La délimitation des écosystèmes riverains : Revue des méthodes existantes et développement d'une méthodologie à l'échelle du Québec méridional

Auteur : Poncelet, Lucien
Promoteur(s) : Lejeune, Philippe; 14589
Faculté : Gembloux Agro-Bio Tech (GxABT)
Diplôme : Master en bioingénieur : sciences et technologies de l'environnement, à finalité spécialisée
Année académique : 2020-2021
URI/URL : http://hdl.handle.net/2268.2/13093

7. ANNEXES

Annexe 1 : Applications de la délimitation d'un cours d'eau pour les domaines de la Science et de la Gestion (Tadaki et al., 2014).

Scientific Applications	Management Applications
Structuring information to facilitate the identification of patterns and relationships among environmental attributes • Describing river variability across an area • Describing behavior/change over time *Categorical induction* • Predicting a river's behavior from its appearance (inductive generalization) • A way to extrapolate site-specific data to other similar locations • Choosing representative rivers or sample sites for observation or experiment *Developing conceptual models* • Generating and testing hypotheses that link form and process • Analyzing system adjustments and interpreting controlling factors (drivers, pressures, stressors, thresholds, etc.) *Framing collaborations* between disciplines, between management agencies and bridging gaps between science and management • Providing common language and consistent concepts/spatial entities • Providing 'manageable' units (e.g., a reach, a river, and a patch) with predictable characteristics (e.g., a braided river, a riffle, and an ephemeral stream) • Enables development of meta-frameworks that can be applied across classes (e.g., environmental flows)	*Resource inventory and allocation* • Human uses/values—commercial, transport, leisure, aesthetics, ecosystem services • Risk assessment—flood potential, erosion potential, etc. • Water allocation management (quantity, quality) • Ecosystem management, biodiversity, habitat availability/viability, presence of rare/endangered species *Designing monitoring systems* • Assessing and benchmarking ecosystem condition • Appraisal of the 'range of variability' in space/time • Standardizing monitoring procedures across jurisdictions and institutions to facilitate comparison *Setting conservation/restoration priorities* • Identification of sites or streams with representative or rare/unique ecological characteristics • Setting restoration/conservation objectives (a reference condition) • Diagnosing restoration needs and pathways • Indication of methods likely to succeed on a given stream type • Identifying sensitivities/threats to streams of a given type *Strategic decision-making* • Scoping river futures and the likely range of evolutionary trajectories • Foresighting impacts of land use practices, climate change, and responses to management activities

Figure 30 - Applications proposées par Tadaki et al. (2014) pour la délimitation d'un cours d'eau dans les domaines de la Science et de la Gestion

Annexe 2 : Variables reprises dans la méthode de délimitation de Ouellet et al. (2019).

8. Tables

Table 1 Variables included in the PCA analyses and in each sub-classification. Variable abbreviations (last column) followed by a star (*) indicate variables that were included in the final sub-classifications. Variables with the expression "log" in the abbreviation have been logged before being used in the analyses. Discharge indices are based on long-term monthly averages for the period 1971-2000. The long-term maximum and minimum monthly discharge is represented by the highest and lowest month, respectively, of the annual flow regime based on the same time period. Unless stated otherwise, values were collected at the representative downstream pixel for each spatial scale unit.

Sub-classification	Variables	Spatial scale of data collection	Dataset used	Abbreviation for figures
Hydrological	Long-term average discharge	Upstream watershed	WaterGAP (Döll et al. 2003 downscaled to HydroSHEDS (Lehner and Grill 2013)	log.av*
	Long-term maximum monthly discharge	Upstream watershed		log.ma
	Long-term minimum monthly discharge	Upstream watershed		log.mi
	Flow regime variability 1 (log(ma/av))	Upstream watershed		log.ma.av
	Flow regime variability 2 (log(ma/mi))	Upstream watershed		log.ma.mi*
	Long-term average land surface runoff	Upstream watershed		Av.Run
	First frost free day (average Julian Day when minimum air temperature exceeds 0°C)	River reach	Pedlar et al. (2015)	JDab0*
Physio-climatic	Temperature seasonality (as calculated by WorldClim: standard deviation of daily temperature * 100)	River reach	WorldClim (Hijmans et al. 2005)	T.seas
	Average annual range of temperature	River reach		T.range*
	Average daily range of temperature	River reach		T.day.rg
	Average temperature of the warmest quarter	River reach		T.av.warm
	Average temperature of the coldest quarter	River reach		T.av.cold
	Average maximum temperature of the warmest quarter	River reach		T.mx.warm
	Average minimum temperature of the coldest quarter	River reach		T.mn.cold
	Average temperature	River reach		T.av*
	Climate moisture index (calculated as defined by Willmott and Fedema, 1992)	River reach	Own calculation based on data by Hijmans et al. (2005) and Zomer et al. (2008)	CMI*
	Long-term average maximum snow depth categorical from small to large)	River reach	Natural Resources Canada (2006)	Snow.av
	Average number of winter months defined as months with snow on the ground during a year	River reach	MODIS snow cover (Hall et al. 2002)	Winter.mo
	Range of elevation in the hillslope	River reach	HydroSHEDS elevation	Log.rg.dem
	Average elevation	River reach	(Lehner et al. 2008)	Log.dem*
	Average terrain slope	5km radius from the river reach		Log. slope
Geomorphic	Total stream power	Upstream watershed	Own calculation based on WaterGAP discharge (Döll et al. 2003) downscaled to HydroSHEDS network (Lehner and Grill 2013) and stream gradients calculated from HydroSHEDS elevation data (Lehner et al. 2008)	Log.sp*
	Percentage of upstream watershed area covered by peatlands	Upstream watershed	Peatland map of Canada (Tarnocai et al. 2011)	Peat*
	Percentage of upstream watershed area covered by lakes	Upstream watershed	HydroLAKES (Messager et al. 2016)	lake.pc*

Figure 31 - Variables reprises dans la méthode de délimitation de Ouellet et al. (2019)

Annexe 3 : Variables associées à chacune des classes reprises dans la méthode de délimitation de délimitation de Church (2006)

Table 1 Classification of river channels and riverine landscapes modified from Church (2006)

