureCollection « Benin » (où chaque Feature est un département). La fonction `size()` compte le nombre d'éléments dans la propriété « pointsWDPA » de chaque Feature. Cette valeur est ajoutée à chaque Feature sous le nom « nb_points ».

```
var WDPApointsauBenin_somme =
WDPApointsauBenin.map(function(departement) {
    var npoints = ee.List(departement.get('pointsWDPA')).size();
    return departement.set('nb_points', npoints);
});
print(WDPApointsauBenin_somme);
```

Pour facilement visualiser le nombre d'aires protégées par département, il est possible de créer un plot avec la fonction `ui.Chart.feature.byFeature()`. Ici, la propriété « ADM1_NAME » est utilisée pour l'axe x et la propriété « nb_points ». Le type de figure souhaitée doit être renseigné dans la fonction `setChartType()` et les noms des axes et le titre sont indiqués dans la fonction `setOptions()`.

```
var figure = ui.Chart.feature.byFeature(WDPApointsauBenin_somme,
'ADM1_NAME', 'nb_points')
    .setChartType('ColumnChart')
    .setOptions({
        title: 'Aires protégées par département',
        hAxis: {title: 'Département'},
        vAxis: {title: 'Nombre'}});
print(figure);
```

Séries temporelles

Cette section décrit les étapes pour visualiser le changement des valeurs de la réflectance dans le visible et de la précipitation au niveau d'un point (ici, Parakou).

Le point d'intérêt, « parakoupoint » est créé à partir de coordonnées longtitude-latitude.

```
var parakoupoint = ee.Geometry.Point(2.6167, 9.338);
Map.addLayer(parakoupoint);
Map.centerObject(parakoupoint, 12);
```

Les données Landsat 8 sont filtrées pour ne sélectionner que les Images produites en 2023 et incluant le point d'intérêt. Les bandes B4, B3, et B2 (rouge, vert, bleu) sont sélectionnées.

```
var L8filtre = ee.ImageCollection('LANDSAT/LC08/C02/T1_TOA')
    .filterDate('2023-01-01', '2024-01-01')
    .filterBounds(parakoupoint)
```

```
    .select(['B4', 'B3', 'B2']);
```

La fonction `ui.Chart.image.series()` permet de visualiser l'évolution de la valeur de plusieurs bandes au niveau d'un point d'intérêt (*region*) avec une ImageCollection. Un *reducer* doit être spécifié dans le cas où il existe plusieurs Image pour la même date : ici nous utilisons le *reducer* `ee.Reducer.first()` qui sélectionne la première Image pour chaque date (il n'existe qu'une seule Image par date dans l'ImageCollection « L8filtre » de toute façon). L'argument *scale* indique la résolution spatiale en mètre, ici de 30 mètres car ce sont des images Landsat 8.

```
var graph = ui.Chart.image.series({
   imageCollection: L8filtre,
   region: parakoupoint,
   reducer: ee.Reducer.first(),
   scale: 30
});
print(graph);
```

Dans certains cas, les données sont fournies avec une fine résolution temporelle qui doit être modifiée afin d'obtenir des moyennes ou sommes par mois. Dans cet exemple, nous transformons une ImageCollection avec une résolution de 5 jours concernant la précipitation (jeu de données CHIRPS Pentad) en des sommes mensuelles à partir d'une agrégation mensuelle. D'abord, l'ImageCollection est importée et filtrée pour ne garder que les Images de 2023 :

```
var chirps = ee.ImageCollection('UCSB-CHG/CHIRPS/PENTAD');
var debut = '2023-01-01';
var fin = '2024-01-01';
var chirps2023 = chirps.filter(ee.Filter.date(debut, fin))
print(chirps2023, 'chirps2023');
```

Il existe d'autres façons de manipuler les données de date comme ci-dessous. Ici, la variable « debut » est créée avec la fonction `ee.DatefromYMD()` qui prend trois arguments : l'année (Y), le mois (M), et le jour (D). La variable « fin » est créée à partir de la variable « debut » à l'aide de la fonction `advance()` qui permet d'ajouter des années (*year*), mois (*month*), ou jours (*day*).

```
var annee = 2023;
var debut = ee.Date.fromYMD(annee, 1, 1);
var fin = debut.advance(1, 'year');
var chirps2023_bis = chirps
   .filter(ee.Filter.date(debut, fin));
print(chirps2023_bis, 'chirps2023_bis');
```

Les objets EarthEngine ont leurs propriétés *system:time_start* et *system:time_end* en format nombre et non en format date. De plus, ce format nombre est en fait le nombre de

millisecondes depuis 01/01/1970 (une convention). Ainsi, la conversion de la date de début et de la date de fin en millisecondes depuis le 01/01/1970 avec la fonction `millis()` permet de facilement filtrer les objets EarthEngine.

```
print(debut, 'date de debut');
print(fin, 'date de fin');
print(debut.millis(), 'date de debut (millis)');
print(fin.millis(), 'date de fin (millis)');
```

Ainsi, cette fonction `millis()` va être incluse dans une fonction générale qui va convertir les dates de début et de fin (chaque mois) en millisecondes pour toute l'année 2023 et filtrer la collection « chirps2023_bis » pour appliquer le *reducer* somme (`ee.Reducer.sum()`) qui somme la précipitation entre ces deux dates (Figure 11).

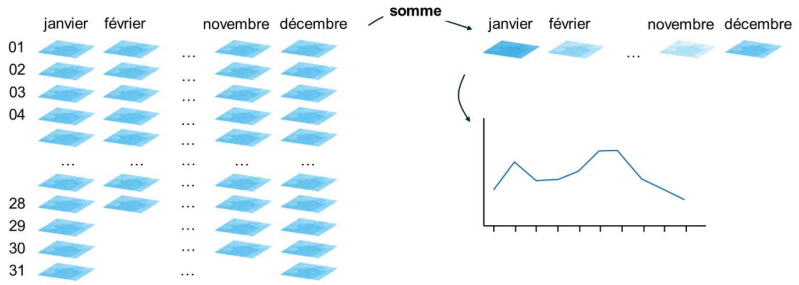


Figure 11 Schéma de l'utilisation d'une fonction produisant des sommes de valeurs par mois entre janvier et décembre.

```
var total_mensuel_fun = function(mois) {
   var debut = ee.Date.fromYMD(annee, mois, 1);
   var fin = debut.advance(1, 'month');
   var mois_filtre = chirps2023_bis
      .filter(ee.Filter.date(debut, fin));
   var total = mois_filtre.reduce(ee.Reducer.sum());
    return total.set({
      'system:time_start': debut.millis(),
      'system:time_end': fin.millis(),
      'year': annee,
      'month': mois
    });
};
```

Cette fonction est appliquée à une liste allant de 1 à 12 créée avec la fonction `ee.List.sequence()` qui compte du premier argument (1) au dernier argument (12). On obtient alors une liste de 12 Images qui correspondent aux 12 mois de l'année 2023.

```
var mois_liste = ee.List.sequence(1, 12);
```

```
var images_mensuelles = mois_liste.map(total_mensuel_fun);
print(images_mensuelles);
```

Cette liste est convertie en ImageCollection avec la fonction `ee.ImageCollection.fromImages()` :

```
var images_mensuelles_collection =
ee.ImageCollection.fromImages(images_mensuelles);
print(images_mensuelles_collection);
```

Enfin, le point « parakoupoint » est utilisé pour calculer la valeur moyenne de la précipitation totale pour chaque mois. Des options supplémentaires sont ajoutées au graphique avec les arguments *lineWidth* (épaisseur de la ligne) et *pointSize* (taille du point).

```
var graph = ui.Chart.image.series({
   imageCollection: images_mensuelles_collection,
   region: parakoupoint,
   reducer: ee.Reducer.mean(),
   scale: 5566,
}).setOptions({
   lineWidth: 1,
   pointSize: 3,
   title: 'Précipitation mensuelle pour Parakou (2023)',
   vAxis: {title: 'Précipitation (mm)'},
   hAxis: {title: 'mois'}
});
print(graph);
```

Détection de changement

La détection à l'aide d'indices de végétation et de seuils est simple à mettre en place sur GEE. Dans cet exemple, nous produisons des images NDVI à partir de données Landsat 8 et nous nous focalisons sur la forêt classée de Tchachou Gokana (dont le numéro WDPA est 33019) qui semble contenir une plantation d'arbres.

```
var plantation =
ee.FeatureCollection('WCMC/WDPA/current/polygons')
   .filter(ee.Filter.eq('WDPA_PID', '33019'));
Map.addLayer(plantation, {}, 'plantation');
Map.centerObject(plantation, 14);
```

La fonction `select()` permet à la fois de sélectionner des bandes d'intérêt et de les renommer : le premier argument est la liste des bandes à sélectionner, et le second argument est la liste des nouveaux noms pour ces bandes.

```
var landsat8 = ee.ImageCollection('LANDSAT/LC08/C02/T1_L2')
```



```
   .select(
    ['SR_B2', 'SR_B3', 'SR_B4', 'SR_B5', 'SR_B6', 'SR_B7'],
    ['blue', 'green', 'red', 'nir', 'swir1', 'swir2']
    );
```

Nous sélectionnons l'Image avec le moins de couverture nuageuse selon la propriété « CLOUD_COVER » pour les mois de février 2014 et 2023.

```
var L82014 = landsat8
   .filterBounds(plantation)
   .filterDate('2014-02-01', '2014-03-01')
   .sort('CLOUD_COVER', true)
   .first();
var L82023 = landsat8
   .filterBounds(plantation)
   .filterDate('2023-02-01', '2023-03-01')
   .sort('CLOUD_COVER', true)
   .first();
```

Le paramètre de visualisation se basant sur les bandes SWIR2, NIR, et RED est créé et appliqué aux deux Image « L82014 » et « L82023 ».

```
var visParam = {
   'bands': ['swir2', 'nir', 'red'],
   'min': 7750,
   'max': 22200
};
Map.addLayer(L82014, visParam, '2014');
Map.addLayer(L82023, visParam, '2023');
```

