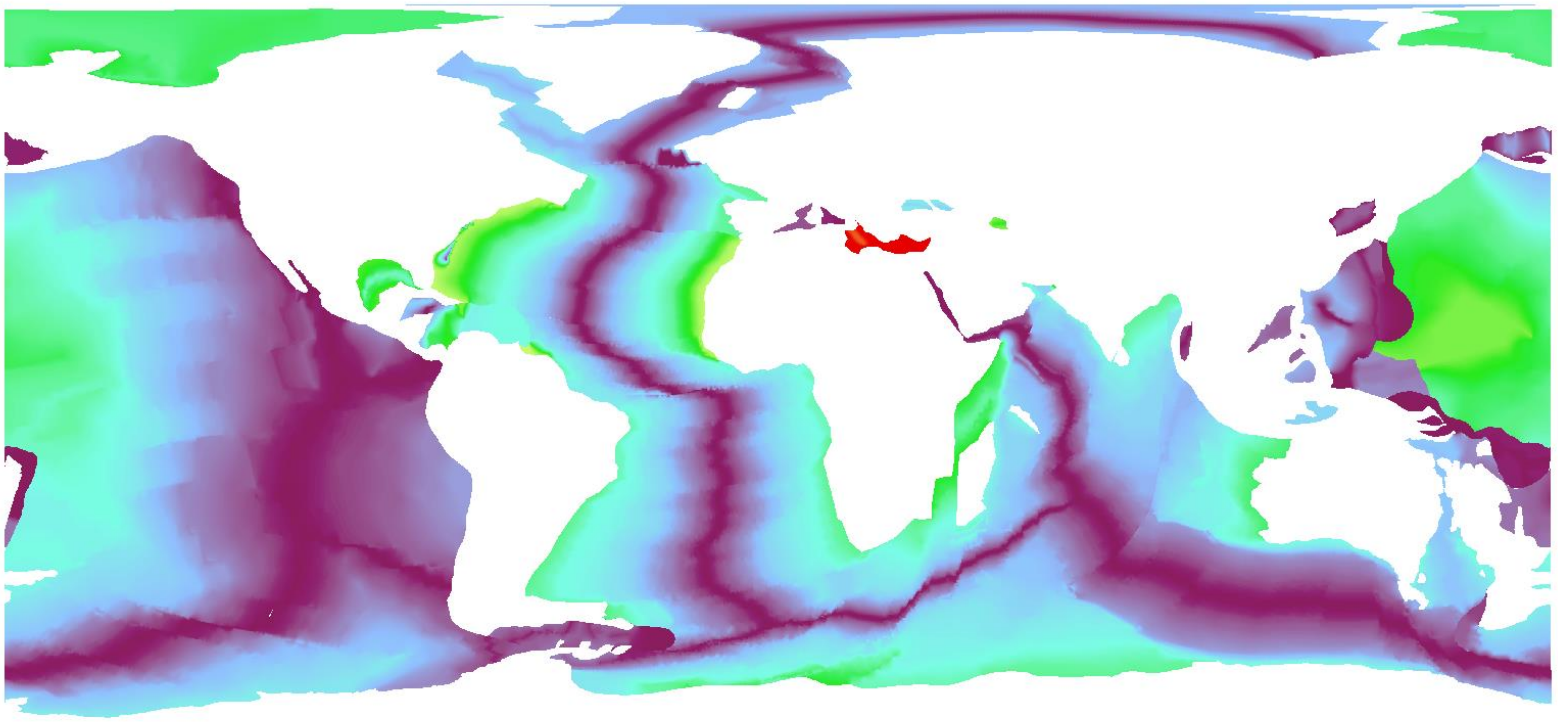
DÉVELOPPEMENT DE L'OUTIL *SEAFLOORAGE* RECONSTRUISANT L'ÂGE DES FONDS OCÉANIQUES SUR ARCGIS PRO



MÉMOIRE DE RECHERCHE

CERTIFICAT COMPLÉMENTAIRE EN GÉOMATIQUE

Etudiante : Célia Barat

Superviseurs : Christian Vérard, Gregory Giuliani

Juin - Octobre 2022

UNIVERSITÉ DE GENÈVE
FACULTÉ DES SCIENCES
Section des sciences de la Terre et de l'environnement

UNIVERSITÉ DE GENÈVE
FACULTÉ DES SCIENCES DE LA SOCIÉTÉ

TABLE DES MATIERES

Résumé 3
Introduction 3
Outil et données 4
Méthodologie 4
Installer l'outil sur ArcGIS Pro 5
Lancer l'outil 6
Partie 1 : Mise en place de l'environnement 7
UTF-8 7
Modules 7
Paramètres de l'environnement 7
Données en entrée 7
Géodatabases 8
Partie 2 : Préparation des données 8
Sélection des âges à reconstruire 8
Créations de vecteurs 9
Liste en cas d'âges non-disponibles 12
Partie 3 : Reconstruction des plaques tectoniques 12
Loop des âges 12
Loop des plaques tectoniques 15
Déterminer le type de la plaque 21
Déplacement des points 23
Interpolation des âges de la croûte 25
Partie 4 : Reconstruction de l'âge 27
Partie 5 : Symbologie et messages d'avertissements 30
Symbologie 30
Messages d'avertissements 32
Résultats 33
Discussion 35
Problèmes rencontrés 35
Erreurs potentielles et solutions 37
Améliorations 38
Conclusion 40
Annexes 40
Références 46

RÉSUMÉ

Il existe aujourd'hui de nombreux modèles reconstruisant les environnements paléogéographiques capables de remonter à des millions d'années dans le passé, tel que le modèle préliminaire Panalesis. Celui-ci se fonde sur la tectonique des plaques, la topographie, le climat et la présence de végétation afin de proposer des reconstructions synthétiques et contraintes par des données géologiques. Ces cartes produites permettent d'étudier d'autres domaines d'intérêts et notamment l'évolution de l'âge de la croûte océanique à l'échelle du globe. Ce mémoire de recherche a pour objectif de créer un outil *SeaFloorAge* capable de reconstruire n'importe quelle période donnée à l'aide du système d'information géographique ArcGIS Pro. Comme de nombreuses étapes sont nécessaires à la production de cartes de l'âge des fonds marins, un tel outil permet d'économiser du temps et de l'énergie. *SeaFloorAge* mériterait néanmoins certaines améliorations afin de rendre les reconstructions plus proches de la réalité.

INTRODUCTION

Durant la deuxième Guerre Mondiale, la marine américaine a indirectement participé à la découverte d'un argument de taille en faveur de la théorie de la tectonique des plaques, récente et encore largement débattue dans le monde scientifique. Loin de là l'intention de révolutionner la géologie, l'armée cherchait plutôt à repérer les sous-marins ennemis en mesurant les anomalies magnétiques induites. Pour se faire, elle finança la mise au point d'un magnétomètre portable et de grande précision qui, après la guerre, permis aux océanographes de cartographier les anomalies magnétiques des fonds marins (Gramling, 2021).

Ainsi, la croûte océanique se sépare aux rides médio-océaniques et le magma qui en ressort forme la nouvelle croûte dont les minéraux contenant du fer s'orientent selon le champ magnétique actuel (Gramling, 2021). Les alternances de polarité observées, symétriques de chaque côté des rides médio-océaniques, en sont le résultat et une preuve importante en faveur de la théorie de l'expansion des fonds marins et en défaveur de celle de la dérive des continents (Gramling, 2021; National Geographic Society, s. d.). En effet, cette dernière défendait que les continents se déplaçaient sur une croûte océanique immobile, ne prenant en compte ni les fonds marins, ni l'influence du manteau (National Geographic Society, s. d.; Verard, 2019b). Cette découverte prouve que la croûte océanique est un lieu actif de la tectonique des plaques, qui quant à elle prend en considération la limite des plaques, la convection du manteau, les fonds marins, les zones de subduction, etc. (Verard, 2019b).

Ainsi, la première notion d'expansion des fonds marins est apparue en 1961 avec Dietz (Dietz, 1961; Verard, 2019b), puis l'âge de la croûte a progressivement été daté en corrélant les isochrones (i.e. : chaque bande d'anomalie magnétique correspond à un âge qu'on appelle « isochrone ») avec les registres des inversions géomagnétiques obtenues à partir de roches continentales déjà datées (Earth Observatory of Singapore, s. d.). Ces mesures ont permis de produire la *Global Isochron Chart* en 1992 et différents modèles de reconstruction de la tectonique des plaques (Verard, 2019b).

En ce qui concerne les données des âges des fonds marins utilisées pour ce projet de géomatique, celles-ci proviennent de différentes publications. Pour les cartes les plus récentes (entre le Mésozoïque et le Cénozoïque), elles proviennent des anomalies magnétiques marines du jeu de données de Müller et al. (1997, 2008), qui provient lui-même d'autres publications telle que Royer et al. (1992). Ce jeu de données est également complété par des isochrones synthétiques (Hochard, 2008; Stampfli & Borel, 2002; Verard, 2019b, 2019a).

Toutes ces isochrones ont été intégrées dans la version préliminaire de PANALESIS de Verard (2019a), un modèle de reconstruction paléogéographique se basant sur la tectonique des plaques et complété par des modèles topographiques, climatiques et de végétation. Il vise à reconstituer le passé géologique de la Terre jusqu'à 600 millions d'années. Une nouvelle version en cours de développement, Panalesis v.1., remonte même jusqu'à 888 Ma. Ce modèle retourne diverses informations telles que les vitesses de subduction des plaques, les taux d'accrétion des rides médio-océaniques, etc., et notamment l'âge de la croûte océanique (Verard, 2019a, 2019b).

Celles-ci ont été utilisées par l'outil *SeaFloorAge* de ce projet pour la reconstruction des âges des fonds marins sur ArcGIS Pro.

L'objectif de ce travail de recherche est de recréer un outil qui génère une carte globale de l'âge des fonds océaniques pour chaque période donnée. En effet, un autre outil produisait de genre de carte sur ArcMap mais les SIG ayant évolué, il a été nécessaire d'en faire un nouveau. À l'aide du logiciel ArcGIS Pro, l'idée est d'écrire un script Python afin d'automatiser la démarche qui nécessite beaucoup d'étapes. Le résultat final doit produire une carte raster pour chacune des reconstructions, soit 48 cartes allant d'aujourd'hui jusqu'à 600 Ma.

Le présent rapport est construit comme suit : après une rapide description des données utilisées, la partie *méthodologie* introduit ce qui a déjà été fait, ce que j'ai apporté à l'outil et explique en détail chaque étape du code. Ensuite, les reconstructions sont présentées et décrites dans la partie *résultats* puis des suggestions d'améliorations et la description des problèmes rencontrés sont présentés dans la *discussion*.

OUTIL ET DONNÉES

Un tel code avait déjà été écrit il y a quelques années au format .net : il ne fonctionne donc plus avec les versions d'ArcGIS les plus récentes. En effet, Esri a apporté certaines nouveautés à son software et il a donc fallu réécrire le code ainsi que son architecture en Python afin de pouvoir utiliser l'outil avec le site-package ArcPy d'ArcGIS. J'avais également la possibilité de créer l'outil à l'aide du ModelBuilder mais comme ce dernier ne permet de faire qu'une seule itération ou condition, j'ai préféré travailler sur un script unique où y figurent toutes les instructions, itérations, conditions, etc.

Les données utilisées pour tester et lancer l'outil *SeaFloorAge* proviennent de la version préliminaire du modèle Panalesis de Christian Verard (2019a), que ce dernier m'a gentiment transmis. Il s'agit de 3 dossiers (*Plates*, *WorldMaps*, *COB_psAbs*), chacun contenant 48 shapefiles - un pour chaque période (de 0 à 600 Ma) - à l'échelle globale. Ces dossiers contiennent les données suivantes :

- Les polygones des plaques tectoniques ;
- Les lignes d'isochrones des âges de la croûte océanique, des limites de zones de subductions, de l'emplacement des rides médio-océaniques, etc. ;
- Les polygones des plaques continentales.

Ce jeu de données permet ainsi de réaliser des reconstructions de l'âge des fonds marins.

MÉTHODOLOGIE

En plus des shapefiles, Christian Vérard m'a expliqué les étapes à suivre pour me lancer dans l'écriture du script, étant donné que la démarche avait déjà été faite une fois.

L'idée est donc de faire la reconstruction de l'âge des fonds marins plaque par plaque à l'aide d'une boucle *for*, puis d'assembler le tout pour en faire une reconstruction complète pour chaque période. Il faut extraire du shapefile *WorldMaps* les isochrones, rides médio-océaniques et marges passives qui sont contenues dans chaque plaque, et en faire un nouveau vecteur ligne dont on en densifie les sommets. Cette couche est ensuite transformée en vecteur points en avant d'en faire une interpolation selon l'âge de chaque point. On obtient un raster qui doit d'abord être découpé selon la forme de sa plaque tectonique puis faire une mosaïque de rasters de toutes les reconstructions de plaque. Enfin, on retire de la reconstruction les plaques continentales qui n'ont rien à voir avec l'âge de la croûte océanique avec les shapefiles *CBO_psAbs*.

Il y a cependant un problème de projection à prendre en compte : certaines plaques sont « coupées en deux » par la ligne de temps internationale, comme la plaque Pacifique, les plaques Fiji, l'Antarctique, l'Arctique, etc. Du point de vue de la projection, les points de ces plaques sont séparés de chaque côté de la carte et l'interpolation

tente de produire une reconstruction évidemment fausse pour l'espace entre les deux morceaux de plaque (voir figure 24). Il faut donc déplacer une partie de ces points de l'autre côté de la carte afin que tous soient rassembler au même endroit.

Christian m'a proposé de trouver une solution par moi-même pour déplacer ces points. Après quelques tentatives, je suis parvenue à déplacer les points ciblés à l'aide de la ligne de changement de date : si un polygone intersecte cette ligne, il est considéré comme « divisé » et ses points à l'ouest de la carte sont déplacés de 360 degrés vers l'est. Cependant, les plaques aux pôles sont par conséquent aussi considérées comme divisées, bien que dans ce cas-ci déplacer les points ne servirait à rien, sauf si la plaque est constituée de deux morceaux au moins (voir figure 26) : il faut alors seulement déplacer les points du petit morceau de la plaque. Ainsi, les plaques sont définies comme *normales*, *divisées*, ou *pôle et divisées* et leurs points sont bougés en fonction.

Une autre complication vient des points proches des limites des plaques, et notamment les zones de subductions. En effet, les plaques ne sont pas entièrement recouvertes de points et il arrive que la reconstruction ait des trous lors de l'interpolation. Christian Vérard m'a proposé deux solutions : la création artificielle de points ou l'intégration des points manquants dans la reconstruction antérieure en prenant compte les mouvements des plaques. J'ai opté pour la première car plus simple bien que moins réaliste. La deuxième requiert la traduction et l'intégration dans ArcGIS Pro d'un tout autre outil pour la rotation des points (voir *Amélioration* dans la discussion pour plus de précision).

Une fois la reconstruction d'une période faite, les valeurs du rasters (c'est-à-dire les âges) sont recalculés afin que la valeur 0 corresponde à la croûte nouvellement formée à cette époque, et la symbologie est modifiée.

Le script retourne également des avertissements ou messages pour informer si certaines époques ou plaques n'ont pas pu être reconstruites.

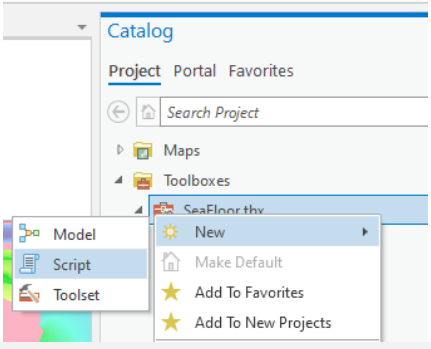
Note : Les pages suivantes détaillent avec autant de précision que nécessaire les différentes parties du code, afin de faciliter sa compréhension dans l'éventualité où une personne serait menée à le modifier pour améliorer l'outil (notamment une potentielle version bêta), modifier des paramètres ou corriger un éventuel bug. La version complète du script est disponible dans l'annexe de ce rapport.

Ce script Python a été développé sur ArcGIS Pro 2.7.0.

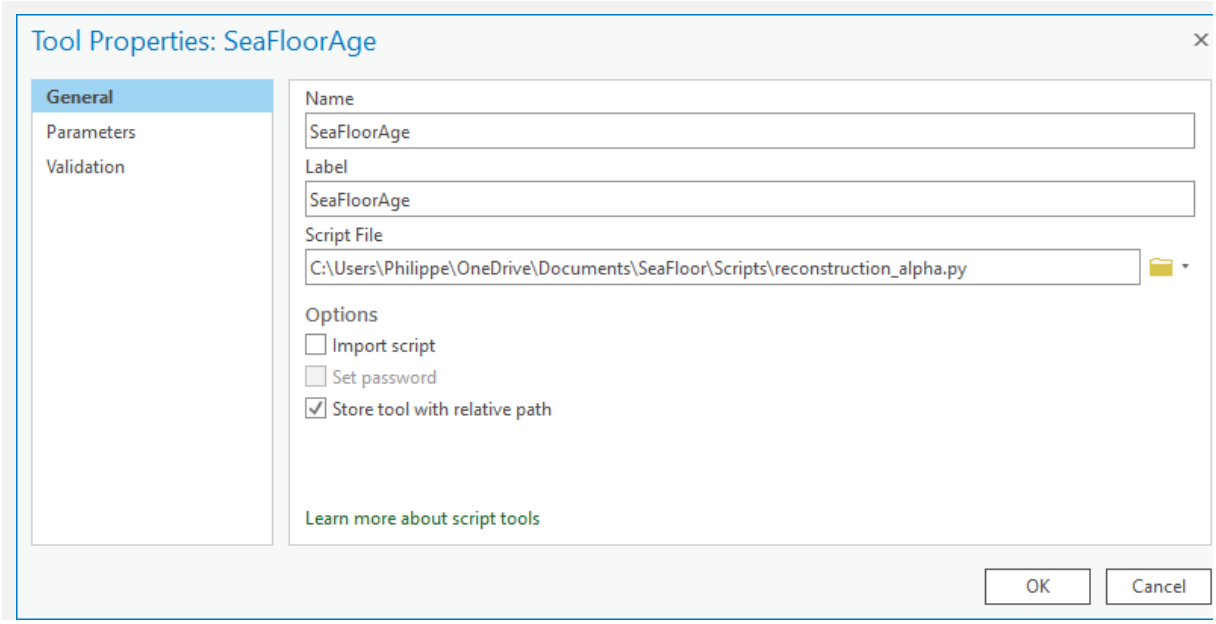
INSTALLER L'OUTIL SUR ARCGIS PRO

Dans le catalogue du projet sur ArcGIS Pro, faire un clic droit sur la boite à outil et sélectionner *New>Script* (figure 1.a). Dans la section *General*, donner un nom à l'outil et sélectionner le script Python de l'outil dans *Script File* (figure 1.b). Puis dans la section *Parameters*, spécifier les informations nécessaires pour les données en entrées (figure 1.c ; voir *Partie 1, Données en entrées* pour plus de détails).

a.



b.



c.

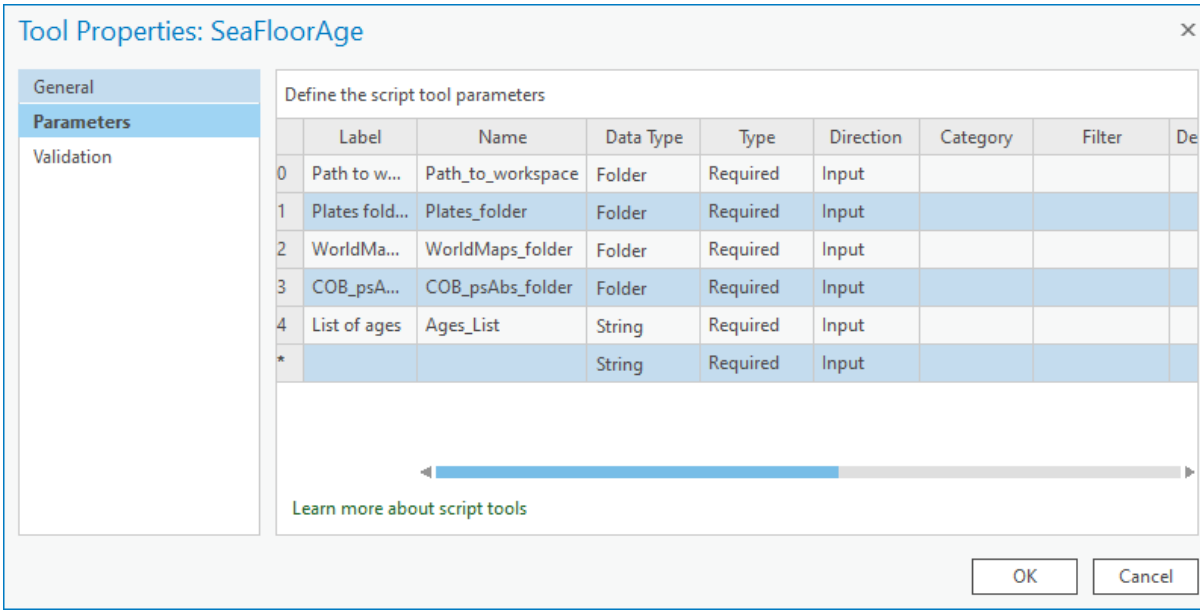


Figure 1 : a. Clic droit sur la boite à outil de *SeaFloorAge*. b. Informations générales des propriétés de l'outil. c. Paramètres des propriétés de l'outil.

LANCER L'OUTIL

Tout d'abord, il faut s'assurer que rien ne soit sélectionné sur le projet dans ArcGIS Pro. Dans l'outil *SeaFloorAge*, les informations à entrer sont les suivantes (figure 2) :

- *Path to workspace* : le chemin jusqu'au dossier contenant la/les géodatabase(s) du projet.
- *Plates folder* : le chemin menant au dossier contenant les fichiers des plaques tectoniques pour chaque âge.
- *WorldMaps* folder : le chemin jusqu'au dossier contenant les couches vecteurs lignes des âges de la croûtes (isochrones), des limites des rides médio-océaniques, des zones de subductions, etc., pour chaque âge.
- *COB_psAbs* folder : le chemin du dossier avec les polygones des croûtes continentales pour chaque âge.