Channel type/Bankfull Shields stress (τ^*_{bf})[a]	*Sediment type*	*Sediment transport regime*[b]	*Channel morphology*[c]	*Channel stability*
Jammed channel; $\tau^*_{bf} = 0.04^+$	Cobble- or boulder-gravel	Low total transport, but subject to debris flows; bed load transport is a high percentage of the total load (q_b/q_t typically >10%)	Step-pool or boulder cascades; width typically a low multiple of largest boulder size; Slope (*S*) >3°	Stable for long periods of time with throughput of bed load finer than structure-forming clasts; subject to catastrophic destabilization in debris flows
Threshold channel; $\tau^*_{bf} = 0.04^+$	Cobble-gravel	Schumm's "bed load" channels; low to moderate total transport, with a high percentage of bed load (q_b/q_t typically >10%), but usually limited to partial mobility (*sensu* Wilcock and McArdell, 1993)	Cobble-gravel channel bed; single thread or wandering; highly structured bed; relatively steep; low sinuosity; width-to-depth (*w/h*) >20, except in headwater boulder channels	Relatively stable for extended periods, but subject to major floods causing lateral channel instability and avulsion; may exhibit serially reoccupied secondary channels
Threshold channel; τ^*_{bf} up to 0.15	Sandy-gravel to cobble-gravel	Moderate total transport, with a moderate to high percentage of bed load (q_b/q_t typically 5–10%); partial transport to full mobility (*sensu* Wilcock and McArdell, 1993)	Gravel to sandy-gravel single thread to braided; limited, local bed structure; complex bar development by lateral accretion; moderately steep; low sinuosity; *w/h* very high (>40)	Subject to avulsion and frequent channel shifting; braid-form channels may be highly unstable, both laterally and vertically; single-thread channels subject to chute cutoffs at bends; deep scour possible at sharp bends
Transitional channel; $\tau^*_{bf} = 0.15–1.0$	Sand to fine gravel	Schumm's "mixed load" channels; moderate to high total transport, with a moderate percentage of bed load (q_b/q_t typically 3–5%); full mobility, with sandy bed forms	Mainly single-thread, irregularly sinuous to meandered; lateral/point bar development by lateral and vertical accretion; levees present; moderate gradient; sinuosity < 2; *w/h* >40	Single-thread channels, irregular lateral instability or progressive meanders; braided channels laterally unstable; degrading channels exhibit both scour & channel widening
Labile channel; $\tau^*_{bf} > 1.0$	Sandy channel bed, fine sand to silt banks	Schumm's "suspended load" channels; high total transport, with a low to moderate percentage of bed load (q_b/q_t typically 1–3%); fully mobile, sand bed forms; sediment transport at most stages	Single thread, meandered with point bar development; significant levees; low gradient; sinuosity >1.5; *w/h* <20; serpentine meanders with cutoffs	Single-thread, highly sinuous channel; loop progression and extension with cutoffs; anastomosis possible, islands are defined by vegetation; vertical accretion in the floodplain; vertical degradation in channel
Labile channel; τ^*_{bf} up to 10	Silt to sandy channel bed, silty to clay-silt banks	High total transport, almost exclusively suspended and wash load (q_b/q_t typically < 1%); minor bed form development	Single-thread or anastomosed channels; prominent levees; very low gradient; sinuosity >1.5; *w/h* <15 in individual channels	Single-thread or anastomosed channels; common in deltas and inland basins; extensive wetlands and floodplain lakes; vertical accretion in floodplain; slow or no lateral movement of individual channels

Figure 32 - Variables associées à chacune des classes reprises dans la méthode de délimitation de Church (2006)

Annexe 4 : Modèle automatique de caractérisation du seuil de HAND maximal (fonctionnement détaillé + script)

Fonctionnement détaillé

Le modèle automatique de caractérisation du seuil de HAND maximal pour la délimitation de l'ELR est basé sur l'identification de sauts de pente brusques et de zones plates le long de transects perpendiculaires au cours d'eau. Il a ainsi été important, dans le cadre du développement de ce modèle, d'isoler l'ELR d'une part, des terrasses alluviales supérieures mais également du lit mineur, deux espaces démontrant une topographie plate également et délimités par des sauts de pente brusque (Mason et al., 2007). Pour cela, des paramètres d'entrée de seuils bas et haut de HAND entre lesquels l'ELR est cherché ont été pris en compte. Ces deux paramètres peuvent être aisément calibrés à partir d'analyses transversales du cours d'eau pour la couche HAND et de photo-interprétations de couches d'imagerie satellitaire. La Figure 33 donne un exemple de transect avec les valeurs de HAND échantillonnées le long de celui-ci sur la rivière Bulstrode. Des exemples de seuils minimaux et maximaux sont montrés. On y voit également le lit mineur, partie plate la plus basse et une terrasse alluviale supérieure.

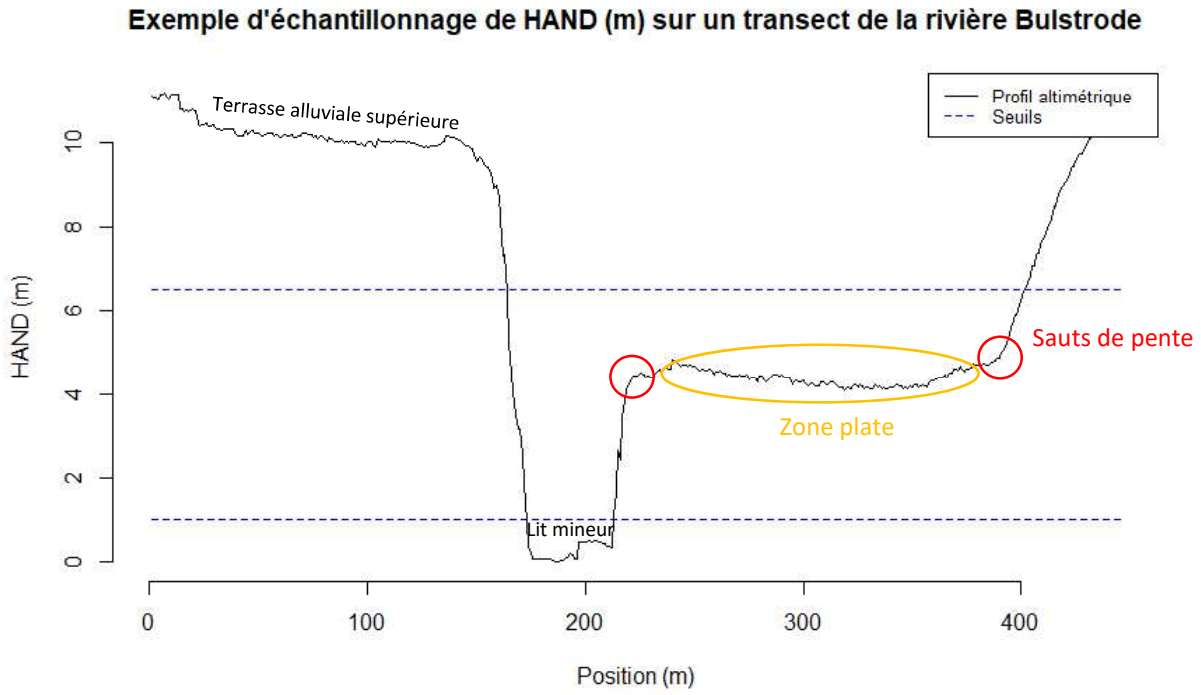

Figure 33 – Exemple d'échantillonnage de la valeur de Height Above Nearest Drainage (HAND) (m) sur une coupe transversale réalisée sur la rivière Bulstrode proche de Saint-Samuel. Une terrasse alluviale supérieure et le lit mineur sont montrés ainsi que les sauts de pente et zones plates pouvant servir à isoler la valeur de HAND de l'Espace Latéral Riverain (ELR).