Pour chaque Image, nous produisons une nouvelle bande correspondant au NDVI à l'aide des bandes NIR et RED et de la fonction `normalizedDifference()`. Chaque Image produite ne contient qu'une bande qui est renommée avec la fonction `rename()`.

```
var NDVI2014 = L82014.normalizedDifference(['nir', 'red'])
   .rename('ndvi_2014');
var NDVI2023 = L82023.normalizedDifference(['nir', 'red'])
   .rename('ndvi_2023');
```

Ces deux Images sont utilisées pour calculer le delta NDVI qui est la différence entre le NDVI de 2023 et celui de 2014. Une valeur négative indiquera donc que le NDVI a diminué entre 2014 et 2023.

```
var deltaNDVI = NDVI2023.subtract(NDVI2014)
   .rename('deltaNDVI');
```

Un paramètre de visualisation allant du rouge (valeur négative) au vert (valeur positive) est créé pour visualiser la variation spatiale du delta NDVI.

```
var NDVIvis = {
   palette: ['red', 'beige', 'green'],
   min: -0.2,
   max: 0.2
};
Map.addLayer(deltaNDVI, NDVIvis, 'Changement en NDVI');
```

Cependant, une valeur faiblement négative n'est pas nécessairement synonyme de réelle chute du NDVI. Nous définissons ainsi un seuil de gain et un seuil de perte.

```
var seuilgain = 0.05;
var seuilperte = -0.05;
```

Ces seuils sont appliqués à l'Image du delta NDVI pour catégoriser le changement en gain (= 1) et perte (= 2). La première étape est de créer une Image vide (`ee.Image(0)`) puis de changer les valeurs de ses pixels où (fonction `where()`) le delta NDVI est en deçà du seuil négatif (-0.05 et donc valeur 2) ou au-dessus du seuil de gain (0.05 et donc valeur 1).

```
var changementclassifie = ee.Image(0);
var changementclassifie =
changementclassifie.where(deltaNDVI.lte(seuilperte), 2);
var changementclassifie =
changementclassifie.where(deltaNDVI.gte(seuilgain), 1);

var changeVis = {
   palette: ['blue', 'red'],
   min: 1,
   max: 2
};
Map.addLayer(changementclassifie.selfMask(),
   changeVis, 'changement lié aux seuils');
```

Il existe une méthode alternative qui ne requiert pas de passer par une Image vide. Ici, l'Image « deltaNDVI » est convertie en une Image binaire 0/1 où 1 est attribué aux pixels plus grands que le seuil de gain (fonction `gte()`). Ensuite, la fonction `where()` (*où*) remplace la valeur des pixels par 2 où l'Image « deltaNDVI » a une valeur en deçà ou égal (fonction `lte()`, *less than or equal*) au seuil de perte.

```
var methodealternative = deltaNDVI.gte(seuilgain)
.where(deltaNDVI.lte(seuilperte), 2);
Map.addLayer(methodealternative.selfMask(),
   changeVis, 'changement lié aux seuils');
```


Régressions

Regression linéaire simple

Il est simple de faire une régression linéaire simple sur GEE. Ici, nous importons d'abord deux jeux de données. MOD44B contient une couche qui renseigne sur le pourcentage de couverture forestière avec une résolution spatiale de 250 mètres pour une année. Le second jeu de données est celui de Landsat 8. Il faut aussi délimiter une zone d'étude, nommée « polygone », avec l'outil de création de polygones. Par exemple autour de la forêt de N'Dali. Les étapes suivantes indiquent comment prédire la couverture forestière à l'aide du NDVI calculé à partir d'images Landsat 8.

```
Map.centerObject(polygone, 9);
var mod44b = ee.ImageCollection('MODIS/006/MOD44B')
   .filterDate('2019-01-01', '2020-01-01')
   .first()
   .clip(polygone)
   .select('Percent_Tree_Cover');
print('Couverture 2019', mod44b_PTC);
Map.addLayer(mod44b_PTC, {max: 100}, 'Couverture 2019');
var L8 = ee.ImageCollection('LANDSAT/LC08/C02/T1_RT')
   .filterBounds(polygone)
   .filterDate('2019-02-01', '2019-03-01')
   .first();
var visL8 = {
   bands: ['B4', 'B3', 'B2'],
   max: 16000
};
Map.addLayer(L8, visL8, 'Image L8');
```

Le NDVI est produit à partir des données Landsat 8 et la bande est nommée « NDVI ».

```
var ndvi = L8.normalizedDifference(['B5', 'B4']).rename('NDVI');
```

Pour utiliser la fonction intégrant la régression linéaire, il faut d'abord que toutes les informations soient dans la même Image.

```
var ImageEntrainement = ndvi.addBands(mod44b_PTC);
print('Image d'entrainement :', ImageEntrainement);
```

La fonction pour établir la régression linéaire est en fait un *reducer*, `ee.Reducer.linearFit()`, qui doit être appliquée dans la fonction `reduceRegion()`. Ici, la fonction est appliquée sur le polygone créé précédemment avec une résolution spatiale de 30 mètres (*scale*). L'argument *bestEffort* permet d'indiquer à GEE qu'il peut retourner une valeur approximative lorsque la taille de l'échantillon dépasse ses limites de calcul.

```
var linearFit = ImageEntrainement.reduceRegion({
```

```
    reducer: ee.Reducer.linearFit(),
    geometry: polygone,
    scale: 30,
    bestEffort: true
});
print('Estimations :', linearFit);
print('Intercept :', linearFit.get('offset'));
print('Pente :', linearFit.get('scale'));
```

Il est alors possible d'utiliser les statistiques calculées pour produire une Image avec les prédictions de la couverture forestière à partir de l'Image NDVI. Ici, la fonction `expression()` prend deux arguments : le premier est la formule qui doit être appliquée avec le nom de différentes variables, le second argument est un dictionnaire avec la définition de chaque variable de l'expression définie dans le premier argument.

```
var predictionPTC = ndvi.expression(
    'intercept + slope * ndvi', {
      'ndvi': ndvi.select('NDVI'),
      'intercept': ee.Number(linearFit.get('offset')),
      'slope': ee.Number(linearFit.get('scale'))
    });
print('predictionPTC', predictionPTC);
Map.addLayer(predictionPTC, {max: 100}, 'Percent Tree Cover
prédit');
```

Régression linéaire multiple

Au lieu d'utiliser le NDVI comme variable indépendante, il est possible d'utiliser toutes les bandes de Landsat 8.

Une liste est créée avec le nom des variables indépendantes (les bandes Landsat 8, donc) :

```
var IndependantesBandes = ['B1', 'B2', 'B3', 'B4', 'B5', 'B6',
'B7', 'B10', 'B11'];
```

Le set d'entraînement est une Image qui doit contenir une bande avec une constante = 1 (`ee.Image(1)`), les variables indépendantes, et la variable dépendante depuis l'Image « mod44b_PTC ».

```
var ImageEntrainement = ee.Image(1)
    .addBands(L8.select(IndependantesBandes))
    .addBands(mod44b_PTC);
print(ImageEntrainement);
```

Comme précédemment, la régression linéaire multiple s'effectue à travers la fonction `reduceRegion()` où le *reducer* est alors `ee.Reducer.linearRegression()` et non

`ee.Reducer.linearFit()`. L'argument *numX* reprend le nombre de variables indépendantes et *numY* reprend le nombre de variables dépendantes.

```
var RegressionLineaire = ImageEntrainement.reduceRegion({
    reducer: ee.Reducer.linearRegression({
      numX: 10,
      numY: 1
    }),
    geometry: polygone,
    scale: 30,
    bestEffort: true
});
print('Résultat RegressionLineaire :', RegressionLineaire);
```

Ensuite, les résultats de la régression linéaire doivent être manipulés pour en extraire les coefficients (fonction `get()`) puis les transformer en Array puis en List. Cette liste contient alors les coefficients des différentes variables indépendantes.

```
var coeff = ee.Array(RegressionLineaire.get('coefficients'))
    .project([0])
    .toList();
```

L'application de ces coefficients se fait en combinant la valeur des bandes (fonction `addBands()`) avec celles des coefficients puis en les sommant.

```
var predictionRLM = ee.Image(1)
    .addBands(L8.select(IndependantesBandes))
    .multiply(ee.Image.constant(coeff))
    .reduce(ee.Reducer.sum())
    .rename('predictedTreeLR')
    .clip(L8.geometry());
Map.addLayer(predictionRLM, {
    min: 0,
    max: 10
}, 'Percent Tree Cover prédit par RLM');
```

Régressions non-linéaires

Les régressions non-linéaires ne sont pas implémentées dans les fonctions *reducer* mais dans les fonctions de type *classifier*. L'exemple ci-dessous est basé sur CART (*classification and regression tree*) qui permet d'effectuer une régression non-linéaire. Comme précédemment, nous commençons avec une Image possédant les variables indépendantes et dépendantes :

```
var CARTEntrainementImage =
ee.Image(L8.select(IndependantesBandes))
    .addBands(mod44b_PTC);
```

Ensuite, un échantillonnage des différentes bandes de cette Image est effectué à partir de la fonction `sample()` qui est délimitée par la région d'intérêt (polygone) et à 1500 pixels. L'argument *seed* est une manière de débuter l'échantillonnage de manière aléatoire.

```
var CARTEntrainement = CARTEntrainementImage.sample({
   region: polygone,
   scale: 30,
   numPixels: 1500,
   seed: 5
});
print('CART Entrainement', CARTEntrainement);
```

Le *classifier* `ee.Classifier.smileCart()` est appliqué en précisant la fonction `setOutputMode()` comme étant une régression et en indiquant les données d'entraînement. Ici, l'argument *features* est l'échantillon, *classProperty* est la valeur à prédire avec les *inputProperties* (les variables indépendantes).

```
var cartRegression = ee.Classifier.smileCart()
   .setOutputMode('REGRESSION')
   .train({
      features: CARTEntrainement,
      classProperty: 'Percent_Tree_Cover',
      inputProperties: IndependantesBandes
   });
```

