APPLICATIONS DE L'ANALYSE FRACTALE À L'IMAGERIE MÉDICALE

CANTALOUBE Théo - N° 35024

Introduction

Imagerie médicale : restituer images du corps humain

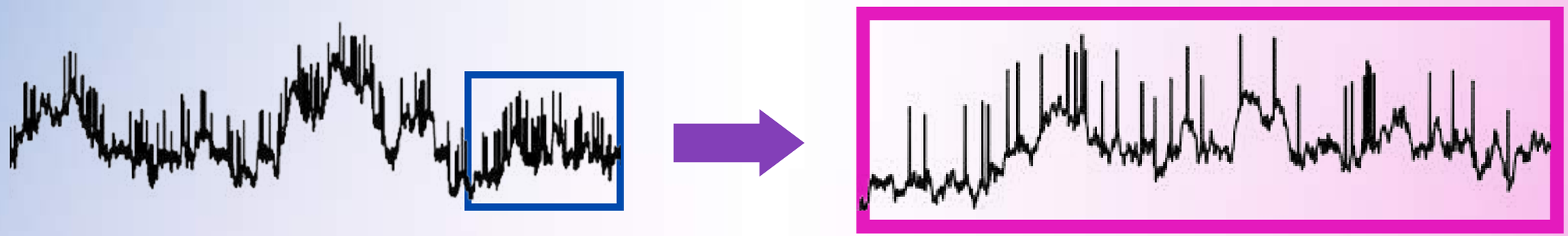
Objectif : analyse de l'image => dépister maladies

Introduction

Imagerie médicale : restituer images du corps humain

Objectif : analyse de l'image => dépister maladies

<u>Caractéristique 1 :</u> Irrégularité à toutes échelles



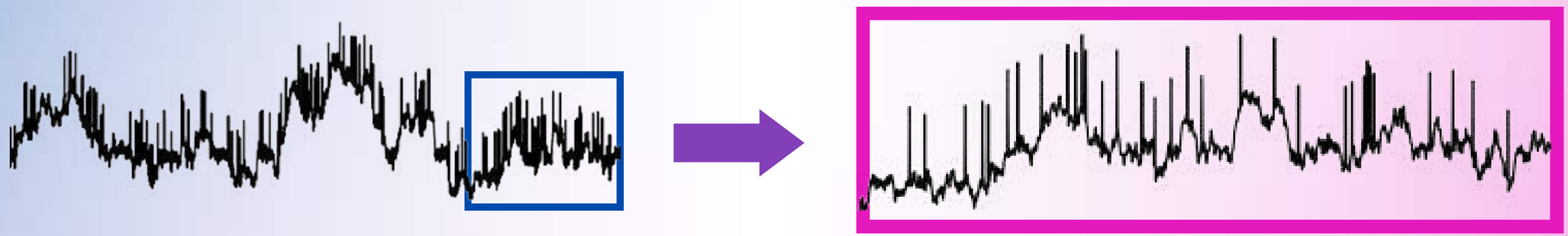
Introduction

Imagerie médicale : restituer images du corps humain

Objectif : analyse de l'image => dépister maladies

Motifs fractaux

<u>Caractéristique 1 :</u> Irrégularité à toutes échelles



Comment exploiter ce phénomène physique ?

Plan

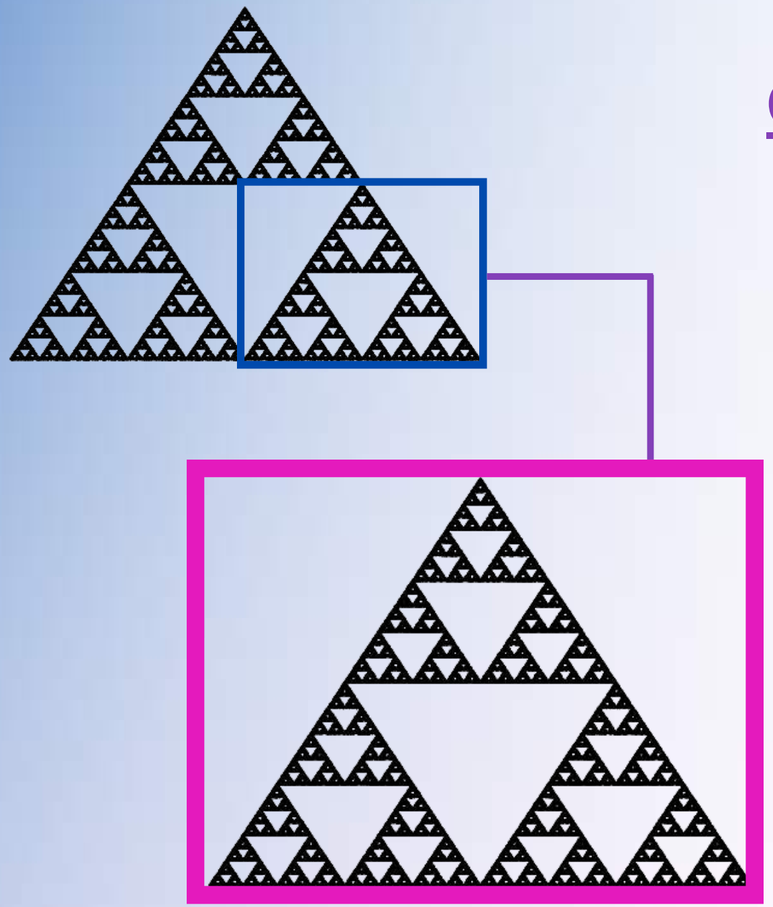
1. Intérêt de l'analyse fractale dans l'imagerie médicale
2. Algorithme de Box-Counting
3. Algorithme des oscillations
4. Algorithmes de déroulement
5. Conclusion

Intérêt de l’analyse fractale dans l’imagerie médicale

Caractéristique 2 :

Auto-similarité

Intérêt de l'analyse fractale dans l'imagerie médicale



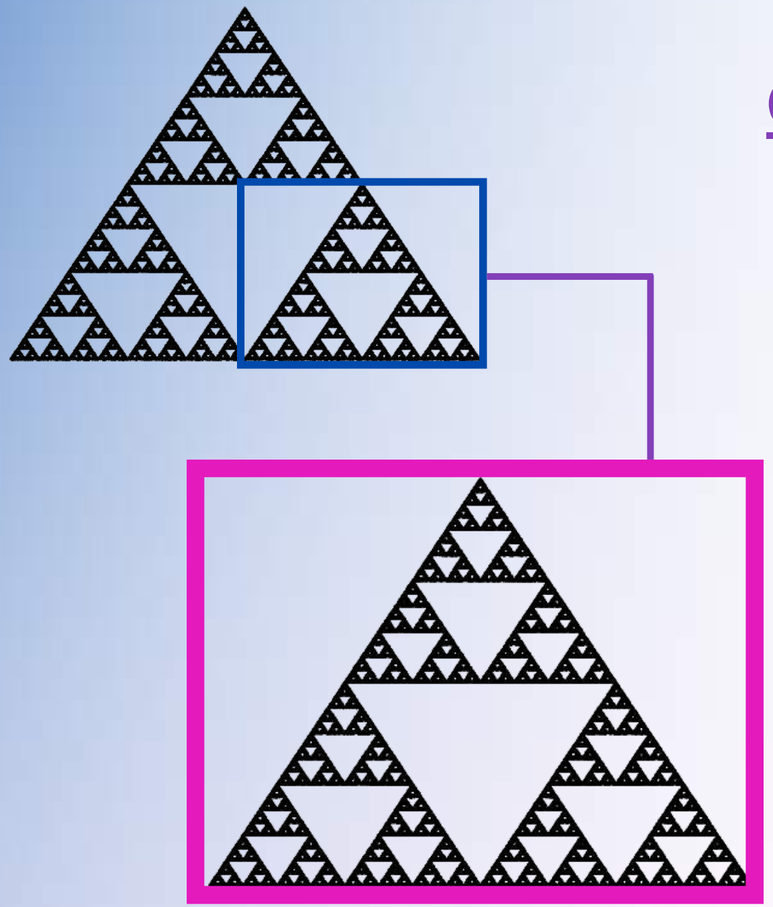
Caractéristique 2 :

Auto-similarité

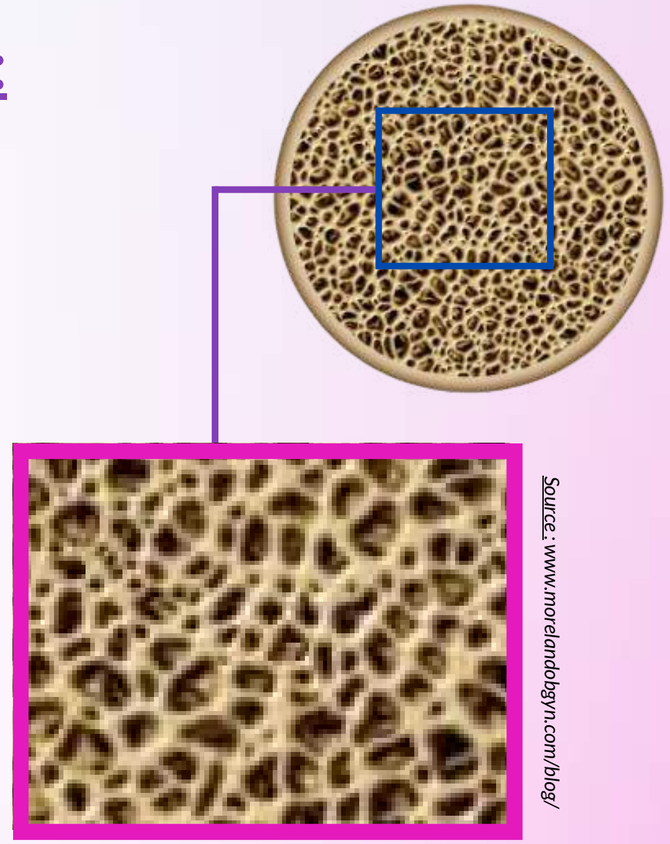
Ex : Flocon de Von Koch

Exacte

Intérêt de l'analyse fractale dans l'imagerie médicale



Caractéristique 2 :
Auto-similarité



Source : www.morelandobgyn.com/blog/

Ex : Flocon de Von Koch

Exacte

Statistique

Ex : Structure trabéculaire de l'os

Intérêt de l'analyse fractale dans l'imagerie médicale

Agrandissement = A

Multiplication = M

Dimension = D

Caractéristique 3 :

Dimension non entière

Intérêt de l'analyse fractale dans l'imagerie médicale

Agrandissement = A

Multiplication = M

Dimension = D

Caractéristique 3 :

Dimension non entière

A = 2
M = 2
D = 1

$$2 = 2^1$$

Intérêt de l'analyse fractale dans l'imagerie médicale

Agrandissement = A

Multiplication = M

Dimension = D

A = 2
M = 2
D = 1

$$2 = 2^1$$

Caractéristique 3 :

Dimension non entière

A = 2
M = 4
D = 2

$$4 = 2^2$$

Intérêt de l'analyse fractale dans l'imagerie médicale

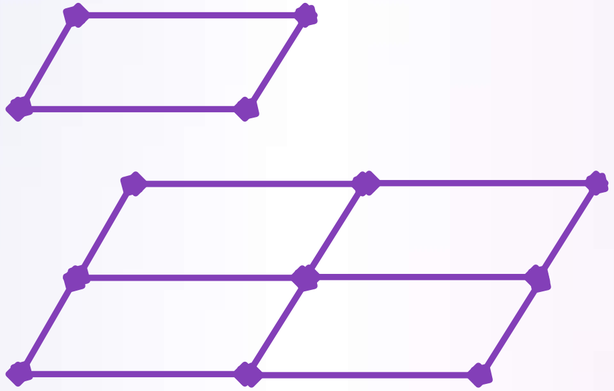
Agrandissement = A

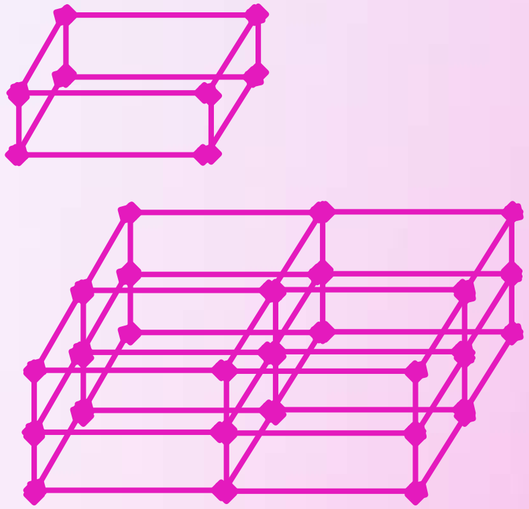
Multiplication = M

Dimension = D

Caractéristique 3 :

Dimension non entière





A = 2
M = 2
D = 1

$$2 = 2^1$$

A = 2
M = 4
D = 2

$$4 = 2^2$$

A = 2
M = 8
D = 3

$$8 = 2^3$$

Intérêt de l'analyse fractale dans l'imagerie médicale

Agrandissement = A

Multiplication = M

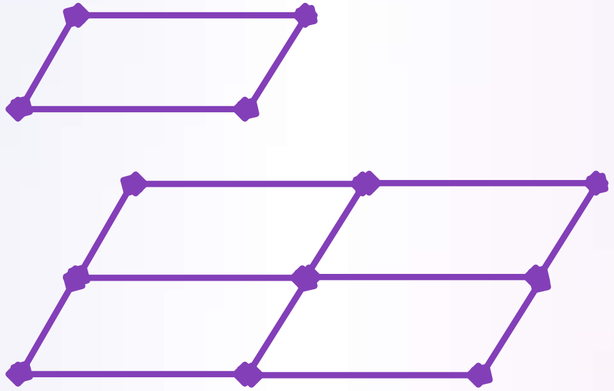
Dimension = D

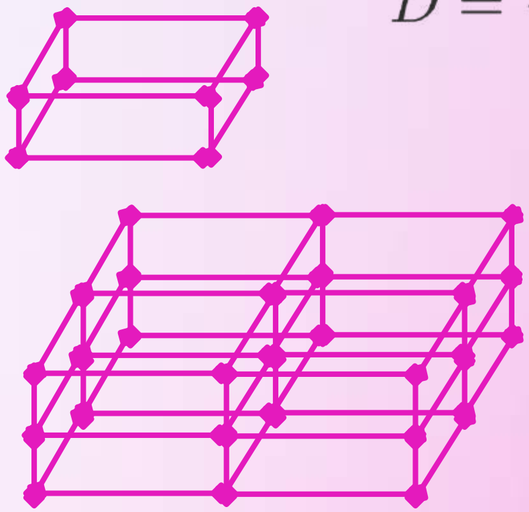
Caractéristique 3 :

Dimension non entière

$$M = A^D$$

$$D = \frac{\ln(M)}{\ln(A)}$$





A = 2
M = 2
D = 1

$$2 = 2^1$$

A = 2
M = 4
D = 2

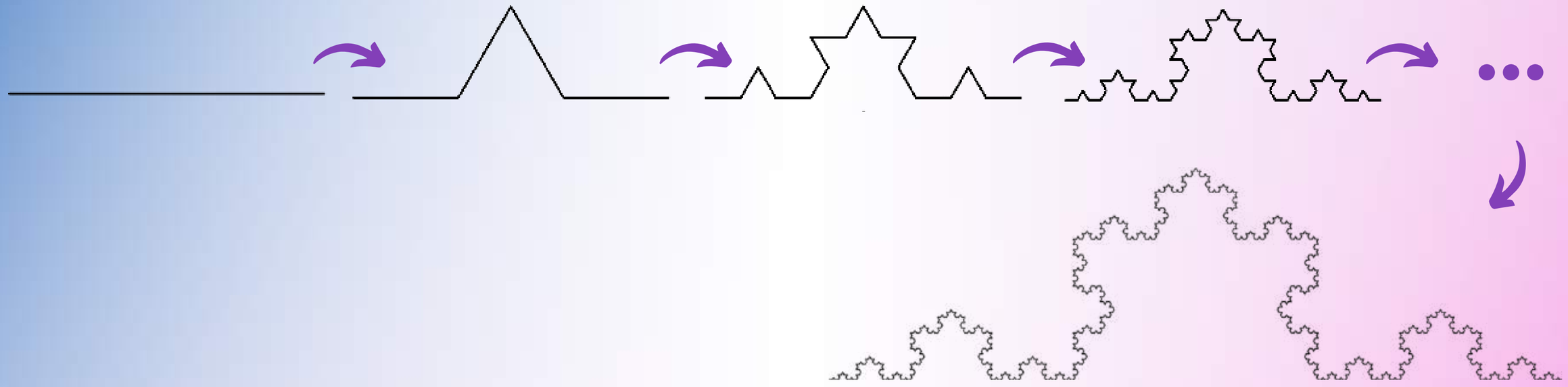
$$4 = 2^2$$

A = 2
M = 8
D = 3

$$8 = 2^3$$

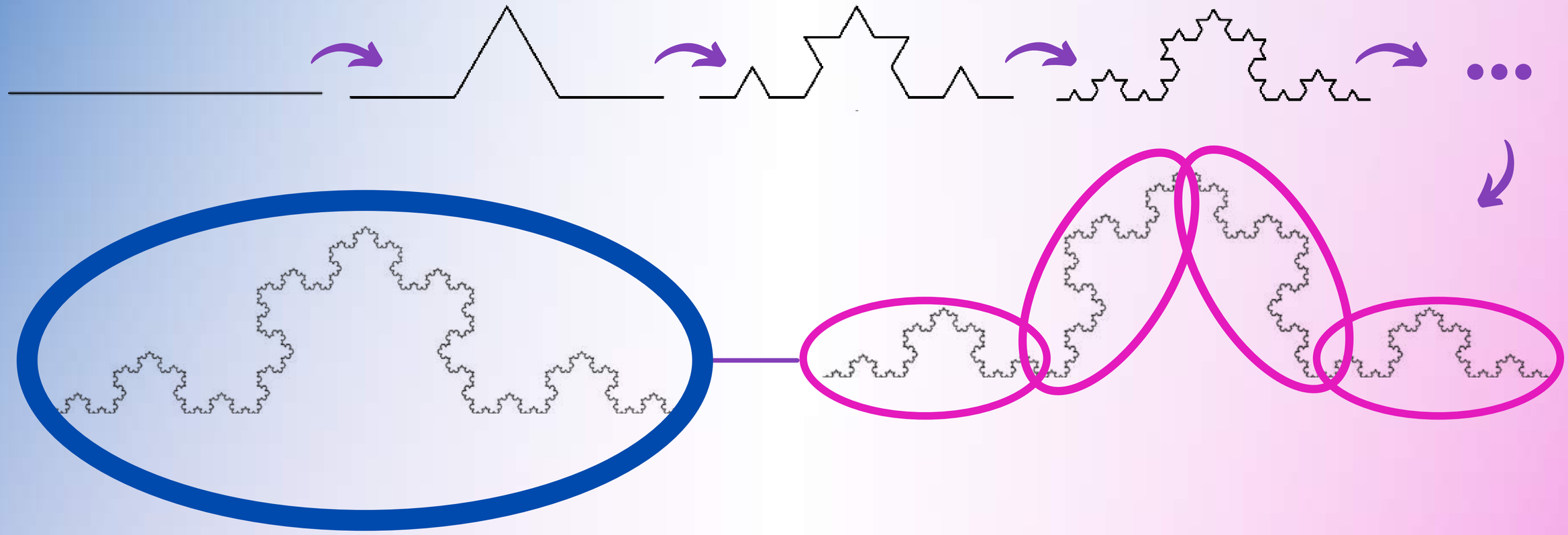
Intérêt de l'analyse fractale dans l'imagerie médicale