La méthodologie générale utilisée pour le modèle est la suivante. Tout d'abord, la couche HAND est modélisée à partir du MNT en utilisant l'outil *Height Above Nearest Drainage* de la boite à outil RTT (Dilts et al., 2010). Ensuite, des transects transversaux au cours d'eau sont générés, les valeurs de HAND sont échantillonnées le long de chacun des transects et un ou plusieurs seuils de HAND sont extraits de chaque transects. Enfin, les statistiques descriptives de l'ensemble des valeurs de HAND caractérisées sur le cours d'eau sont calculées (moyenne, médiane, 3^{e} quartile, etc.) afin d'en tirer une

valeur finale de seuil de HAND pour délimiter l'ELR. Quatre méthodes différentes ont été explorées pour extraire les seuils de HAND de chaque transect. Deux d'entre elles se basent sur l'identification de sauts de pente et les deux autres sont fondées sur l'identification de zones plates tels que visibles sur la Figure 33. Ensuite, parmi ces méthodes, deux se veulent plus automatiques que les autres qui nécessitent plus de paramètres mais qui permettent de donner plus de liberté à l'utilisateur. Les organigrammes suivants (Figure 34) expriment le fonctionnement de chacune des quatre méthodes. Le processus représenté est appliqué sur chaque transect. Les méthodes 1 et 2 sont les méthodes

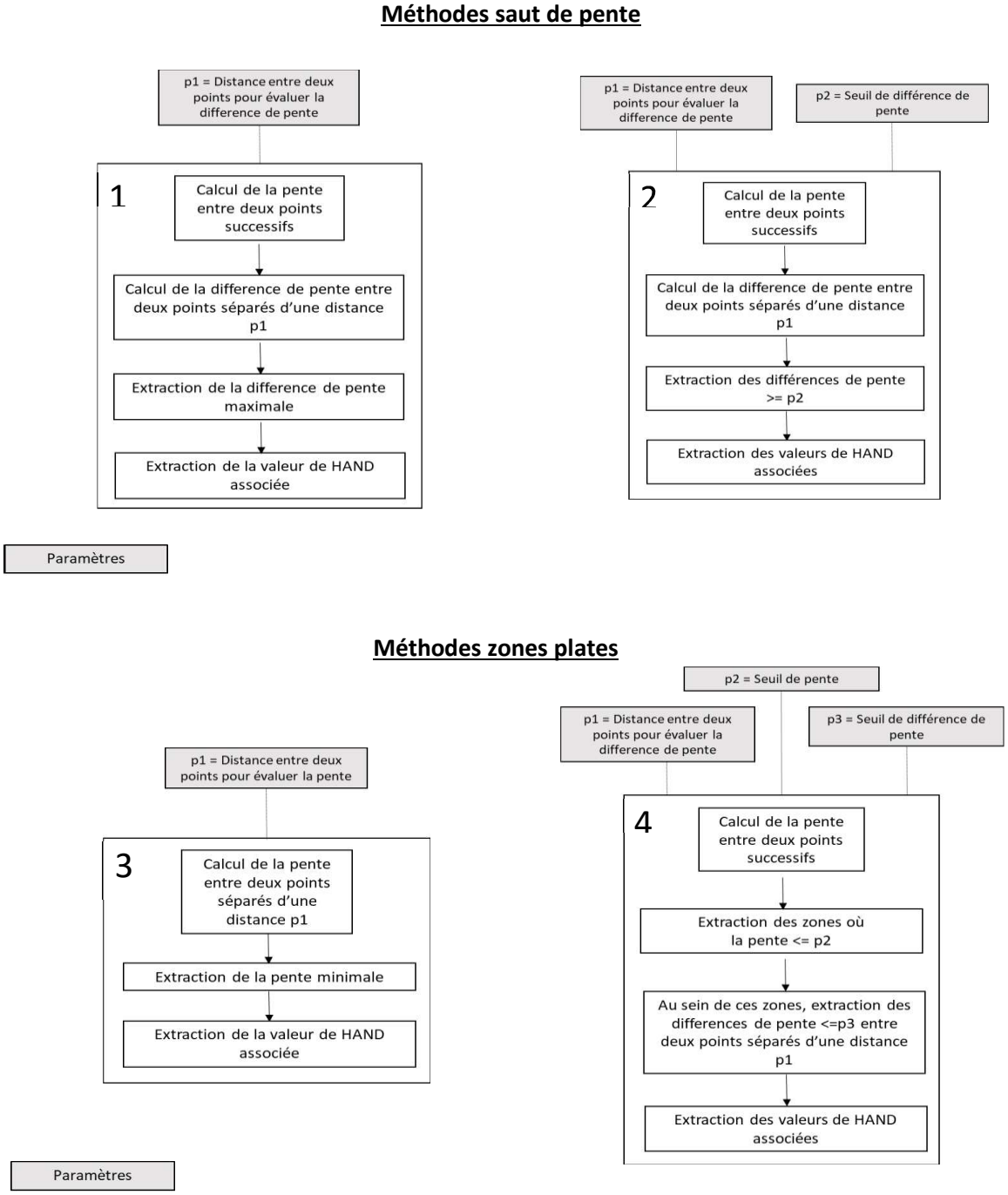

Figure 34 - Schémas des quatre méthodes d'extraction des seuils de HAND maximum pour isoler l'Espace Latéral Riverain (ELR). (1) + (2) = méthodes saut de pente, (3) + (4) = méthodes zones plates

« saut de pente ». Elles sont principalement basées sur la différence de pente entre deux points espacés d'une certaine distance. La méthode 2 se veut plus rigoureuse que la méthode 1 avec

l'introduction d'un paramètre supplémentaire qui, au lieu de prendre la valeur maximale de différence de pente entre deux points sur le transect, permet de fixer un seuil de différence de pente qui caractérise la limite de l'ELR. Les méthodes 3 et 4 sont les méthodes zones plates. La méthode 4 est plus développée que la méthode 3 ; en effet, 3 paramètres différents sont pris en compte. Le premier paramètre p1 permet de déterminer la longueur de la zone plate voulue. Le deuxième paramètre est utile pour déterminer à partir de quel seuil de pente la zone est considéré comme plate. Et enfin, le troisième paramètre permet de déterminer un seuil de différence de pente au sein de la zone plate. Les deux méthodes moins automatiques (2 et 4) ont l'avantage de pouvoir permettre de n'extraire les valeurs de HAND que lorsque certaines conditions hydrogéomorphologiques sont respectées. Ainsi, des problèmes liés à des ELR difficilement identifiables, tels que lorsque les cours d'eau sont fort confinés, peuvent être palliés.