- *List of ages* : le ou les âge(s) à reconstruire. Il faut les écrire avec 3 digits, et on peut en sélectionner un ou plus de plusieurs manières :
    - Un âge : taper simplement l'âge à reconstruire (ex : 011) ;
    - Plusieurs âges spécifiques : les séparer par une virgule (ex : 000,011,417,246) ;
    - Un intervalle d'âge : séparer l'âge minimum et maximum par un trait (ex : 000-315) ;
    - Tout sélectionner : taper un astérisque (*).

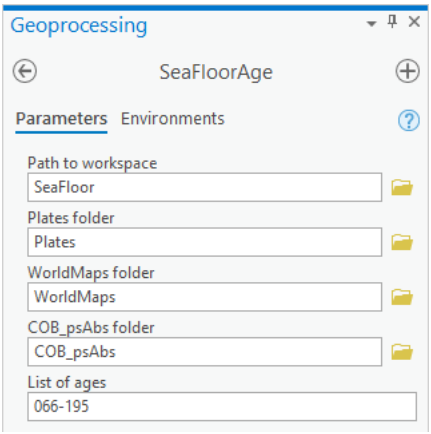


Figure 2 : Screenshot de l'outil *SeaFloorAge* sur ArcGIS Pro.

PARTIE 1 : MISE EN PLACE DE L'ENVIRONNEMENT

UTF-8

Il est toujours mieux de spécifier le codage de caractère informatique (ça règle souvent les erreurs pythons les plus communes).

```
# -*- coding: utf-8 -*-
```

MODULES

Le module *re* permet d'utiliser de la fonction `re.findall()` qui se sert des *regular expressions* (utilisé à l'étape *Sélection des âges à reconstruire* de la partie 2). Pour bénéficier de l'outil *Raster Calculator*, il faut également importer la licence *Spatial Analyst (sa)*.

```
import arcpy, re
from arcpy.sa import *
```

PARAMÈTRES DE L'ENVIRONNEMENT

Cette ligne permet de relancer l'outil sans avoir à effacer ce qui a été précédemment créé. Les fichiers avec les mêmes noms sont ainsi écrasés.

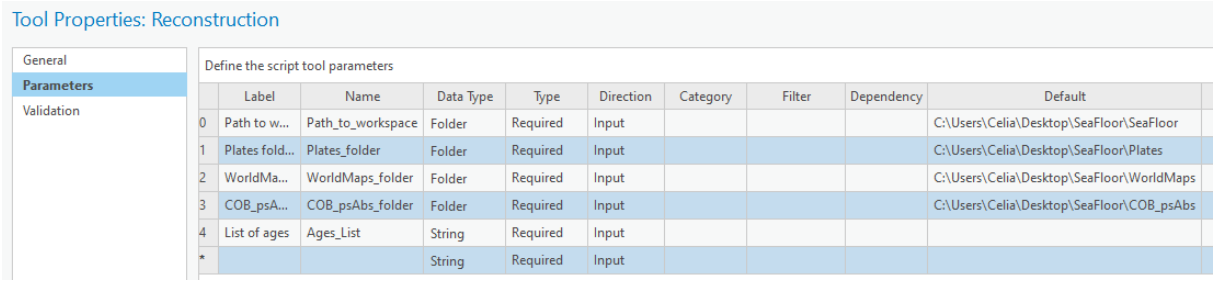
```
arcpy.overwriteOutput = True
```

DONNÉES EN ENTRÉE

La valeur entre parenthèse de la fonction `arcpy.GetParameterAsText(0)` correspond au numéro de la ligne dans le tableau des propriétés de l'outil sur ArcGIS Pro. Depuis ArcGIS Pro, il faut spécifier le label, le nom, quel type de donnée sont en entrée (ex : un dossier, un shapefile, du texte, etc.), entre autres. Ecrire par défaut les chemins jusqu'aux données permet d'éviter de les définir à chaque fois que l'outil est utilisé (figure 3).

```
path_to_GDBs = arcpy.GetParameterAsText(0)
```

```
Plates_folder = arcpy.GetParameterAsText(1)
WorldMaps_folder = arcpy.GetParameterAsText(2)
COB_psAbs_folder = arcpy.GetParameterAsText(3)
Ages_List = arcpy.GetParameterAsText(4)
```



	Label	Name	Data Type	Type	Direction	Category	Filter	Dependency	Default
0	Path to w...	Path_to_workspace	Folder	Required	Input				C:\Users\Celia\Desktop\SeaFloor\SeaFloor
1	Plates fold...	Plates_folder	Folder	Required	Input				C:\Users\Celia\Desktop\SeaFloor\Plates
2	WorldMa...	WorldMaps_folder	Folder	Required	Input				C:\Users\Celia\Desktop\SeaFloor\WorldMaps
3	COB_psA...	COB_psAbs_folder	Folder	Required	Input				C:\Users\Celia\Desktop\SeaFloor\COB_psAbs
4	List of ages	Ages_List	String	Required	Input				
*			String	Required	Input				

Figure 3 : Exemple de chemins par défaut pour les dossiers de shapefiles, dans les paramètres des propriétés de l'outil.

GÉODATABASES

Si ces géodatabases n'existent pas déjà dans le projet ArcGIS, le script les crée : *Process.gdb* recueille toutes les couches temporaires et *Maps.gdb* les reconstructions finales.

```
if not arcpy.Exists(path_to_GDBs +"\\Maps.gdb"):
    arcpy.management.CreateFileGDB(path_to_GDBs, "Maps")

if not arcpy.Exists(path_to_GDBs +"\\Process.gdb"):
    arcpy.management.CreateFileGDB(path_to_GDBs, "Process")

Maps_gdb = path_to_GDBs +"\\Maps.gdb"
Process_gdb = path_to_GDBs + \\Process.gdb
```

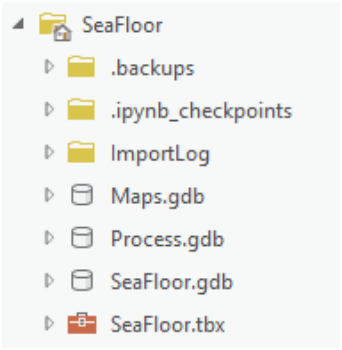


Figure 4 : Emplacement des géodatabases *Process.gdb* et *Maps.gdb* dans le dossier du projet ArcGIS Pro.

PARTIE 2 : PRÉPARATION DES DONNÉES

SÉLECTION DES ÂGES À RECONSTRUIRE

Dans ArcGIS Pro, l'outil demande d'entrer la ou les période(s) à reconstruire, écrites avec 3 digits. Les lignes suivantes permettent de sélectionner :

- Tous les âges disponibles avec « * » ;

```
if Ages_List == "*":
    Ages_List = []
    filename_List = []
    walk = arcpy.da.Walk(Plates_folder, datatype="FeatureClass")
    for dirpath, dirnames, filenames in walk:
        for filename in filenames:
            filename_List.append(filename)
    for filename in filename_List:
```

```
            age = re.findall('[0-9]+', filename)
            Ages_List.append(age[0])
```

- Un intervalle d'âge, [047-078] par exemple. Comme la fonction `int()` enlève les zéros avant un nombre, la for loop présente rajoute ces zéros pour toujours avoir 3 digits ;

```
elif Ages_List.find("-") >= 0:
    interval = Ages_List.split("-")
    nums = range(int(interval[0]), (int(interval[1])+1), 1)
    Ages_List = []
    for num in nums:
        if num < 10:
            num = "00" + str(num)
        elif num < 100:
            num = "0" + str(num)
        else:
            num = str(num)
        Ages_List.append(num)
```

- Des âges spécifiques, séparés par une virgule ;

```
elif Ages_List.find(",") >= 0:
    Ages_List = Ages_List.split(",")
```

- Un seul âge.

```
elif len(Ages_List) == 3 and len(re.findall(r'\b\d{3}\b', Ages_List)) == 1:
    age = Ages_List
    Ages_List = []
    Ages_List.append(age)
```

Si l'intervalle de temps n'a pas été correctement écrit, un message s'affiche pour rappeler les conditions et différentes possibilités et l'outil est interrompu :

```
else:
    arcpy.AddMessage("List of ages incorrect. The age needs to be written with 3 digits.
Select:\n 1) one age\n 2) specific ages separating them with ','\n 3) a range of ages with '-
'\n 4) everything with '*'.")
    exit()
```

Messages

Start Time: jeudi, 25 août 2022 15:53:17

```
List of ages incorrect. The age needs to be written with 3
digits. Select:
 1) one age
 2) specific ages separating them with ','
 3) a range of ages with '-'
 4) everything with '*'.
```

Figure 5 : Message que l'outil renvoie si les conditions des âges ne sont pas respectées.

Si tout est conforme, un message s'affiche avec la liste des âges demandés :

```
arcpy.AddMessage("List of ages (" + str(len(Ages_List)) + ") :")
arcpy.AddMessage(Ages_List)
```

```
List of ages (48) :
['000', '006', '011', '015', '020', '033', '040', '048',
'056', '068', '084', '094', '100', '113', '120', '133',
'140', '154', '165', '180', '200', '210', '220', '230',
'240', '250', '270', '290', '300', '315', '330', '350',
'370', '383', '393', '408', '420', '444', '463', '475',
'489', '500', '518', '535', '545', '560', '580', '600']
```

Figure 6 : Exemple de message de la liste des âges que l'outil renvoie (dans ce cas-ci, « * »).

CRÉATION DE VECTEURS

Afin de pouvoir sélectionner les différentes plaques tectoniques d'une reconstruction et produire les nouvelles couches nécessaires, des vecteurs points, lignes et polygones sont préalablement créés, s'ils n'existent pas déjà dans *Process.gdb* avec `if not` (figure 7).

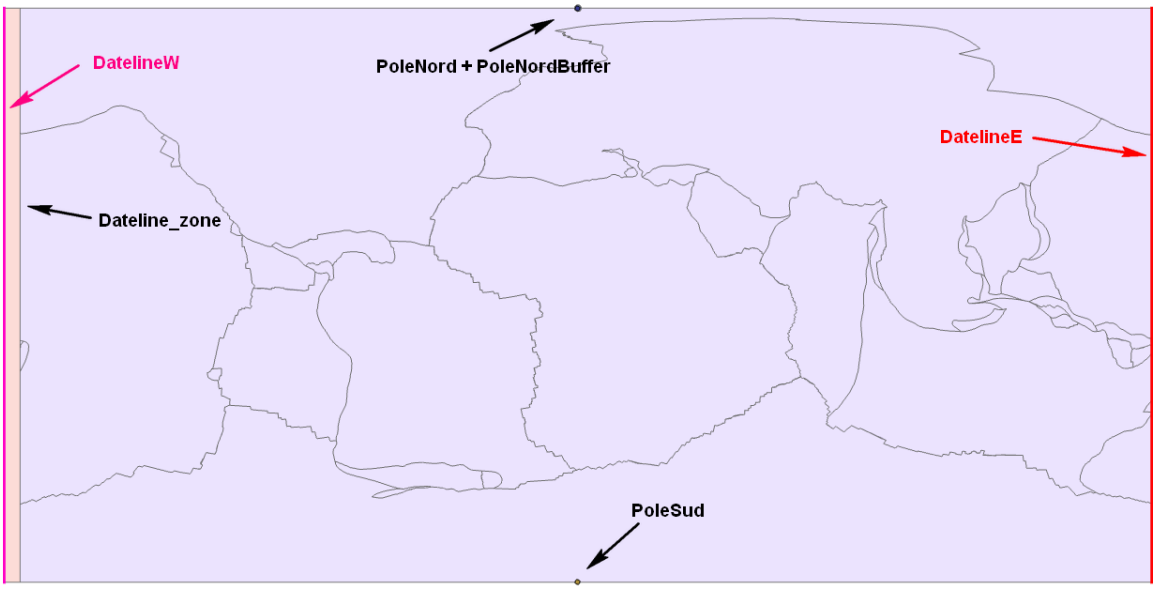


Figure 7 : Vue d'ensemble des vecteurs créés pour la partie 2 « création de vecteur ». Les plaques tectoniques appartiennent à l'âge 000.

POINTS

Les points *PoleNord* et *PoleSud* serviront à identifier les plaques des pôles avec un *SelectByLocation*. Dans le cas de *PoleNord*, j'ai remarqué que les plaques n'intersectent pas avec le point (figure 8.a) : il a donc fallu faire une zone tampon autour du point pour corriger ce problème (figure 8.b).

Donc si les couches n'existent pas déjà, le code utilise les deux listes préalablement créées contenant le nom des couches points ainsi que les coordonnées de ces points. Puis, l'outil itère simultanément dans ces deux listes avec la fonction `zip()` afin de créer les couches avec `CreateFeatureclass()`. À l'aide de `InsertCursor()` et `insertRow()`, l'équivalent du *Edit>Feature>Modify* dans l'interface graphique d'ArcGIS Pro, les points sont ajoutés. Le principe est le même pour la création des lignes et du polygone décrits plus bas.

```
# Create PoleNord et PoleSud feature points layers ----------------------------------------
if not arcpy.Exists(Process_gdb + "\\PoleNordBuffer") or not arcpy.Exists(Process_gdb +
"\\PoleSud"):
    nmList = ["PoleNord", "PoleSud"]
    ptList = [((0,90,0000000),) , ((0,-90),) ]
    for nm,pt in zip(nmList, ptList):
        nm = arcpy.management.CreateFeatureclass(out_path=Process_gdb, out_name=nm,
geometry_type="POINT", template=[], has_m="DISABLED", has_z="DISABLED")
        cursor = arcpy.da.InsertCursor(nm, ["SHAPE@XY"])
        cursor.insertRow(pt)
        del cursor

    # Because PoleNord doesn't intersect with north plate
    PoleNordBuffer = Process_gdb + "\\PoleNordBuffer"
    arcpy.analysis.Buffer(in_features=Process_gdb + "\\PoleNord",
out_feature_class=PoleNordBuffer, buffer_distance_or_field="1 DecimalDegrees",
line_side="FULL", line_end_type="ROUND", dissolve_option="NONE", dissolve_field=[],
method="PLANAR")
```

a. **b.**

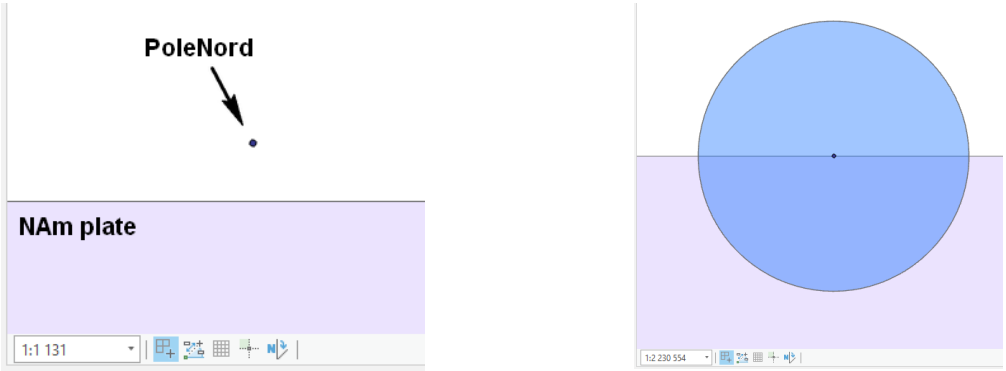


Figure 8 : a. zoom sur le point *PoleNord* : la plaque *Nam* du l'âge 000 n'intersecte pas avec le point, à 13 mètres près. b. *PoleNordBuffer* remplace donc *PoleNord* afin de s'assurer que les plaques au pôle Nord soient sélectionnées par l'outil.

LIGNES

Les lignes *DatelineW* et *DatelineE* permettront d'identifier les plaques « coupées en deux », déplacer les points des âges de la croûte océanique d'un côté ou de l'autre de la carte, ou encore de supprimer les points supplémentaires situés le long de la ligne de changement de date.

```
# Create international Datelines feature line layers ----------------------------------------
if not arcpy.Exists(Process_gdb + "\\DatelineW") or not arcpy.Exists(Process_gdb +
"\\DatelineE"):
    nmList = ["DatelineW", "DatelineE"]
    ptList = [ ((-180, 90),(-180, -90)) , ((180, 90),(180, -90)) ]
    for nm,pt in zip(nmList, ptList):
        fc = arcpy.management.CreateFeatureclass(out_path=Process_gdb, out_name=nm,
geometry_type="POLYLINE", template=[], has_m="DISABLED", has_z="DISABLED")
        cursor = arcpy.da.InsertCursor(fc, ["SHAPE@"])
        array = arcpy.Array([arcpy.Point(pt[0][0],pt[0][1]),
                             arcpy.Point(pt[1][0],pt[1][1])])
        polyline = arcpy.Polyline(array)
        cursor.insertRow([polyline])
        del cursor
```

POLYGONE

Tout comme *PoleNord*, *DatelineW* n'intersecte pas tout le temps avec les plaques les plus à l'ouest de la carte de certains âges, comme c'est le cas pour le shapefile de 154 Ma par exemple.

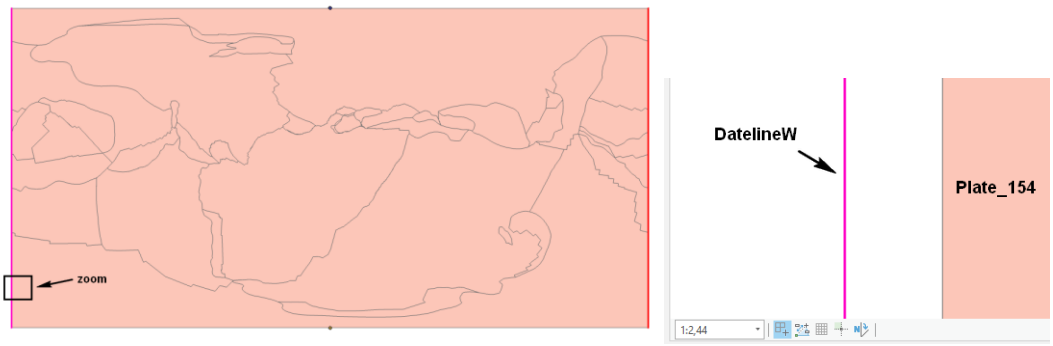


Figure 9 : Zoom sur *DatelineW*, les plaques de l'époque 154 Ma n'intersectent pas avec la ligne.

```
# Create a polygon zone close to DatelineW ----------------------------
if not arcpy.Exists(Process_gdb + "\\Dateline_zone"):
    fc = arcpy.management.CreateFeatureclass(out_path=Process_gdb, out_name="Dateline_zone",
geometry_type="POLYGON", template=[], has_m="DISABLED", has_z="DISABLED")
    cursor = arcpy.da.InsertCursor(fc, ["SHAPE@"])
    array = arcpy.Array([arcpy.Point(-180, 90),
                         arcpy.Point(-175, 90),
                         arcpy.Point(-175, -90),
```

```
                        arcpy.Point(-180, -90)])
    polyline = arcpy.Polygon(array)
    cursor.insertRow([polyline])
    del cursor
```

Une fois les vecteurs ajoutés à Process.gdb, ils sont définis dans le code comme tels :

```
PoleNord = Process_gdb + "\\PoleNordBuffer"
PoleSud = Process_gdb + "\\PoleSud"
DatelineW = Process_gdb + "\\DatelineW"
DatelineE = Process_gdb + "\\DatelineE"
Dateline_zone = Process_gdb + "\\Dateline_zone"
arcpy.AddMessage("Points, lines and polygon added")
```

LISTE EN CAS D'ÂGES NON-DISPONIBLES

Si les âges sont sélectionnés en faisant un intervalle, il se peut que certains n'existent pas dans la base de données. Dans ce cas, ils sont ajoutés à cette liste qui préviendra l'utilisateur quels âges n'ont pas été reconstruits à la fin du code.

```
Not_done = []
```

PARTIE 3 : RECONSTRUCTION DES PLAQUES TECTONIQUES

Les étapes préparatoires sont terminées, la reconstruction des âges de la liste commence.

LOOP DES ÂGES

Pour chaque âge, le code va chercher les 3 shapefiles correspondants : les plaques tectoniques, les isochrones/limites de subduction/rides médio-océaniques et l'espace occupé par les continents.

Note : les prochaines figures montrent des exemples pour la période 000. Dans le cas contraire l'âge sera mentionné.

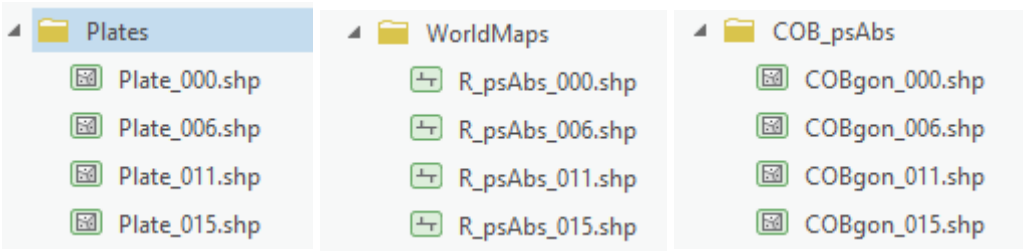


Figure 10 : Extrait des dossiers contenant les différents shapefiles pour chaque âge disponible. Il y a 48 époques au total.