Courbe de Von Koch



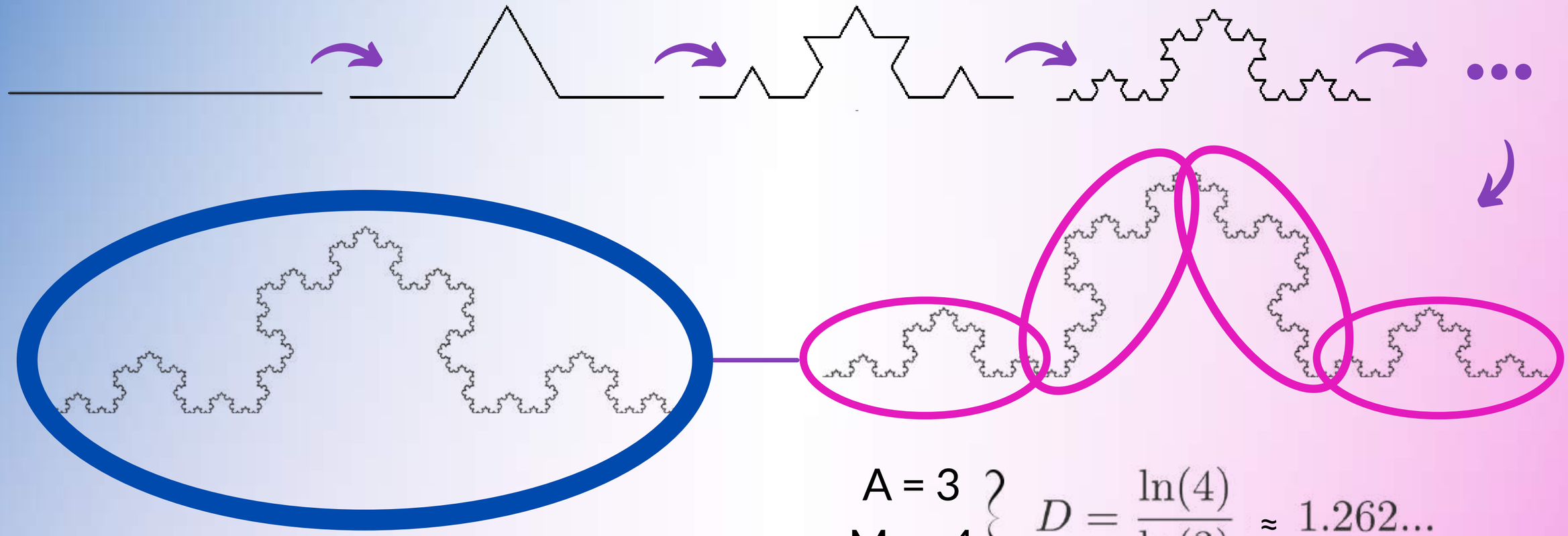
Intérêt de l'analyse fractale dans l'imagerie médicale

Courbe de Von Koch



Intérêt de l'analyse fractale dans l'imagerie médicale

Courbe de Von Koch



A = 3
M = 4

$$D = \frac{\ln(4)}{\ln(3)} \approx 1.262...$$

Intérêt de l'analyse fractale dans l'imagerie médicale

Cas des fractales-plan :

$$1 < d_{fractale} < 2$$

Qualitativement :

↗↗ Irrégularité de la fractale

↓

↗↗ Valeur dimension fractale

Intérêt de l'analyse fractale dans l'imagerie médicale

Cas des fractales-plan :

$$1 < d_{fractale} < 2$$

Qualitativement :

Irrégularité de la fractale

Valeur dimension fractale

En général :

Échantillons malades présentent motifs plus désordonnés

Os normal

Ostéoporose

Ex : Structure trabéculaire de l'os

Intérêt de l'analyse fractale dans l'imagerie médicale

Cas des fractales-plan :

$$1 < d_{fractale} < 2$$

Qualitativement :

Irrégularité de la fractale

Valeur dimension fractale

En général :

Échantillons malades présentent motifs plus désordonnés

Dépister maladie = Comparer dimensions fractales issues de l'analyse du patient et du témoin sain

Os normal

Ostéoporose

Ex : Structure trabéculaire de l'os

Intérêt de l'analyse fractale dans l'imagerie médicale

- Repérer des tumeurs (cancer du sein ou de la prostate)

Intérêt de l’analyse fractale dans l’imagerie médicale

- Repérer des tumeurs (cancer du sein ou de la prostate)
- Dépister l'ostéoporose (analyse de l'architecture trabéculaire de l'os)

Intérêt de l'analyse fractale dans l'imagerie médicale

- Repérer des tumeurs (cancer du sein ou de la prostate)
- Dépister l'ostéoporose (analyse de l'architecture trabéculaire de l'os)
- Détecter l'épilepsie (analyse des tissus cérébraux)

Comment mesurer la dimension fractale ?

Dimension de Minkowski-Bouligand

Dimension de :

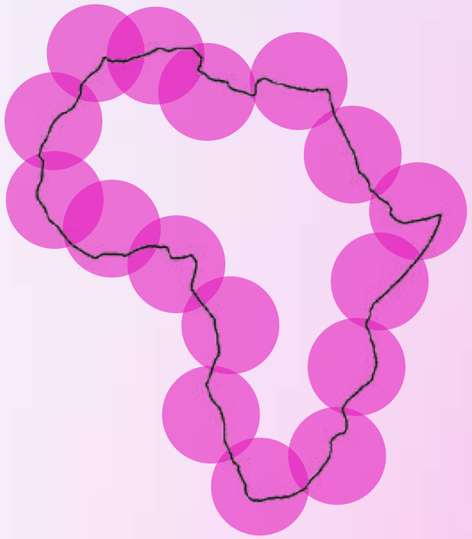
- Hausdorff
- homothétie
- Minkowski-Bouligand /"Box-counting"

Dimension de Minkowski-Bouligand

Dimension de :

- Hausdorff
- homothétie
- Minkowski-Bouligand /"Box-counting"

$$dim(f) = \lim_{\varepsilon \to 0} \frac{\ln(N(\varepsilon))}{\ln(\frac{1}{\varepsilon})}$$

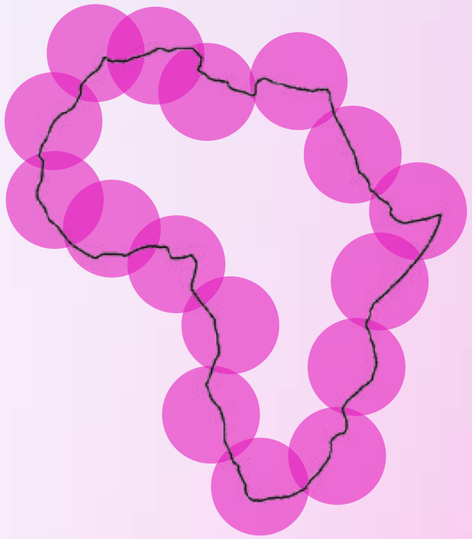


Dimension de Minkowski-Bouligand

Dimension de :

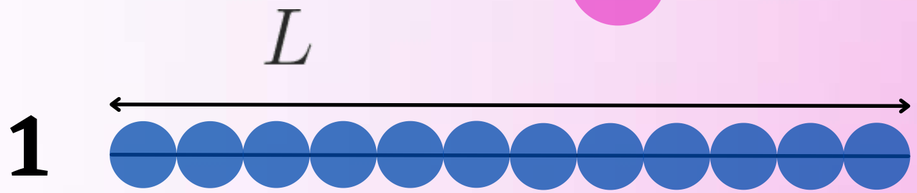
- Hausdorff
- homothétie
- Minkowski-Bouligand /"Box-counting"

$$dim(f) = \lim_{\varepsilon \to 0} \frac{\ln(N(\varepsilon))}{\ln(\frac{1}{\varepsilon})}$$



1 $N(\varepsilon) \approx \frac{L}{\varepsilon} \Rightarrow \ln(N(\varepsilon)) \approx \ln(L) - \ln(\varepsilon) \approx \ln(\frac{1}{\varepsilon})$

$$\text{donc } \frac{\ln(N(\varepsilon))}{\ln(\frac{1}{\varepsilon})} \underset{0}{\sim} \frac{\ln(\frac{1}{\varepsilon})}{\ln(\frac{1}{\varepsilon})} = 1 \text{ donc } \dim(f) = 1$$

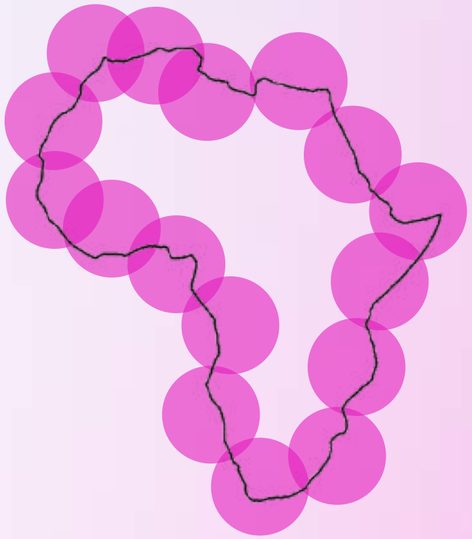


Dimension de Minkowski-Bouligand

Dimension de :

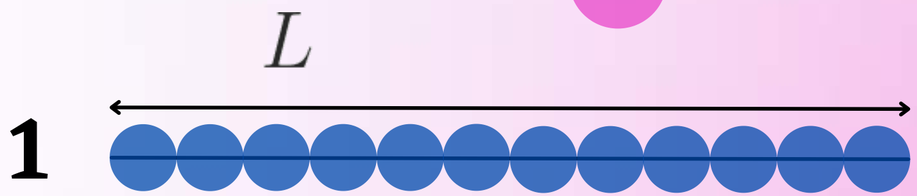
- Hausdorff
- homothétie
- Minkowski-Bouligand /"Box-counting"

$$dim(f) = \lim_{\varepsilon \to 0} \frac{\ln(N(\varepsilon))}{\ln(\frac{1}{\varepsilon})}$$



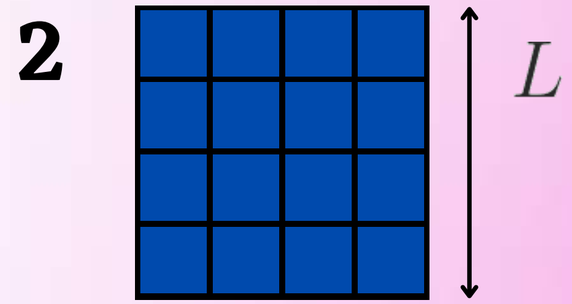
1 $N(\varepsilon) \approx \frac{L}{\varepsilon} \Rightarrow \ln(N(\varepsilon)) \approx \ln(L) - \ln(\varepsilon) \approx \ln(\frac{1}{\varepsilon})$

$$\text{donc } \frac{\ln(N(\varepsilon))}{\ln(\frac{1}{\varepsilon})} \underset{0}{\sim} \frac{\ln(\frac{1}{\varepsilon})}{\ln(\frac{1}{\varepsilon})} = 1 \text{ donc } \dim(f) = 1$$

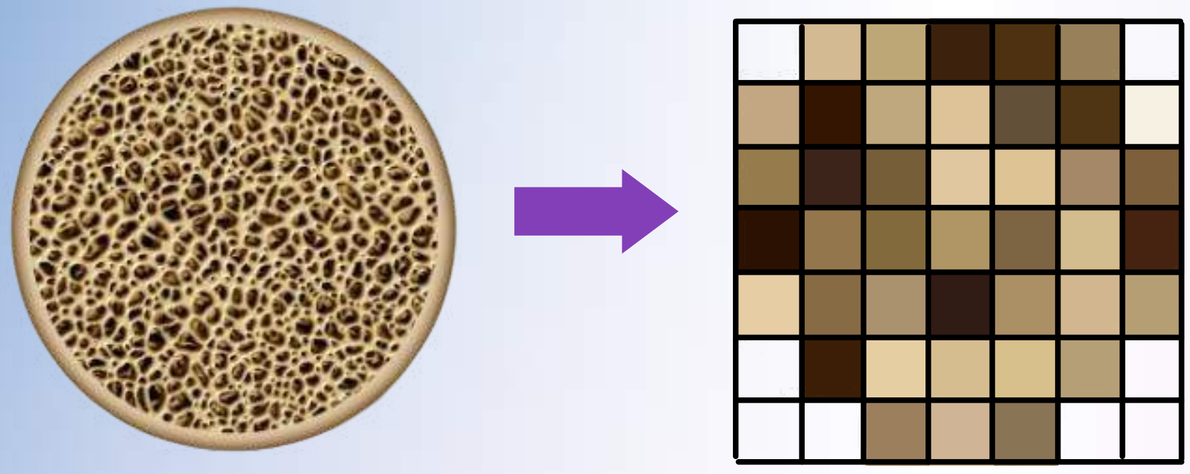


2 $N(\varepsilon) \approx \frac{L^2}{\varepsilon^2} \Rightarrow \ln(N(\varepsilon)) \approx 2 \times (\ln(L) - \ln(\varepsilon)) \approx 2 \times \ln(\frac{1}{\varepsilon})$

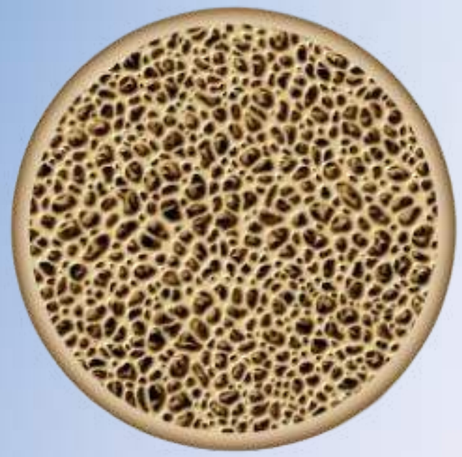
$$\text{donc } \frac{\ln(N(\varepsilon))}{\ln(\frac{1}{\varepsilon})} \underset{0}{\sim} \frac{2 \times \ln(\frac{1}{\varepsilon})}{\ln(\frac{1}{\varepsilon})} = 2 \text{ donc } \dim(f) = 2$$

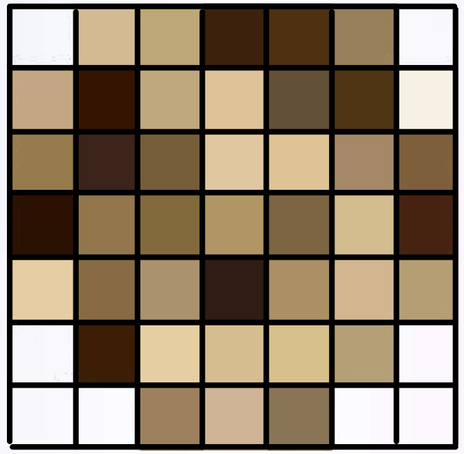


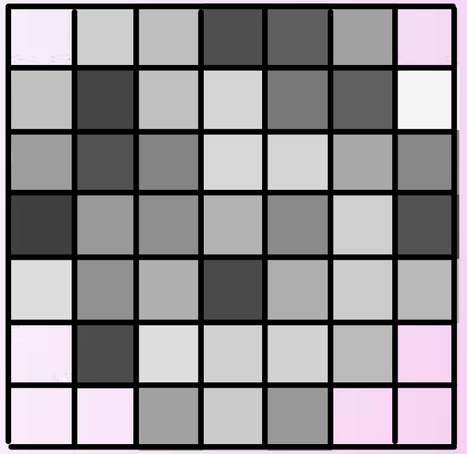
Algorithme de Box-Counting



Algorithme de Box-Counting

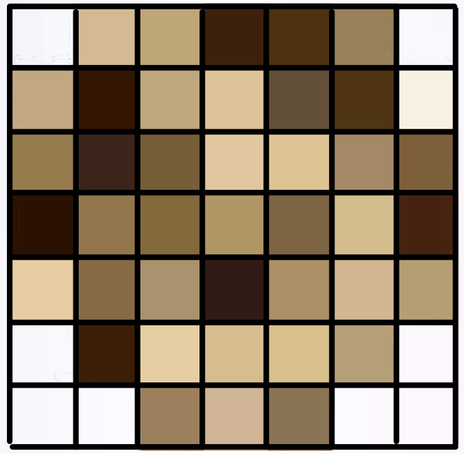


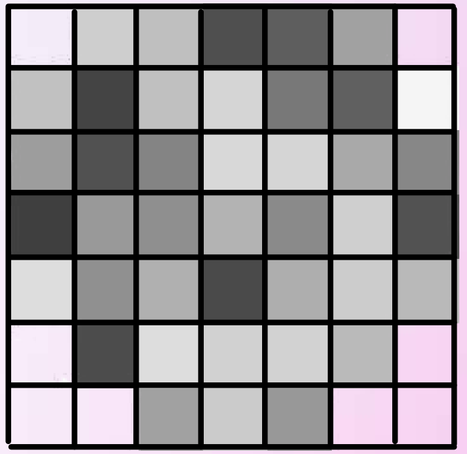




255	206	191	79	94	161	255
193	68	192	213	120	96	245
157	81	132	216	213	169	135
63	153	143	179	138	207	82
221	144	176	74	174	204	185
255	76	221	209	210	186	255
255	255	161	203	152	255	255