Choix de la méthode et paramètres

Après avoir fait marcher le modèle sur les deux sites d'études avec les 4 méthodes et pour différentes valeurs de paramètres, la méthode qui s'est avéré la plus cohérente pour les deux sites est la méthode 2. En effet, celle-ci a fourni les valeurs de seuil de HAND maximales les plus proches des valeurs obtenues par calibration et les moins dispersées autour de leur moyenne pour l'ensemble des valeurs échantillonnées sur les différents transects. Concernant les valeurs de paramètres sélectionnées, celles-ci sont données au sein du Tableau 16. Elles ont été approchées à partir d'analyses transversales rapides du cours d'eau pour la couche HAND.

Tableau 16 - Valeurs optimales des paramètres pour le modèle d'extraction du seuil maximal de Height Above Nearest Drainage (HAND) pour délimiter l'Espace Latéral Riverain (ELR) pour les sites d'études « rivière des Saults » et « rivière Bulstrode »

Méthode	Seuil bas de HAND (m)	Seuil haut de HAND (m)	Distance pour la différence de pente (m)	Seuil de différence de pente (-)
2	0.5	6	5	0.4

Script Python

```
### Ce script a pour but de trouver la valeur de seuil de HAND maximale qui
permet de
### delimiter l'Espace Latéral Riverain a partir d'une couche HAND generee
a partir
### d'une couche de cours d'eau et d'un MNT

import arcpy, sys, os, time, numpy,math
import pandas as pd
from matplotlib import pyplot as plt
from arcpy.sa import *
arcpy.env.overwriteOutput=True

start_time = time.time()
## Etape 1 (QGIS) : Generation de transects le long de la rivière avec
## l'Outil Transect by distance de l'extension Geometric Attributes
# Choisir longueur de transect adaptee
# Choisir espace entre transect adapte

## Etape 2 (QGIS) : Conversion de transects en points avec Outil Points le
```

```
long d'une geometrie
# echantillonage a 1m d'espacement (resolution raster)
# le champ Distance permet d'associer les points a leur transect
# le champ distance_1 referencie les points dans l'ordre depuis le debut de
la ligne jusqu'a la fin
# on importe la couche de points ici et les traitements commencent
#########################################################################
################################
###Parametres a modifier
##parametres d'entree (obligatoire)
highTresh=6.5 #seuil haut pour chercher l'ELR
lowTresh=0.5 # seuil bas pour chercher l'ELR
meth=4 # methode choisie
##couches en entree (obligatoire)
transectPoints='C:/Users/lucie/Documents/GIS/Python/Input/TransectPointsBul
.shp' # a changer par riviere
HAND='C:/Users/lucie/Documents/GIS/Python/Input/handbul' # a changer par
riviere
##parametres lies aux methodes (obligatoire en fonction de la methode
choisie)
# SEULS LES PARAMETRES LIES A LA METHODE DOIVENT ETRE MODIFIES
p1=5 #Methode 1 : Longueur a considerer pour la difference de pente
p2=5 #Methode 2 : Longueur a considerer pour la difference de pente
slpdifTresh2=2 #Methode 2 : Seuil de difference de pente pour isoler l'ELR'
p3=5 #Methode 3 : Longueur a considerer pour la portion plate
p4=6 #Methode 4 : Longueur a considerer pour la portion plate
slpTresh4=0.01 #Methode 4 : Seuil de pente caracterisant une portion plate
slpdifTresh4=0.01 #Methode 4 : Seuil de difference de pente pour la portion
plate
#########################################################################
################################
outXls=os.path.dirname(HAND)+'/outXLS.xls'
outpoint=os.path.dirname(HAND)+'/outPoint.shp'

## Etape 3 : on calcule pour chaque point la valeur de HAND
arcpy.AddMessage('Extraction des valeurs de HAND sur les points du
transect')
arcpy.CheckOutExtension("Spatial")
arcpy.sa.ExtractValuesToPoints(transectPoints,HAND,outpoint)

## Etape 4 : on enleve les transects dans lesquels il y a des points qui
ont des valeurs aberrantes
arcpy.AddMessage('Nettoyage points en dehors de HAND')
wrongRows=[]
with arcpy.da.SearchCursor(outpoint,['RASTERVALU','Distance']) as cursor:
    for row in cursor:
        if row[0]<0:
            wrongRows.append(row[1])
with arcpy.da.UpdateCursor(outpoint,['RASTERVALU','Distance']) as cursor:
    for row in cursor:
        if row[1] in wrongRows:
            cursor.deleteRow()
## Etape 5 : on convertit la table d'attribut du shapefile en dataframe
arcpy.TableToExcel_conversion(outpoint,outXls)
predf = pd.read_excel(outXls,sheetname='outXLS')
df=predf[['Distance','distance_1','RASTERVALU']]

## Etape 6 : on modifie le dataframe pour respecter les seuils haut et bas
ou l'ELR va etre cherche
arcpy.AddMessage('Application des seuils sur les valeurs de HAND')
indexOutTresh=df[ (df['RASTERVALU'] < lowTresh) | (df['RASTERVALU'] >
```

```
highTresh) ].index
df=df.drop(indexOutTresh)
n = len(df)
df.index=range(0,n)
dist=df.iloc[1][1]-df.iloc[0][1]

## Etape 8 : on lance les methodes
## Methode 1 : pente auto-auto : trouver les plus grands sauts de pente sur
chaque transect sur telle distance
if meth==1:
    arcpy.AddMessage('============Methode des seuils de pente
choisie=============')
    listTransects=[df.iloc[0][0]]
    # creer une liste avec toutes les valeurs de Distance
    for i in range(0,n-1):
        if df.iloc[i+1][0]!=df.iloc[i][0]:
            listTransects.append(df.iloc[i+1][0])
    ntrans=len(listTransects)
    NaN=numpy.nan
    df["Slope"]=NaN
    df["Slopedif"]=NaN
    # Calcul de la pente
    arcpy.AddMessage('Calcul de la pente')
    for i in range(0,n-1):
        if df.iloc[i][0]==df.iloc[i+1][0]:
            df.loc[i, 'Slope'] = (df.iloc[i + 1][2] - df.iloc[i][2]) / dist
    # calcul des differences de pente a p metres d'ecart
    arcpy.AddMessage('Calcul des differences de pentes a {0} metres
d\'ecart'.format(p1))
    for i in range(0,n-p1):
        if df.iloc[i][0]==df.iloc[i+p1][0]:
            df.loc[i, 'Slopedif'] = (df.iloc[i + p1][3] - df.iloc[i][3])

    #creer une liste qui reprend le max et les 3 autres params associes
(transectID, position dans transect et surtout valeur de HAND)
    arcpy.AddMessage('Extraction des sauts de pente')
    SMax=[]
    SMin=[]
    tempList=[]
    IDList=[]
    for t in range(0,ntrans):
        for i in range(0,n-1):
            if df.iloc[i][0]==listTransects[t]:
                tempList.append(df.iloc[i][4])
                IDList.append(i)
                if df.iloc[i][0]!=df.iloc[i+1][0]:
                    Max=max(tempList)
                    Min=min(tempList)
                    dfT=df.iloc[min(IDList):max(IDList)]
                    x1=dfT[dfT["Slopedif"]==Max]
                    for j in range(0,len(x1)):

SMax.append([x1.iloc[j][0],x1.iloc[j][1],x1.iloc[j][2],x1.iloc[j][4]])
                    tempList=[]
                    IDList=[]


# on cree une liste des valeurs de HAND obtenus extrait les valeurs finales
voulues
    arcpy.AddMessage('Extractions valeurs de HAND liees aux sauts de
pente')
```

```
    HANDList=[]
    for i in range(0,len(SMax)):
        HANDList.append((SMax[i][2]))
    moy=sum(HANDList)/len(HANDList)
    ndec=numpy.percentile(HANDList,90)