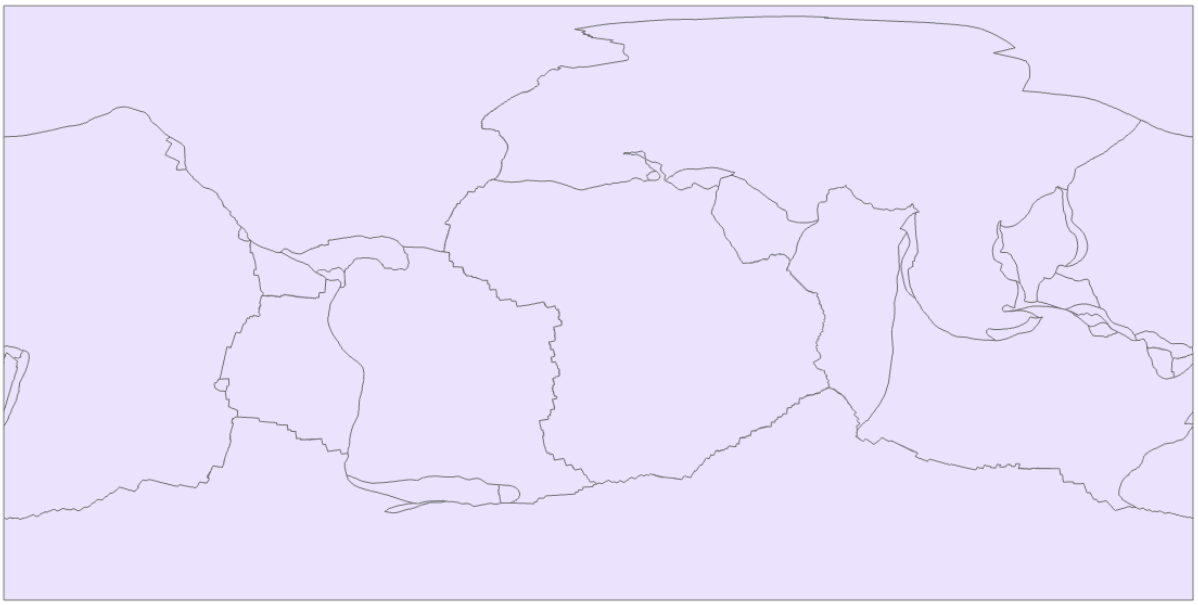


Figure 11 : Plate_000

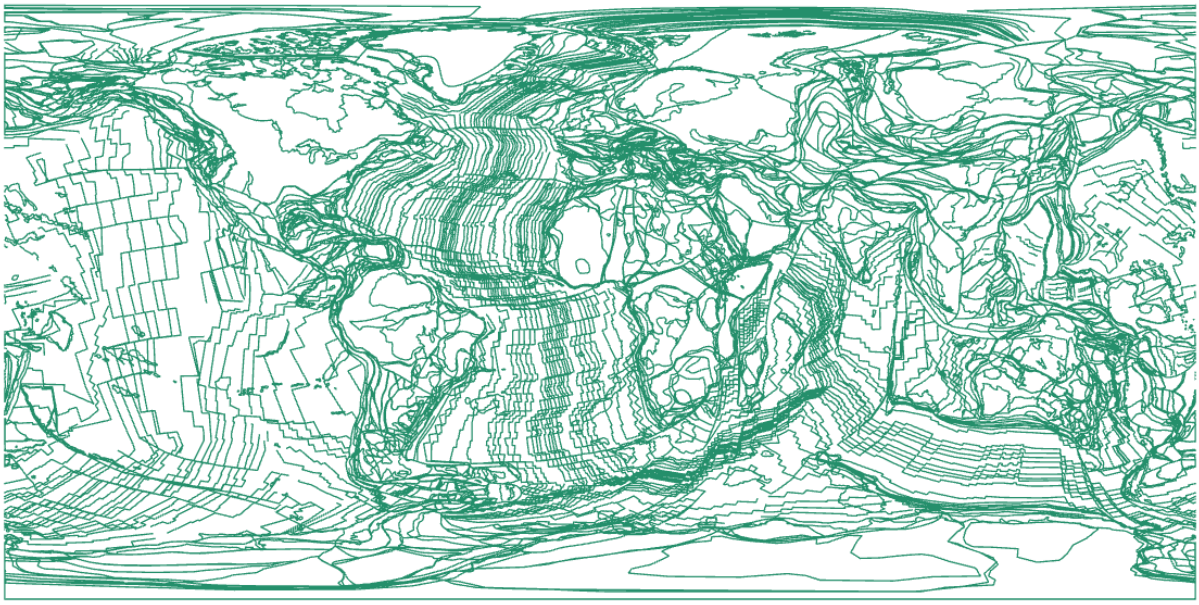


Figure 12 : R_psAbs_000

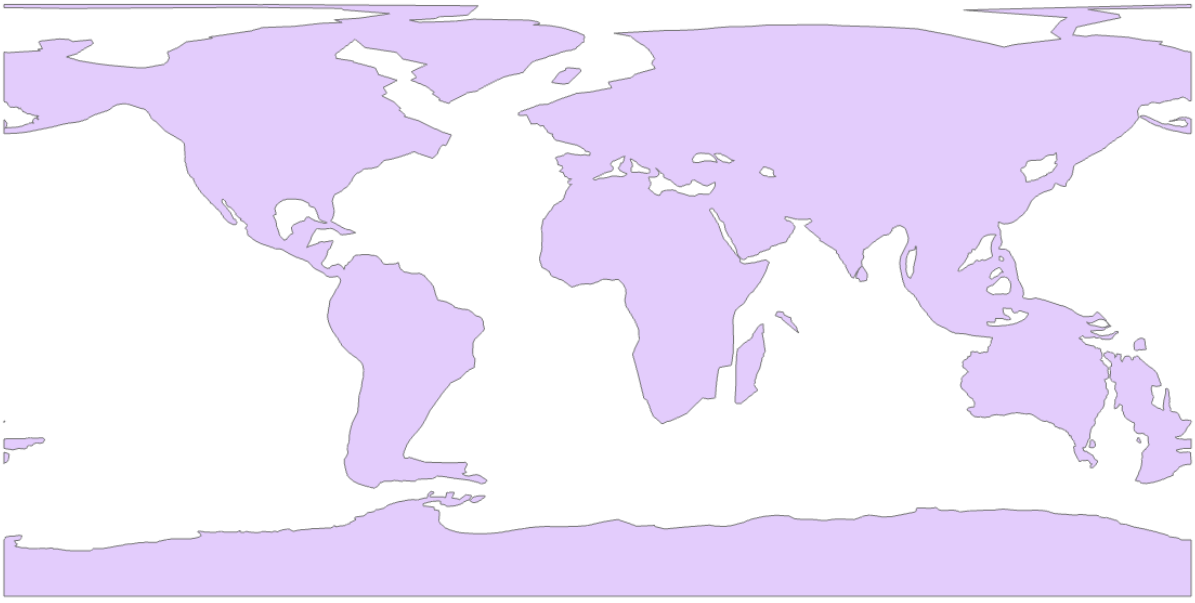


Figure 13 : COBgon_000

Si ces shapefiles existent (i.e. si le `if arcpy.Exists(PlatesMap)` est vrai) :

```
#### ================== Starting the reconstructions ========================================
### Starting the loop for each age in the list: ------------------------------------------------
for age in Ages_List:
    # Go to the shapefiles
    PlatesMap = Plates_folder + "\\Plate_%s"%age + ".shp"
    TimeMap = WorldMaps_folder + "\\R_psAbs_%s"%age + ".shp"
    Continents = COB_psAbs_folder + "\\COBgon_%s"%age + ".shp"

    # Start the process, if age exists in the data:
    if arcpy.Exists(PlatesMap):
        arcpy.AddMessage("========== Reconstructing age %s =========="%age)
```

1. Le code dresse une liste des noms des plaques tectoniques (*Plates*) ;

```
# Make a list of the plates names
        Plates = []
        field = ["PlateName"]
        with arcpy.da.SearchCursor(PlatesMap, field) as cursor:
            for i in cursor:
                Plates.append(i[0])
```

2. Y retire éventuellement la plaque « GAP » qui n'en est pas une ;

```
# Remove "GAP" plates from the list
        if "GAP" in Plates:
            Plates.remove("GAP")
```

3. Prépare deux listes : une qui contiendra les rasters de plaques finies et l'autre les noms des plaques n'ayant pas de croûte océanique ;

```
# Prepare a list for rasters clipped and plates without oceanic crust
        RC_List = []
        No_crust = []
```

4. Prépare une couche vecteur ligne appelée *world_isochrons* contenant seulement les isochrones, les rides médio-océaniques et les limites de subduction avec `SelectLayerByAttribute()` (figure 14). Le `AGE < 999` vise à exclure les artefacts ;

```
# Select isochrons, passive margins and ridges, and make a layer
        SQL = "AGE < 999 AND (TYPE = 'Isochron' OR TYPE = 'Passive_Margin' OR TYPE =
'Ridge')"
        s_isochrons = arcpy.management.SelectLayerByAttribute(TimeMap,
'NEW_SELECTION', SQL)
```

```
        world_isochrons = arcpy.management.CopyFeatures(s_isochrons, Process_gdb +
"\\world_isochrons")
        arcpy.management.SelectLayerByAttribute(TimeMap, 'CLEAR_SELECTION')
```

5. Puis, un nouveau loop commence où chaque plaque est traitée pour obtenir un raster de l'âge de la croûte (expliqué dans la prochaine section *Loop des plaques tectoniques*).

```
        ### Starting the loop for each plate ========================================

        for plt in Plates:
            {...}
```

Si les shapefiles n'existent pas, l'âge est enregistré dans la liste *Not_done* préparée à cet effet et le code passe à l'âge suivant ou bien finit de s'exécuter.

```
else:
        arcpy.AddMessage("There is no shapefile for age %s"%age)
        Not_done.append(age)

    # --- End of looping through all ages ---
```

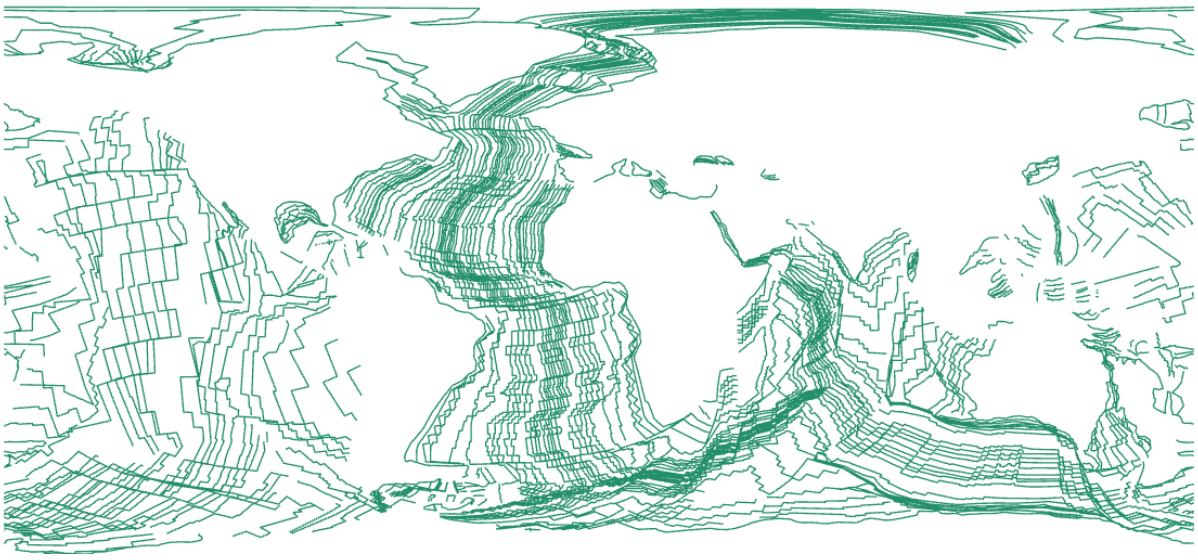


Figure 14 : world_isochrons : la couche contenant seulement les lignes d'isochrones, de rides médio-océaniques et de marges passives.

LOOP DES PLAQUES TECTONIQUES

Cette loop comprend tout ce qu'il y a d'écrit dans l'accolade `{...}` figurant dans l'extrait du code ci-dessus (en point 5).

Polygone des plaques

La plaque en cours d'itération dans la loop est sélectionné et copié avec `SelectLayerByAttribute()` et `CopyFeatures()`, et une fine zone tampon est ensuite faite autour de la plaque (figure 15). Les polygones se chevauchant légèrement, cela permettra d'éviter d'éventuelles lignes blanches (i.e. sans données) qui apparaitraient autrement dans la reconstruction finale (figure 16).

Note : Pour les prochains extraits du code de cette loop, une indentation a été enlevée à chaque ligne afin d'améliorer la visibilité. Le script complet avec les bonnes indentations se trouve en Annexe.

```
### Starting the loop for each plate ============================================================
for plt in Plates:
    ## POLYGON ------------------------------------------------------------------------------
    # Select the plate and make a layer from the selection
    SQL = "PlateName = '" + str(plt) + "'"
    selection = arcpy.management.SelectLayerByAttribute(PlatesMap, 'NEW_SELECTION', SQL)
    plate = arcpy.management.CopyFeatures(selection, Process_gdb + "\\plate")
```

```
    arcpy.management.SelectLayerByAttribute(PlatesMap, 'CLEAR_SELECTION')
    arcpy.AddMessage("Plate : " + str(plt) + " ----------")

    # Make a slight buffer around the Plate
    plate_buffer = Process_gdb + "\\plate_buffer"
    arcpy.analysis.Buffer(in_features=Process_gdb + "\\plate", out_feature_class=plate_buffer,
buffer_distance_or_field="0,05 DecimalDegrees", line_side="FULL", line_end_type="ROUND",
dissolve_option="NONE", dissolve_field=[], method="PLANAR")
```

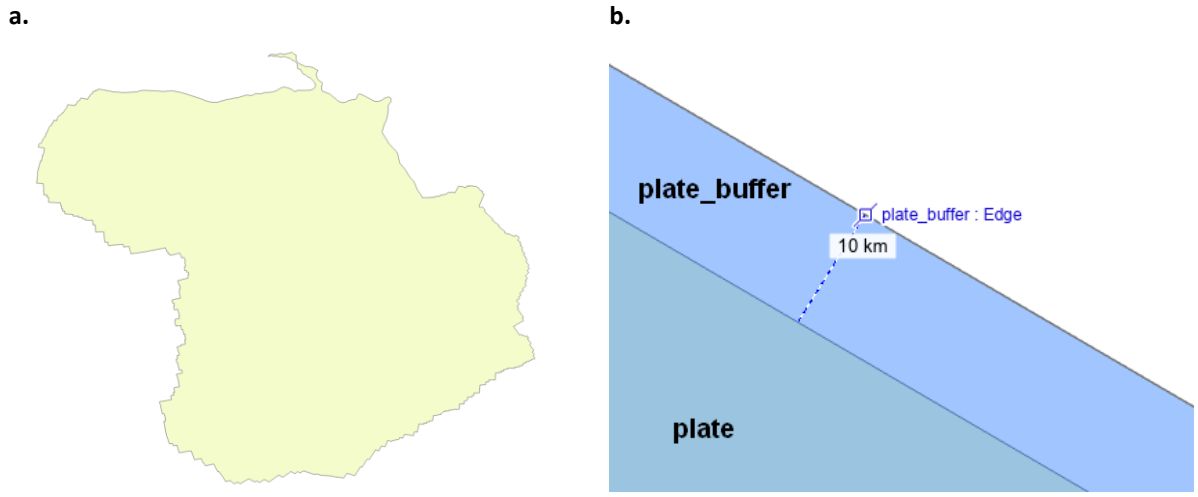


Figure 15 : a. exemple de la plaque d'Afrique. b. Zone tampon de 10 km (= 0,05 degré décimal) autour de la plaque.

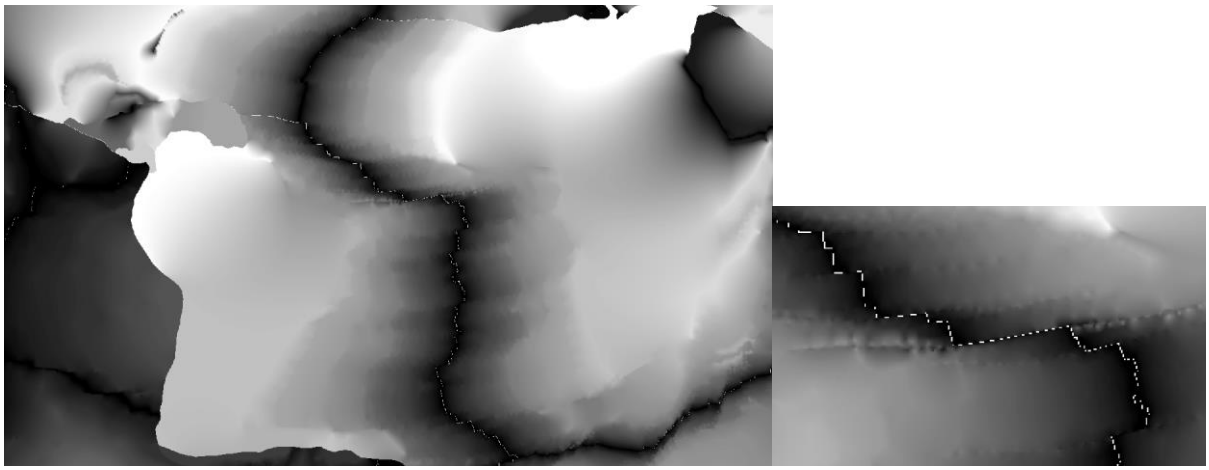


Figure 16 : exemple de ces « lignes blanches » dans le cas où la couche *plate_buffer* n'est pas utilisée.

LIGNES DES ISOCHRONES

Seuls les isochrones à l'intérieur de la plaque sont gardées pour en faire un vecteur ligne avec un `SelectLayerByLocation()` (figure 17.a). Il est possible que certaines plaques n'aient pas du tout de croûte océanique. Dans ce cas aucune ligne n'est sélectionnée dans la couche *world_isochrons* : ces plaques sont ignorées et enregistrées dans la liste *No_crust* et le code passe à la plaque suivante.

```
    ## LINES ---------------------------------------------------------------------------------
    # Select isochron, passive margins and ridge inside the plate, and make a layer
    selection = arcpy.management.SelectLayerByLocation(TimeMap, 'HAVE_THEIR_CENTER_IN',
plate_buffer, "", "NEW_SELECTION", "NOT_INVERT")
    SQL = "AGE < 999 AND (TYPE = 'Isochron' OR TYPE = 'Passive_Margin' OR TYPE = 'Ridge')"
    selection = arcpy.management.SelectLayerByAttribute(selection, 'SUBSET_SELECTION', SQL)
    # Skip this plate if no oceanic crust
    if int(arcpy.GetCount_management(selection)[0]) == 0:
        arcpy.AddMessage("The plate %s has no oceanic crust --> skip"%plt)
        No_crust.append(plt)
        continue
```

```
    flines = arcpy.management.CopyFeatures(selection, Process_gdb + "\\flines")
    arcpy.management.SelectLayerByAttribute(TimeMap, 'CLEAR_SELECTION')
    #arcpy.AddMessage("Lines made")
```

Comme certaines lignes dépassent les frontières du polygone, ce qui déborde est coupé (figure 17.b).

```
    # Cut lines that go beyond the plate
    lines = arcpy.analysis.Clip(in_features=flines, clip_features=plate_buffer,
out_feature_class=Process_gdb+"\\lines", cluster_tolerance="")
```

a. **b.**

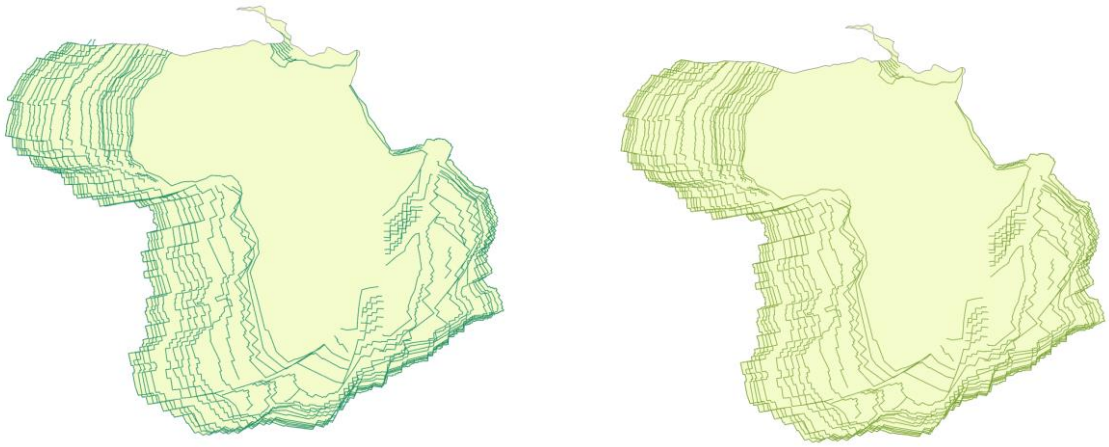


Figure 17 : a. *flines*, la couche dont les isochrones associées à la plaque dépassent encore de celle-ci. b. *lines*, la couche dont les isochrones sont coupés afin de ne pas déborder de la plaque.

Enfin, les sommets des lignes sont densifiés pour obtenir une plus grande précision de l'interpolation qui suivra. Dans la fonction `Densify()`, il est possible de réduire la valeur de distance entre les sommets. Cela améliora la résolution mais le temps de traitement de l'outil augmentera également. D'un point de vue personnelle, je dirais qu'une valeur de 1 est encore correcte pour la résolution.

```
    # Densify vertices
    lines = arcpy.edit.Densify(lines, "DISTANCE", distance="1 DecimalDegrees",
max_deviation="0,1 DecimalDegrees", max_angle=10, max_vertex_per_segment=None)
    #arcpy.AddMessage("Vertices densified")
```

POINTS DES ÂGES

Les sommets du vecteur ligne sont ensuite transformés en vecteur points.

```
    ## POINTS ------------------------------------------------------------------------------
    # Feature vertices to points
    points = arcpy.management.FeatureVerticesToPoints(lines, Process_gdb + "\\points",
point_location="ALL")
    #arcpy.AddMessage("Points made")
```

Figure 18 : points de la plaque Pacifique.

Cependant, la plaque n'est pas entièrement recouverte de points et certaines régions, notamment celles proches des zones de subduction, sont vides. Ce qui produit des trous sur la plaque lors de l'interpolation (figure 19.a). Il y a deux solutions possibles (alpha et bêta) :

A. Créer artificiellement des points qui longent la bordure de la plaque, leurs attribuer un âge en fonction de l'isochrone le plus proche et les intégrer à la couche *points* (figure 19.b). C'est la solution que j'applique dans l'outil, bien qu'elle soit moins fidèle à la réalité que la solution bêta.
B. L'idée est d'aller chercher les points manquants dans l'époque précédente, avant que la croûte océanique entre en subduction (pour plus de détails, voir *Améliorations* dans la discussion).

a.

b.