L'objet produit est ainsi utilisé pour prédire le pourcentage de couverture forestière.

```
var cartRegressionImage = L8.select(IndependantesBandes)
   .classify(cartRegression, 'cartRegression');
Map.addLayer(cartRegressionImage, {
   min: 0,
   max: 10
}, 'Percent Tree Cover CART');
```

L'ensemble suivant permet de calculer la valeur du RMSE afin d'estimer la qualité de l'estimation de la couverture forestière. Le RMSE se calcule de la façon suivante :

$$RMSE = \sqrt{\frac{\sum(Prédite_i - Observée_i)^2}{n}}$$

```
var obspred = ee.Image.cat(mod44b_PTC, predictionPTC,
   predictionRLM, cartRegressionImage)
   .rename(['TCpercent', 'LFprediction',
      'RLMprediction', 'CARTprediction'
   ]);
var echantillonRMSE = obspred.sample({
   region: polygone,
   scale: 30,
   numPixels: 500,
   seed: 5
});
print('Premier échantillon', echantillonRMSE.first());
var calculDiff = function(feature) {
   var TCpercent = ee.Number(feature.get('TCpercent'));
   var diffLF2 = ee.Number(feature.get('LFprediction'))
      .subtract(TCpercent).pow(2);
   var diffRLM2 = ee.Number(feature.get('RLMprediction'))
      .subtract(TCpercent).pow(2);
   var diffCART2 = ee.Number(feature.get('CARTprediction'))
      .subtract(TCpercent).pow(2);
return feature.set('diffLF2', diffLF2).set('diffRLM2', diffRLM2)
      .set('diffCART2', diffCART2)
};
print(echantillonRMSE.map(calculDiff));
var RMSECART = echantillonRMSE.map(calculDiff)
   .aggregate_mean('diffCART2').sqrt();
print('Valeur RMSECART :', RMSECART);
```

Tendances

La détermination de tendances est simple à mettre en place sur GEE. Les séries temporelles sont souvent sous la forme d'ImageCollection, cependant, le nombre d'observations par pixel varie car il y a des nuages, etc. GEE permet d'ajuster une régression linéaire par pixel (Figure 12) de la même manière que la fonction regress dans le package *terra* de R. Ceci permet de suivre des dynamiques très différentes au sein d'une même région. Par exemple, pour la tendance du NDVI, la formule sera :

$$NDVI = a \times date + b$$

Où la date peut être en jours, millisecondes (fonction millis()), etc).

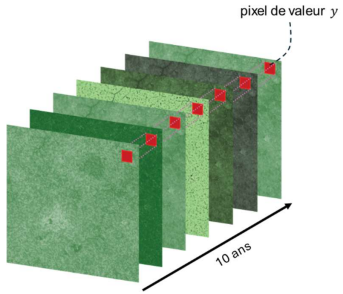


Figure 12 La tendance est calculée pour chaque pixel dans GEE.

Ainsi, comme la régression sera ajustée à chaque pixel, chaque pixel aura ses propres valeurs de a et b. Sur GEE, ces données seront stockées sur une Image avec une couche de type Array qui permet de stocker plusieurs valeurs par pixel sans passer par plusieurs bandes (Figure 13) :

Figure 13 Conversion d'une Image à trois bandes en une Image à une bande avec des valeurs de type array (trois valeurs). Source : Google Earth Engine.

Enfin, il est possible de supprimer la tendance d'un paramètre (a, Figure 14) pour en étudier ensuite la saisonnalité (b) :

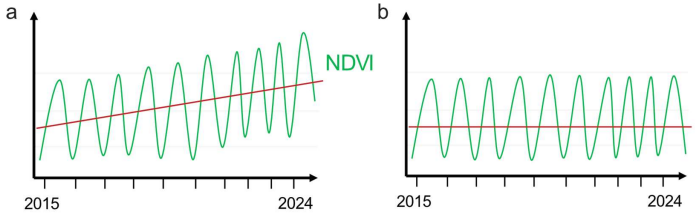


Figure 14 La suppression de la tendance permet de mieux apprécier la variation intra-annuelle.

Tendance pour un point

La première étape est de créer un point appelé « pointdinteret » avec l'outil de création de point. Ce point doit être localisé à l'endroit dont la dynamique du NDVI doit être suivie. Les données Sentinel-2 vont être utilisées (COPERNICUS/S2_HARMONIZED), il est donc nécessaire d'utiliser la fonction `maskS2clouds()` qui est disponible sur la page du jeu de données. La date de chaque Image doit être conservée, il faut ainsi changer la dernière ligne de la fonction `maskS2clouds()` en :

```
return image.updateMask(mask).divide(10000).copyProperties(image,
['system:time_start']);
```

Puis le jeu de données peut être importé comme :

```
var S2_NDVI = ee.ImageCollection('COPERNICUS/S2_HARMONIZED')
    .filterDate('2015-01-01', '2025-01-01')
    .filter(ee.Filter.lt('CLOUDY_PIXEL_PERCENTAGE', 20))
    .filterBounds(pointdinteret)
    .map(maskS2clouds)
```


Cependant, nous ajoutons une fonction à la fin de cette instruction. Cette fonction est appliquée à chaque Image et va produire une Image avec une bande NDVI, une bande avec l'année, une bande avec une valeur constante de 1, et garder la propriété « system:time_start » de l'Image d'origine (c'est-à-dire la date).

```
    .map(function(image) {
    var date = ee.Date(image.get('system:time_start'));
    var years = date.difference(ee.Date('1970-01-01'), 'year');
    return image.normalizedDifference(['B8', 'B4']).rename('ndvi')
    .addBands(ee.Image(years).rename('t')).float()
    .addBands(ee.Image.constant(1))
    .copyProperties(image, ['system:time_start']);
});
print(S2_NDVI);
```

Cette ImageCollection peut ainsi être utilisée pour afficher un plot de l'évolution du NDVI au cours du temps au niveau de ce point. Une nouvelle option est utilisée dans `setOptions()` : l'argument *trendlines* permet d'ajouter une droite de tendance de type linéaire (*linear*) ou polynomiale (*polynomial*) ou exponentielle (*exponential*). L'argument *visibleInLegend* permet d'indiquer si la légende doit aussi afficher les caractéristiques de cette ligne (soit *true* ou *false*).

```
var S2NDVIplot = ui.Chart.image.series({
    imageCollection: S2_NDVI.select('ndvi'),
    region: pointdinteret
})
    .setChartType('ScatterChart')
    .setOptions({
       title: 'NDVI au point dintéret 2015-2024 + tendance',
       trendlines: {0: {type: 'linear', color: 'red',
          visibleInLegend: true}},
       lineWidth: 1,
       pointSize: 3,
   });
print(S2NDVIplot);
```

Correction de la tendance au sein d'une ImageCollection

La correction de la tendance au sein d'une ImageCollection nécessite que les coefficients et résidus soient calculés pour chaque pixel. A partir de l'ImageCollection « S2_NDVI » produite précédemment, nous sélectionnons les bandes d'intérêts que sont les coefficients, résidus, et le NDVI. Ensuite, le *reducer* `ee.Reducer.linearRegression()` est utilisé avec deux variables indépendantes (*constant* et *t*) et une variable dépendante (1).

```
var independantes = ee.List(['constant', 't']);
var dependante = ee.String('ndvi');
var tendance = S2_NDVI.select(independantes.add(dependante))
```

```
   .reduce(ee.Reducer.linearRegression(independantes.length(),
1));
Map.addLayer(tendance, {}, 'tendance en array image', false);
print(tendance);
```

L'Image « tendance » a une bande « coefficients » avec des données de type Array avec deux dimensions. Cette bande est convertie en une Image a deux bandes, qui sont plus simples à utiliser par la suite :

```
var coefficients = tendance.select('coefficients')
   .arrayProject([0])
   .arrayFlatten([independantes]);
print(coefficients);
```

La suppression de la tendance se fait alors en corrigeant la valeur observée du NDVI en soustrayant (fonction `subtract()`) la valeur du NDVI prédite par la tendance. Le NDVI prédit par la tendance est : $coefficient_1 \times date + coefficient_2$.

```
var sanstendance = S2_NDVI.map(function(image)
{return image.select(dependante)
   .subtract(image.select(independantes).multiply(coefficients)
   .reduce('sum'))
   .copyProperties(image, ['system:time_start']);
});
var sanstendanceplot = ui.Chart.image.series(sanstendance,
pointdinteret, null, 20)
   .setOptions({
      title: 'Série du NDVI sans tendance',
      lineWidth: 1,
      pointSize: 3,
      trendlines: {0: {color: 'red'}},
   });
print(sanstendanceplot);
```

Périodicité

Une autre possibilité est de définir l'harmonique décrivant la périodicité du NDVI (série de Fourier, Figure 15). Ici, les valeurs du NDVI varient entre deux extrêmes (l'amplitude A) avec une fréquence donnée (ω).