    tqua=numpy.percentile(HANDList,75)
    med=numpy.percentile(HANDList,50)
    std = numpy.std(HANDList)
    arcpy.AddMessage('Valeur moyenne = {0}'.format(moy))
    arcpy.AddMessage('Valeur mediane = {0}'.format(med))
    arcpy.AddMessage('3e quartile = {0}'.format(tqua))
    arcpy.AddMessage('9eme decile = {0}'.format(ndec))
    arcpy.AddMessage('Ecart-type = {0}'.format(std))
    arcpy.AddMessage('Temps d\'execution: {0} secondes'.format(time.time()
- start_time))

## Methode 2 : pente auto-manuelle # trouver les sauts de pente plus grand
que telle valeur sur telle distance (attention : saut de pente = difference
de pente !)
elif meth==2:
    arcpy.AddMessage('============Methode des seuils de pente
choisie=============')
    NaN=numpy.nan
    df["Slope"]=NaN
    df["Slopedif"]=NaN
    # Calcul de la pente
    arcpy.AddMessage('Calcul de la pente')
    for i in range(0,n-1):
        if df.iloc[i][0]==df.iloc[i+1][0]:
            df.loc[i, 'Slope'] = (df.iloc[i + 1][2] - df.iloc[i][2]) / dist
    # calcul des differences de pente a p2 metres d'ecart
    arcpy.AddMessage('Calcul des differences de pentes a {0} metres
d\'ecart'.format(p2))
    for i in range(0,n-p2):
        if df.iloc[i][0]==df.iloc[i+p2][0]:
            df.loc[i, 'Slopedif'] = math.fabs(df.iloc[i + p2][3] -
df.iloc[i][3])

    arcpy.AddMessage('Extraction des sauts de pente')
    slopeList=[]
    for i in range(0, n - p2):
        if df.iloc[i][0] == df.iloc[i + p2][0]:  # pour ne pas faire
d'operations entre deux transects
            if df.iloc[i][4]>=slpdifTresh2: # critere pente faible
                    slopeList.append([df.iloc[i][0], df.iloc[i][1],
df.iloc[i][2], df.iloc[i][4]])
                    print i


    # on cree une liste des valeurs de HAND obtenues
    arcpy.AddMessage('Extractions valeurs de HAND liees aux sauts de
pente')
    HANDList = []
    for i in range(0,len(slopeList)):
        HANDList.append(slopeList[i][2])
    moy = sum(HANDList) / len(HANDList)
    ndec = numpy.percentile(HANDList, 90)
    tqua = numpy.percentile(HANDList, 75)
    med = numpy.percentile(HANDList, 50)
    std=numpy.std(HANDList)
    arcpy.AddMessage('Valeur moyenne = {0}'.format(moy))
```

```
    arcpy.AddMessage('Valeur mediane = {0}'.format(med))
    arcpy.AddMessage('3e quartile = {0}'.format(tqua))
    arcpy.AddMessage('9eme decile = {0}'.format(ndec))
    arcpy.AddMessage('Ecart-type = {0}'.format(std))

    arcpy.AddMessage('Temps d\'execution: {0} secondes'.format(time.time()
- start_time))


elif meth==3:
    arcpy.AddMessage('============Methode des zones plates en hauteur
choisie=============')
    listTransects = [df.iloc[0][0]]
    # creer une liste avec toutes les valeurs de Distance
    for i in range(0, n - 1):
        if df.iloc[i + 1][0] != df.iloc[i][0]:
            listTransects.append(df.iloc[i + 1][0])
    ntrans = len(listTransects)
    NaN = numpy.nan
    df["Slope"] = NaN
    # Calcul de la pente a p metre d'ecart
    arcpy.AddMessage('Calcul de la pente')
    for i in range(0, n - p3):
        if df.iloc[i][0] == df.iloc[i + p3][0]:
            df.loc[i, 'Slope'] = math.fabs(df.iloc[i + p3][2] -
df.iloc[i][2]) / (dist*p3) # on prend la valeur absolue

    # creer une liste qui reprend le max et les 3 autres params associes
(transectID, position dans transect et surtout valeur de HAND)
    arcpy.AddMessage('Extraction des points les plus plats de chaque
transect')
    flat = []
    tempList = []
    IDList = []
    for t in range(0, ntrans):
        for i in range(0, n - 1):
            if df.iloc[i][0] == listTransects[t]:
                tempList.append(df.iloc[i][3])
                IDList.append(i)
                if df.iloc[i][0] != df.iloc[i + 1][0]:
                    Min = min(tempList)
                    dfT=df.iloc[min(IDList):max(IDList)]
                    x = dfT[dfT["Slope"] == Min]
                    for j in range(0,len(x)):
                        flat.append([x.iloc[j][0], x.iloc[j][1],
x.iloc[j][2], x.iloc[j][3]])
                    tempList = []
                    IDList=[]
    # on cree une liste des valeurs de HAND obtenus extrait les valeurs
finales voulues
    arcpy.AddMessage('Extractions valeurs de HAND liees aux zones plates')
    HANDList = []
    for i in range(0, len(flat)):
        HANDList.append(flat[i][2])
    moy = sum(HANDList) / len(HANDList)
    ndec = numpy.percentile(HANDList, 90)
    tqua = numpy.percentile(HANDList, 75)
    med = numpy.percentile(HANDList, 50)
    std = numpy.std(HANDList)
    arcpy.AddMessage('Valeur moyenne = {0}'.format(moy))
    arcpy.AddMessage('Valeur mediane = {0}'.format(med))
```

```
    arcpy.AddMessage('3e quartile = {0}'.format(tqua))
    arcpy.AddMessage('9eme decile = {0}'.format(ndec))
    arcpy.AddMessage('Ecart-type = {0}'.format(std))
    arcpy.AddMessage('Temps d\'execution: {0} secondes'.format(time.time()
- start_time))

elif meth==4: ## cela ressemble fort a la structure de la methode
precedente mais on recherche, en valeur absolue, les plus petites valeurs
de difference de pente
    arcpy.AddMessage('============Methode des zones plates en hauteur
choisie=============')
    NaN = numpy.nan
    df["Slope"] = NaN
    # Calcul de la pente
    arcpy.AddMessage('Calcul de la pente')
    for i in range(0, n - 1):
        if df.iloc[i][0] == df.iloc[i + 1][0]:
            df.loc[i, 'Slope'] = math.fabs(df.iloc[i + 1][2] -
df.iloc[i][2]) / dist # ici on prend la valeur absolue !

# on a deux conditions, on recherche les pentes faibles en valeur absolue
mais egalement qui restent faibles sur une certaine distance
    arcpy.AddMessage('Extraction des zone plates')
    flatList=[]
    for i in range(0, n - p4):
        if df.iloc[i][0] == df.iloc[i + p4][0]:
            if df.iloc[i][3]<=slpTresh4:
                if math.fabs((df.iloc[i+(p4/2)][3])-
(df.iloc[i][3]))<=slpdifTresh4 and \
                math.fabs((df.iloc[i+(p4)][3])-
(df.iloc[i+(p4/2)][3]))<=slpdifTresh4:
                    flatList.append([df.iloc[i][0], df.iloc[i][1],
df.iloc[i][2], df.iloc[i][3]])
                    print i


    # on cree une liste des valeurs de HAND obtenues
    arcpy.AddMessage('Extractions valeurs de HAND liees aux zones plates')
    HANDList = []
    for i in range(0,len(flatList)):
        HANDList.append(flatList[i][2])
    moy = sum(HANDList) / len(HANDList)
    ndec = numpy.percentile(HANDList, 90)
    tqua = numpy.percentile(HANDList, 75)
    med = numpy.percentile(HANDList, 50)
    std=numpy.std(HANDList)
    arcpy.AddMessage('Valeur moyenne = {0}'.format(moy))
    arcpy.AddMessage('Valeur mediane = {0}'.format(med))
    arcpy.AddMessage('3e quartile = {0}'.format(tqua))
    arcpy.AddMessage('9eme decile = {0}'.format(ndec))
    arcpy.AddMessage('Ecart-type = {0}'.format(std))

    arcpy.AddMessage('Temps d\'execution: {0} secondes'.format(time.time()
- start_time))

arcpy.Delete_management(outXls)
arcpy.Delete_management(outpoint)
```