Algorithme de Box-Counting



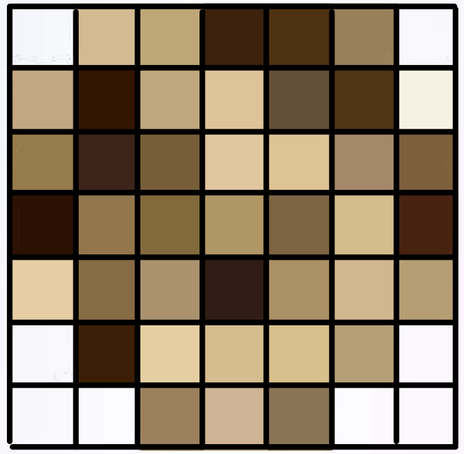


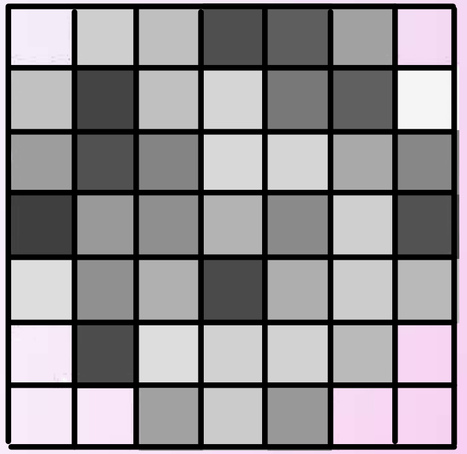
255	206	191	79	94	161	255
193	68	192	213	120	96	245
157	81	132	216	213	169	135
63	153	143	179	138	207	82
221	144	176	74	174	204	185
255	76	221	209	210	186	255
255	255	161	203	152	255	255

$\varepsilon = 2$

0.73	0.53	0.26	0.04
0.39	0.33	0.29	0.21
0.70	0.58	0.14	0.27
0	0.16	0.40	0

Algorithme de Box-Counting





255	206	191	79	94	161	255
193	68	192	213	120	96	245
157	81	132	216	213	169	135
63	153	143	179	138	207	82
221	144	176	74	174	204	185
255	76	221	209	210	186	255
255	255	161	203	152	255	255

Ɛ = 2

0.73	0.53	0.26	0.04
0.39	0.33	0.29	0.21
0.70	0.58	0.14	0.27
0	0.16	0.40	0

$$\varepsilon = 2 \Rightarrow N(\varepsilon) = 5.03 \Rightarrow$$

$$(\ln(\frac{1}{\varepsilon}), \ln(N(\varepsilon))) \simeq (-0.69, 1.62)$$

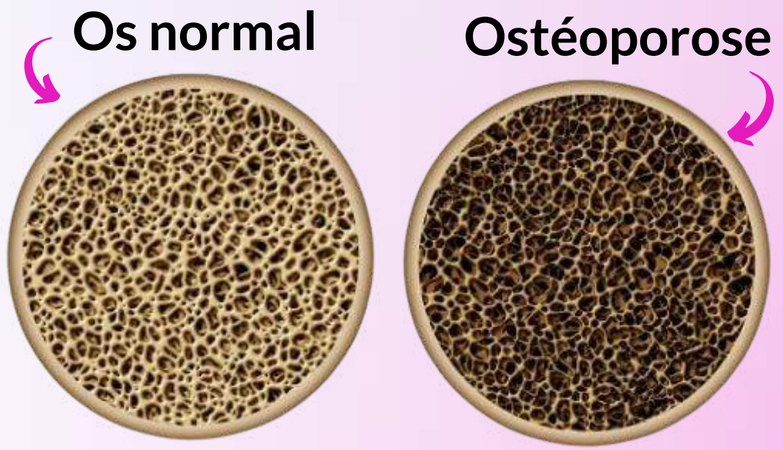
On recommence pour chaque Ɛ

$\Rightarrow$ Régression linéaire

$$dim(f) = \lim_{\varepsilon \to 0} \frac{\ln(N(\varepsilon))}{\ln(\frac{1}{\varepsilon})}$$

Résultats

Ex : Structure trabéculaire de l'os



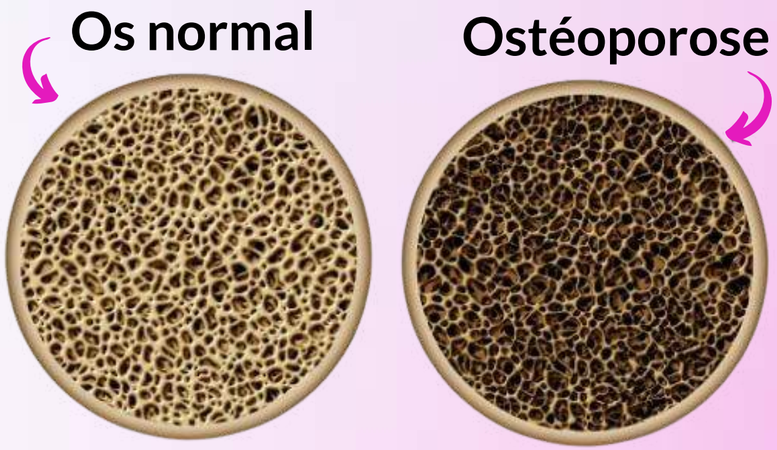
d = 1.234 d = 1.452

Résultats

d = 1.356 d = 1.388

Ex :
Vascularisation reinale normale puis alimentant une tumeur cancéreuse

Ex : Structure trabéculaire de l'os



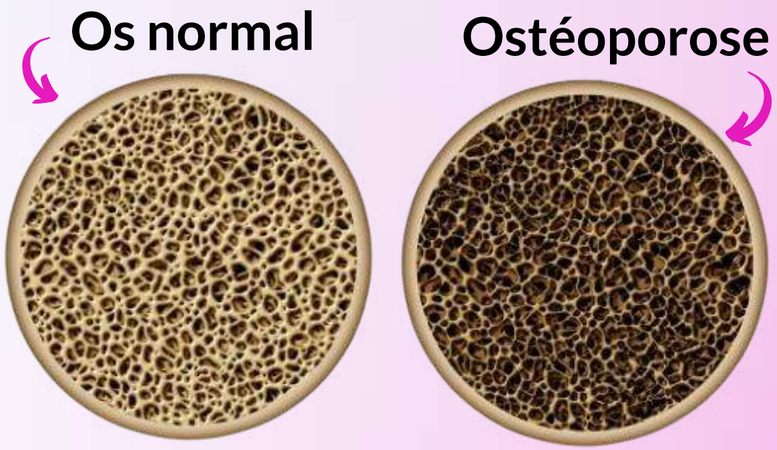
d = 1.234 d = 1.452

Résultats

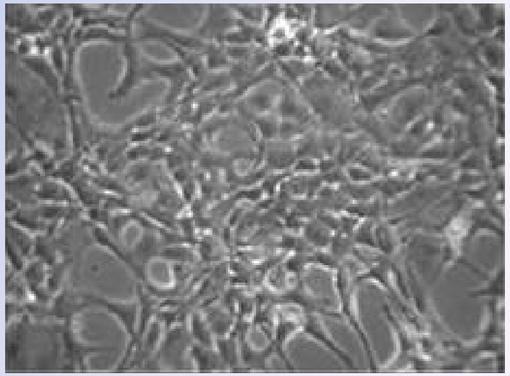
d = 1.356 d = 1.388

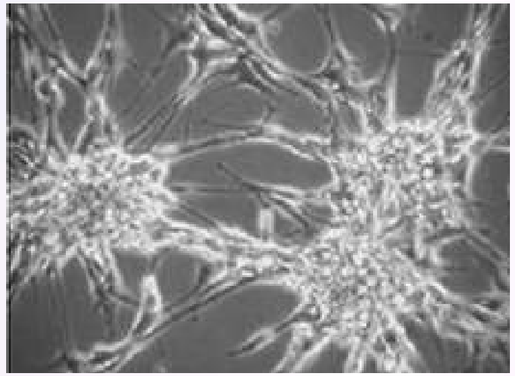
Ex :
Vascularisation reinale normale puis alimentant une tumeur cancéreuse

Ex : Structure trabéculaire de l'os



d = 1.234 d = 1.452





Source : National Cancer institute

d = 1.682 d = 1.798

Ex : Cellule normale puis cellule cancéreuse

Dimension

Algorithme de Box-Counting

$$dim(f) = \lim_{\varepsilon \to 0} \frac{\ln(N(\varepsilon))}{\ln(\frac{1}{\varepsilon})}$$

Dimension

Algorithme de
Box-Counting

$$dim(f) = \lim_{\varepsilon \to 0} \frac{\ln(N(\varepsilon))}{\ln(\frac{1}{\varepsilon})}$$

Algorithme des
oscillations

Dimension

$R(\varepsilon)$: Somme des aires/volumes des boules de taille ε

d : dimension (1, 2, 3)

Algorithme de Box-Counting

$$dim(f) = \lim_{\varepsilon \to 0} \frac{\ln(N(\varepsilon))}{\ln(\frac{1}{\varepsilon})}$$

$$N(\varepsilon) \underset{+\,0}{\sim} k \times \frac{R(\varepsilon)}{\varepsilon^d} \text{ or } \lim_{\varepsilon \to 0} N(\varepsilon) = +\infty$$

$$\Rightarrow \ln(N(\varepsilon)) \underset{+\,0}{\sim} \ln(k \times \frac{R(\varepsilon)}{\varepsilon^d}) = \ln(k) + \ln(\frac{R(\varepsilon)}{\varepsilon^d})$$

$$\underset{+\,0}{\sim} \ln(\frac{R(\varepsilon)}{\varepsilon^d}) = \ln(R(\varepsilon)) - d \times \ln(\varepsilon)$$

$$\Rightarrow \frac{\ln(N(\varepsilon))}{ln(\frac{1}{\varepsilon})} \underset{+\,0}{\sim} \frac{\ln(R(\varepsilon)) - d \times \ln(\varepsilon)}{ln(\frac{1}{\varepsilon})} \underset{+\,0}{\sim} d - \frac{\ln(R(\varepsilon))}{ln(\varepsilon)}$$

Algorithme des oscillations

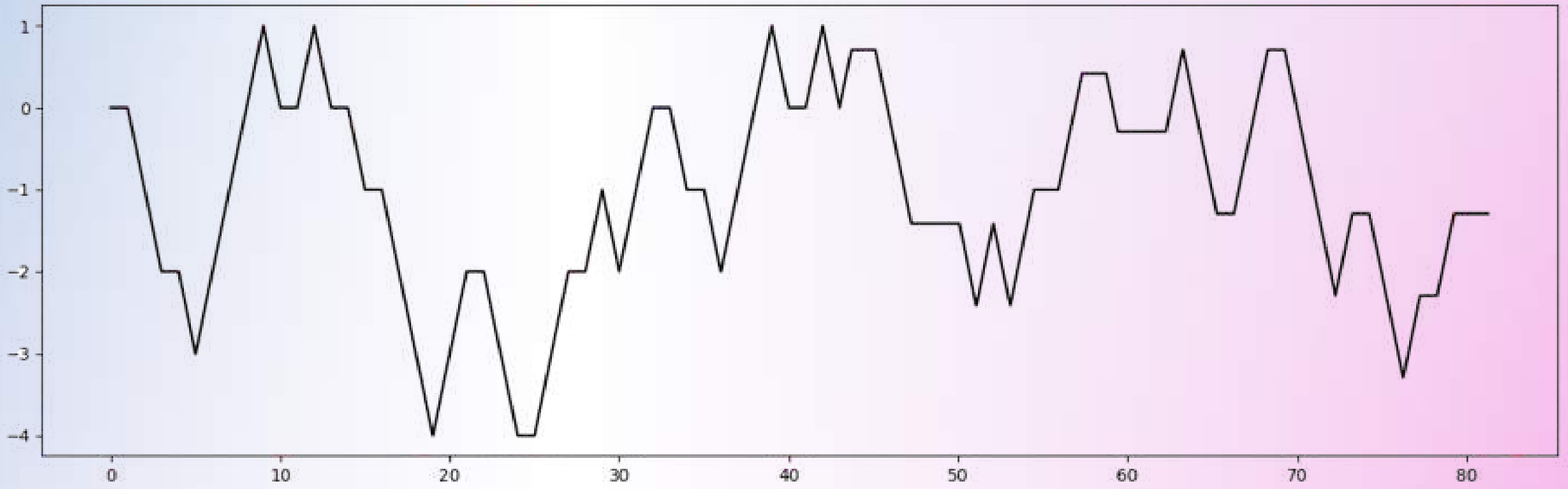
$$dim(f) = \lim_{\varepsilon \to 0}(d - \frac{\ln(R(\varepsilon))}{ln(\varepsilon)})$$