Figure 15 Amplitude (A) et fréquence (ω) d'une fonction périodique.

$$NDVI = a \times date + b + c \times \cos(date \times 2\pi\omega) + d \times \sin(date \times 2\pi\omega)$$

On définit d'abord une List contenant le nom des coefficients :

```
var independantesharmonique = ee.List(['constant', 't', 'cos',
'sin']);
```

Ensuite, l'ImageCollection « S2_NDVI » est traitée avec une fonction qui produit les éléments de l'équation ci-dessus. La variable « timeRadians » est l'équivalent de la partie $date \times 2\pi\omega$. L'expression Math.PI est en fait la valeur de pi dans GEE.

```
var harmoniqueS2_NDVI = S2_NDVI.map(function(image)
   {var timeRadians = image.select('t').multiply(2 * Math.PI);
   return image.addBands(timeRadians.cos().rename('cos'))
   .addBands(timeRadians.sin().rename('sin'));
});
```

Une fois ces coefficients calculés, ils sont combinés avec la variable dépendante et traités à partir du *reducer* `ee.Reducer.linearRegression()` où les 4 premières bandes correspondent aux variables indépendantes et la dernière bande correspond à la variable dépendante.

```
var modelharmonique = harmoniqueS2_NDVI
   .select(independantesharmonique.add(dependante))
.reduce(ee.Reducer.linearRegression(independantesharmonique.length
(), 1));
```

Comme précédemment, l'Image « modelharmonique » a une bande « coefficients » avec des données de type Array avec deux dimensions. Cette bande est convertie en une Image a deux bandes, qui sont plus simples à utiliser par la suite :

```
var coeffmodelharmonique = modelharmonique.select('coefficients')
   .arrayProject([0])
   .arrayFlatten([independantesharmonique]);
```

Enfin, on applique les coefficients calculés à l'aide de la fonction `multiply()` puis on somme les bandes à l'aide la fonction `reduce()` (*sum*) pour arriver à l'estimation du NDVI sur chaque Image et donc date comme dans l'équation précédente. Le plot affiché comprend deux bandes : le NDVI mesuré sur les Image (*ndvi*) et celui estimé selon le modèle (*ndvi_model*). Deux couleurs sont spécifiées à l'aide d'un code hexadécimal.

```
var imageharmonique = harmoniqueS2_NDVI.map(function(image) {
   return image.addBands(
      image.select(independantesharmonique)
      .multiply(coeffmodelharmonique)
      .reduce('sum')
```

```
      .rename('ndvi_model'));
});
var harmoniqueplot = ui.Chart.image.series(
   imageharmonique.select(['ndvi_model', 'ndvi']), pointdinteret,
   ee.Reducer.mean(), 30)
   .setSeriesNames(['ndvi', 'ndvi_model'])
   .setOptions({colors: ['3366cc', 'cb0000']}
);
print(harmoniqueplot);
```

Collaborer sur GEE

Partage de scripts

Une copie du script ouvert peut être partagée à l'aide du bouton Get Link en haut de l'éditeur de code. Cette fonction permet de créer un lien qui contiendra ce même script et qui pourra être ouvert par d'autres utilisateurs de GEE. Cependant, comme précisé par GEE, ce lien ne permet pas de voir les modifications intervenues après la création du lien, et ne permet pas non plus aux autres utilisateurs de mettre ce code à jour.

Partage de données (assets)

Le partage d'assets importés par l'utilisateur est simple et se fait à partir de l'onglet Assets du volet de gauche. Il faut alors survoler l'asset à partager et cliquer sur . La fenêtre qui s'affiche offre la possibilité de partager l'asset avec un utilisateur précis (à partir de l'email) en mode lecteur (*reader*) ou auteur (*writer*, ce qui permet de modifier l'asset). Si *Anyone can read* est coché, alors n'importe qui aura droit de lire cet asset.

Pour utiliser l'asset d'un autre utilisateur, il faut alors l'importer dans l'éditeur de code comme les autres Image et Feature du catalogue Earth Engine, comme :

```
var carteOS = ee.Image('projects/ee-
jauneatemps/assets/pastoralism/OBSYDYA_OS_2023_L1_Awanla');
```

Ceci fonctionne de la même manière pour les données partagées dans les articles scientifiques (par exemple https://doi.org/10.1016/j.rse.2024.114380) ou depuis des catalogues autres que celui de Earth Engine (https://gee-community-catalog.org/projects/).

Partage de dossiers

Pour partager un *repository* (dossier avec plusieurs scripts), cela se fait dans l'onglet Scripts en survolant le nom du dossier à partager et en cliquant sur . Comme précédemment, le *repository* peut-être partagé avec des utilisateurs spécifiques et ces utilisateurs peuvent être soit lecteur ou auteur.

Suivre les modifications des scripts

L'historique des versions d'un dossier ou d'un script est disponible en survolant leur nom dans l'onglet Script et en cliquant sur . Pour les scripts, les modifications sont indiquées en couleur en cliquant sur le bouton Compare. Attention concernant le bouton Revert qui permet de revenir en arrière.

Importer et créer des modules

Importer des codes déjà produits et sauvegardés dans d'autres scripts permet de gagner du temps et de limiter la taille des scripts. Ceci se fait à partir de la fonction `require()`.

Par exemple : utiliser une palette de couleur produite par d'autres utilisateurs pour afficher un modèle numérique de terrain. D'abord, ouvrir le lien https://code.earthengine.google.com/?accept_repo=users/gena/packages et importer son contenu. Une fois importé, il se trouvera dans un dossier « users/gena/packages » dans la section Reader de l'onglet Script. En ouvrant le script « palettes », on peut accéder à de nombreuses palettes de couleurs comme « cmocean » qui en contient de nombreuses (« Thermal », « Haline », etc).

```
var mnt = ee.Image('USGS/SRTMGL1_003');
var palettes = require('users/gena/packages:palettes');
Map.addLayer(mnt, {
   min: 0,
   max: 3000,
   palette: palettes.cmocean.Algae[7]
}, 'cmocean Algae[7]');
```

Ici la variable « palettes » est bien basée sur le script importé précédemment avec la fonction `require()`. Cette variable est ensuite utilisée pour en extraire la palette Algae avec 7 couleurs.

La création d'un module et son application passent par la création d'un nouveau script. Par exemple, pour créer un module reprenant la fonction `maskS2clouds()` souvent utilisée pour identifier et retirer les nuages sur les images Sentinel-2 (voir la page du produit COPERNICUS/S2_SR_HARMONIZED par exemple), il faudra coller cette fonction dans le nouveau script mais au lieu de commencer l'instruction par var, celle-ci commencera par exports. Tel que :

```
exports.maskS2clouds = function (image) {
   var qa = image.select('QA60');
   // Bits 10 and 11 are clouds and cirrus, respectively.
   var cloudBitMask = 1 << 10;
   var cirrusBitMask = 1 << 11;
   // Both flags should be set to zero, indicating clear
conditions.
   var mask = qa.bitwiseAnd(cloudBitMask).eq(0)
      .and(qa.bitwiseAnd(cirrusBitMask).eq(0));
```

```
    return image.updateMask(mask).divide(10000);
};
```

Ce script est alors sauvegardé et nommé « S2masquenuage ». Pour l'utiliser dans un autre script, il sera appelé dans une nouvelle variable avec require où « users/.../nom_du_repository :S2masquenuage » est bien l'adresse du script contenant la fonction :

```
var myCloudFunctions = require(
'users/jauneatemps/cours2025:S2masquenuage');
var S2 = ee.ImageCollection('COPERNICUS/S2_HARMONIZED')
    .filterDate('2024-10-01', '2024-12-01')
    .filterBounds(geometry)
    .filter(ee.Filter.lt('CLOUDY_PIXEL_PERCENTAGE', 20))
    .map(myCloudFunctions.maskS2clouds);
var rgbVis = {
    min: 0.0,
    max: 0.3,
    bands: ['B4', 'B3', 'B2'],
};
Map.addLayer(S2.first(), rgbVis, 'S2 sans nuages');
```

Changer de résolution et projection

GEE gère la résolution des images selon les critères qui lui sont demandés. Si ces critères ne sont pas spécifiés dans le script, telle qu'avec l'argument *scale*, alors GEE se base sur le zoom de la carte. Ceci se manifeste lorsque la carte est déplacée et que l'utilisateur change de zoom. Ce point est particulièrement important lors du calcul de statistiques, etc. car ce calcul sera influencé par la résolution des données. En effet, GEE stocke les données importées selon une pyramide d'images allant de la résolution de l'image initiale (niveau 0) à des résolutions moins détaillées (Figure 16).

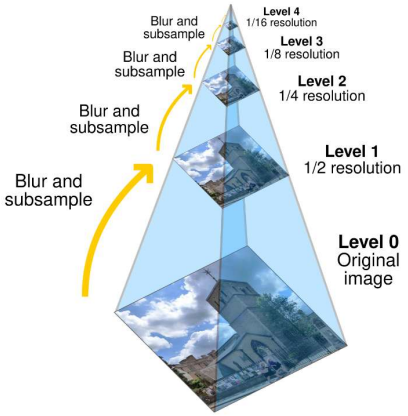


Figure 16 Schéma du concept de pyramide en traitement d'image. Le niveau 0 correspond à l'image originale et des étapes de rééchantillonnage permettent de produire d'autres niveaux de la pyramide. Source : Wikipedia (EN).

Ceci passe par une étape de *resampling* qui est par ailleurs paramétrable lors de l'importation de rasters. Le *resampling* peut se baser sur des variables continues, ainsi la valeur d'un pixel de faible résolution spatiale (niveau 3 par exemple) se basera sur une moyenne des valeurs des multiples pixels du niveau 0. Mais le *resampling* peut aussi utiliser des variables discrètes ainsi la nouvelle valeur sera une de celles déjà existantes au niveau inférieur.

Lorsque GEE applique les instructions contenues dans un script, il identifiera le niveau de la pyramide le plus proche du *scale* demandé puis appliquera une fonction de resampling qui est par défaut le plus proche voisin. Pour utiliser une autre méthode celle-ci doit être alors précisée avec la fonction `resample()` tel que :

```
Image.resample('bilinear'); // 'bicubic'
```

Cette méthode sera ainsi utilisée à la prochaine reprojection ou changement de résolution. La gestion des projections dans GEE peut se faire simplement à la fin d'un processus par exemple lors de l'extraction de données et ou lors du calcul de statistiques.

Trois points importants sont à prendre en compte :

- SI une Image est ajoutée à la carte avec une résolution trop fine par rapport au niveau du zoom (très grande zone mais résolution très fine) cela peut demander trop de calculs à GEE. Ces calculs seront réalisables lors d'une extraction vers Google Drive, mais pas forcément pour afficher une carte dans le volet carte.
- Si les instructions de la fonction reproject sont basées sur une Image, cette fonction sera affectée par le niveau de zoom.
- Il n'est pas recommandé d'accumuler les reprojections.