Annexe 5 : Scripts python pour la délimitation bidimensionnelle

1) Création de transects à l'intersection des UEA

```
## Ce script permet de generer des transects a chaque intersection entre
deux UEA
# Les transects auront une orientation perpendiculaire a la moyenne des
orientations des
# UEA de part et d'autre du point de liaison entre les 2 UEA

## Etape 1 : Import des modules + parametres entree et sortie
import arcpy,sys,os,time,math,numpy
arcpy.env.overwriteOutput=True # choix de l'espace de travail

start_time = time.time() #initiation du temps de calcul (a titre indicatif)

UEA=arcpy.GetParameterAsText(0)
stream=arcpy.GetParameterAsText(1)
tempFolder=arcpy.GetParameterAsText(2)
IDField=arcpy.GetParameterAsText(3)
XSLength_=arcpy.GetParameterAsText(4)
XSLength=int(XSLength_)
outXS=arcpy.GetParameterAsText(5)


outDir=tempFolder+'/'
UEACut=os.path.dirname(outXS)+'/UEA_cut.shp' # couche UEA coupee selon la
couche de GRHQ
spatial_ref=arcpy.Describe(UEA).spatialReference
# listes contenant les futures startpoints, endpoints et les ID des UEA
start=[]
end=[]
ID=[]

## Etape 2 : couper la couche de UEA selon une couche de GRHQ ne prenant
que les rivieres d'interet
arcpy.Clip_analysis(UEA,stream,UEACut)

## Etape 3 : separer la couche UEA en fonction de l'attribut ID
# On va extraire une par une chaque UEA avec l'outil Select_analysis
for row in arcpy.da.SearchCursor(UEACut, ["OID@"]):
    outPut=outDir+'UEA_{0}.shp'.format(row[0])
    expr="{0} = {1}".format(IDField,row[0])
    arcpy.Select_analysis(UEACut,outPut,expr)

## Etape 4 : on va aller extraire, pour chaque couche, les coordonnees X Y
du debut de ligne et de la fin de ligne
    for rows in arcpy.da.SearchCursor(outPut,["SHAPE@"]):
        startPt=rows[0].firstPoint
        startX=startPt.X
        startY=startPt.Y
        startCoord=[startX,startY]
        start.append(startCoord) # creation liste coordonnes startpoint

        endPt=rows[0].lastPoint
        endX=endPt.X
        endY=endPt.Y
        endCoord=[endX,endY]
        end.append(endCoord) # creation liste coordonnes startpoint
```

```
        # on cree la liste des coordonnes de startpoint et de endpoint
        ID.append(row[0]) #on rajoute l'ID de l'UEA pour voir par apres
quelles UEA sont liees

## Etape 5 : on convertit chaque couche en succession de vertex
        outVertex = outDir + 'UEA_Vertex{0}.shp'.format(row[0])
        arcpy.FeatureVerticesToPoints_management(outPut, outVertex, "ALL")
# A ce stade-la, les listes sont crees

features=[]

## Etape 6 : on boucle sur les starts points et end points pour trouver les
couples qui sont similaires (pour trouver les points d'intersection)
for index1, coord1 in enumerate(start):
    for index2, coord2 in enumerate(end):
        if coord1==coord2:
            arcpy.AddMessage('{0},{1}'.format(index1,index2))

## Etape 7 : pour les points qui matchent, on va chercher les coordonnees
du point apres le start point et avant le endpoint
            # On reprend les vertex des deux UEA en question
            outVertexStart = outDir + 'UEA_Vertex{0}.shp'.format(index1)
            outVertexEnd = outDir + 'UEA_Vertex{0}.shp'.format(index2)
            nRowsObj=arcpy.GetCount_management(outVertexEnd)
            nRowsStr=nRowsObj.getOutput(0)
            nRows=int(nRowsStr)
            coEndPtID=nRows-2
            # On initialise les coStartPt ET coEndPt et l'expression qui va
etre determinee pour les trouver
            coStartPt=outDir+'UEA_coStart{0}.shp'.format(index1)
            exprStart="FID = 1"
            coEndPt=outDir+'UEA_coEnd{0}.shp'.format(index2)
            exprEnd="FID = {0}".format(coEndPtID)
            arcpy.Select_analysis(outVertexStart, coStartPt, exprStart)
            arcpy.Select_analysis(outVertexEnd,coEndPt,exprEnd)
            # On obtient les coordonnees X et Y des coPoints et calcule la
direction de la fin de ligne (on prend la direction par rapport a l'axe des
x vers les starts
            for row in arcpy.da.SearchCursor(coStartPt, ["SHAPE@X",
"SHAPE@Y"]):
                coStartX = row[0]
                coStartY = row[1]
                thetaStart = math.atan2((coStartY - coord1[1]), (coStartX -
coord1[0]))
            for row in arcpy.da.SearchCursor(coEndPt, ["SHAPE@X",
"SHAPE@Y"]):
                coEndX = row[0]
                coEndY = row[1]
                thetaEnd = math.atan2((coord2[1]-coEndY), (coord2[0] -
coEndX))
            theta=(thetaStart+thetaEnd)/2

## Etape 8 : on determine les points de limite des transects
            alpha=theta+90*(math.pi/180) #angles perpendiculaires aux
lignes
            beta = theta - 90 * (math.pi / 180)

            limitX1 = coord1[0] + XSLength * math.cos(alpha)
            limitX2 = coord1[0] + XSLength * math.cos(beta)
            limitY1 = coord1[1] + XSLength * math.sin(alpha)
            limitY2 = coord1[1] + XSLength * math.sin(beta)
```

```
## Etape 9 : on cree les transects a partir des coordonnees des points de
limite
            lineCoord = [[limitX1, limitY1], [limitX2, limitY2]]
            features.append(
                arcpy.Polyline(
                    arcpy.Array([arcpy.Point(*coords) for coords in
lineCoord]),spatial_ref))

arcpy.CopyFeatures_management(features, outXS)

## Etape 10 : on supprime les fichiers temporaires
for i in range(0,len(start)):
    arcpy.Delete_management(outDir+'UEA_{0}.shp'.format(i)) #lignes coupees
    arcpy.Delete_management(outDir+'UEA_Vertex{0}.shp'.format(i)) #lignes
sous formes de vertex
    arcpy.Delete_management(outDir+'UEA_coStart{0}.shp'.format(i)) #points
coStart
    arcpy.Delete_management(outDir+'UEA_coEnd{0}.shp'.format(i)) #points
coEnd

arcpy.AddMessage('Temps d\'execution: {0} secondes'.format(time.time() -
start_time))


arcpy.AddMessage('A faire en sortie du script : retravailler les XS au
niveau des points de confluence')
```