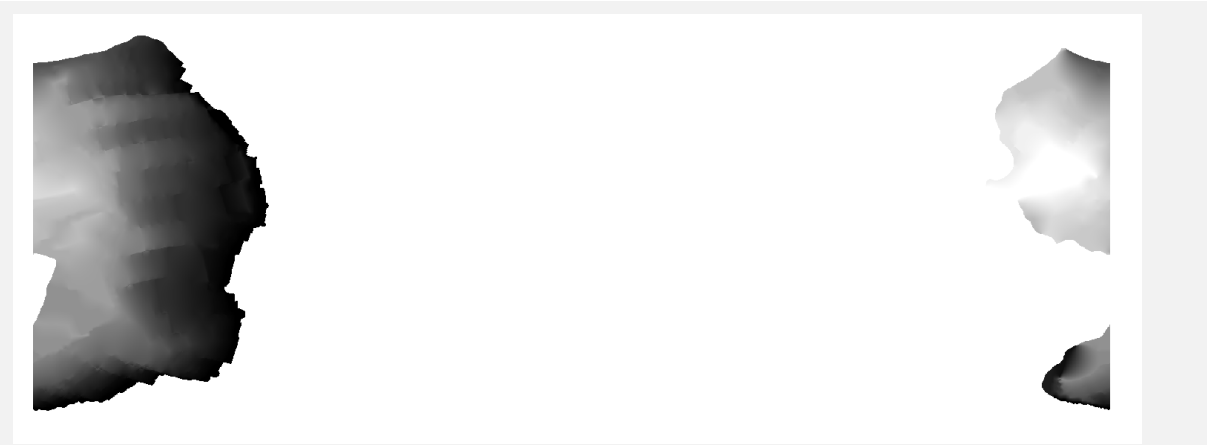


Figure 19 : a. Exemple de ces « trous » vers les zones de subduction pour la plaque Pacifique, si aucun point n'est rajouté. b. Exemple du résultat de la solution alpha. Les trous sont bouchés mais ces âges de la croûte océaniques sont cependant artificiels et ne sont pas tout à fait justes.

REMPLISSAGE DES ZONES SUBDUCTÉES (VERSION ALPHA)

L'idée est de créer une autre couche points contenant seulement ces points artificiels, puis de l'intégrer à la couche de points originale. Une copie est faite du polygone plaque car la densification modifie le polygone en entrée. Puis les sommets de cette copie sont transformés en points appelés *points_polygon* (figure 20).

```
     ## ALPHA VERSION: add the plate's boundary points (in case there are gaps in the
interpolation result): ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    # Make a copy of the plate polygon
    plate_polygon = arcpy.management.CopyFeatures(plate, Process_gdb + "\\plate_polygon")

    # Densify polygon's vertices
    plate_polygon = arcpy.edit.Densify(plate_polygon, "DISTANCE", distance="1 DecimalDegrees",
max_deviation="0,1 DecimalDegrees", max_angle=10, max_vertex_per_segment=None) # 0,1

    # Feature vertices to points
    points_polygon = arcpy.management.FeatureVerticesToPoints(plate_polygon, Process_gdb +
"\\points_polygon", point_location="ALL")
```

Figure 20 : couche points_polygon.

Ensuite, le champ *Age* est retiré (car inutile) ainsi que les points sur la ligne de temps internationale (figure 21). Pour ces derniers, les points proches de la ligne de changement de date à l'ouest et à l'est de la carte sont sélectionnés. Si le `SelectLayerByLocation()` a des éléments sélectionnés, ceux-ci sont supprimés avec `DeleteRows_management()`.

```
    # Remove unwanted AGE field:
```

```
    arcpy.DeleteField_management(points_polygon, ["AGE"])


    # Remove points that might be on the international Datelines
    pointsW = arcpy.management.SelectLayerByLocation(points_polygon, "WITHIN_A_DISTANCE",
DatelineW, "15 Meters", "NEW_SELECTION", "NOT_INVERT")
    if int(arcpy.GetCount_management(pointsW)[0]) > 0:
        arcpy.DeleteRows_management(pointsW)
    arcpy.management.SelectLayerByAttribute(pointsW, 'CLEAR_SELECTION')
    pointsE = arcpy.management.SelectLayerByLocation(points_polygon, "WITHIN_A_DISTANCE",
DatelineE, "15 Meters", "NEW_SELECTION", "NOT_INVERT")
    if int(arcpy.GetCount_management(pointsE)[0]) > 0:
        arcpy.DeleteRows_management(pointsE)
    arcpy.management.SelectLayerByAttribute(pointsE, 'CLEAR_SELECTION')
```

Figure 21 : couche points_polygon sans les points sur la ligne de temps internationale.

Avec l'outil `Near()`, chacun de ces points est associé à l'isochrones géographiquement le plus proche et les champs *NEAR_ID* et *NEAR_DIST* (le OBJECTID de l'isochrone et la distance séparant les deux objets en mètre) sont ajoutés à la couche *points_polygon*. Les deux couches sont ensuite jointes avec `JoinField()` par l'ID des isochrones et chaque point artificiel se voit donné un âge (figure 22). Puis, les points de la couche *points_polygon* sont ajoutés à la couche *points* en conservant tous les champs avec `Append()` (figure 23).

```
        # Attribute these new points an age (based on the nearest point with an age)
    points_polygon = arcpy.analysis.Near(in_features=points_polygon, near_features=lines,
search_radius="", location="NO_LOCATION", angle="NO_ANGLE", method="PLANAR",
field_names=[["NEAR_FID", "NEAR_FID"], ["NEAR_DIST", "NEAR_DIST"]])[0]

    # Join Age column
    points_polygon = arcpy.management.JoinField(in_data=points_polygon, in_field="NEAR_FID",
join_table=lines, join_field="OBJECTID", fields=["AGE"])[0]

    # Merge the 2 points layer
    points = POINTS = arcpy.management.Append(inputs=points_polygon, target=points,
schema_type="NO_TEST", field_mapping="ID \"ID\" true true false 8 Double 0
0,First,#,TROU\\points_polygon,Id,-1,-1;TYPE \"TYPE\" true true false 20 Text 0
0,First,#;LIMIT \"LIMIT\" true true false 3 Text 0 0,First,#;COB_LIMIT \"COB_LIMIT\" true true
false 3 Text 0 0,First,#;AGE \"AGE\" true true false 8 Double 0
0,Max,#,TROU\\points_polygon,AGE,-1,-1;NAME_TERR \"NAME_TERR\" true true false 50 Text 0
0,First,#;POSITION \"POSITION\" true true false 20 Text 0 0,First,#;REF_PLATE \"REF_PLATE\"
true true false 2 Text 0 0,First,#;IDTERRANE \"IDTERRANE\" true true false 4 Long 0
0,First,#;APPEARANCE \"APPEARANCE\" true true false 8 Double 0 0,First,#;DISAPPEARA
\"DISAPPEARA\" true true false 8 Double 0 0,First,#;PLATE \"PLATE\" true true false 20 Text 0
0,First,#;ORIG_FID \"ORIG_FID\" true true false 4 Long 0
0,First,#,TROU\\points_polygon,ORIG_FID,-1,-1", subtype="", expression="")[0]
```

points_polygon

Field: Selection:

NEAR_FID	NEAR_DIST	OBJECTID *	Shape *	Id	PlateName	ORIG_FID	AGE
18	40636,883441	3720	Point	0	Pac	1	3
18	43589,180232	3721	Point	0	Pac	1	3
18	46548,056625	3722	Point	0	Pac	1	3
18	50817,932422	3723	Point	0	Pac	1	3
18	55096,091695	3724	Point	0	Pac	1	3
18	59382,155921	3725	Point	0	Pac	1	3
18	63675,539965	3726	Point	0	Pac	1	3

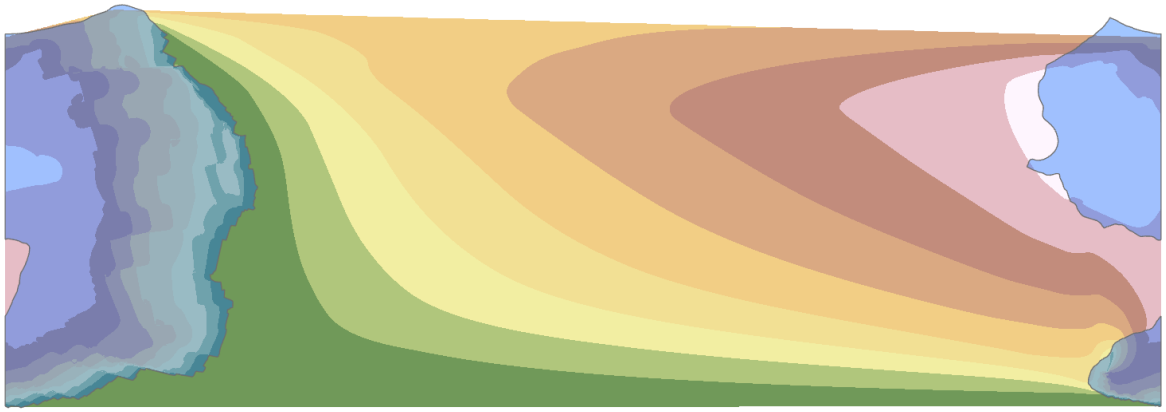
Figure 22 : Table d'attribut de la couche *points_polygon* après l'outil *JoinField()*. Les colonnes en jaunes sont celles ajoutées aux données des points artificiels. La colonne *AGE* à droite correspond donc à l'âge de l'isochrone le plus proche du point.

Figure 23 : couche points après l'outil *Append()*.

DÉTERMINER LE TYPE DE LA PLAQUE

Il y a 3 types de plaques possibles dans ce modèle : *normal*, *divisé* ou *pôle & divisé*. Cette distinction permet de déterminer si les points devront être déplacés d'un côté ou de l'autre de la ligne de temps internationale afin de ne former qu'un seul bloc de données. Cette étape est importante car l'interpolation peut être différente en fonction de l'emplacement des points, comme montré dans la figure 24 ci-dessous.

a.



b.

Figure 24 : Exemple de l'interpolation de la plaque Pacifique sans (a) et avec (b) le déplacement des points. Les valeurs à l'extérieur des plaques sont erronées. On remarque aussi que l'interpolation à l'intérieur des plaque n'est pas tout à fait la même.

Même si le polygone de plaque est visuellement divisé en deux parties ou plus, il est considéré comme un seul objet dans ArcGIS Pro. L'idée est donc de couper le polygone par la ligne de temps internationale avec `FeatureToPolygon()` : si le polygone se retrouve avec plus d'un objet dans sa table d'attribut, alors il s'agit d'une plaque *divisée* ou *pôle et divisée*. En utilisant l'outil `FeatureToPolygon()` cependant, il arrive que des artefacts s'ajoutent au vecteur, comme des polygones en surplus ou lignes dont l'aire égale une valeur proche de zéro. Pour enlever les polygones en trop, l'outil `Clip()` est utilisé. Pour ces lignes supplémentaires dans la table d'attribut, le code sélectionne celles dont l'aire est trop petite et les supprime.

```
    ## Determine the plate's type ----------------------------------------------------------
    # Split plate by international Dateline
    split = arcpy.management.FeatureToPolygon([plate, DatelineW], Process_gdb + "\\split" ,
"", "ATTRIBUTES", "")

    # Clip with original plate to remove extra polygons
    plateNew = arcpy.analysis.Clip(in_features=split, clip_features=plate,
out_feature_class=Process_gdb+"\\plateNew", cluster_tolerance="")

    # Deleting artefact rows
    SQL = "Shape_Area < 0.001"
    s_arte = arcpy.management.SelectLayerByAttribute(plateNew, 'NEW_SELECTION', SQL)
    if int(arcpy.GetCount_management(s_arte)[0]) > 0:
        arcpy.DeleteRows_management(s_arte)
    arcpy.management.SelectLayerByAttribute(s_arte, 'CLEAR_SELECTION')
```

Ensuite, un type est attribué à la plaque en fonction du résultat des sélections faites. Si ArcGIS Pro arrive à sélectionner des lignes avec la condition `if num_rows > 1`, c'est-à-dire s'il y a plus d'un objet dans la table d'attribut, alors la plaque est divisée. Autrement elle est normale. Si elle est divisée, le code regarde s'il est ensuite possible de sélectionner les plaques se situant sur les pôles. Si au moins un élément est sélectionné (il arrive qu'une seule plaque se situe sur les deux pôles tellement elle est grande), alors la plaque est divisée et au pôle (figure 26). Sinon elle juste divisée (figure 25). La fonction `GetCount_management()` permet justement de compter le nombre d'éléments sélectionnés (ou pas) dans une couche.

```
    # What kind of plate do we have?
    num_rows = int(arcpy.GetCount_management(plateNew)[0])
    arcpy.AddMessage("Number of plate(s): " + str(num_rows))
    if num_rows > 1: # "if more than 1 row, then plate is divided"

        # Type ...
        PN = arcpy.management.SelectLayerByLocation(plateNew, 'INTERSECT', PoleNord, "",
"NEW_SELECTION", "NOT_INVERT")
        NPN = int(arcpy.GetCount_management(PN)[0])
        arcpy.management.SelectLayerByAttribute(plate, 'CLEAR_SELECTION')
```

```
        PS = arcpy.management.SelectLayerByLocation(plateNew, 'INTERSECT', PoleSud, "",
"NEW_SELECTION", "NOT_INVERT")
        NPS = int(arcpy.GetCount_management(PS)[0])
        arcpy.management.SelectLayerByAttribute(plate, 'CLEAR_SELECTION')

        if (NPN > 0 or NPS > 0): # "if part of the plate is at the poles..."
            TYPE = "Pole & Divided"
        else:
            TYPE = "Divided"
    else:
        TYPE = "Normal"
    arcpy.AddMessage("Plate's type is %s"%TYPE)
```

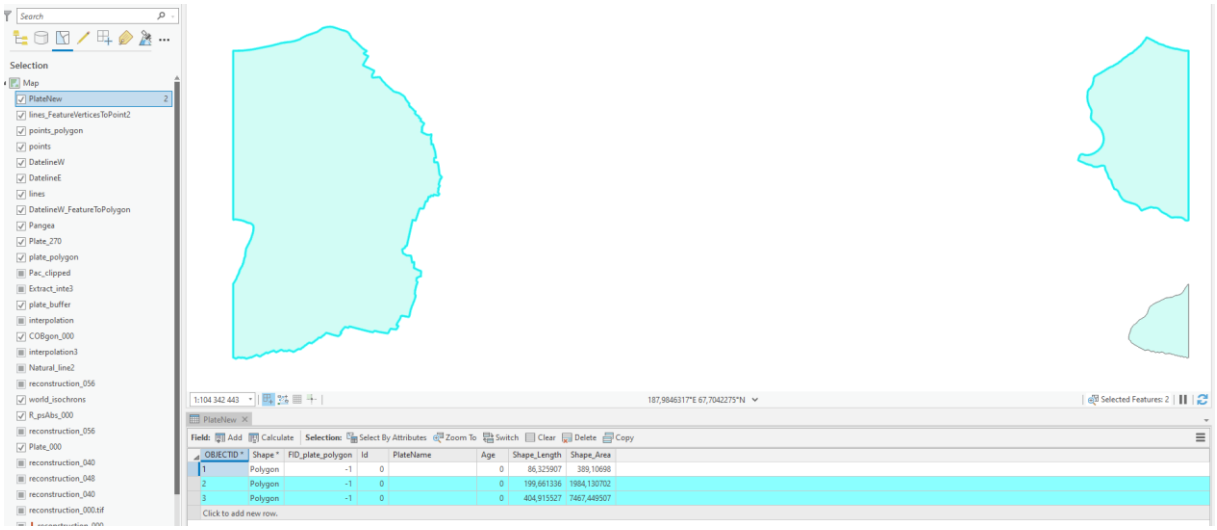


Figure 25 : Exemple de la requête « if num_rows > 1 » pour la plaque Pacifique. Des éléments sont sélectionnés, la plaque est donc *divisée*.

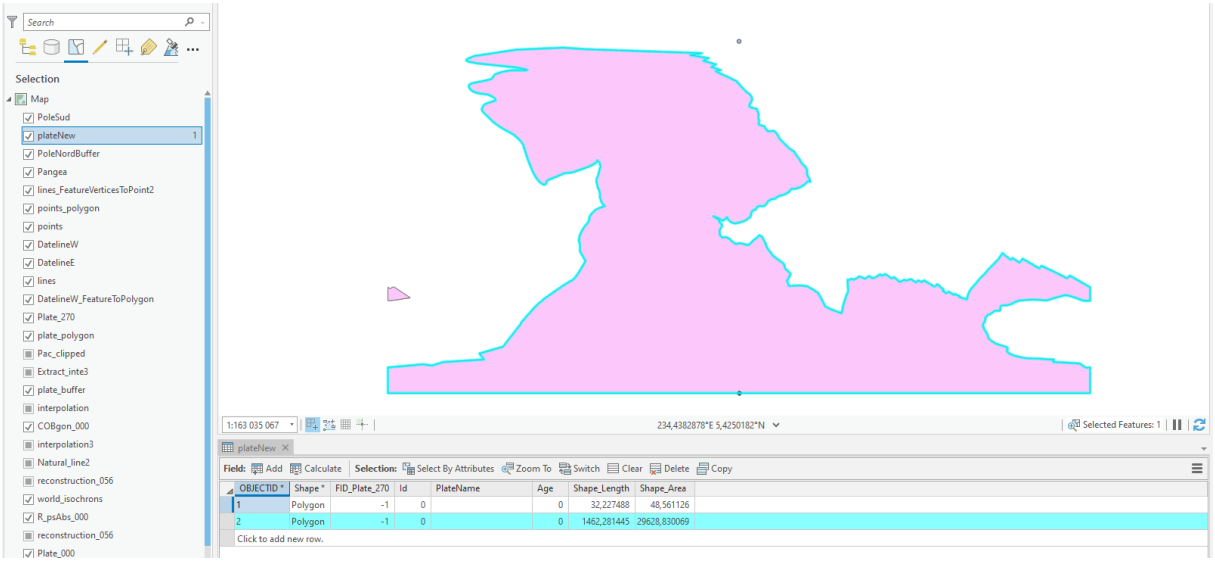


Figure 26 : Exemple d'une plaque *pôle et divisée* avec la Pangée, il y a 270 Ma.

DÉPLACEMENT DES POINTS

Cette partie du code s'occupe de déplacer les points si nécessaire. Ainsi, si la plaque est de type divisé, les points appartenant au(x) polygone(s) touchant la ligne de temps international ouest sont sélectionnés puis déplacés de 360 degrés vers l'est.

Pour effectuer le déplacement avec ArcPy, il faut utiliser la fonction `UpdateCursor()` qui permet de modifier les éléments d'une couche vecteur (ici, la sélection *s_points* de la couche *points*), et dans notre cas, modifier les coordonnées de nos points `SHAPE@XY`. Les points sélectionnés sont ceux à une distance de 50 km de la plaque à l'ouest de la carte. Avec une *for loop*, le code passe sur chaque point appartenant à la couche et grâce à `updateRow()` il modifie les coordonnées X (`row[0][0]`) et Y (`row[0][1]`) en leur ajoutant la valeur des variables *xOffset* et *yOffset*.

```
    ## Move points if needed ----------------------------------------------------------------
    if TYPE == "Divided":
        s_dateline = arcpy.management.SelectLayerByLocation(plateNew, 'INTERSECT',
Dateline_zone, "", "NEW_SELECTION", "NOT_INVERT")
        s_points = arcpy.management.SelectLayerByLocation(points, "WITHIN_A_DISTANCE",
s_dateline, "50 Kilometers", "SUBSET_SELECTION", "NOT_INVERT")
        arcpy.management.SelectLayerByAttribute(s_dateline, 'CLEAR_SELECTION')
        xOffset = 360 # move towards east
        yOffset = 0
        # Move points
        with arcpy.da.UpdateCursor(s_points, ["SHAPE@XY"]) as cursor:
            for row in cursor:
                cursor.updateRow([[row[0][0] + xOffset, row[0][1] + yOffset]])
        arcpy.management.SelectLayerByAttribute(s_points, 'CLEAR_SELECTION')
```

Figure 27 : Couche *points* une fois que les points sélectionnés sont déplacés de 360 degrés vers l'est, pour la plaque *divisée* Pacifique.

Les points d'une plaque au pôle ne peuvent pas être déplacés de la même manière que ceux d'une plaque simplement divisée, car les points de la plaque au pôle (qui touche *DatelineW*) et ceux du morceau de plaque seront tous déplacés, ce qui ne changera absolument rien.

Il faut donc pouvoir sélectionner seulement les points de la plaque isolée qui n'est pas sur l'un des deux pôles. Puis, selon si ce polygone se situe à l'est ou à l'ouest de la carte, ses points sont déplacés vers l'ouest ou l'est, respectivement. Pour rappel, *NPN* et *NPS* sont le nombre de polygones qui intersectent avec le point des pôles Nord et Sud (déterminés lors de l'étape *Déterminer le type de plaque*) et permettent de savoir avec quel pôle nous travaillons. Ainsi, si `NPN > 0`, on sélectionne l'inverse de la plaque du pôle (i.e. le petit morceau de plaque) et on vérifie si ce polygone intersecte avec la ligne de temps internationale ouest. Si un élément est sélectionné, le polygone est à l'ouest et `xOffset = 360`, autrement le morceau est à l'est de la carte et `xOffset = -360`.