Algorithme des oscillations

Algorithme des oscillations

Fonction continue

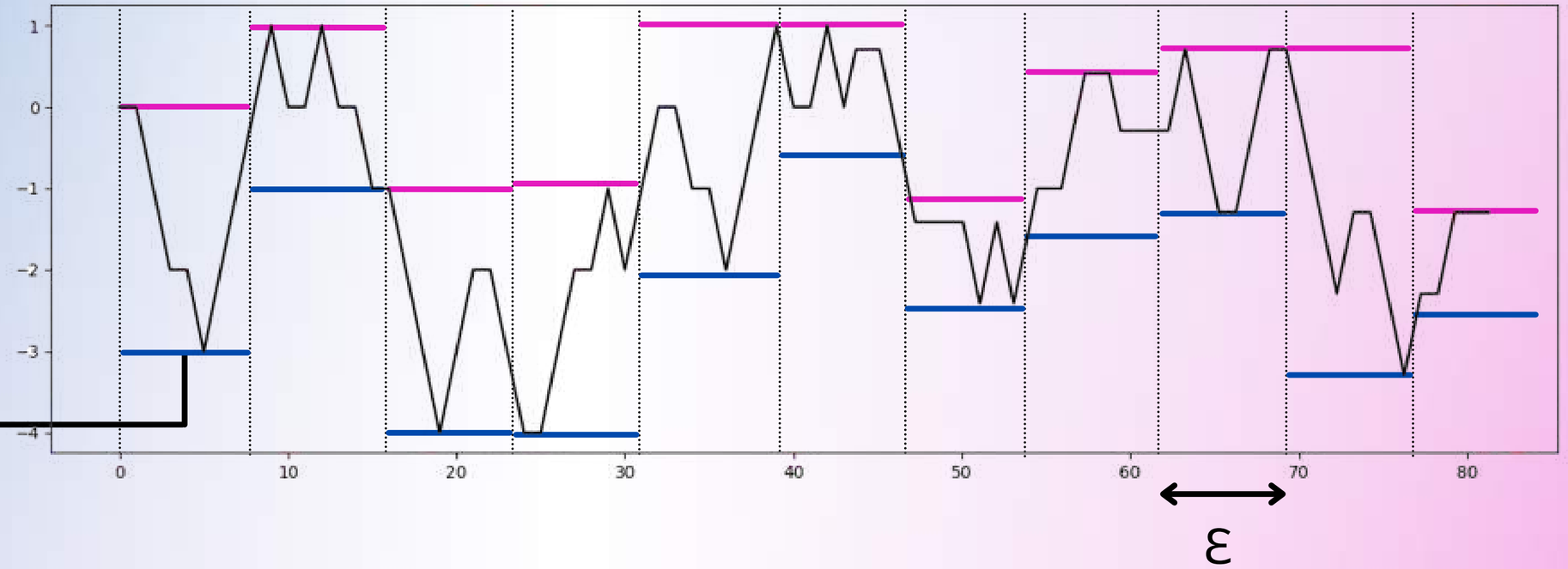
Image en dimension 1



Algorithme des oscillations

Fonction continue

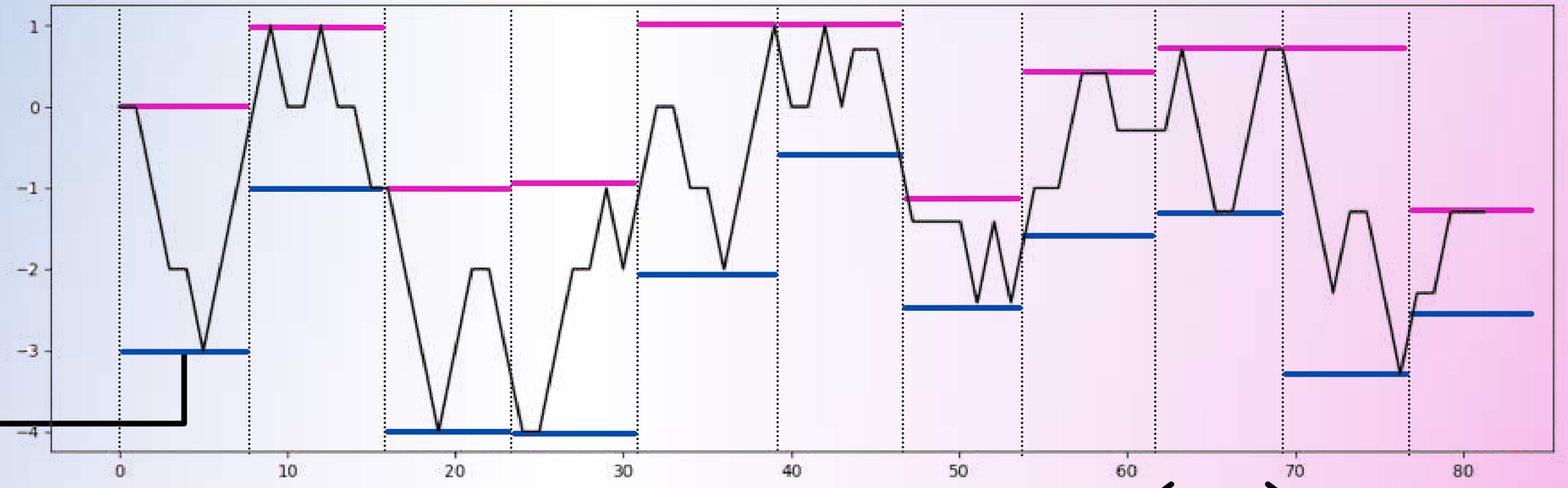
Image en dimension 1

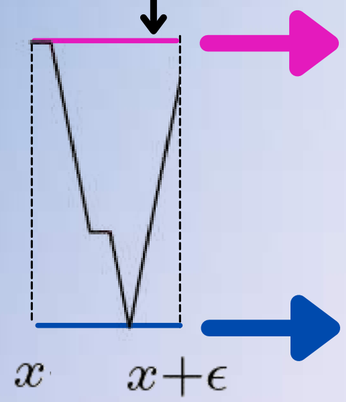


Algorithme des oscillations

Fonction continue

Image en dimension 1



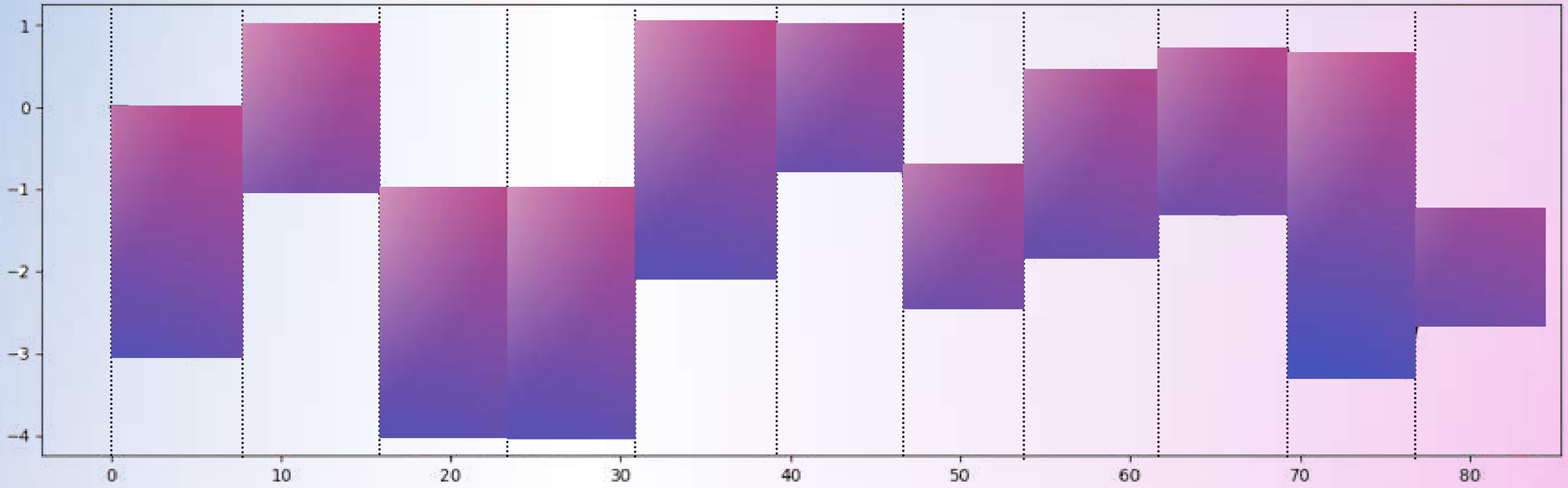


$$\max_{t \in \sqsubset x, x+\epsilon \sqsubset} f(t)$$

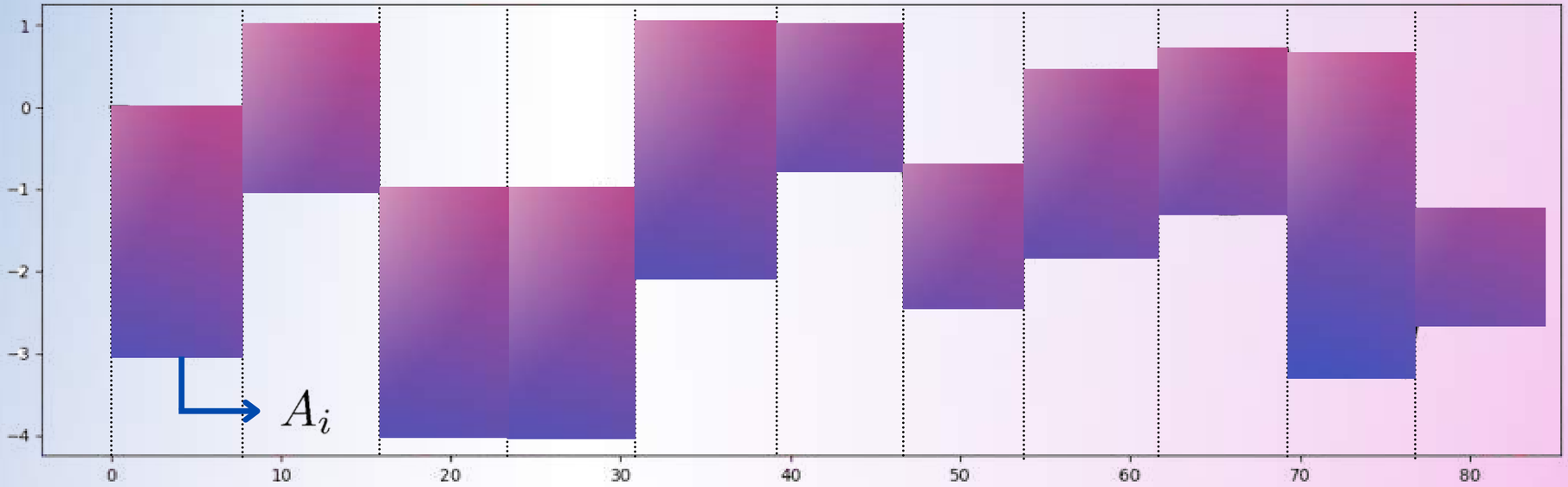
$$\min_{t \in \sqsubset x, x+\epsilon \sqsubset} f(t)$$

$$OSC(f, \epsilon, x) = \max_{t \in \sqsubset x, x+\epsilon \sqsubset} f(t) - \min_{t \in \sqsubset x, x+\epsilon \sqsubset} f(t)$$

Somme des aires des rectangles



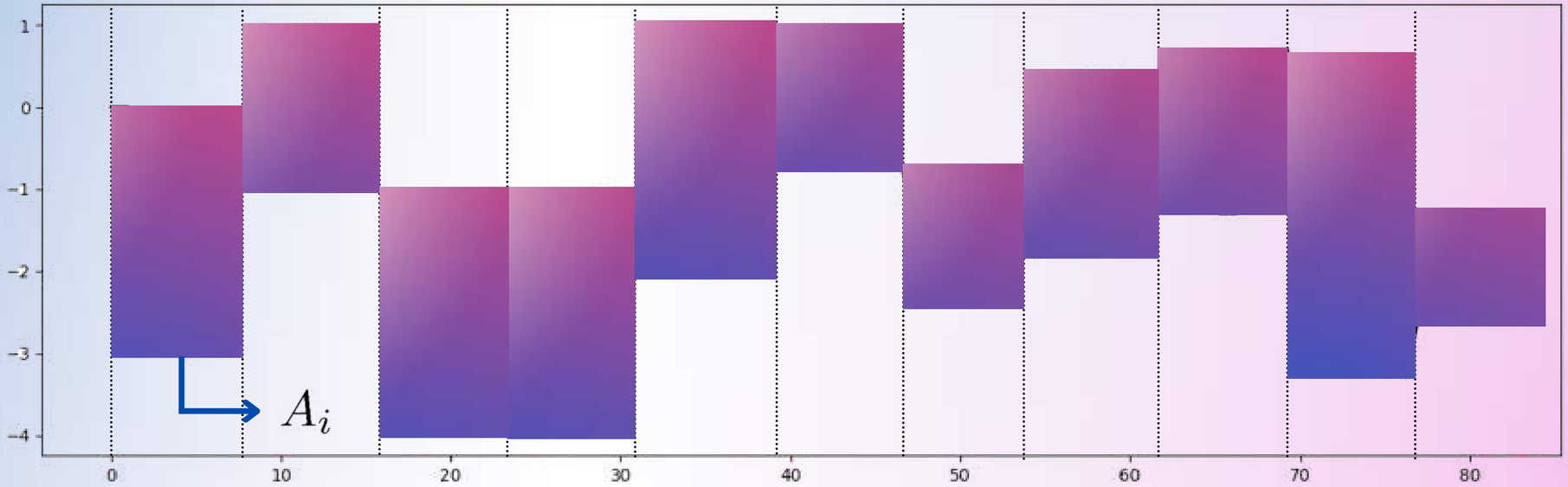
Somme des aires des rectangles



$$Var(f,\epsilon) = \sum_{i=0}^{n} A_i = \sum_{i=0}^{n} OSC(f,\epsilon,xi)\epsilon = \epsilon \sum_{i=0}^{n} OSC(f,\epsilon,xi)$$

$$dim(f) = \lim_{\epsilon \to 0}(2 - \frac{\ln Var(f,\epsilon)}{\ln \epsilon})$$

Somme des aires des rectangles



$$Var(f,\epsilon) = \sum_{i=0}^{n} A_i = \sum_{i=0}^{n} OSC(f,\epsilon,xi)\epsilon = \epsilon \sum_{i=0}^{n} OSC(f,\epsilon,xi)$$

$$dim(f) = \lim_{\epsilon \to 0}(2 - \frac{\ln Var(f,\epsilon)}{\ln \epsilon})$$

Dimension fractale trouvée ≈ 1,15

Restriction :

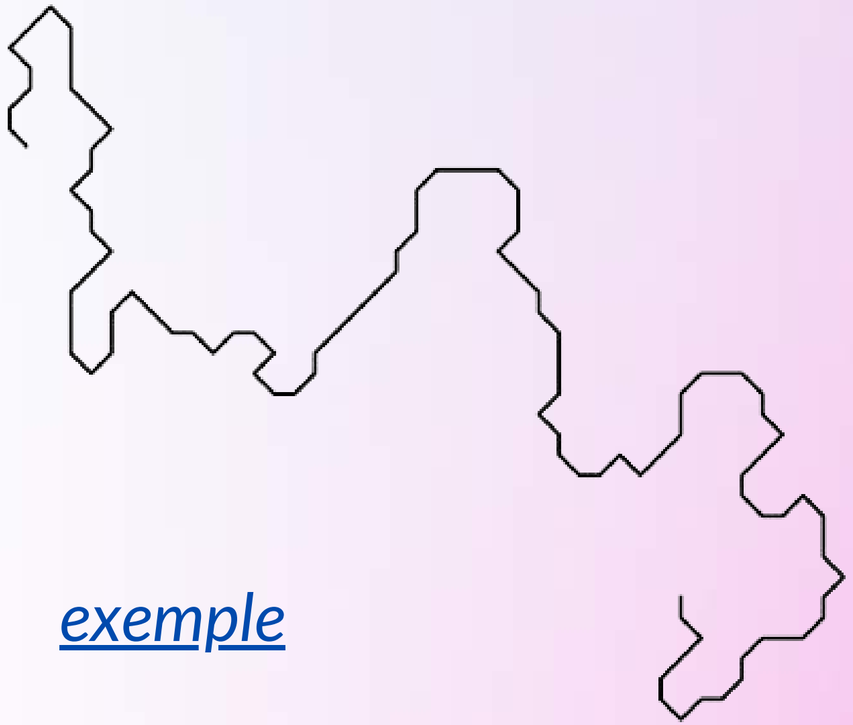
Algorithme des oscillations pour images en 1D

Restriction :

Algorithme des oscillations pour images en 1D

↓

Traiter les images en 2D ?



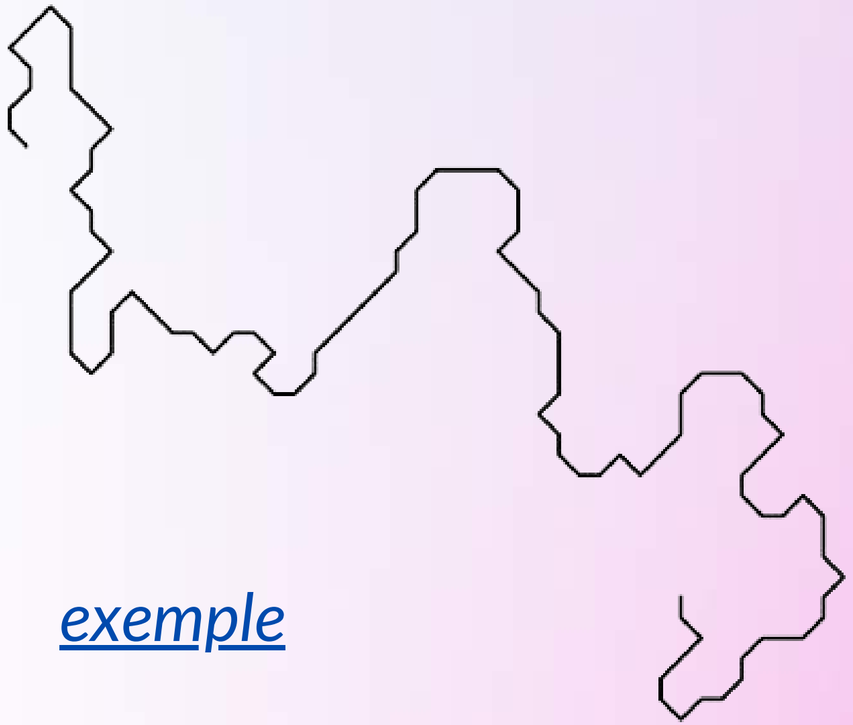
exemple

Restriction :

Algorithme des oscillations pour images en 1D

↓

Traiter les images en 2D ?



exemple

↓

Déroulement

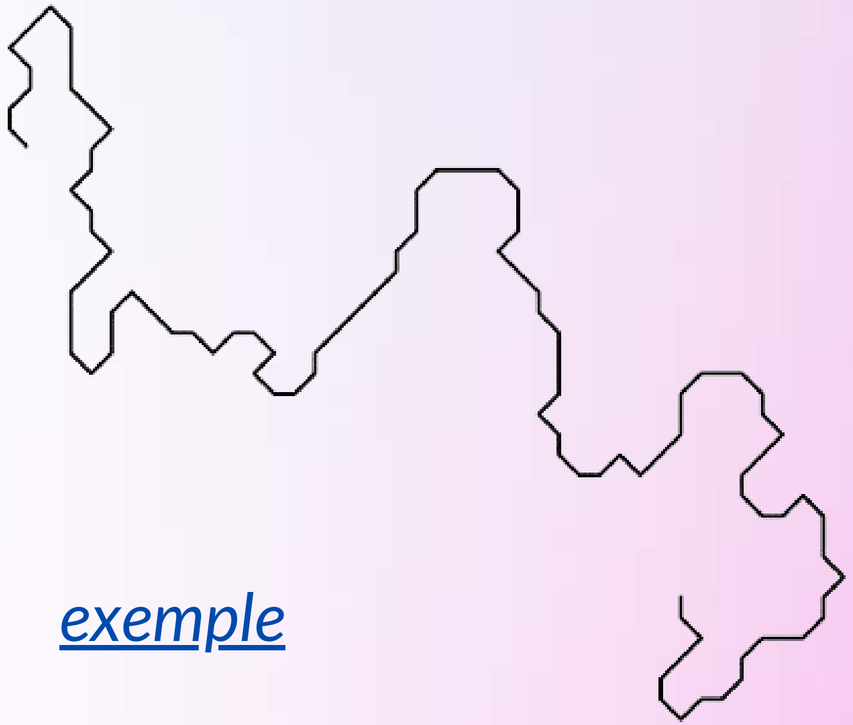
Restriction :

Algorithme des oscillations pour images en 1D

↓

Traiter les images en 2D ?