De plus, lorsque plusieurs images avec des projections différentes sont combinées, GEE utilise WGS84 avec une résolution de 1°. Il faut ainsi bien spécifier les projection et résolution attendues.

Exemple du calcul de l'indice d'hémérobie

Cet exemple est basé sur le travail de Mikhaïl Padonou du Projet OBSYDYA. L'indice d'hémérobie permet d'évaluer le degré d'impact des activités humaines sur l'environnement en attribuant des degrés différents aux classes d'occupation du sol. Les cultures ont ainsi un degré d'hémérobie selon ce que leur culture nécessite et leur impact sur l'environnement.

```
var carteOS = ee.Image('projects/ee-
jauneatemps/assets/pastoralism/OBSYDYA_OS_2023_L1_Awanla')
var classes = [11, 12, 13, 14, 21, 22, 31, 41, 42, 51, 52, 61]
var ind_hemerob = [4, 4, 5, 5, 3, 3, 2, 2, 2, 7, 6, 3]
```

```
var carteOShemerob = carteOS.remap({
   from: classes,
   to: ind_hemerob,
   defaultValue: 0
});
var hemerobVis = {
   min: 1, max: 7,
   palette: ['#1a9850', '#91cf60', '#d9ef8b', '#ffffbf',
'#fee08b', '#fc8d59', '#d73027']
};
Map.addLayer(carteOShemerob, hemerobVis);
```

```
var hemerobie_100 = carteOShemerob.reduceResolution({
   reducer: ee.Reducer.mean(),
   maxPixels: 65536
}).reproject({
   crs: 'EPSG:32631',
   scale: 100
});
Map.addLayer(hemerobie_100, hemerobVis);
```

```
var hemerobie_100_2 = hemerobie_100.reduceResolution({
   reducer: ee.Reducer.mean(),
   maxPixels: 65536
});
Export.image.toDrive({
   image: hemerobie_100_2,
   region: site_etude,
   scale: 1000,
   fileFormat: 'GeoTIFF'
});
```


Client et serveur, exécution différée

Dans l'interface en ligne de GEE, les objets sont soit des objets Javascript au niveau du client (le navigateur), soit des objets EarthEngine au niveau du serveur (reconnaissables car créés avec une fonction commençant par `ee.`). Les objets EarthEngine ne sont pas connus par le client sauf lorsqu'il demandé avec une fonction telle que `print()`. Ainsi, tous les calculs peuvent se dérouler du côté serveur si tous les objets sont de type EarthEngine. La fonction `getInfo()` vue précédemment est une requête au niveau du serveur pour extraire une valeur précise. Cette requête bloque ainsi le reste du calcul le temps que la valeur soit calculée et peut ainsi fortement ralentir le déroulement des calculs par GEE.

C'est pour cela que les boucles de type `for i` ne sont pas recommandées car elles sont exécutées sur le client, tandis que la fonction `map()` permet de tout effectuer au niveau du serveur. Ceci est similaire pour les fonctions `ee.Algorithms.If()` et `Filter()` qui remplacent `if else`.

Dans GEE, l'exécution du code est différée car le code, une fois encodé par le client et envoyé vers le serveur, n'est exécuté que lorsque quelque chose est demandé au niveau du client (print, afficher une image, ... voir Figure 17). Ceci permet la parallélisation des tâches et rend l'utilisation de GEE plus efficace.

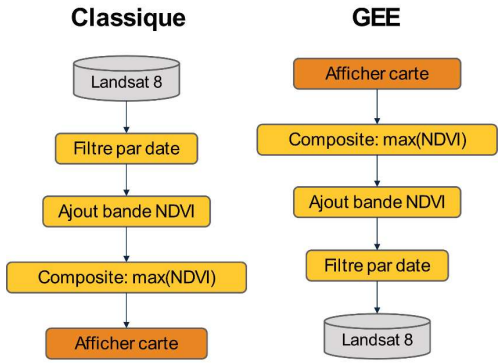


Figure 17 Les étapes de manipulation des données sont différentes des méthodes classiques lorsqu'on utilise GEE. Source : Michael DeWitt & Katie Friis, Geo For Good (2022).

Tendance non paramétrique

Le test de Mann-Kendall peut être implémenté sur GEE pour tester si une variable a une tendance monotone en fonction du temps. Ce test est la somme des signes des différences entre des observations consécutives :

$$S = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} signe(x_j - x_i)$$

La statistique tau est calculée à partir de S et est comprise entre -1 et 1 :

$$\tau = \frac{2S}{n(n-1)}$$

L'exemple suivant montre le calcul de ces statistiques avec la fonction `ee.reducer.kendallsCorrelation()` à partir d'images NDVI produites à partir des données MODIS pour un polygone créé et appelé « site ».

```
var NDVI = ee.ImageCollection('MODIS/061/MOD13A3')
   .filter(ee.Filter.calendarRange(2001, 2024, 'year'))
   .filter(ee.Filter.calendarRange(12, 12, 'month'))
   .filterBounds(site)
   .select('NDVI');
print(NDVI.first());
Map.centerObject(site, 12);
var NDVItau = NDVI.reduce(ee.Reducer.kendallsCorrelation());
print(NDVItau);
var vistau = {min: -1, max: 1,
   palette: ['red', 'white', 'green']};
Map.addLayer(NDVItau.select('NDVI_tau'), vistau);
```

Cependant, GEE ne calcule par de valeur de *p*, et il n'est ainsi pas possible de directement identifier les pixels où la tendance est significative. Ceci peut se faire en calculant la statistique Z qui est une approximation où *n* est le nombre d'observations sur la période d'étude (valeur calculée avec la fonction `count()`):

$$Z = \frac{3\tau\sqrt{n(n-1)}}{\sqrt{2(2n+5)}}$$

```
var count = NDVI.count().rename('n');
var NDVItau = NDVItau.addBands(count);
var Zformule = '(3 * Ntau * sqrt(nobs * (nobs - 1))) / (sqrt(2 *
((2 * nobs) + 5)))';
var Z = ee.Image().expression({
   expression: Zformule,
   map:  {Ntau: NDVItau.select('NDVI_tau'),
      nobs: NDVItau.select('n')}
}).rename('Z');
Map.addLayer(Z.select('Z'));
```

Selon le tableau des scores Z, la tendance est significative si Z > 1,64 ou si Z < -1,64. Ainsi, les pixels avec une tendance significative sont représentés en prenant la valeur absolue de Z avec la fonction `abs()` et en masquant les pixels dont la valeur absolue de Z est < 1,64 :

```
Map.addLayer(NDVItau.select('NDVI_tau')
   .updateMask(Z.abs().gte(1.64)), vistau);
```


A noter que GEE permet d'estimer la pente de la droite de régression avec la méthode de Sen en utilisant le *reducer* `ee.Reducer.sensSlope()`.

Les « Kernels » (noyaux)

Les noyaux (ou *kernel*) sont utilisés pour transformer des pixels en se basant sur le voisinage. Cet outil permet de traiter des outliers ou l'absence de données, mais est aussi utilisé pour effectuer du lissage à partir de moyennes glissantes. Dans GEE, un kernel est définit par sa taille, sa forme, et le poids des différents pixels du voisinage. Ceci est visible utilisant la fonction `print()` qui affichera une matrice avec le poids des pixels :

```
print('Kernel uniforme', ee.Kernel.square(2));
```

L'effet des kernels est particulièrement visible lors du lissage d'images hautes résolutions comme celles de Planet. Créer un polygone au niveau de l'Université de Parakou et appelé « UP » :

```
var nicfi = ee.ImageCollection('projects/planet-
nicfi/assets/basemaps/africa')
   .filter(ee.Filter.date('2021-10-01','2021-12-01'))
   .first();
Map.centerObject(UP, 16);
var vis = {bands:['R','G','B'], min:64, max:5454, gamma:1.8};
Map.addLayer(nicfi, vis, '2021-1011');
```

```
var gaussianKernel = ee.Kernel.gaussian({
   radius: 5,
   units: 'meters',
});
var lisse = nicfi.convolve(gaussianKernel);
Map.addLayer(lisse, vis, '2021-1011 lissé');
```

```
var laplacianKernel = ee.Kernel.laplacian8();
print('Laplacian kernel :', laplacianKernel);
var contours = nicfi.convolve(laplacianKernel);
Map.addLayer(contours, vis, '2021-1011 contours');
```

La fonction `reduceNeighborhood()` permet d'appliquer un *reducer* à un ensemble de pixels à l'aide d'un kernel comme :

```
var mediane = nicfi.reduceNeighborhood({
   reducer: ee.Reducer.median(),
   kernel: uniformKernel
});
var vis = {bands:['R_median','G_median','B_median'],
```

```
min:64, max:5454, gamma:1.8};
Map.addLayer(mediane, vis, 'mediane');
```

Un exemple d'application d'un kernel est d'ajouter une zone tampon autour des zones occupée par du bâti sur une carte d'occupation du sol. La carte d'occupation du sol d'Awanla est d'abord importée avec une palette de couleur :

```
var carteOS = ee.Image('projects/ee-
jauneatemps/assets/pastoralism/OBSYDYA_OS_2023_L1_Awanla')
var couleursOS = ['eebc1d', 'f1e5ac', 'f531c1', 'bb270d',
'4cbb17', '40e0d0', '355e3b', '008000', '8a9a5b', 'ff0000',
'fff8dc', '0000ff'];
Map.addLayer(carteOS, {min: 11, max: 61, palette: couleursOS},
'carteOS', false);
```

Nous définissons ensuite un kernel carré avec un rayon de 20 mètres.

```
var uniformKernel = ee.Kernel.square({radius: 20, units:
'meters'});
```