Ensuite, il faut à nouveau sélectionner le polygone isolé (le code ne marchait pas si je ne répétais pas cette ligne une nouvelle fois), et à partir de cette sélection on peut sélectionner les points appartenant à la petite plaque. Puis, comme pour les plaques divisées, les coordonnées des points sont modifiées (figure 28).

```
    elif TYPE == "Pole & Divided":
        if NPN > 0: # if at North pole
            s_not_Npole = arcpy.management.SelectLayerByLocation(plateNew, 'INTERSECT',
PoleNord, "", "NEW_SELECTION", "INVERT")
            s_micro_plate = arcpy.management.SelectLayerByLocation(s_not_Npole, 'INTERSECT',
Dateline_zone, "", "SUBSET_SELECTION", "NOT_INVERT")
            if int(arcpy.GetCount_management(s_micro_plate)[0]) > 0: # if microplate is at the
west of the map
```

```
                xOffset = 360 # move towards east
            else:
                xOffset = -360 # move towards west
            s_not_Npole = arcpy.management.SelectLayerByLocation(plateNew, 'INTERSECT',
PoleNord, "", "NEW_SELECTION", "INVERT")
            s_points = arcpy.management.SelectLayerByLocation(points, "WITHIN_A_DISTANCE",
s_not_Npole, "50 Kilometers", "SUBSET_SELECTION", "NOT_INVERT")
            arcpy.AddMessage(int(arcpy.GetCount_management(s_points)[0]))

            arcpy.management.SelectLayerByAttribute(s_not_Npole, 'CLEAR_SELECTION')
            arcpy.management.SelectLayerByAttribute(s_micro_plate, 'CLEAR_SELECTION')

        elif NPS > 0: # if at South pole
            s_not_Spole = arcpy.management.SelectLayerByLocation(plateNew, 'INTERSECT',
PoleSud, "", "NEW_SELECTION", "INVERT")
            s_micro_plate = arcpy.management.SelectLayerByLocation(s_not_Spole, 'INTERSECT',
Dateline_zone, "", "SUBSET_SELECTION", "NOT_INVERT")
            if int(arcpy.GetCount_management(s_micro_plate)[0]) > 0: # if microplate is at the
west of the map
                xOffset = 360 # move towards east
            else:
                xOffset = -360 # move towards west
            s_not_Spole = arcpy.management.SelectLayerByLocation(plateNew, 'INTERSECT',
PoleSud, "", "NEW_SELECTION", "INVERT")
            s_points = arcpy.management.SelectLayerByLocation(points, "WITHIN_A_DISTANCE",
s_not_Spole, "50 Kilometers", "SUBSET_SELECTION", "NOT_INVERT")
            arcpy.AddMessage(int(arcpy.GetCount_management(s_points)[0]))

            arcpy.management.SelectLayerByAttribute(s_not_Spole, 'CLEAR_SELECTION')
            arcpy.management.SelectLayerByAttribute(s_micro_plate, 'CLEAR_SELECTION')

        # Move points
        yOffset = 0
        with arcpy.da.UpdateCursor(s_points, ["SHAPE@XY"]) as cursor:
            for row in cursor:
                cursor.updateRow([[row[0][0] + xOffset, row[0][1] + yOffset]])
        arcpy.AddMessage("Points moved")
        arcpy.management.SelectLayerByAttribute(s_points, 'CLEAR_SELECTION')
```

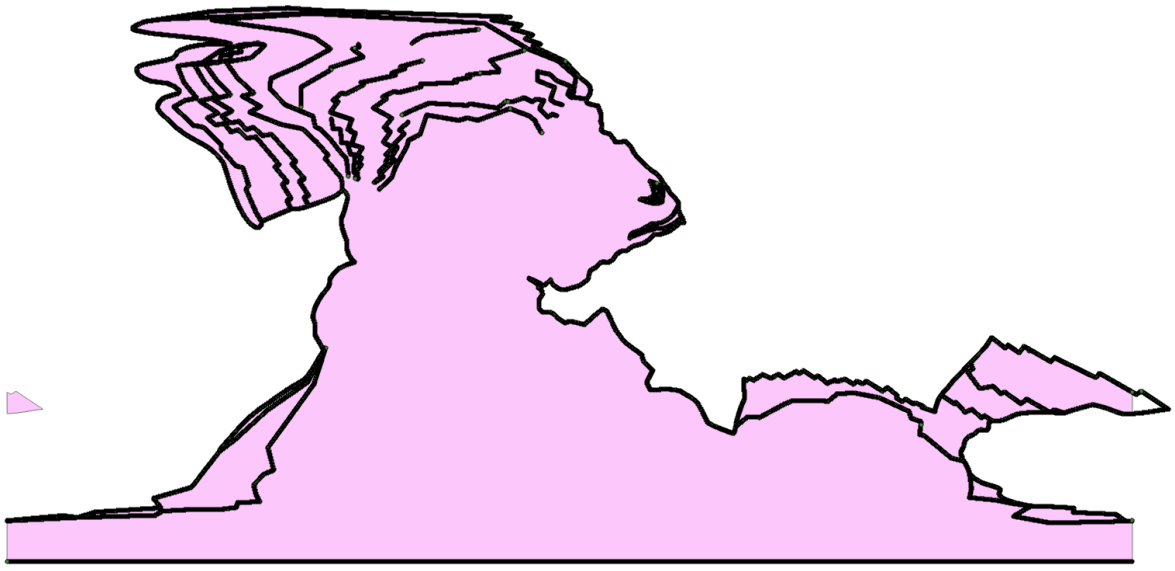


Figure 28 : Couche *points* de la plaque *Pangea* d'il y a 270 Ma après avoir déplacé les points de la petite plaque à l'ouest.

Si la plaque est normale, les points restent inchangés.

```
    else: # TYPE == "Normal":
        arcpy.AddMessage("Points unchanged")
```

INTERPOLATION DES ÂGES DE LA CROÛTE

Les différents âges de la croûte son interpolés avec la fonction `NaturalNeighbor()` (figure 29). Il est important d'utiliser un *try – except bloc* car certaines plaques n'ont pas de croûte océanique, comme c'est le cas pour la

plaque *Turkish* par exemple. Ainsi, le code ne provoque pas d'erreur, la plaque peut être ignorée et le code passe à la plaque suivante.

```
    # Interpolation ---------------------------------------------------
    try: # because not working for plates without oceanic crust
        arcpy.ddd.NaturalNeighbor(points, "AGE", Process_gdb + "\\interpolation", "0,1")
        interpolation = arcpy.Raster(Process_gdb + "\\interpolation")
        arcpy.AddMessage("Interpolation")
    except Exception as e:
        arcpy.AddMessage("No interpolation done for %s :"%plt)
        arcpy.AddMessage("ERROR: "+str(e))
        continue
```

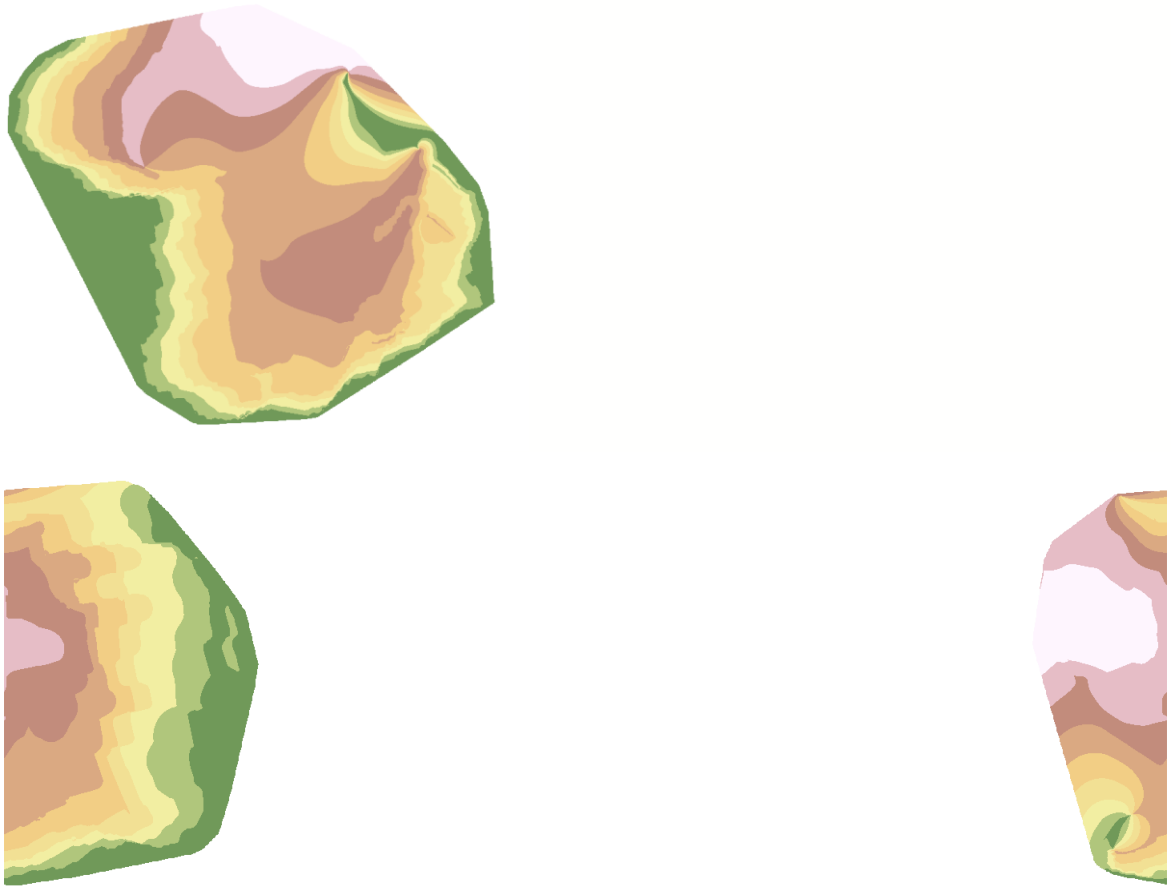


Figure 29 : Exemple d'interpolation avec l'outil *NaturalNeighbor()* pour les plaques d'Afrique et du Pacifique.

Enfin, le résultat de l'interpolation est découpé à l'emporte-pièce avec le polygone *plate_buffer* (figure 30). Une fois assemblées, les plaques se chevaucheront donc légèrement afin d'éviter de potentiels trous de données dans la reconstruction.

Ce raster de plaque final est ensuite enregistré dans la liste *RC_List* qui sera utilisée à l'étape suivante.

Le code passe ensuite à la plaque suivante, ou sort du *loop* pour passer à l'étape de reconstruction de l'âge si toutes les plaques ont été itérées.

```
# Remove what overflows from the plate's boundaries ---------------------------------------
    Clip = arcpy.sa.ExtractByMask(in_raster=interpolation, in_mask_data=plate_buffer)
    Clip.save(Process_gdb + "\\%s_clipped"%plt)
    arcpy.AddMessage("Clipped")

    # Get a list of all the rasters of the reconstruction
    RC_List.append(Clip)
```

```
    # --- End of the plates loop ---
```

Figure 30 : Exemple des reconstructions de plaque, pour l'Afrique et le Pacifique.

PARTIE 4 : RECONSTRUCTION DE L'ÂGE

Maintenant que toutes les plaques ont été reconstruites, il est temps de les assembler pour donner la reconstruction complète de la période avec l'outil `MosaicToNewRaster()` (figure 31). Les rasters en entrée sont contenus dans la liste *RC_List* qui enregistrait chaque plaque au fur et à mesure de la *loop* des plaques.

```
    ### Finalizing the reconstruction ============================================================
    arcpy.AddMessage(RC_List)
    # Merge the rasters
    mozaic = arcpy.management.MosaicToNewRaster(input_rasters=RC_List,
output_location=Process_gdb, raster_dataset_name_with_extension="mozaic",
coordinate_system_for_the_raster="", pixel_type="32_BIT_UNSIGNED", cellsize=None,
number_of_bands=1, mosaic_method="LAST", mosaic_colormap_mode="FIRST")[0]
    mozaic = arcpy.Raster(mozaic)
    arcpy.AddMessage("Mozaic")
```

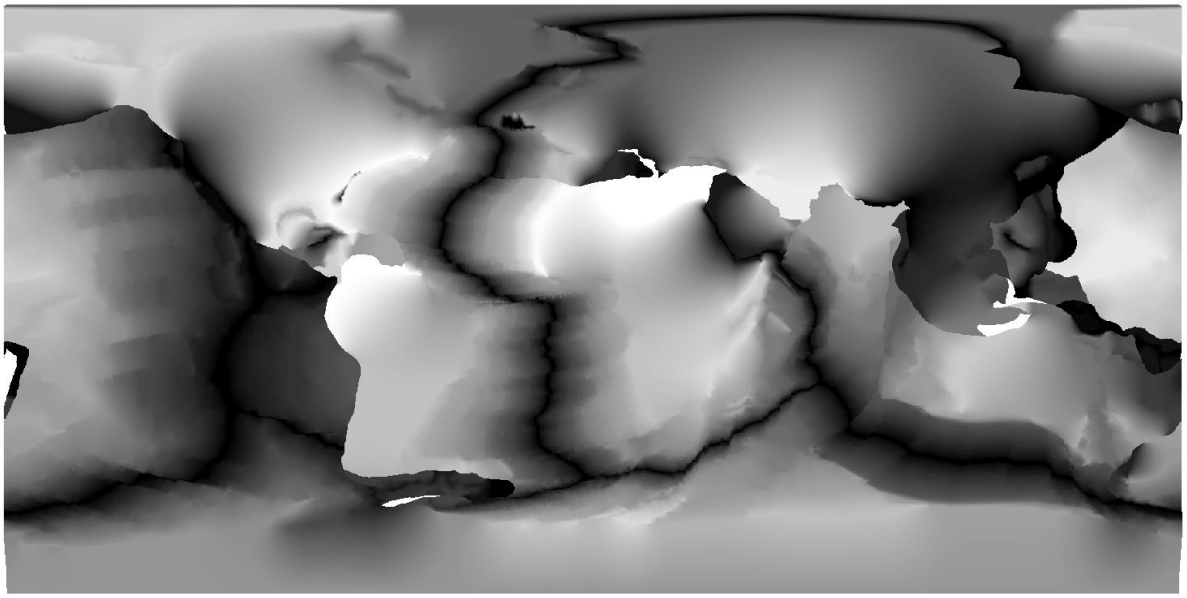


Figure 31 : couche *mozaic*.

Ensuite, il faut enlever la croûte continentale dont l'interpolation des âges ne fait aucun sens. Pour cela, il faut dans un premier temps découper à l'emporte-pièce le raster *mozaic* avec le polygone de la surface des continents en utilisant `ExtractByMask()`. La zone tampon d'environ 5 km préalablement produite autour des continents est utilisée afin de complètement retirer les pixels associés à la croûte continentale (figures 32 et 33). Cependant, en utilisant l'outil `ExtractByMask()`, le raster en sortie appelé *inverse* est l'interpolation de l'âge de la croûte continentale, et non pas de la croûte océanique (figure 34). La ligne XXX est l'équivalent de *Environnements* dans l'outil de l'interface graphique d'ArcGIS et en spécifiant l'extension à MAXOF, on s'assure que le raster de sortie s'étendra sur toutes les données en entrée.

```
    # Remove continents
    Continents_buffer = Process_gdb + "\\Continents_buffer"
    arcpy.analysis.Buffer(in_features=Continents, out_feature_class=Continents_buffer,
buffer_distance_or_field="0,05 DecimalDegrees", line_side="FULL", line_end_type="ROUND",
dissolve_option="NONE", dissolve_field=[], method="PLANAR")

    # Make a raster of the continents only
    with arcpy.EnvManager(extent="MAXOF"):
        inverse = arcpy.sa.ExtractByMask(in_raster=mozaic, in_mask_data= Continents_buffer)
        inverse.save(Process_gdb + "\\inverse")
    #arcpy.AddMessage("Inverse")
```

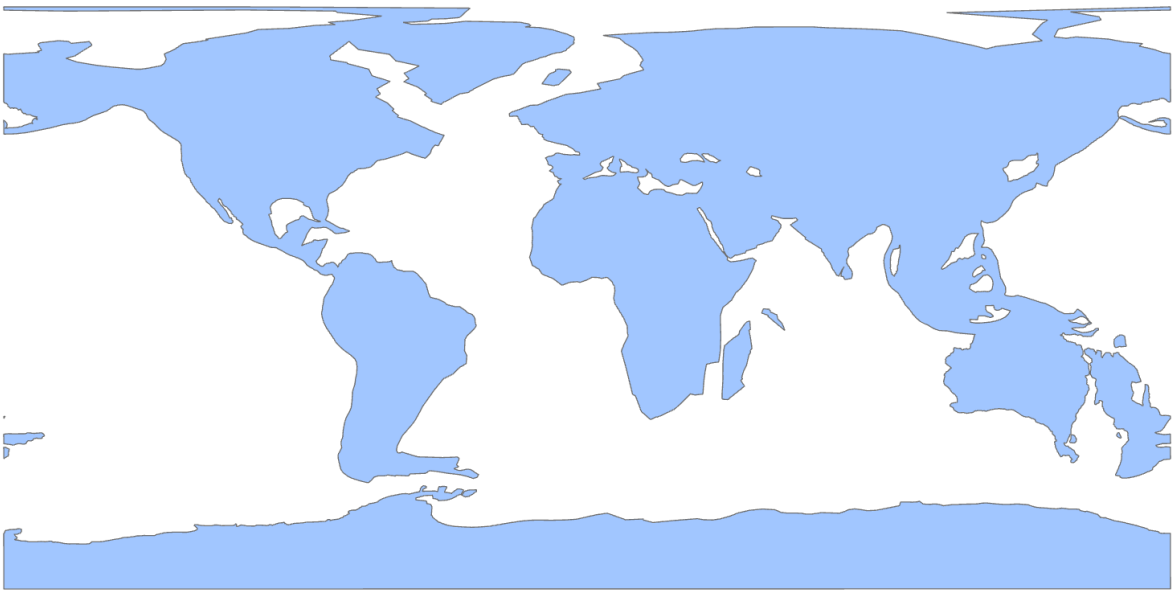


Figure 32 : Couche *Continents_buffer*.

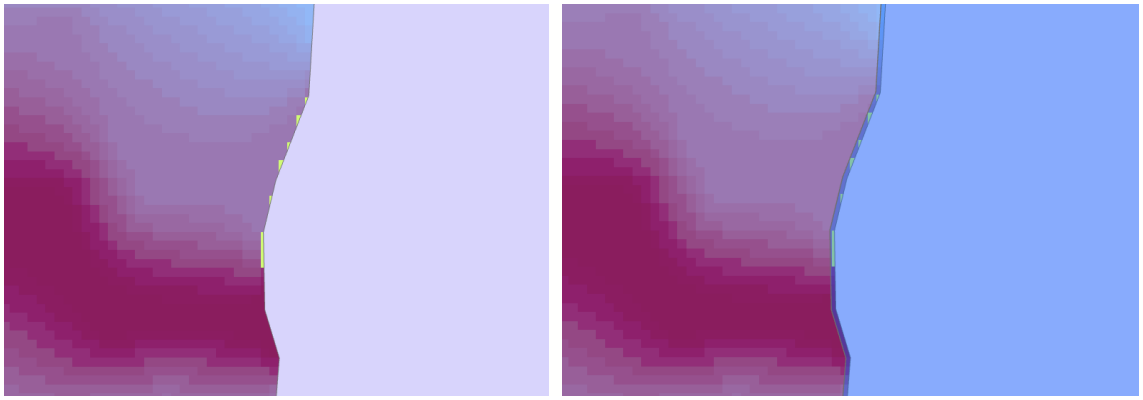


Figure 33 : À gauche : couche de la croûte continentale sur la couche *mozaic* : on remarque que certains pixels appartenant au continent ne sont pas complètement recouverts par le polygone. À droite : le polygone *Continents_buffer* recouvre ces pixels indésirables.

Figure 34 : couche *inverse*.

Il faut donc, dans un deuxième temps, inverser la zone d'intérêt avec l'outil `RasterCalculator()` en utilisant les couches *inverse* et *mozaic*. La commande `(SetNull(~(IsNull( inverse)), mozaic))` permet de retirer de la couche *mozaic* la surface de la couche *inverse* et d'obtenir la surface de la croûte océanique, appelée *reconstruction* (figure 35). La suite de la commande `- %s"%min` permet de modifier la valeur des pixels du rasters de sortie afin qu'un pixel d'âge 0 corresponde à l'âge de la croûte nouvellement formée pour la période considérée. La variable *min* est la plus petite valeur du raster *mozaic*, et tous ses pixels sont ensuite soustraits par *min* pour obtenir l'âge de la croûte à l'époque de la reconstruction (figure 36).

```
    # Recontruction
    min = mozaic.minimum
    expression = "(SetNull(~(IsNull( inverse)), mozaic)) - %s"%min
    reconstruction = RasterCalculator([inverse, mozaic], ["inverse", "mozaic"], expression,
"FirstOf")
    reconstruction.save(Maps_gdb + "\\reconstruction_%s"%age)

    arcpy.AddMessage("Continental crust removed")
```

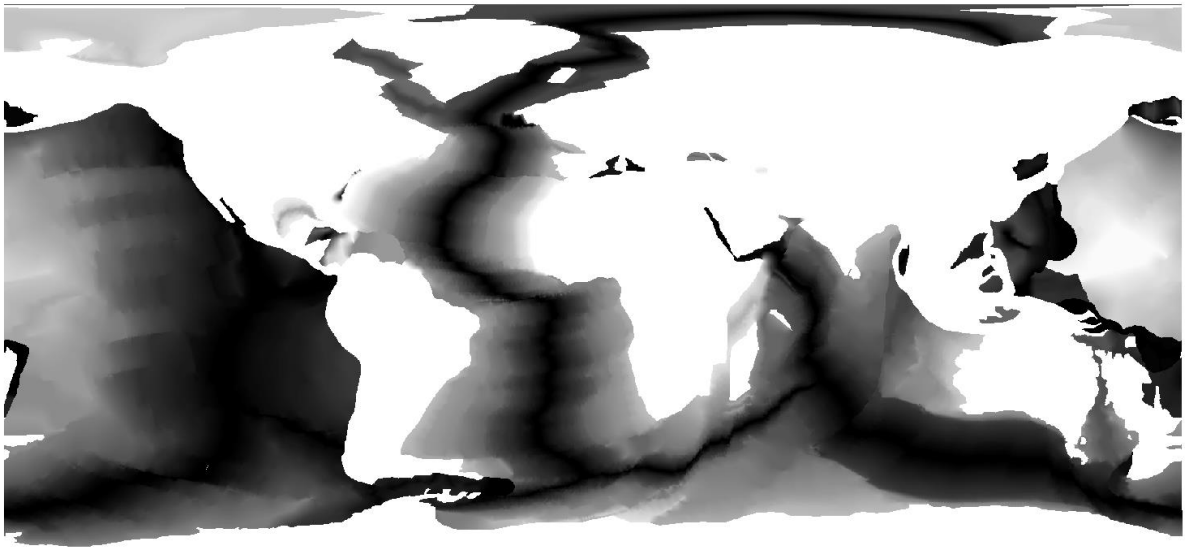


Figure 35 : Couche *reconstruction*.