Problème : Impossible de tout traiter



exemple

Déroulement

Création de contours aléatoires

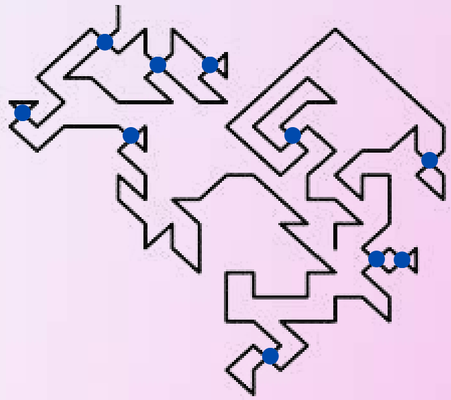
Conditions préalables :

Création de contours aléatoires

Conditions préalables :

1. Pas d'auto-intersection

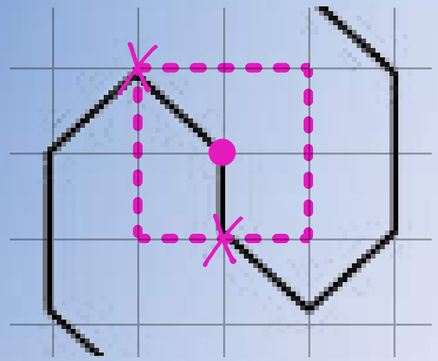
Intersections



Création de contours aléatoires

Conditions préalables :

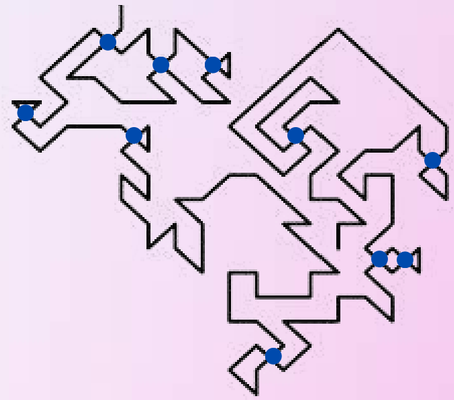
1. Pas d'auto-intersection



● X

Point et voisins associés

● *Intersections*

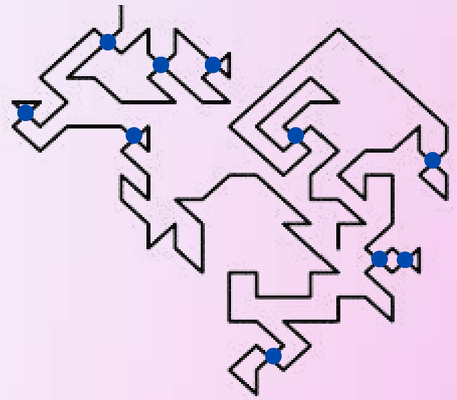


2. Tout point possède exactement deux voisins

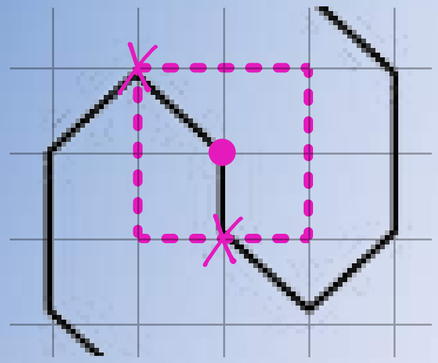
Création de contours aléatoires

Conditions préalables :

1. Pas d'auto-intersection



Intersections



● X

Point et voisins associés

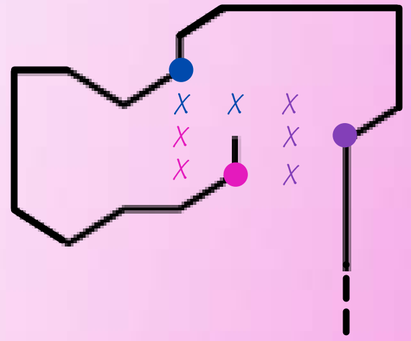
2. Tout point possède exactement deux voisins

3. Si le contour ne peut avancer

=> fin

X ●

Points et voisins associés innateignables



Algorithme de déroulement

Objectif : Appliquer localement algo des oscillations à une succession de morceaux en 1D du contour

Algorithme de déroulement

Objectif : Appliquer localement algo des oscillations à une succession de morceaux en 1D du contour

3 étapes :

Découpage

+

Rotation

+

Translation

Algorithme de déroulement

Objectif :

Appliquer localement algo des oscillations à une succession de morceaux en 1D du contour

3 étapes :

Découpage

+

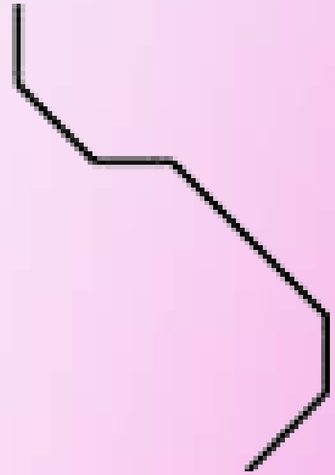
Rotation

+

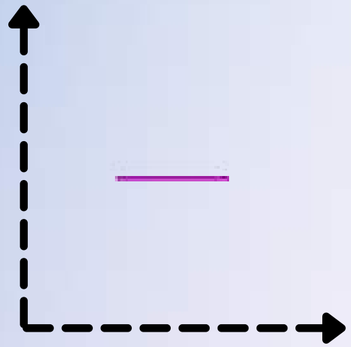
Translation

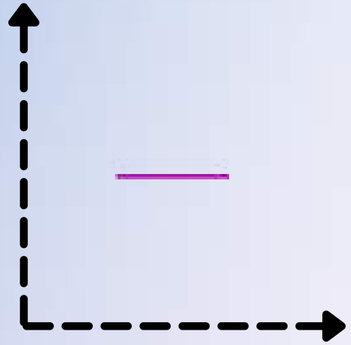
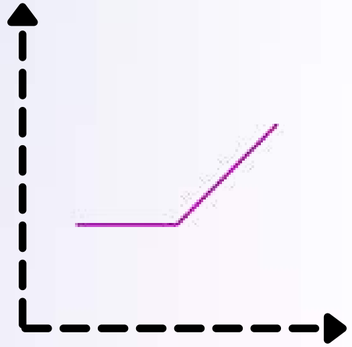
Contour "cartésien" :

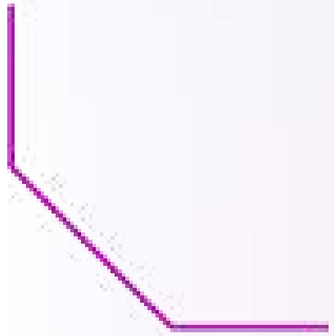
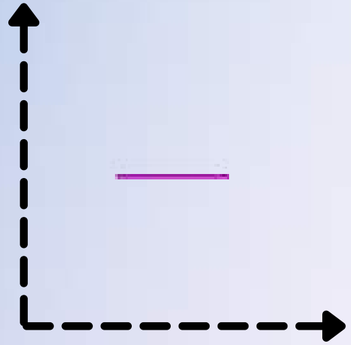
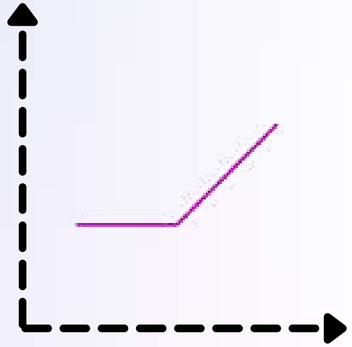
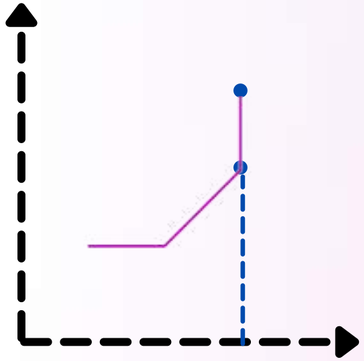
Il existe une orientation (horizontale / verticale) telle que dans un repère cartésien, aucune abscisse n'a deux images

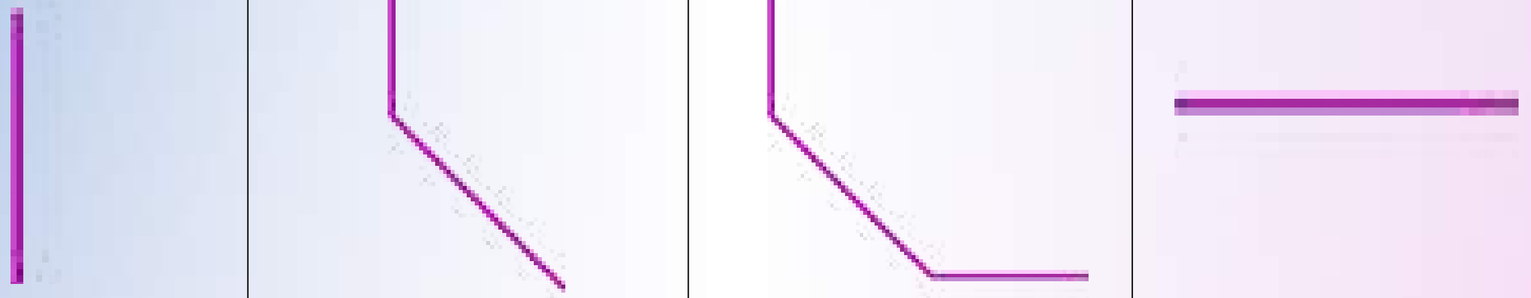
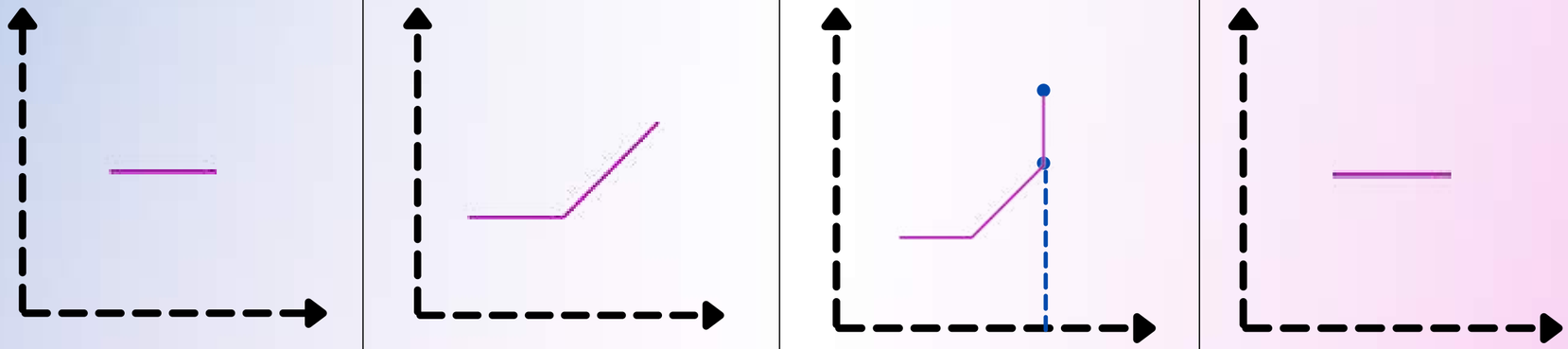


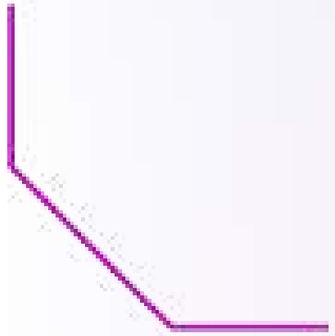
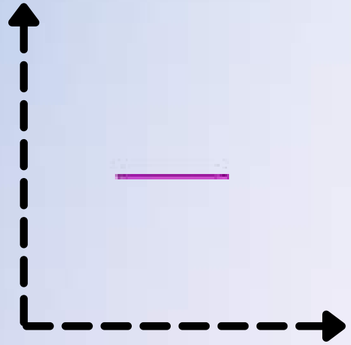
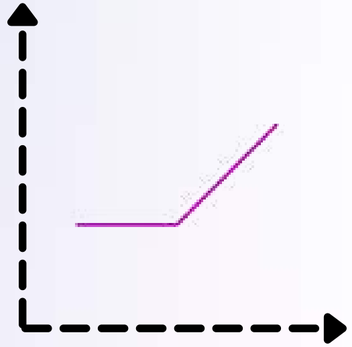
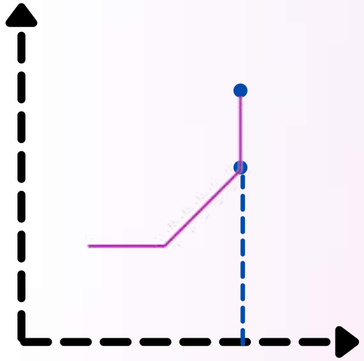
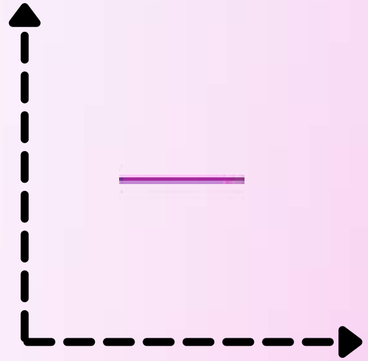
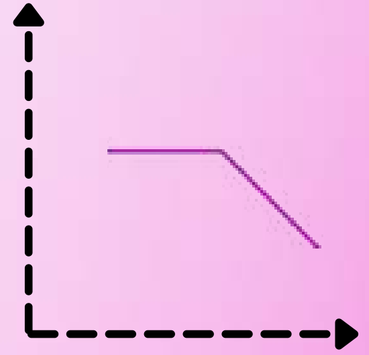
Exemple de contour non cartésien

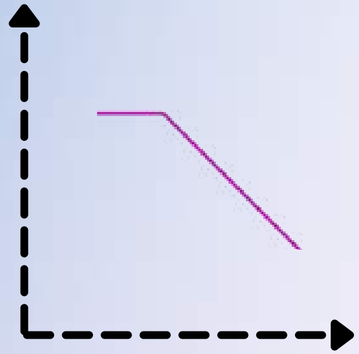
Étape	1	2	3	4	5
Morceau considéré					
Dans un repère					
Cartésien	OUI				

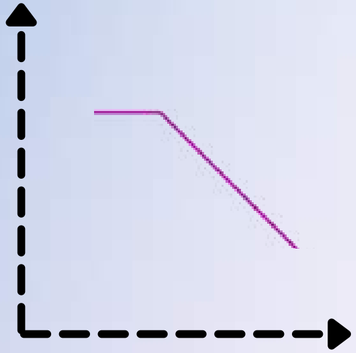
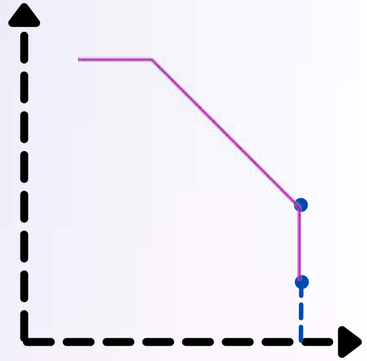
Étape	1	2	3	4	5
Morceau considéré					
Dans un repère					
Cartésien	OUI	OUI			17

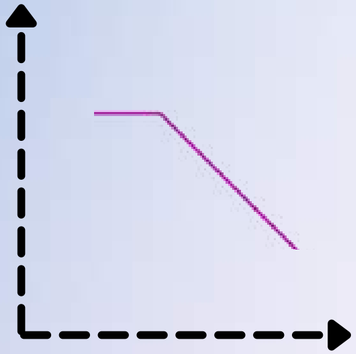
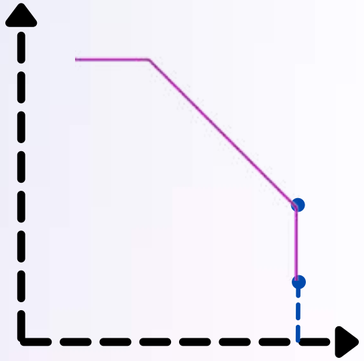
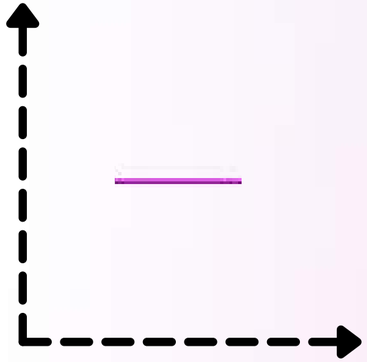
Étape	1	2	3	4	5
Morceau considéré					
Dans un repère					
Cartésien	OUI	OUI	NON		

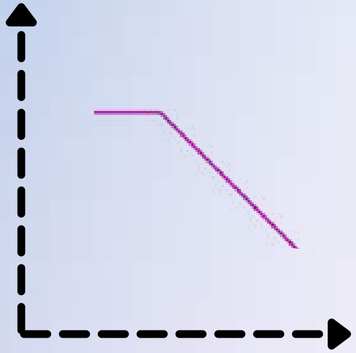
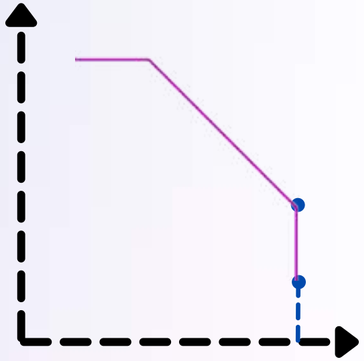
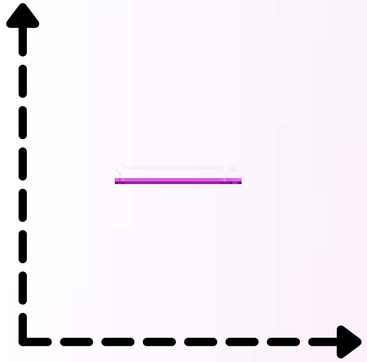
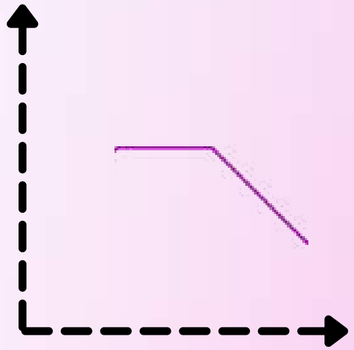
Étape	1	2	3	4	5
Morceau considéré					
Dans un repère					
Cartésien	OUI	OUI	NON	OUI	