La carte d'occupation du sol est transformée en une carte binaire 0/1 où 1 est la valeur des pixels qui correspondent à du bâti (valeur 51 sur la carte d'occupation du sol). Cette carte binaire est ensuite utilisée avec la fonction `reduceNeighborhood()` pour appliquer le kernel avec un *reducer* maximum (`ee.Reducer.max()`) pour établir une zone tampon autour des pixels = 1.

```
var carteOS51 = carteOS.eq(51);
var carteOS51_c = carteOS51.reduceNeighborhood({
   reducer: ee.Reducer.max(),
   kernel: uniformKernel
});
```

Cette Image est utilisée pour modifier la carte d'occupation du sol originale. Cependant les valeurs correspondant à du bâti sur carteOS51_c varient et ne correspondent pas à la valeur 51 de la carte d'occupation du sol originale. Ainsi, nous multiplions les valeurs de ces pixels par 100 pour obtenir des valeurs > 100. Ces valeurs sont ajoutées à la carte d'occupation du sol originale : les pixels ne correspondant pas à du bâti ne changent pas de valeur car sur carteOS51_c, leur valeur = 0. Enfin, nous appliquons un clamp avec une limite supérieure de 62 pour que tous ces pixels > 100 soient égaux à 62 (la nouvelle valeur pour le bâti).

```
var carteOS_2 = carteOS.add(carteOS51_c.multiply(100)).clamp(0,
62);
```

Mosaïques

Les mosaïques permettent de combiner de nombreuses images plus petites que la surface totale voulue et d'éviter la couverture nuageuse. Leur création a déjà été vue précédemment, cependant il est aussi possible d'utiliser des kernels pour lisser les nuages dans le cas où il n'est pas possible de trouver des scènes sans nuages comme pendant la saison des pluies.

```
var benin =
ee.FeatureCollection('FAO/GAUL/2015/level0').filter(ee.Filter.eq('
ADM0_NAME', 'Benin')).first();
var myCloudFunction = require(
   'users/jauneatemps/cours2025:S2masquenuage');
var S2 = ee.ImageCollection('COPERNICUS/S2_HARMONIZED')
   .filter(ee.Filter.calendarRange(2022, 2024, 'year'))
   .filter(ee.Filter.calendarRange(7, 9, 'month'))
   .filterBounds(benin.geometry())
   .filter(ee.Filter.lt('CLOUDY_PIXEL_PERCENTAGE', 20))
   .map(myCloudFunction.maskS2clouds);
var composite = S2.median().clip(benin);
var visS2 = {
   min: 0.0,
   max: 0.3,
   bands: ['B4', 'B3', 'B2'],
};
Map.centerObject(benin, 7);
Map.addLayer(composite, visS2, 'Benin composite');
```

Un kernel est alors défini avec un rayon de 300 mètres car l'image finale couvre une grande surface et car les nuages sont en général larges.

```
var uniformKernel = ee.Kernel.square({
   radius: 300,
   units: 'meters',
});
```

Pour lisser, le *reducer* médiane est utilisé et permettra ainsi d'éviter les valeurs extrêmes associées aux nuages et à leurs ombres.

```
var lisse = composite.reduceNeighborhood({
   reducer: ee.Reducer.median(),
   kernel: uniformKernel
});
var vis = {
   min: 0.0, max: 0.3,
   bands: ['B4_median', 'B3_median', 'B2_median'],
};
Map.addLayer(lisse, vis, 'Benin composite lissée');
```

Unmixing

Le démixage spectral (ou unmixing) permet des analyses à une échelle en deçà de celle du pixel. En effet, un pixel ne recouvre pas forcément une occupation du sol homogène et est bien souvent composé d'un mélange de type de surface (par exemple, eau, forêt et un peu de sol nu) comme sur la Figure 18. Pour cela, cette technique décompose un pixel mixte en une série de spectres (*endmembers*) et une série de valeurs d'abondance qui caractérisent chacun de ses endmembers. Evidemment, cela se base sur la signature spectrale de pixel couvrant une surface homogène (par exemple 100% forêt).

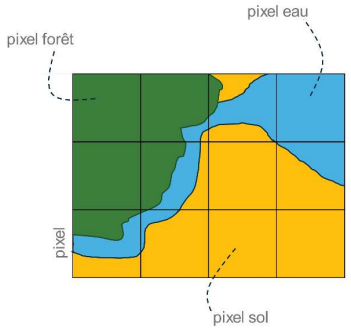


Figure 18 Les pixels ne correspondent pas tous à une surface unie. Plusieurs classes d'occupation du sol peuvent être retrouvées dans un même pixel.

Dans cet exemple, nous utilisons des images Sentinel-2 pour identifier la portion de différents types de surface par pixel. Comme dans les exemples précédents, l'importation des données se fait ainsi et la fonction `maskS2clouds()` est utilisée avec un polygone appelé « site » :

```
var S2 = ee.ImageCollection('COPERNICUS/S2_HARMONIZED')
    .filterDate('2022-01-01', '2022-01-31')
    .filter(ee.Filter.lt('CLOUDY_PIXEL_PERCENTAGE', 20))
    .filterBounds(site)
    .map(maskS2clouds);
var rgbVis = {min: 0.0, max: 0.3, bands: ['B4', 'B3', 'B2'],};
Map.addLayer(S2.median(), rgbVis, 'RGB');
```

Il faut ensuite créer une liste avec les noms des bandes d'intérêt dans les Image contenues dans S2.

```
var bandes = ['B2', 'B3', 'B4', 'B5', 'B6', 'B7', 'B8', 'B8A',
'B11', 'B12'];
```


La médiane des valeurs de chaque pixel sur la période d'étude (janvier 2022) est calculée pour chaque bande, et nous sélectionnons un sous-échantillon des bandes à partir de la liste créée juste avant.

```
var S2median = S2.median().select(bandes);
```

Comme lors de la classification supervisée, nous allons créer trois sets de points (à partir de l'outil de création de points sur la carte) correspondant aux catégories d'occupation du sol eau, forêt et sol. Ces geometries sont appelées « vegetation », « eau », et « sol » et doivent chacune contenir au moins 10 points. Ces points doivent être placé sur des surfaces homogènes correspondant au nom de la géométrie. Une fois créés, ces points sont utilisés avec la fonction `reduceRegion()` pour en extraire la valeur moyenne dans l'Image « S2median » et les valeurs sont ensuite collectées à partir de la fonction `values()`.

```
var solMean = S2median.reduceRegion(ee.Reducer.mean(), sol,
10).values();
var eauMean = S2median.reduceRegion(ee.Reducer.mean(), eau,
10).values();
var vegMean = S2median.reduceRegion(ee.Reducer.mean(), vegetation,
10).values();
print(solMean, 'sol mean');
```

Il est intéressant d'afficher ces données sur un plot. Cela est facilement réalisable avec la fonction `ui.Chart.image.regions()` qui combinera l'Image avec les valeurs médianes (« S2median ») et les points créés et combinés en une FeatureCollection. Les bandes sont extraites dans l'ordre dans lequel elles sont dans l'Image « S2median » et donc selon leur longueur d'onde. Ici, les valeurs de la longueur d'onde du début de chaque bande sont ajoutées comme label pour l'axe des ordonnées.

```
print(ui.Chart.image.regions(S2median, ee.FeatureCollection([
   ee.Feature(eau, {label: 'eau'}),
   ee.Feature(sol, {label: 'sol'}),
   ee.Feature(vegetation, {label: 'vegetation'})]),
   ee.Reducer.median(), 20, 'label',
    [496.6, 560, 664.5, 703.9, 740.2,
   782.5, 835.1, 864.8, 1613.7, 2202.4]));
```

Pour l'unmixing, il est nécessaire de combiner les trois listes de valeurs extraites en un array ou chaque bande aura trois valeurs (sol, forêt, eau).

```
var endmembers = ee.Array.cat([solMean, vegMean, eauMean], 1);
```

Cette étape est compliquée à visualiser : elle produit une Image avec une seule bande où chaque pixel contient un array contenant 10 valeurs, soit celles des 10 bandes de l'Image S2median.

```
var arrayImage = S2median.toArray().toArray(1);
```

Ce sont ces valeurs qui seront combinées avec celles des pixels homogènes (*endmembers*) pour en estimer la proportion de sol nu, forêt, et eau. Ici, `ee.Image(endmembers)` est la transformation des *endmembers* en une Image array avec 10x3 valeurs par pixel (par exemple [moyenne sol pour bande 2, moyenne forêt pour bande 2, moyenne eau pour bande 2]). La fonction `matrixSolve(arrayImage)` résoud l'équation $valeur\ Bande\ 2 = A \times x$.

```
var unmixed = ee.Image(endmembers).matrixSolve(arrayImage);
```

L'Image produite est avec une bande à deux dimensions, celle-ci doit être transformée en une Image à 3 bandes avec une dimension pour être affichée.

```
var unmixedImage = unmixed.arrayProject([0]).arrayFlatten([['sol',
'vegetation', 'eau']]);
Map.addLayer(unmixedImage, {}, 'fractions sol, vegetation, eau');
```

Le démixage des valeurs médianes des 10 bandes à l'aide des *endmembers* peut se faire pour estimer la fraction de chaque type de couverture du sol. Ici, la fonction `unmix()` prend deux arguments *true* afin que la somme des proportions = 1. Ceci est simpliste car il existe d'autres types de couverture du sol que l'eau, les forêts, et le sol nu.

```
var constrained = S2median.unmix([solMean, vegMean, eauMean],
true, true);
Map.addLayer(constrained, {}, 'fractions de chaque type');
```

Exporter de grands rasters

Il n'est souvent pas possible d'exporter un très grand raster unique à partir de GEE. Dans certains cas, GEE découpera le raster en plusieurs tuiles avec des noms différents. On peut prévenir cela en découpant le raster préalablement telle qu'en suivant la méthode (Figure 19) :

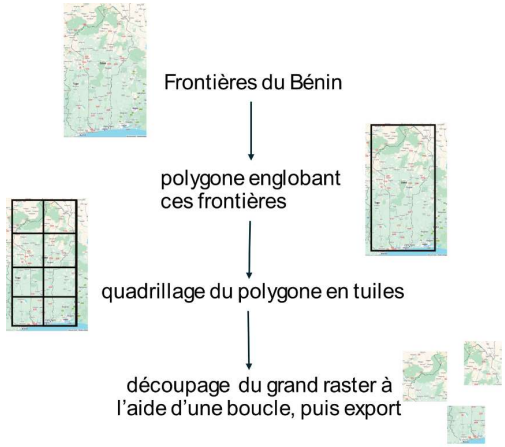


Figure 19 Les étapes de l'exportation d'un grand raster en plusieurs tuiles.