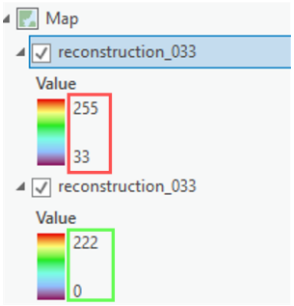


Figure 36 : Résultat de l'instruction `- %s"%min` dans l'outil *RasterCalculator*, avant (encadré en rouge) et après (encadré en vert) son application pour la reconstruction de la période 033.

PARTIE 5 : SYMBOLOGIE ET MESSAGES D'AVERTISSEMENTS

SYMBOLOGIE

Pour une meilleure visualisation, la symbologie de la reconstruction est changée. Pour cela, il faut référencer le projet, la géodatabase, la carte et la couche raster.

```
    ### Change symbology ================================================================
```

```
    # Reference the project
    aprx = arcpy.mp.ArcGISProject("CURRENT")

    # Set the default geodatabase
    aprx.defaultGeodatabase = Maps_gdb

    # Reference items in the project
    mp = aprx.listMaps("Map")[0]

    # Add the reconstruction to the map (drag and drop)
    mp.addDataFromPath(Maps_gdb + "\\reconstruction_%s"%age)
    arcpy.AddMessage("Raster added to the map")

    # Select the raster
    lyr = mp.listLayers()[0]
```

Puis à partir de ce raster, on modifie sa symbologie en définissant la couleur et le type d'affichage des valeurs (dans notre cas *Minimum Maximum* ; figure 37). Pour changer la couleur de la symbologie, il suffit de remplacer `'Bathymetric Scale'` par un autre nom de couleur, comme ceux proposés dans la figure 38.

```
    ## Change symbology
    sym = lyr.symbology
    if hasattr(sym, "colorizer"):
        if sym.colorizer.type == "RasterStretchColorizer":
            sym.colorizer.colorRamp = aprx.listColorRamps('Bathymetric Scale')[0] # or
'Multipart Color Scheme' for example
            sym.colorizer.stretchType = "MinimumMaximum"
            lyr.symbology = sym
    else:
        arcpy.AddMessage("Symbology: no hasattr")
```

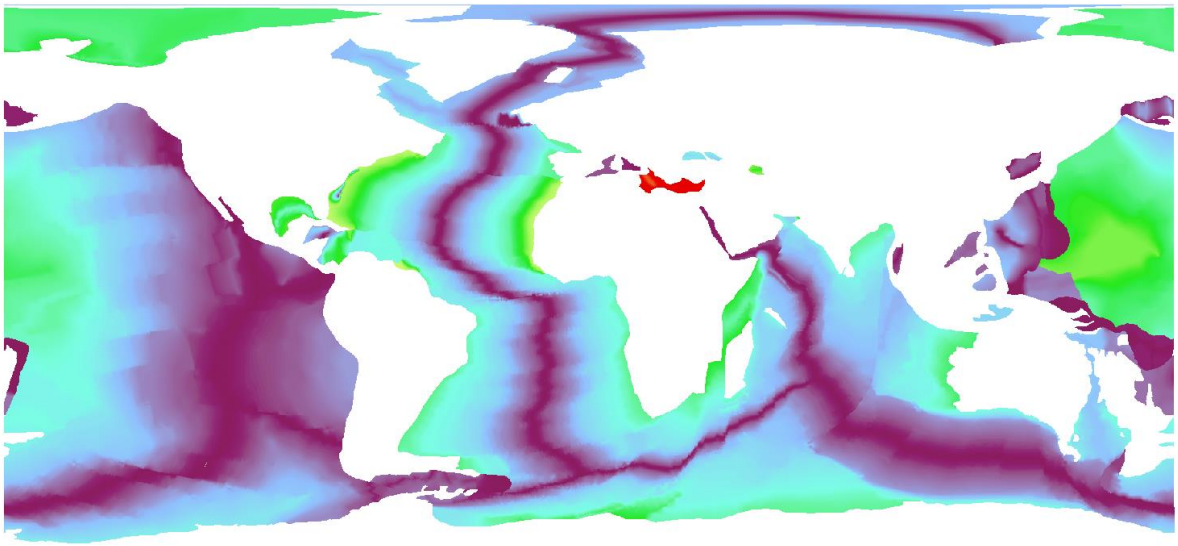
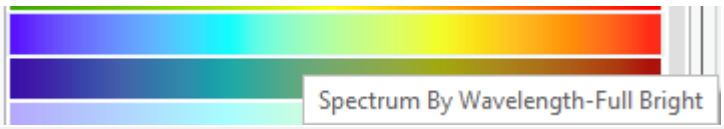
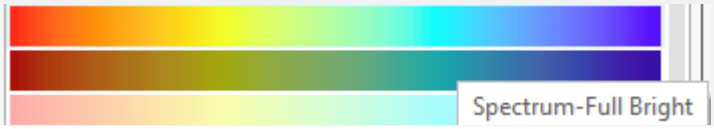


Figure 37 : couche *reconstruction*, une fois la symbologie appliquée.

a.

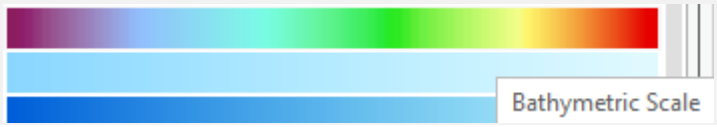


b.



c.

d.



e.

Figure 38 : a. *Spectrum by Wavelength-Full Bright*. b. *Spectrum-Full Bright*. c. *Precipitation*. d. *Bathymetric Scale*. e. *Temperature*.

MESSAGES D'AVERTISSEMENTS

La reconstruction de l'âge se termine par un message indiquant le nom des plaques sans croûte océanique, à conditions que cette liste contienne des noms (figure 39).

Puis un second message apparaît pour indiquer la fin de la reconstruction de l'âge en cours.

Le `else` fait écho au `if arcpy.Exists(PlatesMap):` au début de la partie 3 *Loop des âges*. Si le code tente de reconstruire un âge qui n'a pas de shapefiles associés, alors un message s'affiche pour nous avertir que la reconstruction n'est pas possible et cet âge est enregistré dans la liste *Not_done*, préalablement créée. Le code passe ensuite à l'âge suivant ou sort du *loop*. Lorsque toutes les reconstructions sont faites, le code termine avec un potentiel message recensant la liste *Not_done* si celle-ci contient des âges (figure 40). Cette liste peut être surtout utile lorsque que l'on demande les reconstructions de plusieurs âges simultanément.

```
        ### Warnings ================================================
        if len(No_crust) > 0:
            arcpy.AddMessage("Plates without oceanic crust (which will results in gaps):")
            arcpy.AddMessage(No_crust)

        arcpy.AddMessage("=== Reconstruction completed ===")
        # --- End of the Age loop ---

    else:
        arcpy.AddMessage("There is no shapefile for age %s"%age)
        Not_done.append(age)

    # --- End of looping through all ages ---

if len(Not_done) > 0:
    arcpy.AddMessage("Ages for which no reconstruction has been made:")
    arcpy.AddMessage(Not_done)
```

```
Plates without oceanic crust (which will results in gaps):
['Turkish', 'Snow', 'Banda', 'Molucca']
=== Reconstruction completed ===
Succeeded at mardi, 30 août 2022 18:33:12 (Elapsed Time: 12 minutes 6 seconds)
```

Figure 39 : Exemple de la liste *No_crust*.

```
Ages for which no reconstruction has been made:
['155', '156', '157', '158', '159', '160', '161', '162', '163', '164', '166',
'167', '168', '169', '170']
```

Figure 40 : Exemple de la liste *Not_done*.

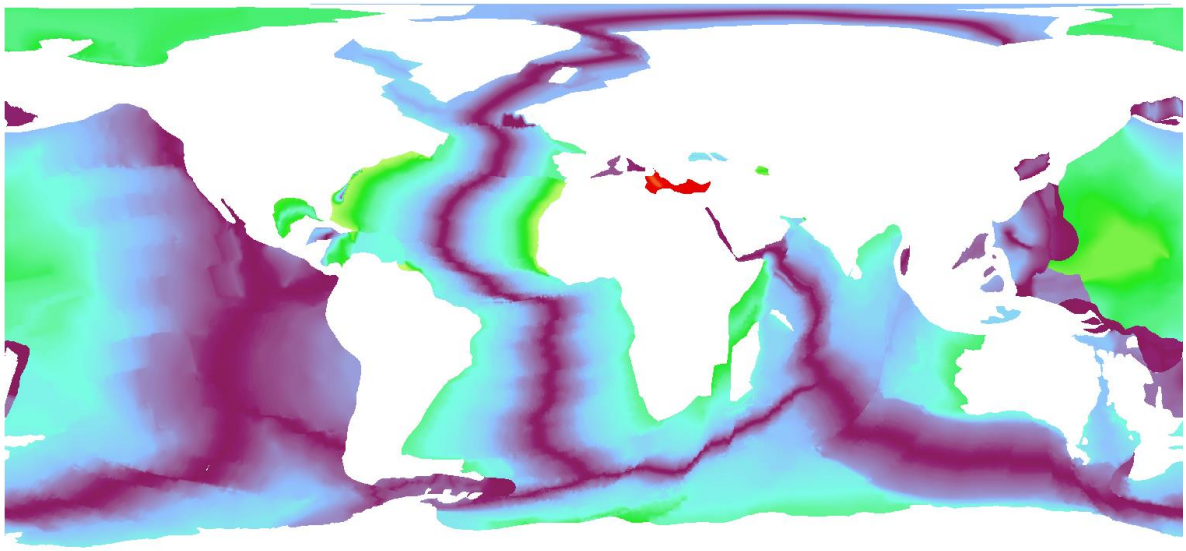
RÉSULTATS

L'outil *SeaFloorAge* produit donc un raster de l'âge de la croûte océanique pour chaque période donnée, au format FGDBR. La taille du raster fait environ 50 MB et la résolution est de 0,1 x 0,1. Les figures suivantes sont quelques exemples de reconstruction que l'outil a produits.

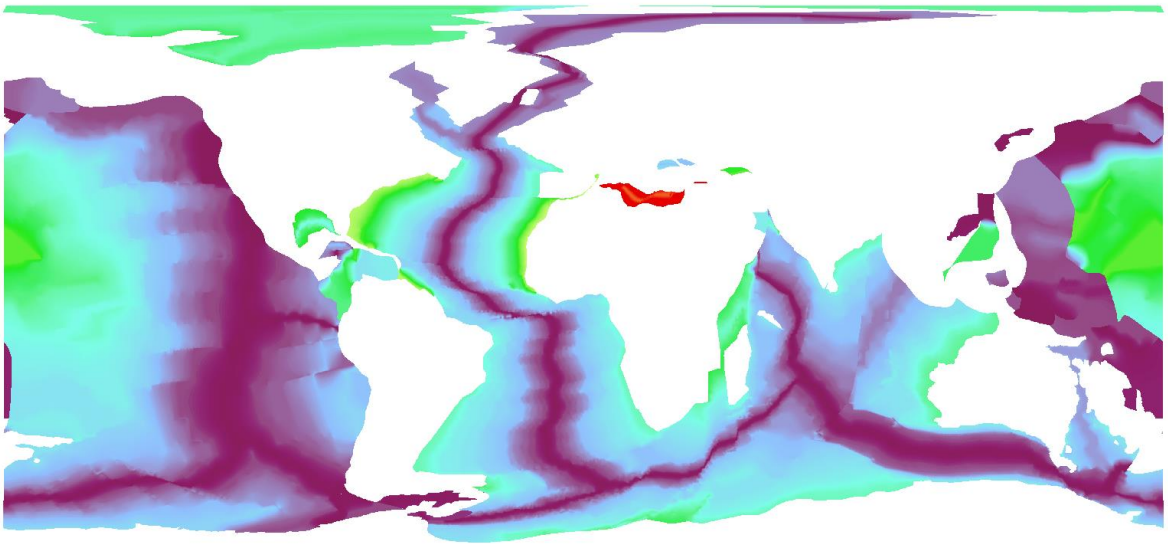
Le modèle Panalesis, dont proviennent les données, se base sur un modèle d'héritage. C'est-à-dire que les données de la reconstruction antérieure sont prises en compte afin de produire la suivante. De ce fait, les reconstructions des plus anciennes périodes (proche de 600 Ma) pourraient être moins fiables car moins contraintes par les âges de croûte antérieurs.

Il est recommandé de faire tourner le code sur un ordinateur ayant plusieurs cœurs. En effet, faire une reconstruction prend beaucoup de temps : environ 35 minutes pour un PC de 4 processeurs logistiques et 4 cœurs. Pour un ordinateur ayant 16 processeurs logistiques et 8 cœurs, la reconstruction d'une carte prend 20 minutes environ.

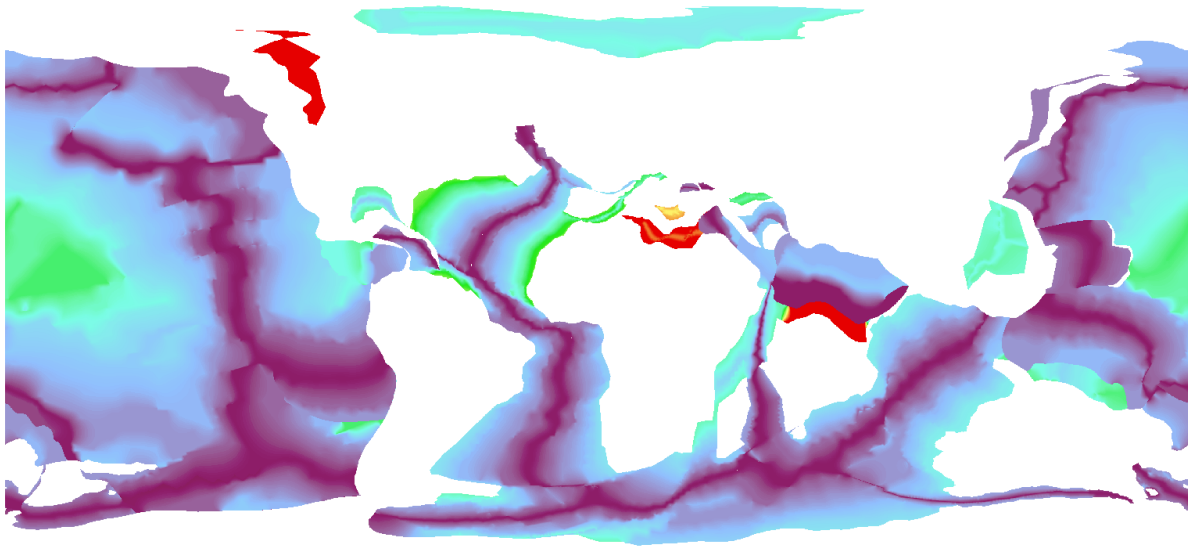
a. *reconstruction_000*



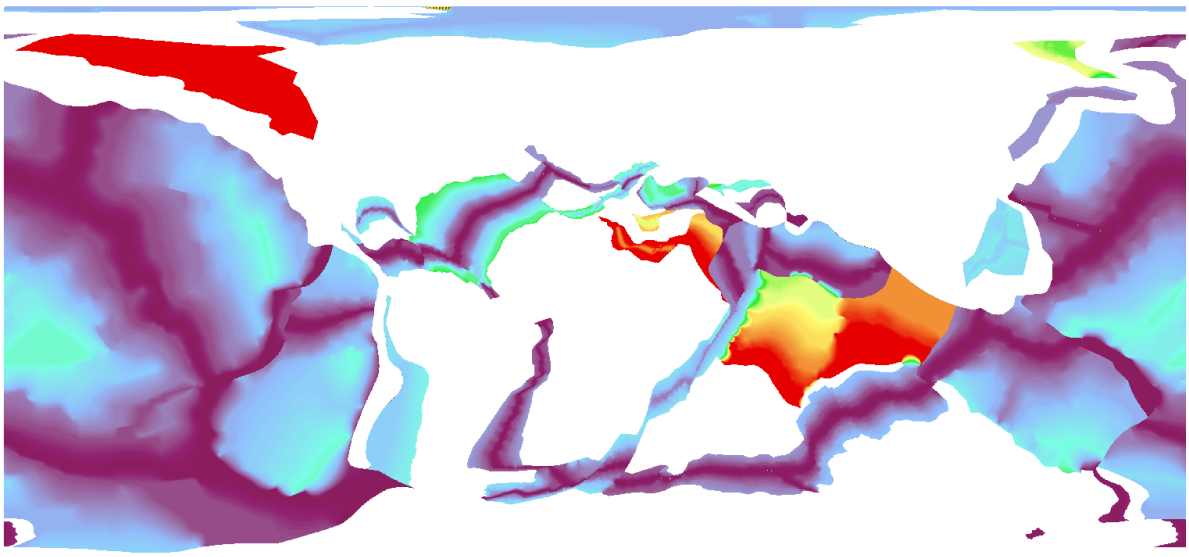
b. *reconstruction_020*



c. *reconstruction_068*



d. *reconstruction_100*



e. *reconstruction_140*

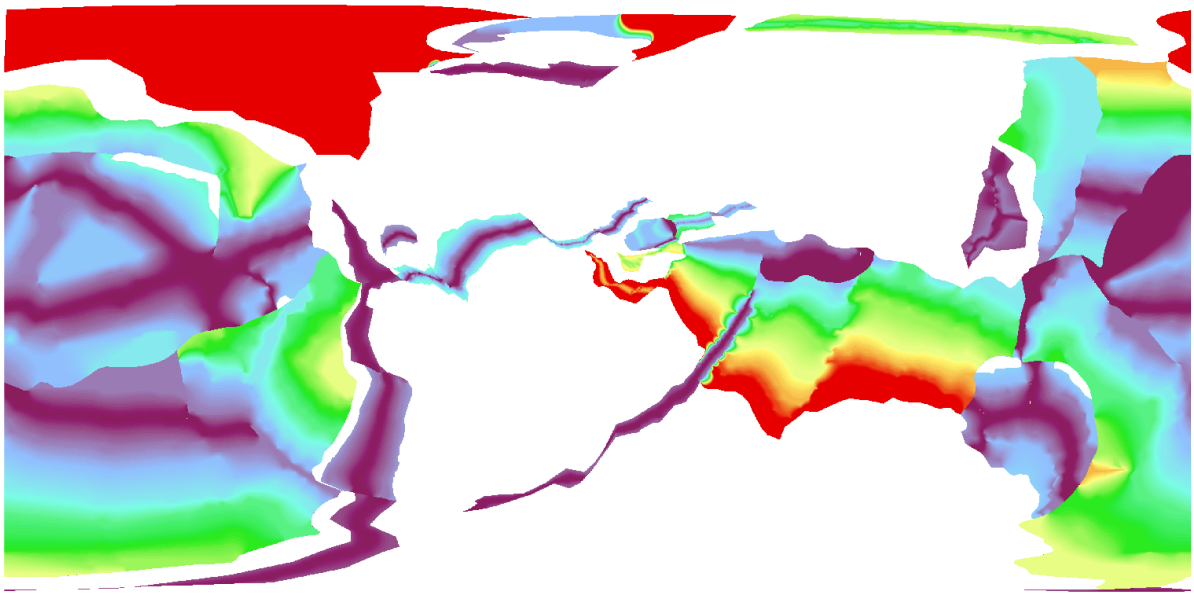


Figure 41. Reconstruction de l'âge de la croûte océanique pour les périodes : 000 (a), 020 (b), 068 (c), 100 (d), 140 (e).

DISCUSSION

Cette partie du rapport se concentre sur les problèmes rencontrés, les erreurs qui peuvent potentiellement apparaître lorsque le code est en cours d'exécution, et des suggestions d'amélioration.

PROBLÈMES RENCONTRÉS

L'écriture de ce script ne s'est pas faite de manière simple et direct. J'ai rencontré beaucoup d'erreurs Python qui ont facilement été résolues en lisant simplement le message d'erreur. D'autres problèmes ont nécessité que je cherche un peu plus sur internet, notamment vers la communauté Esri, pour trouver de potentielles solutions.

Par exemple, un problème facilement résolu était le manque de points pour l'interpolation de certaines plaques (comme c'est le cas de la plaque Panama de la période 000, figure 42). Comme mentionné plus haut (partie 3, *Interpolation des âges de la croûte*), ajouter un bloc *try – except* a permis d'éviter que le problème empêche l'outil de tourner. Plus tard lors de l'avancement du script, j'ai ajouté des lignes de code permettant de créer des points artificiels le long des plaques tectoniques, ce qui a eu pour effet d'annuler l'erreur du manque de points.

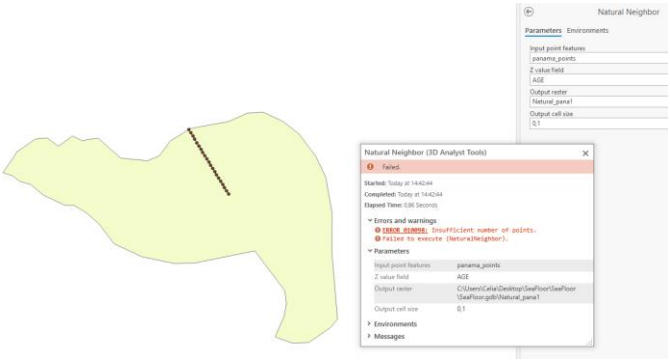


Figure 42 : Panama n'a pas assez de point pour faire une interpolation.