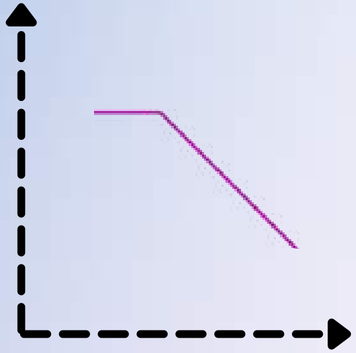
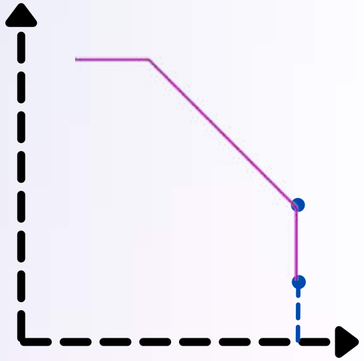
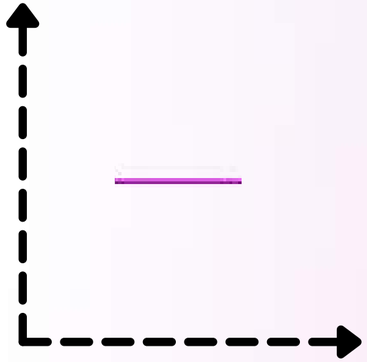
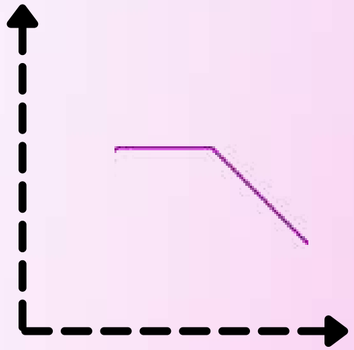
Étape	1	2	3	4	5
Morceau considéré					
Dans un repère					
Cartésien	OUI	OUI	NON	OUI	OUI

Étape	6	7	8	9
Morceau considéré				
Dans un repère				
Cartésien	OUI			

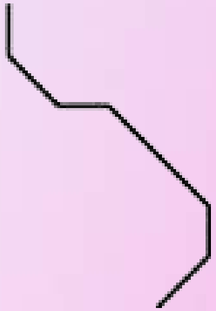
Étape	6	7	8	9
Morceau considéré				
Dans un repère				
Cartésien	OUI	NON		

Étape	6	7	8	9
Morceau considéré				
Dans un repère				
Cartésien	OUI	NON	OUI	

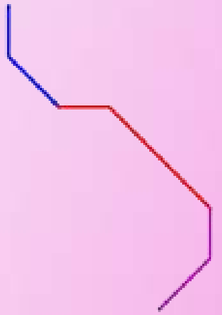
Étape	6	7	8	9
Morceau considéré				
Dans un repère				
Cartésien	OUI	NON	OUI	OUI

Étape	6	7	8	9
Morceau considéré				
Dans un repère				
Cartésien	OUI	NON	OUI	OUI

Résultat :

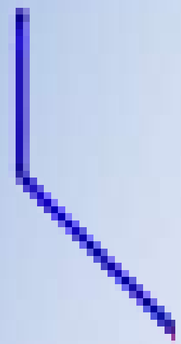


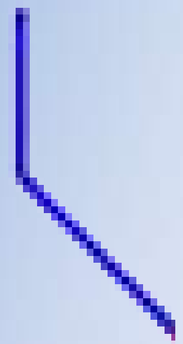
1 contour non cartésien

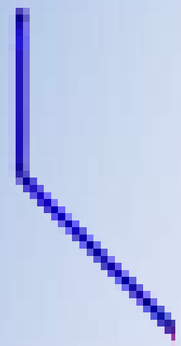
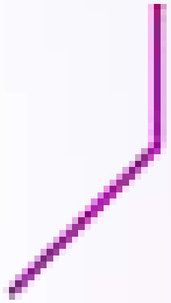


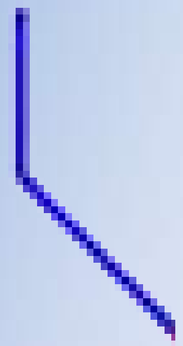
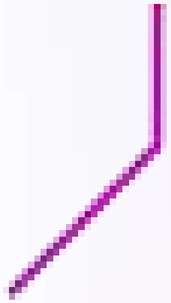
3 contours cartésiens

-› Algorithme de rotation

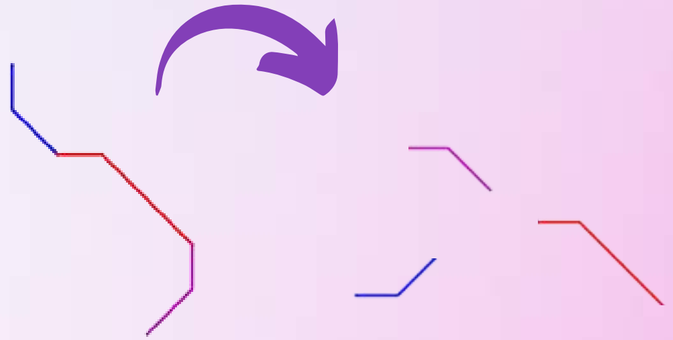
Étape	1	2	2
Morceau considéré			
Direction ordonnée			
Rotation	+ π/2		

Étape	1	2	2
Morceau considéré			
Direction ordonnée			
Rotation	+ π/2	+ 0	

Étape	1	2	2
Morceau considéré			
Direction ordonnée			
Rotation	+ π/2	+ 0	+ π/2

Étape	1	2	2
Morceau considéré			
Direction ordonnée			
Rotation	+ π/2	+ 0	+ π/2

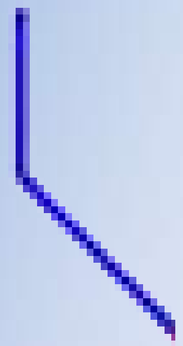
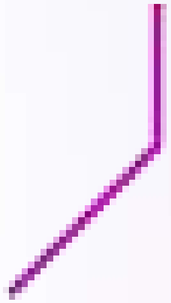
Résultat :



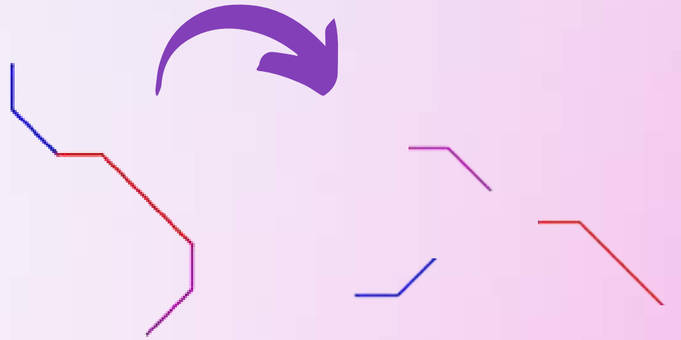
3 contours cartésiens non oritentés

3 contours cartésiens orientés

-› Algorithme de translation

Étape	1	2	2
Morceau considéré			
Direction ordonnée			
Rotation	+ π/2	+ 0	+ π/2

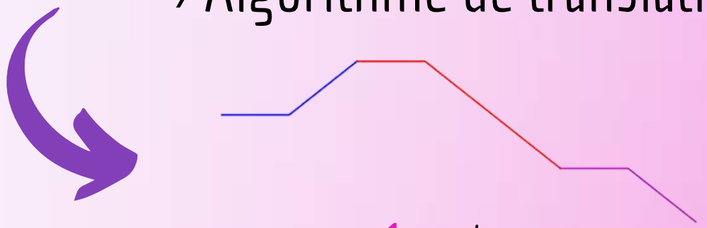
Résultat :



3 contours cartésiens non oritentés

3 contours cartésiens orientés

-› Algorithme de translation

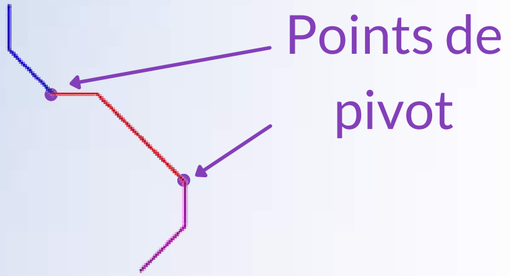


1 contour cartésien

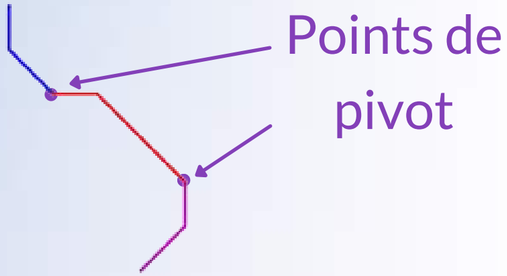
Fin de l'algorithme.

Algorithme des oscillations applicable sur ce nouveau contour cartésien

Inconvénient sur l'exactitude de la valeur :



Inconvénient sur l'exactitude de la valeur :

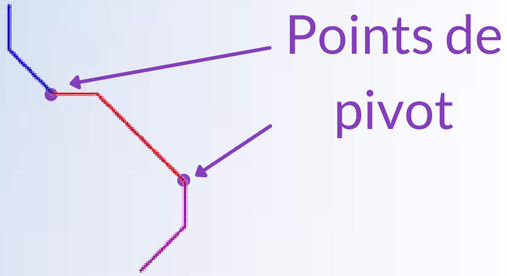


Points de pivot

Précision algorithme des oscillations

Algorithme de déroulement complexe

Inconvénient sur l'exactitude de la valeur :



Points de pivot

Précision algorithme des oscillations

Comment les minimiser ?

Algorithme de déroulement complexe

Inconvénient sur l'exactitude de la valeur :

Points de pivot

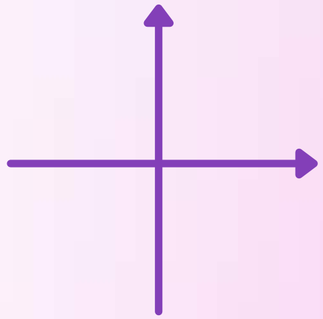
Points de pivot

Précision algorithme des oscillations

Comment les minimiser ?

Redéfinition de "contour cartésien" :

"Il existe une orientation telle que dans un repère cartésien, aucune abscisse n'a deux images"

Algorithme	Déroulement simple	Déroulement complexe
Orienta-tions consi-dérées		

Algorithme de déroulement complexe

Inconvénient sur l'exactitude de la valeur :

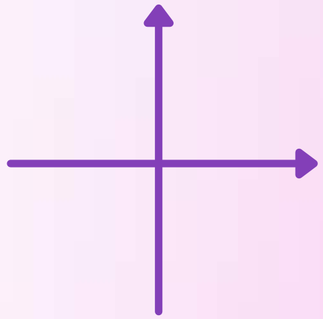
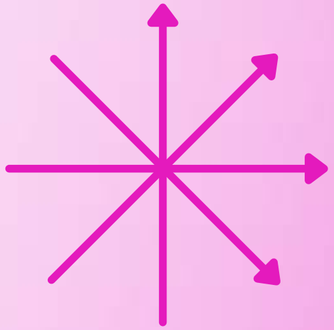
Points de pivot

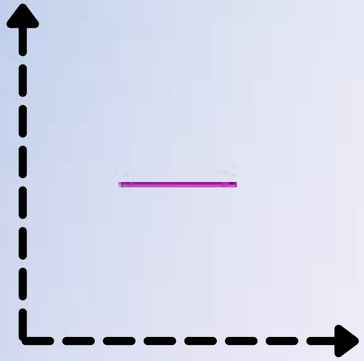
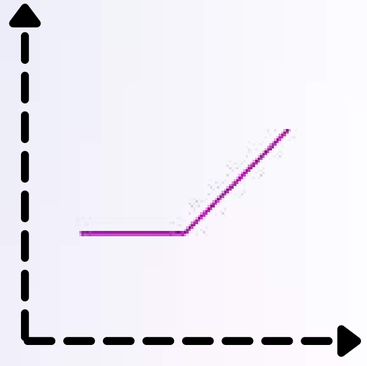
Précision algorithme des oscillations

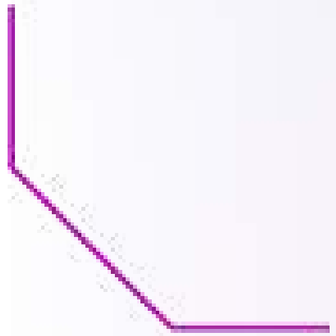
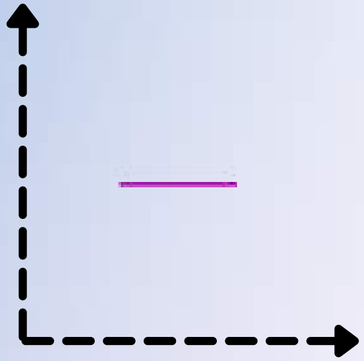
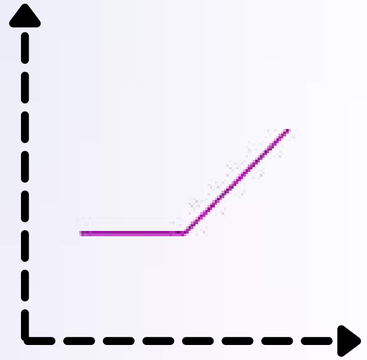
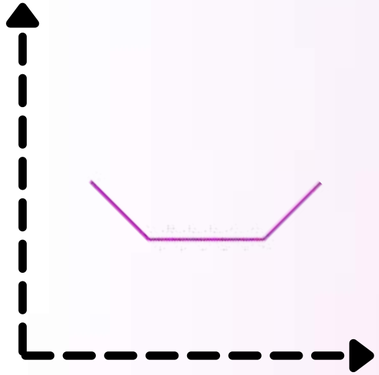
Comment les minimiser ?

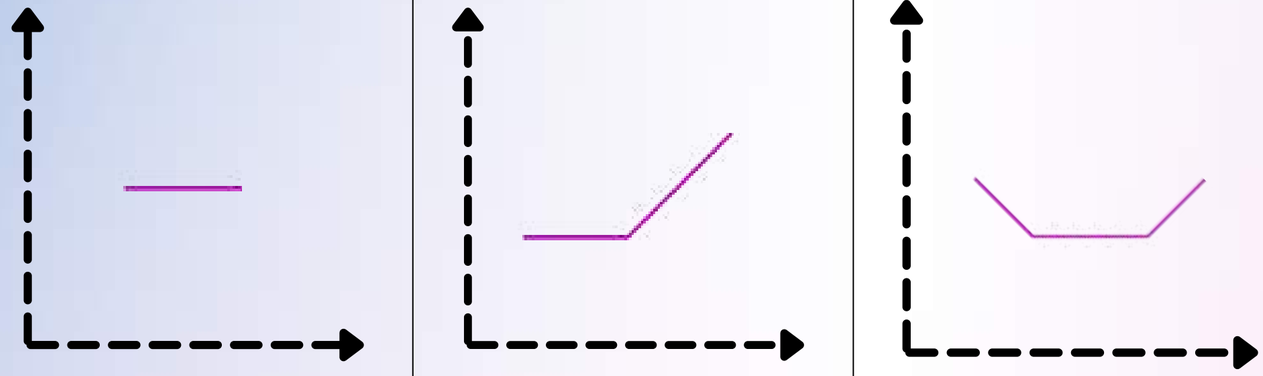
Redéfinition de "contour cartésien" :

"Il existe une orientation telle que dans un repère cartésien, aucune abscisse n'a deux images"

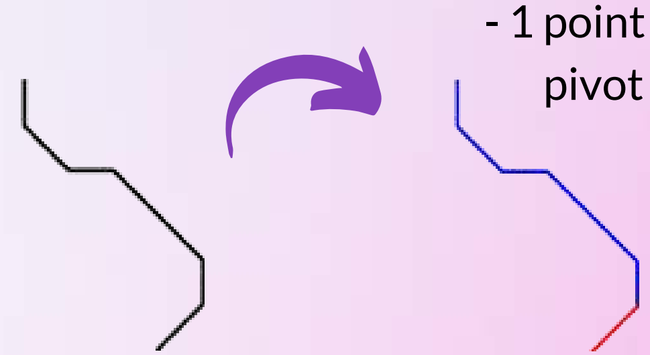
Algorithme	Déroulement simple	Déroulement complexe
Orienta-tions consi-dérées		

Étape	1	2	3
Morceau considéré			
Dans un repère			
Cartésien	OUI	OUI	

Étape	1	2	3
Morceau considéré			
Dans un repère			
Cartésien	OUI	OUI	OUI

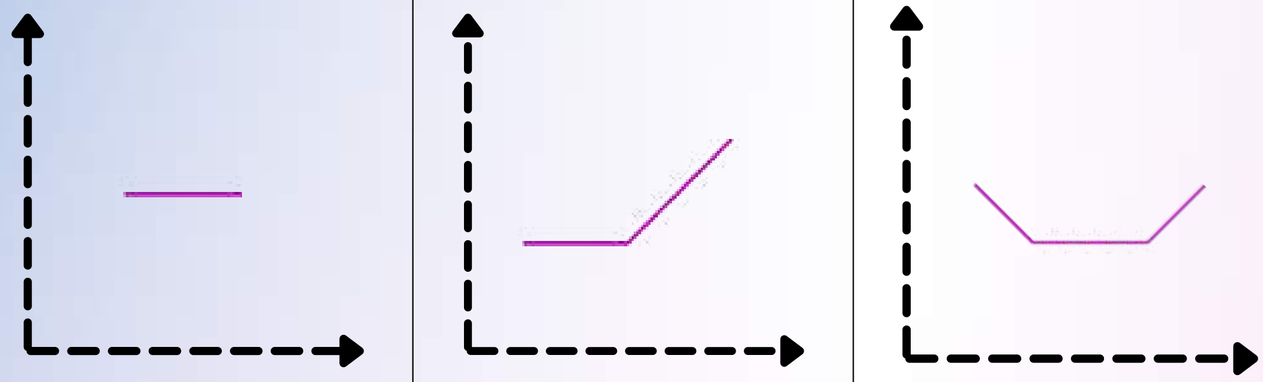
Étape	1	2	3
Morceau considéré			
Dans un repère			
Cartésien	OUI	OUI	OUI