Pour commencer, nous importons les frontières du Bénin ainsi que l'ImageCollection ESA WorldCover qui est une carte de l'occupation du sol à l'échelle mondiale.

```
var benin = ee.FeatureCollection('FAO/GAUL/2015/level0')
   .filter(ee.Filter.eq('ADM0_NAME', 'Benin'))
   .first();
print(benin, 'Bénin');
var esaworldcover = ee.ImageCollection('ESA/WorldCover/v100')
   .first();
print(esaworldcover, 'ESA WorldCover');
```

Pour créer le polygone qui entoure les frontières du Bénin, il faut d'abord définir un crs. Le crs EPSG 32631 correspond à celui du Bénin et limite les distorsions.

```
var crs = 'EPSG:32631';
```

La fonction `geometry()` converti le Feature « benin » en objet de type geometry et la fonction `bounds()` produit un rectangle englobant l'entièreté du Bénin avec la projection définie préalablement et une erreur de maximum 1 mètre.

```
var bounds = benin.geometry().bounds({
   proj: crs,
   maxError: 1
});
```

On souhaite ensuite définir la taille du pixel lors de l'export (la résolution spatiale, en mètre) ainsi que la taille de la tuile qui doit être exportée. La taille de la grille (*gridSize*) est ainsi la taille de la tuile multipliée par la taille du pixel.

```
var pixelSize = 10;
var tileSize = 10000;
var gridSize = ee.Number(tileSize).multiply(pixelSize);
```

La fonction `coveringGrid()` produit alors une grille avec une taille de maille correspondant à *gridSize* défini avant. L'objet « grid » est ainsi une FeatureCollection.

```
var grid = bounds.coveringGrid({
   proj: crs,
   scale: gridSize
});
Map.addLayer(grid, {}, 'Grille');
```

La grille n'aura pas la même taille que le rectangle qui entoure les frontières du Bénin. En effet, les mailles de la grille ont une taille fixe définie précédemment et ainsi ne sont pas parfaitement ajustée à la taille du pays. Ainsi, les tuiles couvriront légèrement les pays voisins.

Afin de connaître le nombre de cellules contenues dans cette grille, il faut d'abord convertir la FeatureCollection en List puis en extraire la taille à partir de la fonction `getInfo()`. Contrairement à `print()`, cette fonction intervient du côté client et permet d'utiliser la valeur obtenue dans une boucle de type for loop.

```
var gridList = grid.toList(ee.Number(grid.size()));
var gridLength = ee.Number(gridList.size()).getInfo();
```

La boucle est définie ainsi et va aller de Feature (cellule) en Feature dans la List de Feature « gridList » pour en extraire la Feature *i*. Cette Feature est alors utilisée pour découper l'Image d'occupation du sol et pour l'exporter vers Google Drive avec la fonction `Export.image.toDrive()`. La boucle continue jusqu'à ce que *i* = le nombre de cellules dans la grille. GEE permet de nommer les fichiers de manière dynamique en combinant plusieurs mots et nombres avec l'expression « 'tuile' + i + 'esabenin' » (où *i* est évidemment un nombre). Les tâches d'export se trouvent dans l'onglet *Tasks* et doivent être lancées manuellement.

```
for (var i = 0; i < gridLength; i++) {
   var feature_i = gridList.get(i);
   var esabenincover_clip_i = esabenincover.clip(feature_i);
      Export.image.toDrive({
      image: esabenincover_clip_i,
      description: 'tuile' + i + 'esabenin',
      scale: 10,
      crs: 'EPSG:32631',
      fileNamePrefix: 'tuile' + i + 'esabenin'
```



```
   });
}
```

Utiliser GEE dans QGIS

GEE est accessible sur QGIS à travers le plugin « Google Earth Engine » de Donchyts et al. Ce plugin permet d'utiliser des scripts directement dans QGIS, cependant il se base sur du python et non du javascript. La documentation de GEE donne les exemples dans ces deux langages, bien que le catalogue n'ait pas toujours des informations en python.

Une fois installé, le plugin requiert d'indiquer le nom du projet EE (« ee-... » en haut à droite de la page https://code.earthengine.google.com/) et de valider l'accès sur la page web qui s'ouvre. Dans QGIS, la console python s'ouvre directement à partir du raccourci Ctrl+Alt+P (Windows) ou avec l'onglet Plugins > Python. L'éditeur de code s'ouvre dans la console avec le bouton et le code se lance avec le bouton .

Par exemple, pour afficher une carte, il faudra écrire :

```
import ee

from ee_plugin import Map

image = ee.Image('USGS/SRTMGL1_003')

vis_params = {

    'min': 0, 'max': 4000,

    'palette': ['006633', 'E5FFCC', '662A00', 'D8D8D8', 'F5F5F5']

}

Map.addLayer(image, vis_params, 'Digital Elevation Model')
```

Cependant, il n'est pas possible d'accéder directement aux données dans QGIS comme s'il s'agissait d'un fichier local. Il est seulement possible d'afficher les données, et elles s'actualiseront lorsque la carte est déplacée avec le curseur.

```
dataset = ee.FeatureCollection('FAO/GAUL/2015/level1')
   .filter(ee.Filter.eq('ADM0_NAME', 'Benin'))
   .filter(ee.Filter.eq('ADM1_NAME',
'Atakora')).first().geometry()

dem_clip = image.clip(dataset)

Map.centerObject(dem_clip)

Map.addLayer(dem_clip, vis_params, 'Atakora DEM')
```

Ainsi, il faudra passer par l'étape de l'export de données vers un dossier Google Drive comme ci-dessous (Figure 20) :

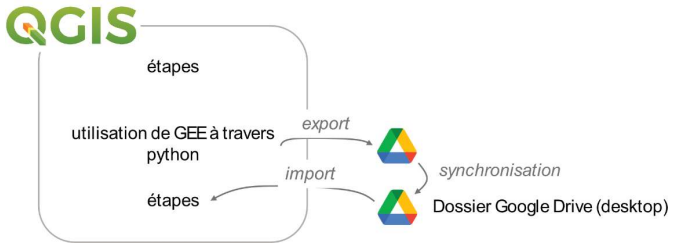


Figure 20 QGIS peut être utilisé en combinaison avec une exportation sur Google Drive puis une importation de ces données dans QGIS.

```
exportdem = ee.batch.Export.image.toDrive(
    image = dem_clip,
    description = 'Atakora_export',
    folder = 'UPGEE2025',
    region = dataset,
    scale = 500,
    crs ='EPSG:32631'
)
exportdem.start()
```

Debugging

Des bugs apparaîtront sans doute lors de l'utilisation de GEE. Les indications à gauche des lignes de code et les messages d'erreurs produits par GEE sont souvent un bon moyen pour identifier la cause des bugs. Toutefois, ces bugs ont souvent de nombreuses origines possibles, alors il est nécessaire d'utiliser les fonctions `print()`, l'outil Inspector, la fonction `Map.addLayer()`, l'exportation, etc. afin de détecter quelles sont les étapes qui doivent être corrigées. Les fonctions `first()`, `limit()`, et `filter()` permettent aussi de limiter la taille d'une collection qui devra être imprimée avec `print()` pour vérifier que les données sont correctes. Enfin, la plupart des erreurs rencontrées ont déjà été rapportées dans les forums en ligne, et une recherche Google permet souvent d'identifier des conversations qui indiquent des façons de résoudre ces problèmes.

Il y a de multiples erreurs courantes telles que :

- Mauvaise syntaxe comme le manque de crochets, guillemets qui ne sont pas fermés, nom erroné de l'argument
- Oublier de définir une variable avec « var variable1 =… »
- Utiliser une variable qui n'a pas encore été définie
- Ne pas créer un objet EE tel que ee.Image() alors qu'il est nécessaire de le faire
- Utiliser le mauvais nom d'une bande

De plus, il est déconseillé d'utiliser des boucles telles que `for i`, et la fonction `getInfo()` qui demande le calcul de données du côté client (ordinateur). Ainsi, GEE peut avoir des problèmes avec la quantité de données qu'il doit utiliser. Les messages tels que *User Memory Limit Exceeded*, *Computation Timeout*, *Too Many Aggregations*, ou *Internal Server Error* indiquent qu'il y a trop de données à traiter pour GEE.
Pour résoudre cela, il faut réduire la taille de la région, éliminer les calculs inutiles (par exemple des conversions inutiles), et exporter plutôt que de visualiser directement dans la carte.

Données Radar

L'imagerie radar est de type actif car elle nécessite la production d'un signal radio vers la cible puis la réception de la partie qui est réfléchie. Ainsi, cela permet de contrôler la longueur d'onde (par exemple, bandes C ou L) et la polarisation de ce signal. Ce type de télédétection à plusieurs avantages : il est possible de l'utiliser la nuit, il n'y a pas beaucoup d'effet de la couverture nuageuse (avec les bandes C et L), et il est possible de suivre les caractéristiques physiques de la surface sur la polarisation de l'onde. En mai 2025, il y a deux collections correspondant à des données radar sur GEE : celle de Sentinel-1 (bande C) et celle de PALSAR-2 ScanSAR (bande L). Ces données ont une double polarisation, avec une partie verticale (V) et une partie horizontale (H), et si le signal réfléchit ne change pas de polarisation, il est ainsi VV ou HH, mais lorsqu'il change de polarisation il est nommé VH ou HV.
Les données Sentinel-1 sur GEE sont de type produit GRD, cela correspond à seulement une partie des données produites à partir de Sentinel-1. Cependant, elles peuvent répondre à de nombreuses applications selon le wiki de Copernicus (https://sentiwiki.copernicus.eu/web/s1-applications).