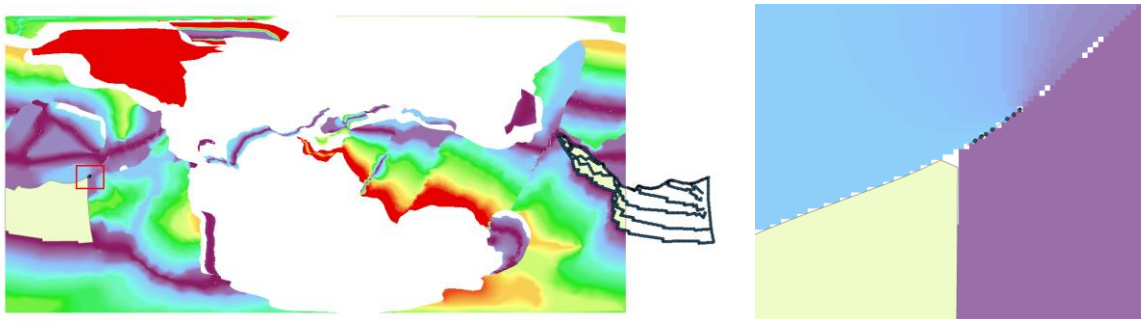
Celui qui m'a sûrement pris le plus de temps était l'erreur de l'outil `MosaicToNewRaster()`. Il manquait au raster de sortie de nombreux pixels (figure 43). Après avoir tester plusieurs solutions proposées sur le net (créer un nouveau dossier contenant les rasters des plaques, créer une nouvelle géodatabase, recaclculer les pyramides,

projeter le raster, etc.), j'ai finalement réalisé que j'avais mal paramétré l'outil : si le *mozaic_method* est MAXIMUM ou MINIMUM il y aura des trous, mais s'il est LAST ou FIRST le problème est réglé.

Figure 43 : Exemple de reconstruction produite avec les mauvais paramètres du *MosaicToNewRaster()*. Même en zoomant sur le raster, les pixels n'apparaissaient pas.

Certains des shapefiles *Plates* et *Worldmaps* contiennent quelques erreurs et artefacts dans leur données, et par conséquent certaines des reconstructions comportent des erreurs. Par exemple, la reconstruction d'il y a 154 Ma a un morceau de plaque manquant (Chatham) car le polygone plaque de celle-ci a un défaut. Ainsi lors du déplacement des points, certains restent à l'ouest de la carte car non-sélectionnés par l'outil, et l'interpolation se fait donc mal (figure 44). Aussi, plusieurs reconstructions ont des artefacts dans les âges de la croûte à cause notamment de doublons d'isochrones (figure 45). Ces défauts dans la reconstruction ont néanmoins l'avantage d'aider à repérer plus facilement les erreurs dans les données et de pouvoir les corriger par la suite.

a.



c.

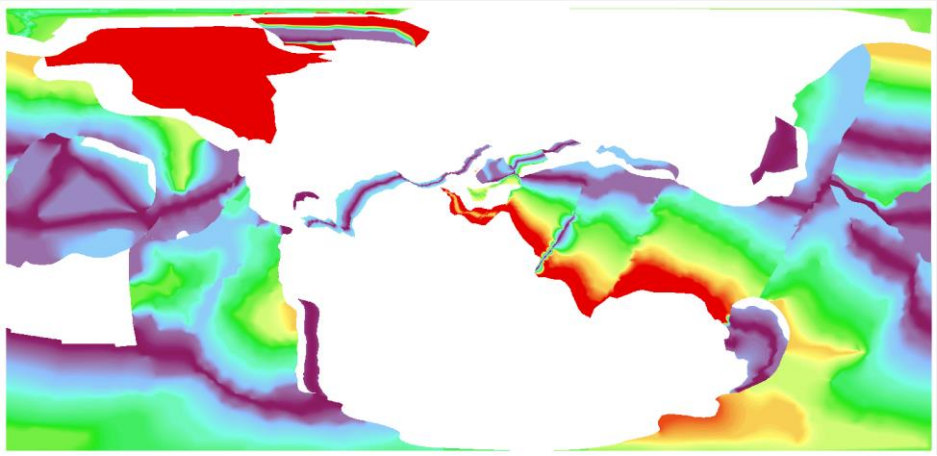


Figure 44 : a. Zoom sur le polygone *Chatham* de l'époque 154 Ma. On voit que le polygone n'est pas bien défini et que des points dépasse de la plaque. b. *Reconstruction_154*, la partie ouest de la plaque *Chatham* est inexistante.

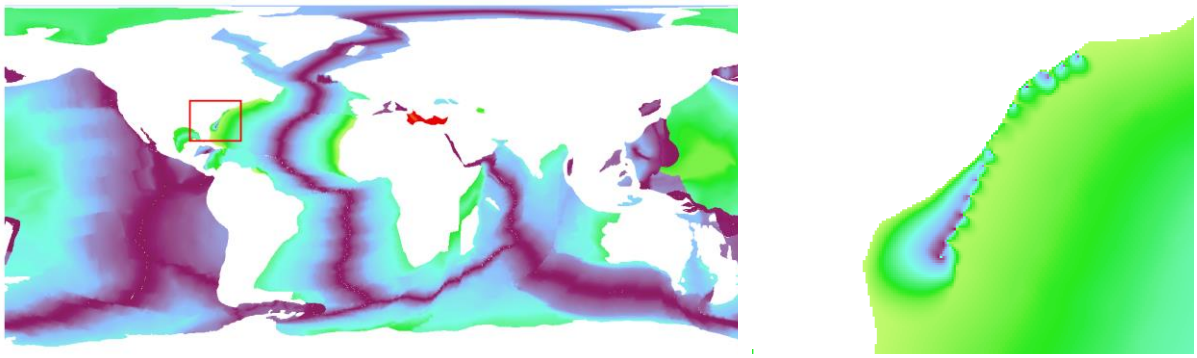


Figure 45 : Zoom sur la côte est des Etats-Unis de la reconstruction 000. Un doublon d'isochrone est responsable pour cet artefact dans l'âge de la croûte.

ERREURS POTENTIELLES ET SOLUTIONS

Il peut arriver que l'erreur 010240 survienne lors du lancement de l'outil *SeaFloorAge*. Malgré la ligne de code `arcpy.overwriteOutput = True`, il arrive qu'Arcpy n'arrive pas à écraser un shapefile ou un raster de la géodatabase temporaire pour le remplacer par une couche du même nom. Une solution à cette erreur est de supprimer les couches de la géodatabase, ou la géodatabase elle-même, sauvegarder et fermer ArcGIS Pro. Puis de relancer l'outil.

Il arrive que l'erreur persiste cependant. Dans ce cas, il se peut qu'il y ait un problème avec les shapefiles en entrée comme un caractère spécial dans le nom d'une plaque, ce qui empêche la sauvegarde du fichier (figure 46).

Un autre problème vient de la mise en veille de mon ordinateur (Windows 10 et 11) : malgré le fait que l'outil soit lancé, il se met en pause si l'ordinateur se met en veille, et reprend lorsque l'ordinateur est à nouveau utilisé. La seule solution que j'ai trouvée est de mettre la mise en veille sur « jamais » dans les paramètres de l'ordinateur pendant que l'outil est en cours de travail.

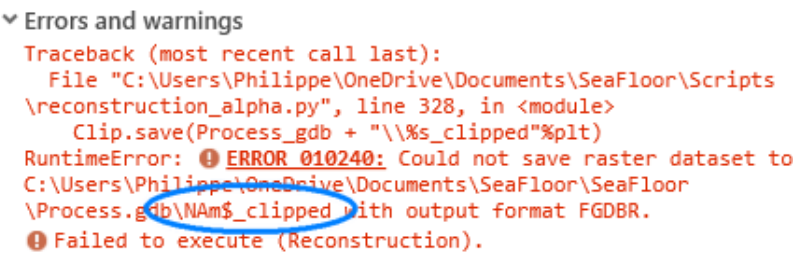


Figure 46 : Exemple de l'erreur 010240 due à un caractère spécial dans le nom d'une plaque.

AMÉLIORATIONS

Cet outil n'est évidemment pas parfait, et certaines modifications pourraient être à envisager afin de l'améliorer.

Comme précédemment mentionné, les points disparaissant dans les zones de subduction laissent des trous dans l'interpolation de l'âge de la croûte (figure 47.3).

Le problème de la version alpha (voir la section *Remplissage des zones subductées*) est que les points artificiellement créés ne correspondent pas à l'âge « véritable » de la croûte là où ils se situent, car ils ne font que prendre l'âge de l'isochrone la plus proche même si celle-ci est très éloignée (figure 47.5).

La version bêta, pensée par Christian Vérard, récupère les points entrés en subduction de la reconstruction antérieure afin d'étendre l'interpolation au-delà des bordures de la plaque. Cela nécessiterait sûrement la création d'un autre outil cependant. Pour un âge donné, l'outil irait chercher les points de la période antérieure (figure 47.6). À l'aide d'un fichier Excel recensant les informations nécessaires pour tourner toutes les plaques tectoniques d'une reconstruction à une autre, il déplacerait les points selon la rotation de la plaque afin de déterminer leur emplacement à la période d'intérêt (figure 47.7). Seuls les points situés à l'extérieur de la plaque (i.e. entrés en subduction) seraient gardés et stockés dans une géodatabase (figure 47.8). Puis, l'outil *SeaFloorAge* irait chercher ces points additionnels pour les ajouter à la couche *points*, ferait l'interpolation et couperait les pixels qui déborderaient de la plaque (figure 47.9-10-11).

Cette solution est plus proche de la réalité mais nécessite les outils supplémentaires créés par Christian Vérard et qui permettent d'appliquer la rotation exacte des points. Mais ces outils (au format VB.Net) n'ont pas encore été implantés sur ArcGIS Pro et doivent d'abord être convertis au langage Python.

Une option que j'aurais aimé pouvoir ajouter au code est l'envoi d'emails. Comme l'exécution du script prend beaucoup de temps, recevoir un courriel informant que les reconstructions sont terminées pourrait être très utile. En théorie, il suffirait d'ajouter quelques lignes à la fin du code et d'en faire un paramètre optionnel dans les propriétés de l'outil sur ArcGIS Pro. En pratique, c'est un peu plus compliqué à cause notamment des pare-feux et je n'y suis malheureusement pas parvenue malgré plusieurs tentatives.

Même lorsqu'une carte a correctement été reconstruite par l'outil, on remarque que certains endroits de l'interpolation ne sont pas lisses ou continus, comme on devrait s'y attendre. Ces petites imperfections se situent notamment sur les failles transformantes proches des rides médio-océaniques, i.e. aux discontinuités. De plus, les isochrones se chevauchent à ces endroits, ce qui rend probablement l'interpolation plus confuse (figure 48). Peut-être qu'augmenter leur nombre ou ajouter une étape de lissage régleraient ce « problème ».

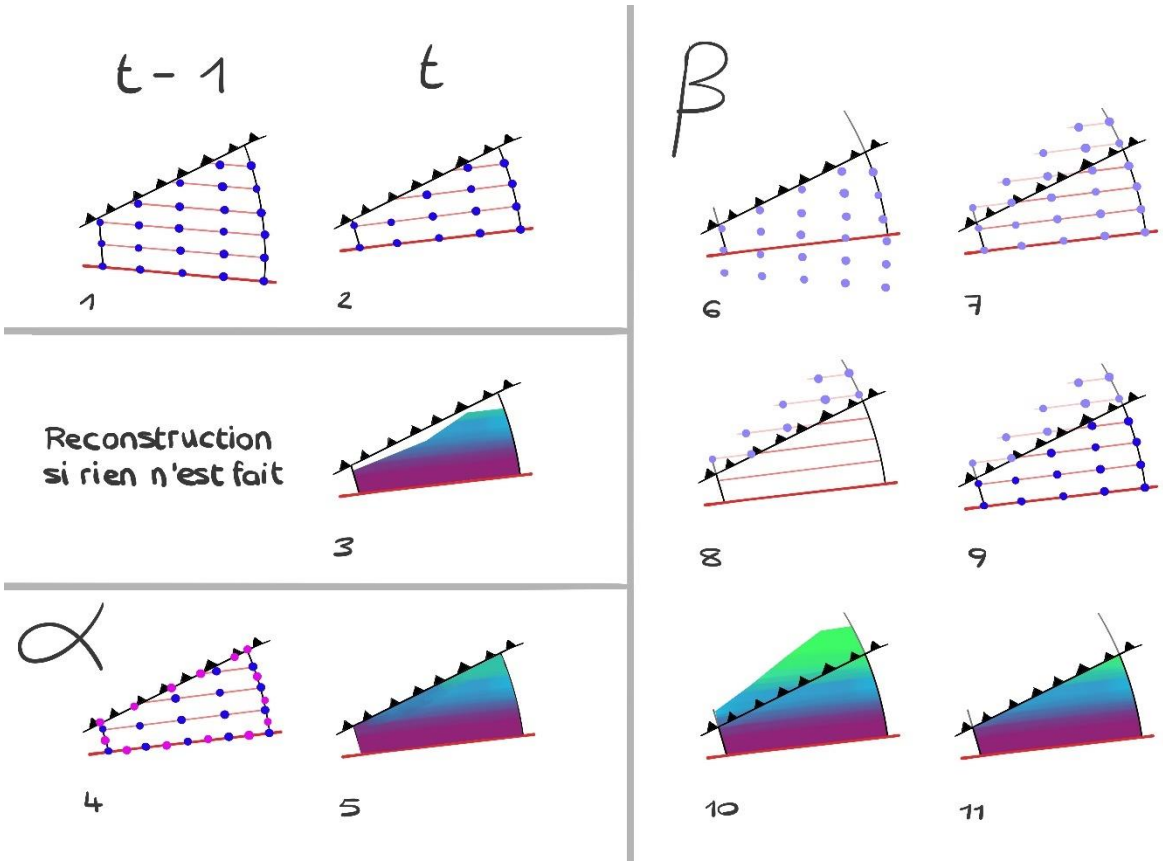


Figure 47 : 1. Plaque au temps t-1. 2. Plaque au temps t. 3. Interpolation de l'âge sans ajout de points. 4. Ajout de points artificiels (en rose) aux bordures de la plaque. 5. Interpolation de la version alpha. 6. Plaque au temps t et points au temps t-1. 7. Les points au temps t-1 sont déplacés (rotation + translation) afin de se superposés aux points au temps t. 8. Seuls les points en dehors de la plaque sont gardés. 9. Les points en dehors de la plaque sont ajouté à la couche des points au temps t. 10. Interpolation de la version bêta. 11. Les pixels en dehors de la plaque sont supprimés. Légende : zone de subduction (trait avec triangles), ride médio-océanique (trait épais rouge), lignes d'isochrones (traits rouges), points des âges (points bleus), points artificiels (points roses), points des âges au temps t-1 pour la version bêta (points bleu clair), interpolation (violet : plus jeune ; vert : plus vieux).

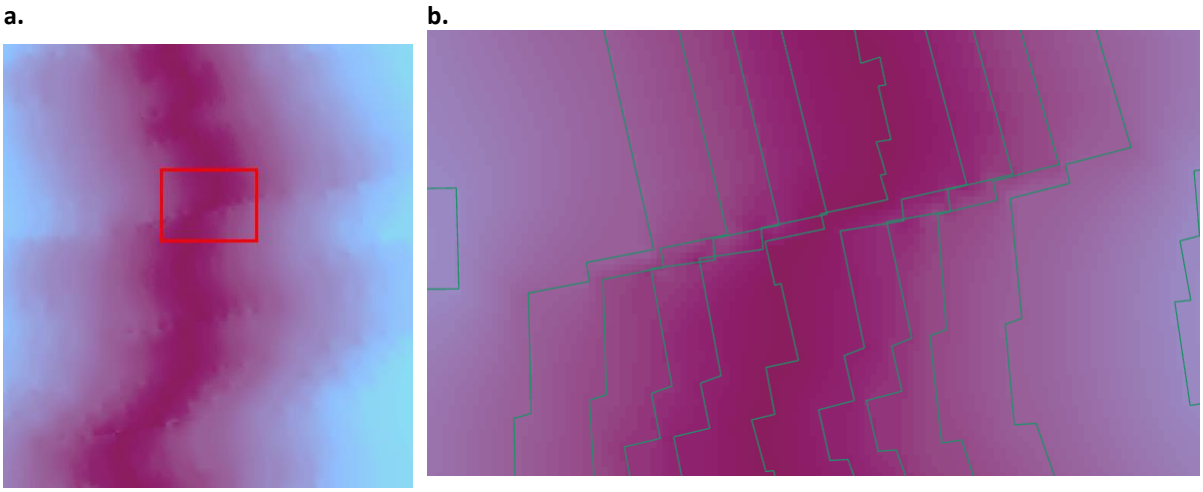


Figure 48 : a. Zoom de la *reconstruction_000* sur la ride médio-océanique entre l'Afrique et l'Amérique latine. L'interpolation n'est pas très « lisse ». b. Zoom sur le rectangle rouge avec la couche *world_isochrons* ajoutée, les isochrones se chevauchent et peuvent être en parti responsables de ces imperfections dans la reconstruction.

CONCLUSION

Comme le montrent ces belles cartes de l'âge des fonds marins, l'outil *SeaFloorAge* fonctionne et est facilement utilisable sur ArcGIS Pro. Malgré quelques imperfections et inexactitudes dans les reconstructions produites, celles-ci transmettent efficacement l'information générée par le modèle Panalesis de Christian Vérard (2019a). L'outil mériterait néanmoins encore quelques améliorations, notamment l'application de la solution bêta, afin de pouvoir produire des cartes plus fidèles et plus proches de la réalité. Néanmoins, *SeaFloorAge* est capable de produire de nombreuses reconstructions nécessitant plusieurs étapes et permet ainsi de gagner une quantité de temps non-négligeable qui pourra être mieux investie dans la recherche académique.

La reconstruction de telles cartes porte avant tout un intérêt scientifique : l'âge de la croûte étant directement lié à la théorie de la divergence des plaques, l'étudier permet de mieux comprendre l'évolution de la Terre et la tectonique des plaques. De plus, ces reconstructions permettent d'observer les résultats que produit le modèle Panalesis et de déceler d'éventuelles erreurs dans les données en entrée (duplicatas d'isochrones, fautes de frappe, etc.).

Enfin, d'un point de vue plus personnel, un tel travail a été un excellent exercice pour me familiariser avec ArcGIS Pro et ArcPy, découvrir de nouveaux outils, apprendre des compétences en programmation, à travailler sur la résolution de problèmes et à réfléchir à la stratégie à adopter avant d'écrire un script pour obtenir le résultat désiré.