Résultat :

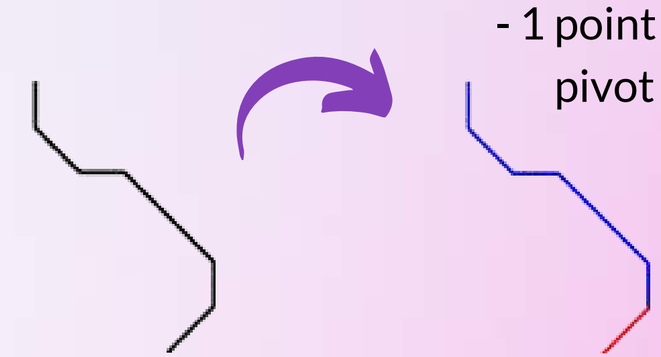


1 contour non cartésien

2 contours cartésiens

Étape	1	2	3
Morceau considéré			
Dans un repère			
Cartésien	OUI	OUI	OUI

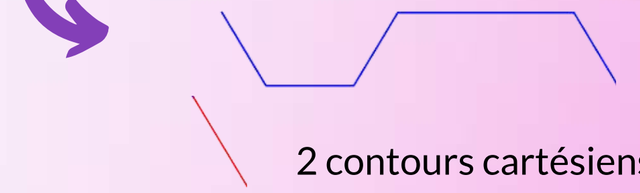
Résultat :



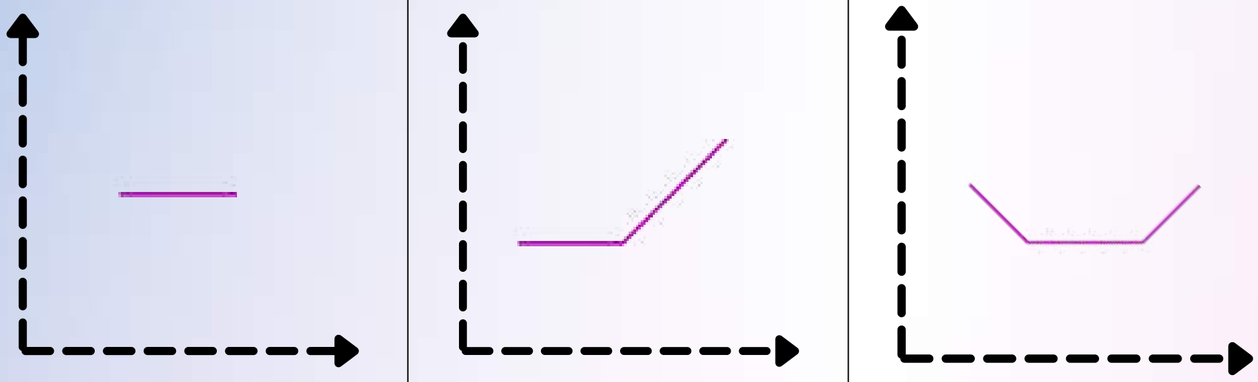
1 contour
non cartésien

2 contours
cartésiens

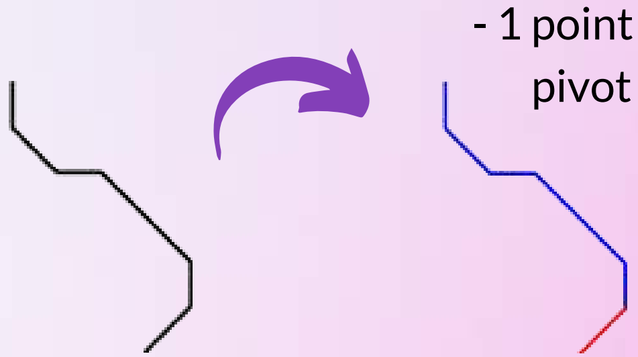
-› Algorithme de rotation



2 contours cartésiens
orientés

Étape	1	2	3
Morceau considéré			
Dans un repère			
Cartésien	OUI	OUI	OUI

Résultat :



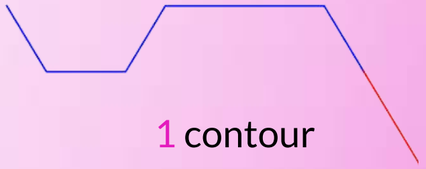
1 contour non cartésien

2 contours cartésiens

-› Algorithme de rotation

2 contours cartésiens orientés

-› Algorithme de translation



Fin de l'algorithme

1 contour cartésien

Efficacité du déroulement complexe

Taille contour		109	301	299	158	205	238	382
Nombre points de pivot	Déroulement simple	19	54	62	28	48	40	64
	Déroulement complexe							

Perte moyenne sur nombre de points de pivot ≈

Efficacité du déroulement complexe

Taille contour		109	301	299	158	205	238	382
Nombre points de pivot	Déroulement simple	19	54	62	28	48	40	64
	Déroulement complexe	14	50	59	24	40	35	54

Perte moyenne sur nombre de points de pivot ≈

Efficacité du déroulement complexe

Taille contour		109	301	299	158	205	238	382
Nombre points de pivot	Déroulement simple	19	54	62	28	48	40	64
	Déroulement complexe	14	50	59	24	40	35	54

Perte moyenne sur nombre de points de pivot ≈ **-13,9 %**

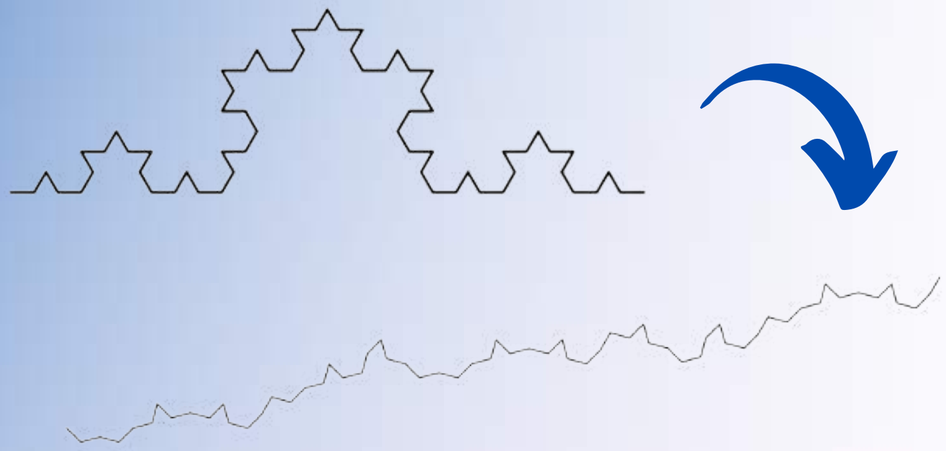
Confrontation des résultats

Courbe de Von Koch $d \approx 1.2618595$

Confrontation des résultats

Courbe de Von Koch d ≈ 1.2618595

Itération 3



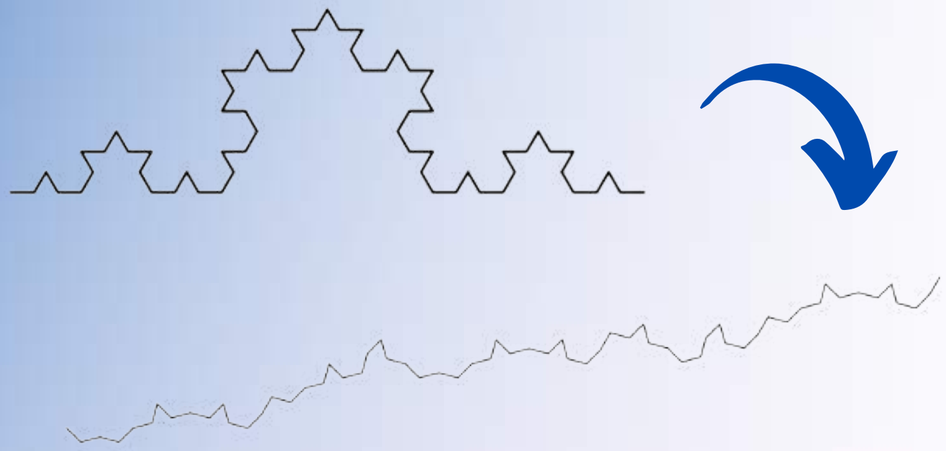
d3 ≈ 1.1143147

Écart relatif itération 3 ≈ **11,7 %**

Confrontation des résultats

Courbe de Von Koch d ≈ 1.2618595

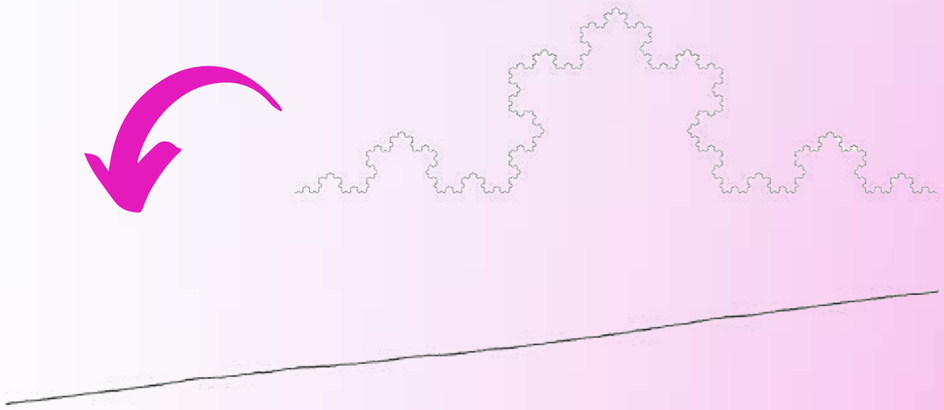
Itération 3



d3 ≈ 1.1143147

Écart relatif itération 3 ≈ **11,7 %**

Itération 6



d6 ≈ 1.2618593

Écart relatif itération 6 ≈ **10^{-5} %**

Confrontation des résultats

Valeur obtenue	Déroulement simple							
	Déroulement complexe							
	Méthode du counting-box	1,291	1,450	1,449	1,398	1,448	1,413	1,412

Écart relatif déroulement simple ≈ Écart relatif déroulement complexe ≈

Confrontation des résultats

Valeur obtenue	Déroulement simple	1,346	1,377	1,428	1,371	1,284	1,384	1,384
	Déroulement complexe	1,352	1,311	1,302	1,296	1,239	1,300	1,281
	Méthode du counting-box	1,291	1,450	1,449	1,398	1,448	1,413	1,412

Écart relatif déroulement simple ≈ Écart relatif déroulement complexe ≈

Confrontation des résultats

Valeur obtenue	Déroulement simple	1,346	1,377	1,428	1,371	1,284	1,384	1,384
	Déroulement complexe	1,352	1,311	1,302	1,296	1,239	1,300	1,281
	Méthode du counting-box	1,291	1,450	1,449	1,398	1,448	1,413	1,412

Écart relatif déroulement simple ≈ **4 %**

Écart relatif déroulement complexe ≈ **9,1 %**

Algorithme de déroulement complexe :

Meilleur gain sur le nombre de pivots

Algorithme de déroulement complexe :

Meilleur gain sur le nombre de pivots

MAIS

Valeur de la dimension fractale moins fiable

Algorithme de déroulement infini

Algorithme de déroulement complexe :

Meilleur gain sur le nombre de pivots

MAIS

Valeur de la dimension fractale moins fiable

Quels autres voies pour l'améliorer ?

Algorithme de déroulement infini

Algorithme de déroulement complexe :

Meilleur gain sur le nombre de pivots

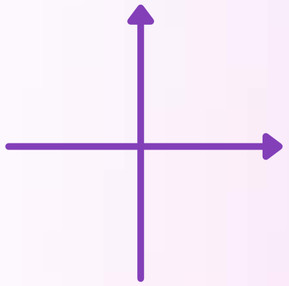
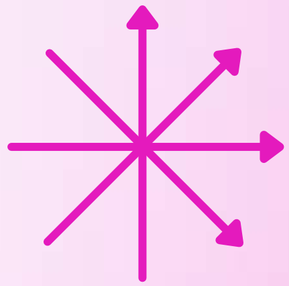
MAIS

Valeur de la dimension fractale moins fiable

Quels autres voies pour l'améliorer ?

Redéfinition de "contour cartésien" :

"Il existe une orientation telle que dans un repère cartésien, aucune abscisse n'a deux images"

Algorithme	Déroulement simple	Déroulement complexe	Déroulement infini
Orienta-tions consi-dérées			

Algorithme de déroulement infini

Algorithme de déroulement complexe :

Meilleur gain sur le nombre de pivots

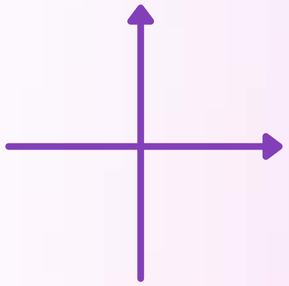
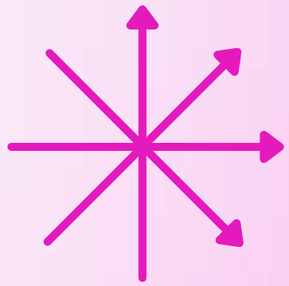
MAIS

Valeur de la dimension fractale moins fiable

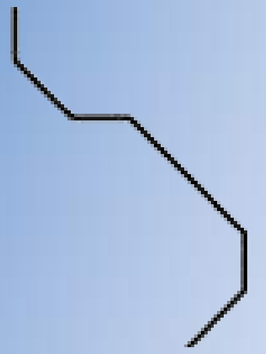
Quels autres voies pour l'améliorer ?

Redéfinition de "contour cartésien" :

"Il existe une orientation telle que dans un repère cartésien, aucune abscisse n'a deux images"

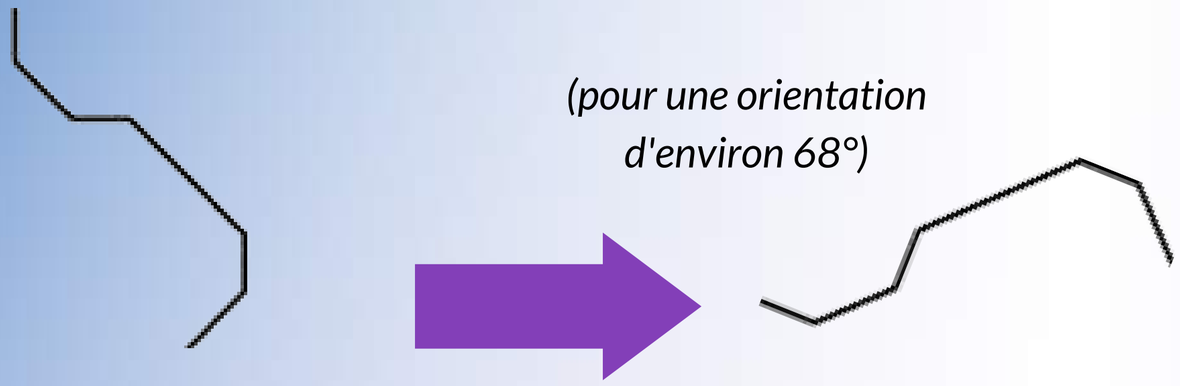
Algorithme	Déroulement simple	Déroulement complexe	Déroulement infini
Orienta-tions consi-dérées			*Toutes*

Résultat :



1 contour non cartésien

Résultat :

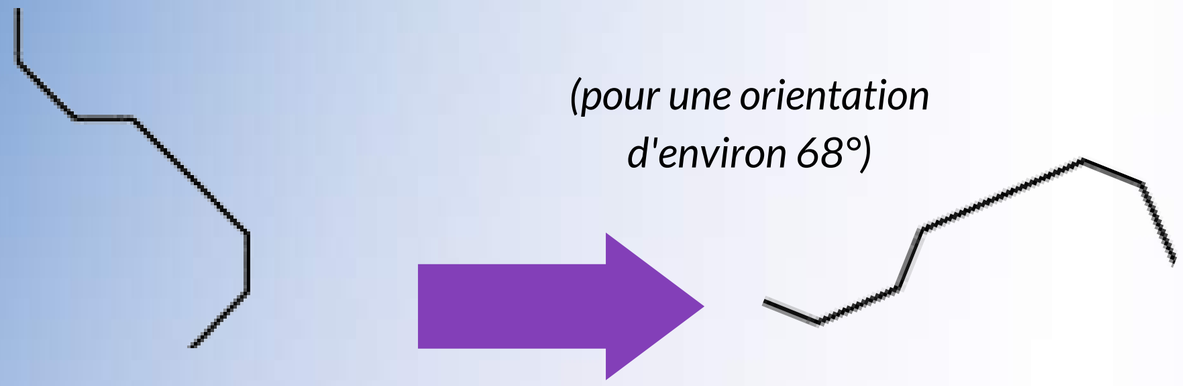


1 contour non cartésien

1 contour cartésien

Algorithme des oscillations applicable sur ce nouveau contour cartésien

Résultat :



1 contour non cartésien

1 contour cartésien

Inconvénient :

- Avec ma méthode de codage :
Complexité infinie sur le nombre de directions (découpage)