Dans cet exemple, nous nous intéressons à la région de Parakou : un polygone doit être créé pour définir cette zone d'étude et appelé « parakou ». Ensuite, l'ImageCollection est filtrée plusieurs fois à partir des propriétés des Images : d'abord nous cherchons celles acquises en mode IW (Interferometric Wide Swath), ensuite nous sélectionnons les Images avec la polarisation VV ou VH (deux collections sont produites : « collectionVV » et « collectionVH »). La fonction `ee.Filter.listContains` est utilisée dans ce cas car la propriété « transmitterReceiverPolarisation » ne contient pas un mot seul (*String*) mais une liste. Enfin, nous cherchons les Images acquises à partir d'une orbite ascendante car la plupart des Images Sentinel-1 pour le Bénin sont acquises dans ce sens.

```
var collectionVV = ee.ImageCollection('COPERNICUS/S1_GRD')
   .filter(ee.Filter.eq('instrumentMode', 'IW'))
.filter(ee.Filter.listContains('transmitterReceiverPolarisation',
'VV'))
   .filter(ee.Filter.eq('orbitProperties_pass', 'ASCENDING'))
   .filterBounds(parakou)
   &.select('VV');

var collectionVH = ee.ImageCollection('COPERNICUS/S1_GRD')
   .filter(ee.Filter.eq('instrumentMode', 'IW'))
 .filter(ee.Filter.listContains('transmitterReceiverPolarisation',
'VH'))
```

```
   .filter(ee.Filter.eq('orbitProperties_pass', 'ASCENDING'))
   .filterBounds(parakou)
   .select('VH');
print(collectionVV); print(collectionVH);
```

Une visualisation rapide est faite en utilisant la valeur médiane des signaux :

```
var VV = collectionVV.median();
var VH = collectionVH.median();
Map.addLayer(VV, {}, 'VV');
Map.addLayer(VH, {}, 'VH');
```

Si on souhaite représenter le changement du signal au cours de l'année, il est possible de créer une image composite où les trois bandes correspondent à des périodes différentes. Les pixels noirs ou blancs correspondent logiquement aux pixels où le signal a très peu changé.

```
var VV1 = ee.Image(collectionVV.filterDate('2023-01-01', '2023-04-
30').median());
var VV2 = ee.Image(collectionVV.filterDate('2023-05-01', '2023-08-
31').median());
var VV3 = ee.Image(collectionVV.filterDate('2023-09-01', '2023-12-
31').median());
Map.addLayer(VV1.addBands(VV2).addBands(VV3), {min: -12, max: -7},
'Composite');
```

Bae et al. (2022 ; https://doi.org/10.1111/2041-210X.13726) ont utilisé un indice simple, le Canopy Development Index (CDI), pour suivre la défoliation de la canopée causée par des insectes :

$$Canopy\ Development\ Index = VV - VH$$

Pour cela, d'abord créer un point appelé « point » :

```
var collectionCDI = ee.ImageCollection('COPERNICUS/S1_GRD')
   .filterDate('2023', '2024')
   .filter(ee.Filter.eq('instrumentMode', 'IW'))
.filter(ee.Filter.listContains('transmitterReceiverPolarisation',
'VH'))
.filter(ee.Filter.listContains('transmitterReceiverPolarisation',
'VV'))
   .filter(ee.Filter.eq('orbitProperties_pass', 'ASCENDING'))
   .filterBounds(point)
   .map(function(image) {
      var imageCDI =
image.select('VV').subtract(image.select('VH'));
```



```
      return imageCDI.rename('CDI').copyProperties(image,
['system:time_start']);
   })
   .select('CDI');
print(collectionCDI, 'collectionCDI');
```

Avec la fonction `ui.Chart.image.series()` il est possible de suivre la fluctuation du CDI au niveau de « point ». Cependant, ces variations semblent complexes à interpréter même en déplaçant « point » dans différents types de végétation :

```
var graph = ui.Chart.image.series({
   imageCollection: collectionCDI,
   region: point,
   reducer: ee.Reducer.mean(),
   scale: 20,
}).setOptions({
   lineWidth: 1,
   pointSize: 3,
   title: 'CDI (2023)',
   vAxis: {title: 'CDI'},
   hAxis: {title: 'Date'}
});
print(graph);
```

Données LiDAR

Tout comme les données RADAR, les données LiDAR sont un type de télédétection actif se basant sur du laser (Light Detection and Ranging : LiDAR) émis par une source et le capteur mesure le signal réfléchi. Ceci permet d'étudier la structure d'une surface car le signal ne sera pas réfléchi au même moment par des éléments à des altitudes différentes (comme la canopée de la forêt en comparaison avec les strates sous-jacentes et le sol) : le premier signal réfléchi viendra des éléments les plus hauts tandis que le dernier signal réfléchi proviendra du sol.

Dans GEE, bien que beaucoup de jeux de données se basent sur des données LiDAR (par exemple, NEON Canopy Height Model, ou le WHRC Pantropical National Level Carbon Stock Dataset), peu de jeux de données fournissent des données LiDAR « brutes ». En effet, seules les données de GEDI sont disponibles sur GEE et couvrent la période 2019-2023 de ce senseur bien qu'il ait a nouveau été utilisé à partir de 2024. Ces jeux de données ne couvrent pas les hautes latitudes car GEDI est attaché à la station spatiale internationale (ISS). De plus, GEDI ne produit pas d'images continuent mais ce sont des points de 25 mètres de diamètre qui sont mesurés à plusieurs intervalles le long de la trajectoire de l'ISS. Les données disponibles vont du niveau L2A (la hauteur de la canopée) à des niveaux plus travaillés comme le L4B qui correspond à la biomasse modélisée.

Avant d'importer l'ImageCollection L2A Monthly, ajouter une fonction qui permet de faire le tri entre les données de bonne ou mauvaise qualité selon la bande « quality_flag » et créer un polygone appelé « geometry » :

```
var qualityMask = function(im) {
  return im.updateMask(im.select('quality_flag').eq(1))
      .updateMask(im.select('degrade_flag').eq(0));
};

var GEDI = ee.ImageCollection('LARSE/GEDI/GEDI02_A_002_MONTHLY')
   .filterDate('2022-01-01', '2023-01-01')
   .filterBounds(geometry)
   .map(qualityMask);
print(GEDI);
Map.centerObject(geometry);
```

En consultant les informations sur GEDI avec `print()`, on remarque qu'il y a de nombreuses bandes. Celles-ci correspondent à la hauteur à laquelle x% du signal aura été réfléchi. Ainsi, la bande « rh100 » correspond à la hauteur maximale car 100% du signal aura été réfléchi à cette hauteur où en deçà (Figure 21).

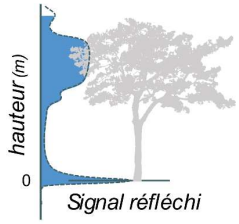


Figure 21 quantité de signal réfléchi en fonction de la structure verticale de la forêt.

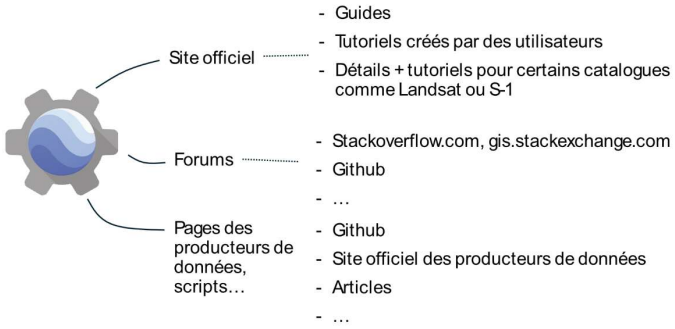
Schneider et al. (2020 ; https://doi.org/10.1088/1748-9326/ab9e99) ont proposé un ratio, le Canopy Ratio, qui décrit la structure verticale de la forêt. Si la plupart de la biomasse est dans la canopée, alors RH_{98}et RH_{25} sont proches et le ratio est alors petit. Ce ratio augmente s'il y a plus de biomasse dans les basses strates de la végétation.

$$Canopy\ Ratio = \frac{RH_{98} - RH_{25}}{RH_{98}}$$

```
   .map(function(image) {
     var CH98 = image.select('rh98');
     var CH25 = image.select('rh25');
     var CR = CH98.subtract(CH25).divide(CH98).rename('CRatio')
     return image.addBands(CR)
   });
```


Limitations de GEE

Ressources



Fiches de cas simples et courants

Filtrer une collection selon une propriété	
Filtrer une collection selon une date	
Filtrer une collection selon l'emprise d'un objet	
Créer une Image à partir de plusieurs Image d'une ImageCollection (somme, etc)	
Extraire toutes les valeurs d'une ImageCollection au niveau d'un point	
Extraire toutes les valeurs des pixels dans un polygone	
Extraire une valeur à partir de plusieurs pixels dans un polygone (*e.g.*, une moyenne)	
Extraire la valeur d'une propriété	
Faire un indice à partir de plusieurs bandes	
Ajouter une bande à une Image	
Ajouter une Image à une ImageCollection	
Découper une Image avec un polygone	
Créer un polygone représentant l'emprise d'une Image	
Masquer certaines valeurs sur une Image	
Masquer certaines valeurs sur les Image d'une ImageCollection	
Créer un point à partir de coordonnées	

Faire un calcul à partir de deux Image	
Exporter un raster	
Exporter un tableau	
Exporter un vecteur	