ANNEXES

```
# -*- coding: utf-8 -*-

"""
This code produces reconstructions of the sea floor'age for the selected periods of time.
Written by Célia Barat
"""

### Preparation ================================================================================
# Importing system module
import arcpy, re
from arcpy.sa import *

# Environment parameters
arcpy.overwriteOutput = True

#Input parameters
path_to_GDBs = arcpy.GetParameterAsText(0)
Plates_folder = arcpy.GetParameterAsText(1)
WorldMaps_folder = arcpy.GetParameterAsText(2)
COB_psAbs_folder = arcpy.GetParameterAsText(3)
Ages_List = arcpy.GetParameterAsText(4)

# Workspaces
# Create databases if not there yet
if not arcpy.Exists(path_to_GDBs +"\\Maps.gdb"):
    arcpy.management.CreateFileGDB(path_to_GDBs, "Maps")

if not arcpy.Exists(path_to_GDBs +"\\Process.gdb"):
    arcpy.management.CreateFileGDB(path_to_GDBs, "Process")

Maps_gdb = path_to_GDBs +"\\Maps.gdb"
Process_gdb = path_to_GDBs + "\\Process.gdb"

# List of ages ---------------------------------------------------------------------------------
if Ages_List == "*":
    Ages_List = []
    filename_List = []
    walk = arcpy.da.Walk(Plates_folder, datatype="FeatureClass")
    for dirpath, dirnames, filenames in walk:
        for filename in filenames:
            filename_List.append(filename)
    for filename in filename_List:
```

```
        age = re.findall('[0-9]+', filename)
        Ages_List.append(age[0])

elif Ages_List.find("-") >= 0:
    interval = Ages_List.split("-")
    nums = range(int(interval[0]), (int(interval[1])+1), 1)
    Ages_List = []
    for num in nums:
        if num < 10:
            num = "00" + str(num)
        elif num < 100:
            num = "0" + str(num)
        else:
            num = str(num)
        Ages_List.append(num)

elif Ages_List.find(",") >= 0:
    Ages_List = Ages_List.split(",")

elif len(Ages_List) == 3 and len(re.findall(r'\b\d{3}\b', Ages_List)) == 1: # if just one age
    age = Ages_List
    Ages_List = []
    Ages_List.append(age)

else:
    arcpy.AddMessage("List of ages incorrect. The age needs to be written with 3 digits.
Select:\n 1) one age\n 2) specific ages separating them with ','\n 3) a range of ages with '-
'\n 4) everything with '*'.")
    exit()

arcpy.AddMessage("List of ages (" + str(len(Ages_List)) + ") :")
arcpy.AddMessage(Ages_List)

# Create PoleNord et PoleSud feature points layers ----------------------------------------
if not arcpy.Exists(Process_gdb + "\\PoleNordBuffer") or not arcpy.Exists(Process_gdb +
"\\PoleSud"):
    nmList = ["PoleNord", "PoleSud"]
    ptList = [((0,90,0000000),) , ((0,-90),) ]
    for nm,pt in zip(nmList, ptList):
        nm = arcpy.management.CreateFeatureclass(out_path=Process_gdb, out_name=nm,
geometry_type="POINT", template=[], has_m="DISABLED", has_z="DISABLED")
        cursor = arcpy.da.InsertCursor(nm, ["SHAPE@XY"])
        cursor.insertRow(pt)
        del cursor

    # Because PoleNord doesn't intersect with north plate
    PoleNordBuffer = Process_gdb + "\\PoleNordBuffer"
    arcpy.analysis.Buffer(in_features=Process_gdb + "\\PoleNord",
out_feature_class=PoleNordBuffer, buffer_distance_or_field="1 DecimalDegrees",
line_side="FULL", line_end_type="ROUND", dissolve_option="NONE", dissolve_field=[],
method="PLANAR")

# Create international Datelines feature line layers ----------------------------------------
if not arcpy.Exists(Process_gdb + "\\DatelineW") or not arcpy.Exists(Process_gdb +
"\\DatelineE"):
    nmList = ["DatelineW", "DatelineE"]
    ptList = [ ((-180, 90),(-180, -90)) , ((180, 90),(180, -90)) ]
    for nm,pt in zip(nmList, ptList):
        fc = arcpy.management.CreateFeatureclass(out_path=Process_gdb, out_name=nm,
geometry_type="POLYLINE", template=[], has_m="DISABLED", has_z="DISABLED")
        cursor = arcpy.da.InsertCursor(fc, ["SHAPE@"])
        array = arcpy.Array([arcpy.Point(pt[0][0],pt[0][1]),
                             arcpy.Point(pt[1][0],pt[1][1])])
        polyline = arcpy.Polyline(array)
        cursor.insertRow([polyline])
        del cursor

# Create a polygon zone close to DatelineW ----------------------------------------------------
if not arcpy.Exists(Process_gdb + "\\Dateline_zone"):
    fc = arcpy.management.CreateFeatureclass(out_path=Process_gdb, out_name="Dateline_zone",
geometry_type="POLYGON", template=[], has_m="DISABLED", has_z="DISABLED")
    cursor = arcpy.da.InsertCursor(fc, ["SHAPE@"])
    array = arcpy.Array([arcpy.Point(-180, 90),
                         arcpy.Point(-175, 90),
                         arcpy.Point(-175, -90),
                         arcpy.Point(-180, -90)])
    polyline = arcpy.Polygon(array)
```

```
    cursor.insertRow([polyline])
    del cursor

PoleNord = Process_gdb + "\\PoleNordBuffer"
PoleSud = Process_gdb + "\\PoleSud"
DatelineW = Process_gdb + "\\DatelineW"
DatelineE = Process_gdb + "\\DatelineE"
Dateline_zone = Process_gdb + "\\Dateline_zone"
arcpy.AddMessage("Points, lines and polygon added")

# In case some ages weren't possible to reconstruct: ----------------------------------------
Not_done = []

#### ================== Starting the reconstructions ========================================
### Starting the loop for each age in the list: ------------------------------------------------
for age in Ages_List:
    # Go to the shapefiles
    PlatesMap = Plates_folder + "\\Plate_%s"%age + ".shp"
    TimeMap = WorldMaps_folder + "\\R_psAbs_%s"%age + ".shp"
    Continents = COB_psAbs_folder + "\\COBgon_%s"%age + ".shp"

    # Start the process, if age exists in the data:
    if arcpy.Exists(PlatesMap):
        arcpy.AddMessage("========== Reconstructing age %s =========="%age)

        # Make a list of the plates names
        Plates = []
        field = ["PlateName"]
        with arcpy.da.SearchCursor(PlatesMap, field) as cursor:
            for i in cursor:
                Plates.append(i[0])

        # Remove "GAP" plates from the list
        if "GAP" in Plates:
            Plates.remove("GAP")

        # Prepare a list for rasters clipped and plates without oceanic crust
        RC_List = []
        No_crust = []

        # Select isochrons, passive margins and ridges, and make a layer
        SQL = "AGE < 999 AND (TYPE = 'Isochron' OR TYPE = 'Passive_Margin' OR TYPE = 'Ridge')"
        s_isochrons = arcpy.management.SelectLayerByAttribute(TimeMap, 'NEW_SELECTION', SQL)
        world_isochrons = arcpy.management.CopyFeatures(s_isochrons, Process_gdb +
"\\world_isochrons")
        arcpy.management.SelectLayerByAttribute(TimeMap, 'CLEAR_SELECTION')

        ### Starting the loop for each plate ================================================
        for plt in Plates:
            ## POLYGON ---------------------------------------------------------------------
            # Select the plate and make a layer from the selection
            SQL = "PlateName = '" + str(plt) + "'"
            selection = arcpy.management.SelectLayerByAttribute(PlatesMap, 'NEW_SELECTION',
SQL)
            plate = arcpy.management.CopyFeatures(selection, Process_gdb + "\\plate")
            arcpy.management.SelectLayerByAttribute(PlatesMap, 'CLEAR_SELECTION')
            arcpy.AddMessage("Plate : " + str(plt) + " ----------")

            # Make a slight buffer around the Plate
            plate_buffer = Process_gdb + "\\plate_buffer"
            arcpy.analysis.Buffer(in_features=Process_gdb + "\\plate",
out_feature_class=plate_buffer, buffer_distance_or_field="0,05 DecimalDegrees",
line_side="FULL", line_end_type="ROUND", dissolve_option="NONE", dissolve_field=[],
method="PLANAR")

            ## LINES -----------------------------------------------------------------------
            # Select world-isochrons inside the plate, and make a layer
            selection = arcpy.management.SelectLayerByLocation(world_isochrons,
'HAVE_THEIR_CENTER_IN', plate_buffer, "", "NEW_SELECTION", "NOT_INVERT")
            # Skip this plate if no oceanic crust
            if int(arcpy.GetCount_management(selection)[0]) == 0:
                arcpy.AddMessage("The plate %s has no oceanic crust --> skip"%plt)
                No_crust.append(plt)
                continue
            flines = arcpy.management.CopyFeatures(selection, Process_gdb + "\\flines")
            arcpy.management.SelectLayerByAttribute(world_isochrons, 'CLEAR_SELECTION')
            #arcpy.AddMessage("Lines made")
```

```
            # Cut lines that go beyond the plate
            lines = arcpy.analysis.Clip(in_features=flines, clip_features=plate_buffer,
out_feature_class=Process_gdb+"\\lines", cluster_tolerance="")

            # Densify vertices
            lines = arcpy.edit.Densify(lines, "DISTANCE", distance="1 DecimalDegrees",
max_deviation="0,1 DecimalDegrees", max_angle=10, max_vertex_per_segment=None)
            #arcpy.AddMessage("Vertices densified")

            ## POINTS ------------------------------------------------------------------------
            # Feature vertices to points
            points = arcpy.management.FeatureVerticesToPoints(lines, Process_gdb + "\\points",
point_location="ALL")
            #arcpy.AddMessage("Points made")

            ## ALPHA VERSION: add the plate's boundary points (in case there are gaps in the
interpolation result): ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
            # Make a copy of the plate polygon
            plate_polygon = arcpy.management.CopyFeatures(plate, Process_gdb +
"\\plate_polygon")

            # Densify polygon's vertices
            plate_polygon = arcpy.edit.Densify(plate_polygon, "DISTANCE", distance="1
DecimalDegrees", max_deviation="0,1 DecimalDegrees", max_angle=10,
max_vertex_per_segment=None) # 0,1

            # Feature vertices to points
            points_polygon = arcpy.management.FeatureVerticesToPoints(plate_polygon,
Process_gdb + "\\points_polygon", point_location="ALL")

            # Remove unwanted AGE field:
            arcpy.DeleteField_management(points_polygon, ["AGE"])

            # Remove points that might be on the international Datelines
            pointsW = arcpy.management.SelectLayerByLocation(points_polygon,
"WITHIN_A_DISTANCE", DatelineW, "15 Meters", "NEW_SELECTION", "NOT_INVERT")
            if int(arcpy.GetCount_management(pointsW)[0]) > 0:
                arcpy.DeleteRows_management(pointsW)
            arcpy.management.SelectLayerByAttribute(pointsW, 'CLEAR_SELECTION')
            pointsE = arcpy.management.SelectLayerByLocation(points_polygon,
"WITHIN_A_DISTANCE", DatelineE, "15 Meters", "NEW_SELECTION", "NOT_INVERT")
            if int(arcpy.GetCount_management(pointsE)[0]) > 0:
                arcpy.DeleteRows_management(pointsE)
            arcpy.management.SelectLayerByAttribute(pointsE, 'CLEAR_SELECTION')

            # Attribute these new points an age (based on the nearest point with an age)
            points_polygon = arcpy.analysis.Near(in_features=points_polygon,
near_features=lines, search_radius="", location="NO_LOCATION", angle="NO_ANGLE",
method="GEODESIC", field_names=[["NEAR_FID", "NEAR_FID"], ["NEAR_DIST", "NEAR_DIST"]])[0]

            # Join Age column
            points_polygon = arcpy.management.JoinField(in_data=points_polygon,
in_field="NEAR_FID", join_table=lines, join_field="OBJECTID", fields=["AGE"])[0]

            # Merge the 2 points layer
            points = POINTS = arcpy.management.Append(inputs=points_polygon, target=points,
schema_type="NO_TEST", field_mapping="ID \"ID\" true true false 8 Double 0
0,First,#,TROU\\points_polygon,Id,-1,-1;TYPE \"TYPE\" true true false 20 Text 0
0,First,#;LIMIT \"LIMIT\" true true false 3 Text 0 0,First,#;COB_LIMIT \"COB_LIMIT\" true true
false 3 Text 0 0,First,#;AGE \"AGE\" true true false 8 Double 0
0,Max,#,TROU\\points_polygon,AGE,-1,-1;NAME_TERR \"NAME_TERR\" true true false 50 Text 0
0,First,#;POSITION \"POSITION\" true true false 20 Text 0 0,First,#;REF_PLATE \"REF_PLATE\"
true true false 2 Text 0 0,First,#;IDTERRANE \"IDTERRANE\" true true false 4 Long 0
0,First,#;APPEARANCE \"APPEARANCE\" true true false 8 Double 0 0,First,#;DISAPPEARA
\"DISAPPEARA\" true true false 8 Double 0 0,First,#;PLATE \"PLATE\" true true false 20 Text 0
0,First,#;ORIG_FID \"ORIG_FID\" true true false 4 Long 0
0,First,#,TROU\\points_polygon,ORIG_FID,-1,-1", subtype="", expression="")[0]

            ## Determine the plate's type ----------------------------------------------------
            # Split plate by international Dateline
            split = arcpy.management.FeatureToPolygon([plate, DatelineW], Process_gdb +
"\\split" , "", "ATTRIBUTES", "")

            # Clip with original plate to remove extra polygons
            plateNew = arcpy.analysis.Clip(in_features=split, clip_features=plate,
out_feature_class=Process_gdb+"\\plateNew", cluster_tolerance="")
```

```
            # Deleting artefact rows
            SQL = "Shape_Area < 0.001"
            s_arte = arcpy.management.SelectLayerByAttribute(plateNew, 'NEW_SELECTION', SQL)
            if int(arcpy.GetCount_management(s_arte)[0]) > 0:
                arcpy.DeleteRows_management(s_arte)
            arcpy.management.SelectLayerByAttribute(s_arte, 'CLEAR_SELECTION')

            # What kind of plate do we have?
            num_rows = int(arcpy.GetCount_management(plateNew)[0])
            arcpy.AddMessage("Number of plate(s): " + str(num_rows))
            if num_rows > 1: # "if more than 1 row, then plate is divided"

                # Type ...
                PN = arcpy.management.SelectLayerByLocation(plateNew, 'INTERSECT', PoleNord,
"", "NEW_SELECTION", "NOT_INVERT")
                NPN = int(arcpy.GetCount_management(PN)[0])
                arcpy.management.SelectLayerByAttribute(plate, 'CLEAR_SELECTION')

                PS = arcpy.management.SelectLayerByLocation(plateNew, 'INTERSECT', PoleSud,
"", "NEW_SELECTION", "NOT_INVERT")
                NPS = int(arcpy.GetCount_management(PS)[0])
                arcpy.management.SelectLayerByAttribute(plate, 'CLEAR_SELECTION')

                if (NPN > 0 or NPS > 0): # "if part of the plate is at the poles..."
                    TYPE = "Pole & Divided"
                else:
                    TYPE = "Divided"
            else:
                TYPE = "Normal"
            arcpy.AddMessage("Plate's type is %s"%TYPE)

            ## Move points if needed -------------------------------------------------------------
            if TYPE == "Divided":
                s_dateline = arcpy.management.SelectLayerByLocation(plateNew, 'INTERSECT',
DatelineW, "", "NEW_SELECTION", "NOT_INVERT")
                s_points = arcpy.management.SelectLayerByLocation(points, "WITHIN_A_DISTANCE",
s_dateline, "50 Kilometers", "SUBSET_SELECTION", "NOT_INVERT")
                arcpy.management.SelectLayerByAttribute(s_dateline, 'CLEAR_SELECTION')
                xOffset = 360 # move towards east
                yOffset = 0
                # Move points
                with arcpy.da.UpdateCursor(s_points, ["SHAPE@XY"]) as cursor:
                    for row in cursor:
                        cursor.updateRow([[row[0][0] + xOffset, row[0][1] + yOffset]])
                arcpy.management.SelectLayerByAttribute(s_points, 'CLEAR_SELECTION')

            elif TYPE == "Pole & Divided":
                if NPN > 0: # if at North pole
                    s_not_Npole = arcpy.management.SelectLayerByLocation(plateNew,
'INTERSECT', PoleNord, "", "NEW_SELECTION", "INVERT")
                    s_micro_plate = arcpy.management.SelectLayerByLocation(s_not_Npole,
'INTERSECT', Dateline_zone, "", "SUBSET_SELECTION", "NOT_INVERT")
                    if int(arcpy.GetCount_management(s_micro_plate)[0]) > 0: # if microplate
is at the west of the map
                        xOffset = 360 # move towards east
                    else:
                        xOffset = -360 # move towards west
                    s_not_Npole = arcpy.management.SelectLayerByLocation(plateNew,
'INTERSECT', PoleNord, "", "NEW_SELECTION", "INVERT")
                    s_points = arcpy.management.SelectLayerByLocation(points,
"WITHIN_A_DISTANCE", s_not_Npole, "50 Kilometers", "SUBSET_SELECTION", "NOT_INVERT")
                    arcpy.AddMessage(int(arcpy.GetCount_management(s_points)[0]))

                    arcpy.management.SelectLayerByAttribute(s_not_Npole, 'CLEAR_SELECTION')
                    arcpy.management.SelectLayerByAttribute(s_micro_plate, 'CLEAR_SELECTION')

                elif NPS > 0: # if at South pole
                    s_not_Spole = arcpy.management.SelectLayerByLocation(plateNew,
'INTERSECT', PoleSud, "", "NEW_SELECTION", "INVERT")
                    s_micro_plate = arcpy.management.SelectLayerByLocation(s_not_Spole,
'INTERSECT', Dateline_zone, "", "SUBSET_SELECTION", "NOT_INVERT")
                    if int(arcpy.GetCount_management(s_micro_plate)[0]) > 0: # if microplate
is at the west of the map
                        xOffset = 360 # move towards east
                    else:
                        xOffset = -360 # move towards west
```

```
                    s_not_Spole = arcpy.management.SelectLayerByLocation(plateNew,
'INTERSECT', PoleSud, "", "NEW_SELECTION", "INVERT")
                    s_points = arcpy.management.SelectLayerByLocation(points,
"WITHIN_A_DISTANCE", s_not_Spole, "50 Kilometers", "SUBSET_SELECTION", "NOT_INVERT")
                    arcpy.AddMessage(int(arcpy.GetCount_management(s_points)[0]))

                    arcpy.management.SelectLayerByAttribute(s_not_Spole, 'CLEAR_SELECTION')
                    arcpy.management.SelectLayerByAttribute(s_micro_plate, 'CLEAR_SELECTION')

                # Move points
                yOffset = 0
                with arcpy.da.UpdateCursor(s_points, ["SHAPE@XY"]) as cursor:
                    for row in cursor:
                        cursor.updateRow([[row[0][0] + xOffset, row[0][1] + yOffset]])
                arcpy.AddMessage("Points moved")
                arcpy.management.SelectLayerByAttribute(s_points, 'CLEAR_SELECTION')

            else: # TYPE == "Normal":
                arcpy.AddMessage("Points unchanged")

            # Interpolation ------------------------------------------------------------------
            try: # because not working for plates without oceanic crust
                arcpy.ddd.NaturalNeighbor(points, "AGE", Process_gdb + "\\interpolation",
"0,1")
                interpolation = arcpy.Raster(Process_gdb + "\\interpolation")
                arcpy.AddMessage("Interpolation")
            except Exception as e:
                arcpy.AddMessage("No interpolation done for %s :"%plt)
                arcpy.AddMessage("ERROR: "+str(e))
                continue

            # Remove what overflows from the plate's boundaries ------------------------------
            Clip = arcpy.sa.ExtractByMask(in_raster=interpolation, in_mask_data=plate_buffer)
            Clip.save(Process_gdb + "\\%s_clipped"%plt)
            arcpy.AddMessage("Clipped")

            # Get a list of all the rasters of the reconstruction
            RC_List.append(Clip)

            # --- End of the plates loop ---

        ### Finalizing the reconstruction ================================================
        arcpy.AddMessage(RC_List)
        # Merge the rasters
        mozaic = arcpy.management.MosaicToNewRaster(input_rasters=RC_List,
output_location=Process_gdb, raster_dataset_name_with_extension="mozaic",
coordinate_system_for_the_raster="", pixel_type="32_BIT_UNSIGNED", cellsize=None,
number_of_bands=1, mosaic_method="LAST", mosaic_colormap_mode="FIRST")[0]
        mozaic = arcpy.Raster(mozaic)
        arcpy.AddMessage("Mozaic")

        # Remove continents
        Continents_buffer = Process_gdb + "\\Continents_buffer"
        arcpy.analysis.Buffer(in_features=Continents, out_feature_class=Continents_buffer,
buffer_distance_or_field="0,05 DecimalDegrees", line_side="FULL", line_end_type="ROUND",
dissolve_option="NONE", dissolve_field=[], method="PLANAR")

        # Make a raster of the continents only
        with arcpy.EnvManager(extent="MAXOF"):
        inverse = arcpy.sa.ExtractByMask(in_raster=mozaic, in_mask_data= Continents_buffer)
        inverse.save(Process_gdb + "\\inverse")
        #arcpy.AddMessage("Inverse")

        # Reconstruction
        min = mozaic.minimum
        expression = "(SetNull(~(IsNull( inverse)), mozaic)) - %s"%min
        reconstruction = RasterCalculator([inverse, mozaic], ["inverse", "mozaic"],
expression, "FirstOf")
        reconstruction.save(Maps_gdb + "\\reconstruction_%s"%age)

        arcpy.AddMessage("Continental crust removed")

        ### Change symbology ============================================================
        # Reference the project
        aprx = arcpy.mp.ArcGISProject("CURRENT")

        # Set the default geodatabase
```

```
        aprx.defaultGeodatabase = Maps_gdb

        # Reference items in the project
        mp = aprx.listMaps("Map")[0]

        # Add the reconstruction to the map (drag and drop)
        mp.addDataFromPath(Maps_gdb + "\\reconstruction_%s"%age)
        arcpy.AddMessage("Raster added to the map")

        # Select the raster
        lyr = mp.listLayers()[0]

        ## Change symbology
        sym = lyr.symbology
        if hasattr(sym, "colorizer"):
            if sym.colorizer.type == "RasterStretchColorizer":
                sym.colorizer.colorRamp = aprx.listColorRamps('Bathymetric Scale')[0] # or
'Multipart Color Scheme' for example
                sym.colorizer.stretchType = "MinimumMaximum"
                lyr.symbology = sym
        else:
            arcpy.AddMessage("Symbology: no hasattr")


        ### Warnings ========================================================================
        if len(No_crust) > 0:
            arcpy.AddMessage("Plates without oceanic crust (which will results in gaps):")
            arcpy.AddMessage(No_crust)

        arcpy.AddMessage("=== Reconstruction completed ===")
        # --- End of the Age loop ---

    else:
        arcpy.AddMessage("There is no shapefile for age %s"%age)
        Not_done.append(age)

    # --- End of looping through all ages ---

if len(Not_done) > 0:
    arcpy.AddMessage("Ages for which no reconstruction has been made:")
    arcpy.AddMessage(Not_done)
```

RÉFÉRENCES

Dietz, R. S. (1961). Continent and ocean basin evolution by spreading of the sea floor. *Nature*, *190*(4779), 854-857.

Earth Observatory of Singapore. (s. d.). *How do we know the age of the seafloor*. Earth Observatory of Singapore. Consulté 21 septembre 2022, à l'adresse https://prod.earthobservatory.sg/earth-science-education/earth-science-faqs/geology-and-tectonics/how-do-we-know-the-age-of-the-seafloor

Gramling, C. (2021, juillet 2). A WWII submarine-hunting device helped prove the theory of plate tectonics. *Science News*. https://www.sciencenews.org/article/fluxgate-magnetometer-submarine-plate-tectonics

Hochard, C. (2008). *GIS and geodatabases application to global scale plate tectonics modelling*.

Müller, R. D., Roest, W. R., Royer, J.-Y., Gahagan, L. M., & Sclater, J. G. (1997). Digital isochrons of the world's ocean floor. *Journal of Geophysical Research: Solid Earth*, *102*(B2), 3211-3214.

Müller, R. D., Sdrolias, M., Gaina, C., & Roest, W. R. (2008). Age, spreading rates, and spreading asymmetry of the world's ocean crust. *Geochemistry, Geophysics, Geosystems*, *9*(4).

National Geographic Society. (s. d.). *Seafloor Spreading*. National Geographic. Consulté 21 septembre 2022, à l'adresse https://education.nationalgeographic.org/resource/seafloor-spreading

Royer, J.-Y., Müller, D. R., Gahagan, L. M., Lawver, L. A., Mayes, C. L., Nürnberg, D., & Sclater, J. G. (1992). *A global isochron chart* [PhD Thesis]. Institute for Geophysics, The University of Texas at Austin.

Stampfli, G. M., & Borel, G. D. (2002). A plate tectonic model for the Paleozoic and Mesozoic constrained by dynamic plate boundaries and restored synthetic oceanic isochrons. *Earth and Planetary science letters*, *196*(1-2), 17-33.

Verard, C. (2019a). Panalesis : Towards global synthetic palaeogeographies using integration and coupling of manifold models. *Geological Magazine*, *156*(2), 320-330.

Verard, C. (2019b). Plate tectonic modelling : Review and perspectives. *Geological Magazine*, *156*(2), 208-241.