2) Séparation couche d'ELR en fonction des transects (création des UREC)

```
## Ce script permet de separer une couche d'ELR en fonction des UEA avec
des transects perpendiculaires
# a la moyenne des orientations des deux UEA se succedant.
# Une fois séparés, il s'agit de mettre l'information de table d'attributs
# des UEA dans celle des UREC nouvellement modélisées

## Etape 1 : Import des modules + parametres entree et sortie
import arcpy, sys,os, time

arcpy.env.overwriteOutput=True

start_time=time.time() #initiation du temps de calcul (a titre indicatif)

UEA= arcpy.GetParameterAsText(0)
floodPlain=arcpy.GetParameterAsText(1)
XS=arcpy.GetParameterAsText(2)
outPut=arcpy.GetParameterAsText(3)
outDir=os.path.dirname(outPut)+'/'

spatial_ref=arcpy.Describe(UEA).spatialReference
outPoly=outDir+'tmp_Flood_UEA.shp'

## Etape 2 : on coupe la couche d'ELR selon les sections transversales
arcpy.FeatureToPolygon_management([floodPlain,XS],outPoly,attributes="NO_AT
TRIBUTES")
with arcpy.da.UpdateCursor(outPoly,["SHAPE@AREA"]) as cursor:
    for row in cursor:
        if row[0]<500: #On enleve les petits polygones inutiles
            cursor.deleteRow()
```

```
# Etape 2 : on coupe les UEA selon la couche d'ELR pour ne pas qu'il y ai
dans l'etape suivante de MIDpoint qui soit en dehors de la couche d'ELR

outCut=outDir+'UEA_prePt.shp'
arcpy.Clip_analysis(UEA,floodPlain,outCut)


# Etape 3 : On enleve les UEA tres courtes
with arcpy.da.UpdateCursor(outCut,["SHAPE@LENGTH"]) as cursor:
    for row in cursor:
        if row[0]<10:
            cursor.deleteRow()


# Etape 4 : on remplace chaque UEA par son milieu de ligne pour ne pas
avoir les problemes lies au spatial join (que chaque
# UEA ne soit liee qu'a un seul polygone)
UEAMiddlePt = outDir+'UEAMiddlePt.shp'
arcpy.FeatureVerticesToPoints_management(outCut,UEAMiddlePt,"MID")

# Etape 5 : on fait la jointure spatiale sur les points
arcpy.SpatialJoin_analysis(outPoly,UEAMiddlePt,outPut,"JOIN_ONE_TO_ONE","KE
EP_ALL")

# Etape 6 : on elimine les polygones qui ne sont pas associes a un UEA ID
(iles, ...)
with arcpy.da.UpdateCursor(outPut,["ID_UEA"]) as cursor:
    for row in cursor:
        if row[0]==" ":
            cursor.deleteRow()


# Etape 7 : On enleve les couches temporaires
arcpy.Delete_management(outPoly)
arcpy.Delete_management(outCut)
arcpy.Delete_management(UEAMiddlePt)

arcpy.AddMessage('Couche de plaine inondable séparée crée')
arcpy.AddMessage('Temps d\'execution: {0} secondes'.format(time.time() -
start_time))
```

Annexe 6 : Choix d'une méthode de partitionnement pour le développement d'une typologie pour les UEA.

Trois algorithmes de partitionnement de données ont été analysés avant de faire un choix final quant à celui qui serait utilisé pour créer une typologie pour les UEA. Ceux-ci sont: le partitionnement en k-moyennes (MacQueen, 1967), le Partitionnement Autour des Médoïdes (PAM) (Kaufman & Rousseeuw, 1990) et la méthode de partitionnement CLARA *(Clustering for Large Applications)* (Kaufman & Rousseeuw, 1990). Les trois algorithmes sont basés sur le même principe : créer des groupes d'objets tels que la similarité intra-groupe est maximisée et la similarité inter-groupes est minimisée. La différence réside dans la manière de définir les différents groupes. Dans le partitionnement en k-moyennes, chaque groupe est représenté par un centroïde qui correspond à la moyenne des points assignés au groupe (MacQueen, 1967). Pour PAM et CLARA chaque groupe est représenté non plus par un centre fictif qui résulte de la moyenne des points mais par un médoïde comme centre du groupe. Un médoïde étant un des points de l'échantillon de données (Kaufman & Rousseeuw, 1990). Le fait de prendre en compte un des points du groupe pour le représenter rend les méthodes PAM et CLARA moins sensibles au bruit et aux valeurs aberrantes que la méthode des k-moyennes où la moyenne peut être fortement influencée par des valeurs extrêmes. (Gupta et al., 2018). Ceci est visible sur la Figure 35 où sont représentés la moyenne et le médoïde d'un échantillon de données. Le partitionnement en k-moyennes a donc été rejetée pour cette raison.

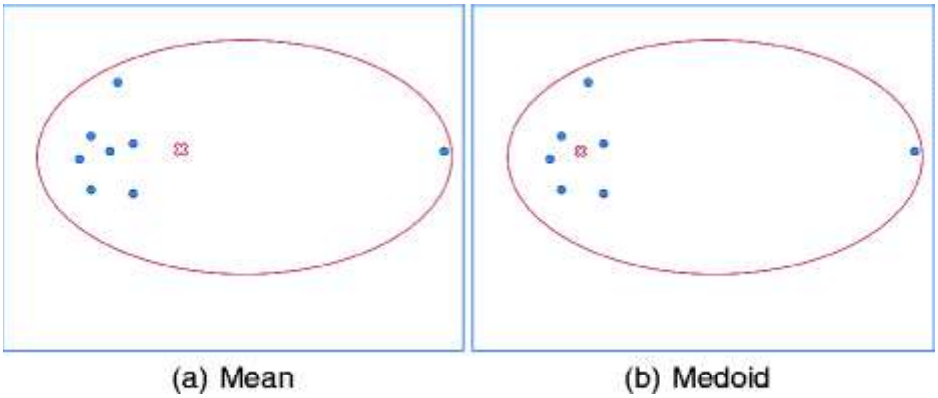

Figure 35 – Représentation d'ensembles de points pour illustrer la différence entre la moyenne et le médoïde d'un échantillon de données (Source : Jin & Han, 2011)

CLARA est une version de PAM optimisée pour les échantillons de données de taille importante en termes de temps de traitement (Kaufman & Rousseeuw, 1990). Celle-ci n'a pas été conservée car elle fournissait une qualité de partitionnement légèrement inférieure à PAM. Ainsi, la méthode prise en compte pour le partitionnement des UEA est la méthode PAM. Pour quantifier, à chaque itération de l'algorithme, la distance entre un point et le médoïde, c'est-à-dire la dissimilarité entre ces deux points, la distance de Manhattan a été considérée. En effet, selon Aggarwal et al. (2001), celle-ci serait plus adaptée, que la distance Euclidienne par exemple, pour des échantillons de données de dimension importante. La formule de Minkowski est la forme généralisée des distances de Manhattan et Euclidienne :

$$D = (\sum_{i=1}^{n} |c_i - d_i|)^{1/p} \qquad (4)$$

c et d sont deux points et p est l'ordre (p=1 pour la distance de Manhattan et p=2 pour la distance Euclidienne). Aggarwal et al. (2001) affirment que plus la dimension de l'échantillon de données est

important, plus une faible valeur de p donne une meilleure discrimination. Ensuite, afin de choisir le nombre de groupes d'UEA à prendre en compte, l'algorithme PAM a été appliqué sur l'échantillon de données avec différents nombre de groupes imposés et la qualité de la classification a été calculée à chaque itération. L'indice pris en compte pour juger de la qualité de partitionnement est le coefficient de silhouette. Sa formule est la suivante pour chaque observation i:

$$S_i = (b_i - a_i)/\max(a_i, b_i) \quad (5)$$

où a_i est la dissimilarité moyenne entre l'observation i et tout les autres points du groupe dans lequel i se trouve et $b_i = \min\ c(i, C)$ où C représente tout les groupes dans lesquels i n'appartient pas, et $c(i, C)$ est la dissimilarité moyenne entre i et toutes les observations de C (Lengyel & Botta-Dukát, 2019). Le coefficient de silhouette moyen sur l'ensemble des observations a donc été considéré. La Figure 36 représente la valeur de coefficient de silhouette moyen en fonction du nombre de groupes considéré. Bien que la meilleure qualité de partitionnement ait été observée pour un nombre de groupes de 2 avec un coefficient de silhouette moyen de 0.37, c'est un nombre de groupes de 3 qui a finalement été pris en compte. La motivation derrière ce choix réside dans la volonté de vouloir favoriser la diversité au sein des analyses à la qualité de représentation. En effet au moins trois groupes d'UEA différents étaient voulus. Un nombre de groupes de 4 donnait une qualité de partitionnement très légèrement supérieure à 3 mais pour cette configuration, l'un des groupes formés possédait un coefficient de silhouette de 0.01 ce qui n'était pas acceptable pour le développement d'une typologie des UEA.

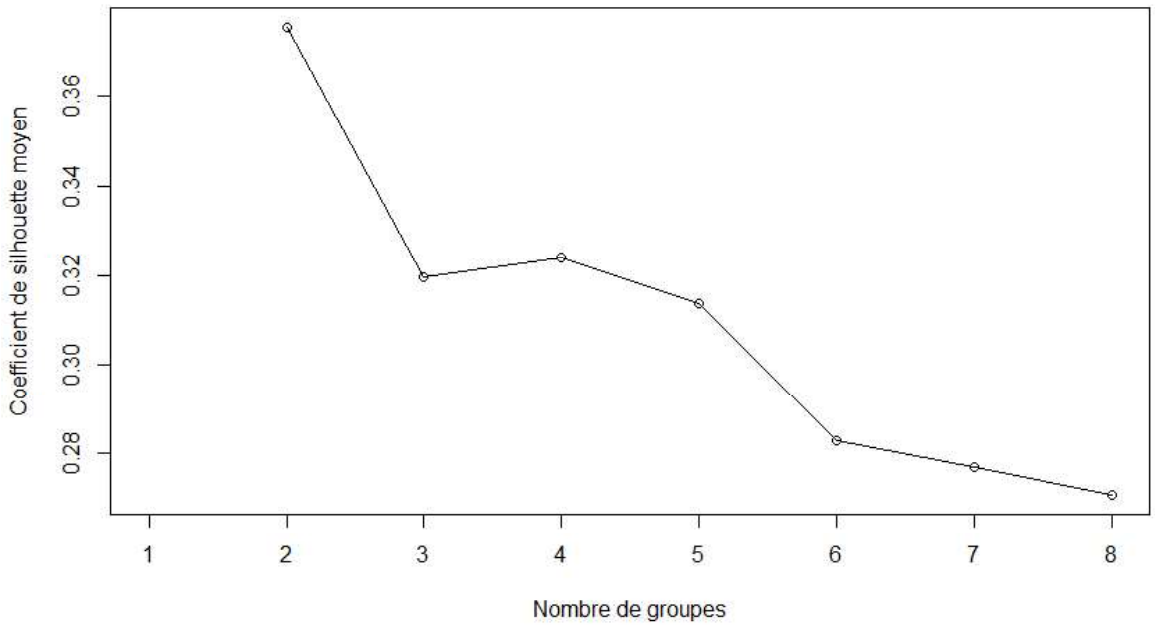

Figure 36 - Représentation graphique du coefficient de silhouette moyen en fonction du nombre de groupes crées lors du processus de partitionnement des Unités Ecologiques Aquatiques (UEA)