Algorithme des oscillations applicable sur ce nouveau contour cartésien

Conclusion

Conclusion

Objectifs atteints :

- Concevoir un algorithme de Counting-Box et l'appliquer
- Concevoir un algorithme des oscillations
- Concevoir deux algorithmes de déroulement simple et complexe

Conclusion

Objectifs atteints :

- Concevoir un algorithme de Counting-Box et l'appliquer
- Concevoir un algorithme des oscillations
- Concevoir deux algorithmes de déroulement simple et complexe

Points à améliorer :

- Restriction sur le type de contours déroulés
- Déroulement complexe moins efficace

Programmes - CANTALOUBE Théo

```python
import numpy as np
import matplotlib.pyplot as plt
from random import choice

###_Détermination des couples (dim(epsilon), epsilon))

def oscileps(fonction,epsilon) : # renvoie la somme des aires délimitées par epsilon
    m, Listevar, dim = int(len(fonction)/epsilon), [], 0
    for i in range(0,m) :
        Fi = fonction[epsilon*i : epsilon*(i+1) + 1]
        Listevar.append((max(Fi), min(Fi)))
    for j in Listevar :
        dim += j[0]-j[1]
    dim = dim*epsilon
    return dim

def dimension(fonction) : # renvoie la dimension de la fonction considérée (cartésienne)
    n = len(fonction)
    limn, Listedim = int(n/4), []
    for epsilon in range(1,limn) :
        Listedim.append((epsilon, (1/n)*oscileps(fonction,epsilon)))
    Listedim = np.log(Listedim)
    return 2 - regLin(Listedim)

def regLin(Liste): # calcule les coefficients de la régression linéaire d'une liste de couples (x,y)
    sommex, sommex2, sommey, sommeyx, n = 0, 0, 0, 0, len(Liste)
    for element in Liste :
        sommex += element[0]
        sommex2 += element[0]**2
        sommey += element[1]
        sommeyx += element[0] * element[1]
    return (n * sommeyx - sommex * sommey) / (n * sommex2 - sommex**2)
```

```python
def dimensionreel(fonction) : # améliore la qualité de l'algorithme des oscillations en introduisant un facteur d'importance du couple considéré (croissant avec la décroissance de epsilon)
    n = len(fonction)
    limn, Listedim = int(n/4), []
    for epsilon in range(1,limn) :
        importance = n//(2*(epsilon+1))
        dim = oscileps(fonction,epsilon)
        for i in range(0,importance) :
            Listedim.append((epsilon, dim))
    Listedim = np.log(Listedim)
    return 2 - regLin(Listedim)

### Création de contour
# On appelle contour tout liste de points juxtaposés telle que n'importe quel point (excepté le premier et dernier) possède exactement deux éléments du la liste dans son voisinage.

def voisin(point,contour) : # Renvoie la liste des voisins du point ne faisant pas partie du contour
    voisins = [[point[0],point[1]-1], [point[0],point[1]+1], [point[0]-1,point[1]], [point[0]+1,point[1]], [point[0]-1,point[1]-1], [point[0]+1,point[1]+1], [point[0]-1,point[1]+1], [point[0]+1,point[1]-1]]
    voisinslibres = []
    for element in voisins :
        if not(element in contour) :
            voisinslibres.append(element)
    return voisinslibres

def contours(n) : # Renvoie un contour aléatoire de n éléments ou moins si problème d'auto-intersection
    condition, voisinspris, contour, i = True, [[0,0],[0, -1], [0, 1], [-1, 0], [1, 0], [-1, -1], [-1, 1]], [[0,0],[0,1]], n
    while condition == True and i != 0 :
        voisinslibres = voisin(contour[-1], voisinspris)
        if voisinslibres != [] :
```

```
            element = choice(voisinslibres)
            contour.append(element)
            voisinspris.append(element)
            voisinsaux = voisin(contour[-2], [])
            for i in voisinsaux :
                if not(i in voisinspris) :
                    voisinspris.append(i)
        else :
            condition = False
        i -= 1
    return contour

### Déroulement simple
# Découpage : Un contour est découpé lorsqu'il est décomposé en sous-listes pouvant toutes être traitées en coordonnées cartésiennes (donc avec un seul antécédent pour chaque image).

def decoupage(liste) :
    n, listedecoupee  = 1, []
    while n < len(liste)-2 : # Construit une liste de sous-listes
        n -= 1
        initialisateur = n
        horizontal, vertical = (liste[n+1])[0] - (liste[n])[0], (liste[n+1])[1] - (liste[n])[1]
        conditionh, conditionv = True, True
        while (conditionh or conditionv) and n < len(liste)-2 :
            n += 1
            diffh, diffv = (liste[n+1])[0] - (liste[n])[0], (liste[n+1])[1] - (liste[n])[1]
            horizontal += diffh
            vertical += diffv
            conditionh = conditionh and (abs(horizontal)==(n+1)-initialisateur)
            conditionv, ancienv = conditionv and (abs(vertical)==(n+1)-initialisateur), conditionv
        sousliste = liste[initialisateur:n+1]
        if ancienv :
            orthogonal = "ord"
        else :
            orthogonal = "abs"
        listedecoupee.append([sousliste,orthogonal])
        n += 1
    return listedecoupee

# Recollage : Le recollage consiste tout d'abord en la réorientation de chaque sous-liste vers la gauche puis en la translation des segments pour les mettre bout à bout.

def rotation(liste) : # Réoriente chaque sous-liste vers la gauche
    listerotatee = []
    for sub in liste :
        sublist = sub[0].copy()
        n = len(sublist)
        if sub[1] == "ord" :
            if sublist[1][1] - sublist[0][1] == 1 : # Direction de la sublist Nord
                for i in range(0,n) :
                    sublist[i] = [sublist[i][1], -sublist[i][0]] # Rotation de -pi/2
            else : # Direction de la sublist Sud
                for i in range(0,n) :
                    sublist[i] = [-sublist[i][1], sublist[i][0]] # Rotation de pi/2
        elif sublist[0][0] - sublist[1][0] == 1 : # Direction de la sublist Est
                for i in range(0,n) :
                    sublist[i] = [-sublist[i][0], -sublist[i][1]] # Rotation de pi
        listerotatee.append(sublist)
    return listerotatee # Pas besoin de changer les sublists de direction Ouest : elles sont déjà bien orientées

def translation(liste) : # Translate chaque liste au bout du contour cartésien créé
    listetranslatee = [liste.pop(0)]
    for sublist in liste :
        vecteur = [listetranslatee[-1][-1][0] - sublist[0][0],
        listetranslatee[-1][-1][1] - sublist[0][1]]
        newlist = []
        for element in sublist :
            newlist.append([element[0] + vecteur[0], element[1] + vecteur[1]])
        listetranslatee.append(newlist)
    return listetranslatee

def final(liste) : # Renvoie une liste aux bonnes dimensions pour appliquer l'algorithme des oscillations
    listefinale = []
    for sublist in liste :
        for element in sublist :
            listefinale.append(element[1])
    return listefinale

def tout(contour) : # Transforme complètement un contour en contour cartésien avec un déroulement simple
    return final(translation(rotation(decoupage(contour))))

### Représentations spécifiques
# Permet de représenter graphiquement les listes à différentes étapes de l'algorithme.
```

```python
def representationcontour(a): # Représente un contour ou une liste translatée
    l1, l2 = [], []
    for element in a :
        l1.append(element[0])
        l2.append(element[1])
    plt.plot(l1,l2,'k')
    plt.axis("equal")
    plt.show()

def representationdecoupage(b): # Représente un contour découpé
    for list in b :
        l1, l2 = [], []
        for points in list[0] :
            l1.append(points[0])
            l2.append(points[1])
        plt.plot(l1,l2)
    plt.axis("equal")
    plt.show()

def representationrotation(c): # Représente une liste rotatée
    for list in c :
        l1, l2 = [], []
        for points in list :
            l1.append(points[0])
            l2.append(points[1])
        plt.plot(l1,l2)
    plt.grid(True)
    plt.axis("equal")
    plt.show()

### Dimension complexe
# Le calcul de dimension complexe diffère dans la délimitation des aires par epsilon. En effet, les abscisses ne sont plus entières à cause de l'homothétie de rapport sqrt(2) induite par les rotations à pi/4 près (à cause des diagonales NOSE et NESO (cf découpage complexe)). Il faut donc introduire un accumulateur.

def oscilepscomplexe(fonction,epsilon,n) : # renvoie la somme des aires délimitées par epsilon
    m, Listevar, dim, acc = int(n/epsilon), [], 0, 0
    for i in range(0,m) :
        init, Fi = acc, []
        while fonction[acc][0] < epsilon*(i+1) :
            acc += 1
        Acci = fonction[init : acc]
        for tuple in Acci :
            Fi.append(tuple[1])
        if Fi != [] :
            Listevar.append((max(Fi), min(Fi)))
    for j in Listevar :
        dim += j[0]-j[1]
    dim = dim*epsilon
    return dim

def dimensioncomplexe(fonction) : # renvoie la dimension complexe de la fonction considérée (cartésienne)
    n = np.floor(fonction[-1][0])
    limn, Listedim = int(n/4), []
    for epsilon in range(1,limn) :
        dim = oscilepscomplexe(fonction,epsilon,n)
        if dim != 0 :
            Listedim.append((epsilon, (1/n)*dim))
    Listedim = np.log(Listedim)
    return 2-regLin(Listedim)

def dimensioncomplexereel(fonction) : # améliore la qualité de l'algorithme des oscillations complexe en introduisant un facteur d'importance du couple considéré (croissant avec la décroissance de epsilon)
    n = len(fonction)
    n = np.floor(fonction[-1][0])
    limn, Listedim = int(n/4), []
    for epsilon in range(2,limn) :
        importance = int(n//(2*(epsilon+1)))
        dim = oscilepscomplexe(fonction,epsilon,n)
        if dim != 0 :
            for i in range(0,importance) :
                Listedim.append((epsilon, (1/n)*dim))
    Listedim = np.log(Listedim)
    return 2-regLin(Listedim)

### Déroulement complexe
# Découpage complexe : le découpage complexe consiste en l'application de l'algorithme de découpage en prenant en compte non seulement les directions Nord et Sud mais également les diagonales NOSE (correspondant à la diagonale allant du Nord-Ouest au Sud-Est) et NESO (correspond à celle allant du Nord-Est au Sud-Ouest).

def decoupagecomplexe(liste) :
    n, listedecoupee  = 1, []
    while n < len(liste)-2 : # Construit une liste de sous-listes
        n -= 1
        initialisateur = n
        horizontal = (liste[n+1])[0] - (liste[n])[0]
        vertical = (liste[n+1])[1] - (liste[n])[1]
        NOSE = pNOSE(horizontal,vertical) # Seule différence vraiment notable avec le découpage simple
        NESO = pNESO(horizontal,vertical)
        conditionh, conditionv, conditionNOSE, conditionNESO = True, True, True, True
```

```
        while (conditionh or conditionv or conditionNOSE or conditionNESO) and n < len(liste)-2 :
            n += 1
            diffh = (liste[n+1])[0] - (liste[n])[0]
            diffv = (liste[n+1])[1] - (liste[n])[1]
            diffNOSE = pNOSE(diffh,diffv)
            diffNESO = pNESO(diffh,diffv)
            horizontal += diffh
            vertical += diffv
            NOSE += diffNOSE
            NESO += diffNESO
            conditionh, ancienh = conditionh and (abs(horizontal)==(n+1)-initialisateur), conditionh
            conditionv, ancienv = conditionv and (abs(vertical)==(n+1)-initialisateur), conditionv
            conditionNOSE, ancienNOSE = conditionNOSE and (abs(NOSE)==(n+1)-initialisateur), conditionNOSE
            conditionNESO = conditionNESO and (abs(NESO)==(n+1)-initialisateur)
        sousliste = liste[initialisateur:n+1]
        if ancienh :
            orthogonal = "abs"
        elif ancienv :
            orthogonal = "ord"
        elif ancienNOSE :
            orthogonal = "NOSE"
        else :
            orthogonal = "NESO"
        listedecoupee.append([sousliste,orthogonal])
        n += 1
    return listedecoupee

def pNOSE(h,v) : # Projection de la distance du segment |liste[n+1],liste[n]| sur la diagonale NOSE
    n = h - v
    if n != 0 : # Normalisation de la projection
        return n/abs(n)
    else :
        return n

def pNESO(h,v) : # Projection de la distance du segment |liste[n+1],liste[n]| sur la diagonale NESO
    n = h + v
    if n != 0 : # Normalisation de la projection
        return n/abs(n)
    else :
        return n
```

```
# Rotation complexe : Le recollage diffère seulement au niveau de la rotation, qui va prendre en compte les directions NOSE et NESO en permettant des rotations à pi/4 près (et non plus seulement des rotations à pi/2 près).

def rotationcomplexe(liste) : # Réoriente chaque sous-liste vers la gauche
    listerotatee = []
    for sub in liste :
        sublist = sub[0].copy()
        n, a, b, c, d = len(sublist), sublist[1][1], sublist[0][1], sublist[0][0], sublist[1][0]
        e, f = (c - d) - (a - b), (c - d) + (a - b)
        r = 1/np.sqrt(2)
        if sub[1] == "ord" :
            if a - b == 1 : # Direction de la sublist Nord
                for i in range(0,n) :
                    sublist[i] = [sublist[i][1], -sublist[i][0]] # Rotation de -pi/2
            else : # Direction de la sublist Sud
                for i in range(0,n) :
                    sublist[i] = [-sublist[i][1], sublist[i][0]] # Rotation de pi/2
        elif sub[1] == "NOSE" :
            if f/abs(f) == 1 : # Direction de la sublist NOSE
                for i in range(0,n) :
                    sublist[i] = [r*(-sublist[i][0]+sublist[i][1]),
                    -r*(sublist[i][0]+sublist[i][1])] # Rotation de -3pi/4
            else : # Direction de la sublist SENO
                for i in range(0,n) :
                    sublist[i] = [r*(sublist[i][0]-sublist[i][1]),
                    r*(sublist[i][0]+sublist[i][1])] # Rotation de pi/4
        elif sub[1] == "NESO" :
            if e/abs(e) == 1 : # Direction de la sublist NESO
                for i in range(0,n) :
                    sublist[i] = [-r*(sublist[i][0]+sublist[i][1]),
                    r*(sublist[i][0]-sublist[i][1])] # Rotation de 3pi/4
            else : # Direction de la sublist SONE
                for i in range(0,n) :
                    sublist[i] = [r*(sublist[i][0]+sublist[i][1]),
                    r*(-sublist[i][0]+sublist[i][1])] # Rotation de -pi/4
        elif c - d == 1 : # Direction de la sublist Est
                for i in range(0,n) :
                   sublist[i] = [-sublist[i][0], -sublist[i][1]] # Rotation de pi
        listerotatee.append(sublist)
    return listerotatee # Pas besoin de changer les sublists de direction Ouest : elles sont déjà bien orientées
```

```python
def finalcomplexe(liste) : # Renvoie une liste de couples adaptée à l'algorithme des oscillations complexe
    listefinale, n = [element for element in liste[0]], len(liste)
    for i in range(1,n) :
        l = len(liste[i])
        for sublist in liste[i][1:l+1] :
            listefinale.append(sublist)
    return listefinale

def toutcomplexe(contour) : # Transforme complètement un contour en contour cartésien avec un déroulement complexe
    return finalcomplexe(translation(rotationcomplexe(decoupagecomplexe(contour))))


### Compilation de toutes les étapes
# Juxtapose sur le même plot un contours aléatoire, son découpage et son recollage réorienté et translaté (simple ou complexe).

def compilation(nature) : # La nature est "simple" ou "complexe"
    a = contoursexigeant(10000)
    plt.subplot(221)
    l1, l2 = [], []
    for element in a :
        l1.append(element[0])
        l2.append(element[1])
    plt.plot(l1,l2)
    if nature == "simple" :
        b = decoupage(a)
    else :
        b = decoupagecomplexe(a)
    plt.subplot(222)
    for list in b :
        l1, l2 = [], []
        for points in list[0] :
            l1.append(points[0])
            l2.append(points[1])
        plt.plot(l1,l2)
    plt.subplot(212)
    if nature == "simple" :
        c = translation(rotation(b))
    else :
        c = translation(rotationcomplexe(b))
    for list in c :
        l1, l2 = [], []
        for element in list :
            l1.append(element[0])
            l2.append(element[1])
        plt.plot(l1,l2)
    plt.grid(True)
    plt.show()
```

```python
from random import choice

### Fractales en tout genre
# Courbe de Von Koch

def branche(point1,point2) :
    x,y = point1
    w,z = point2
    pointg, pointd = ((x+2*w)/3,(y+2*z)/3), ((w+2*x)/3,(z+2*y)/3)
    a,b = pointg
    c,d = pointd
    m1, m2 = (a+c)/2,(b+d)/2
    sommet = ((d-b)*np.sqrt(3)/2+m1,(a-c)*np.sqrt(3)/2+m2)
    return pointg, pointd, sommet

def courbeVonKoch(n) :
    X,Y = [-1],[1]
    def VonKoch(point1,point2,n):
        if n!=0 :
            pointd, pointg, sommet = branche(point1,point2)
            VonKoch(point1,pointg,n-1)
            X.append(pointg[0])
            Y.append(pointg[1])
            VonKoch(pointg,sommet,n-1)
            X.append(sommet[0])
            Y.append(sommet[1])
            VonKoch(sommet,pointd,n-1)
            X.append(pointd[0])
            Y.append(pointd[1])
            VonKoch(pointd,point2,n-1)
    VonKoch((-1,1),(1,1),n)
    plt.plot(X,Y,'k',linewidth=0.5)
    plt.axis('equal')
    plt.show()
    return X,Y

def dimensionVonKoch1(n) :
    frac, a = courbeVonKoch(n), 3**n
    fracagrandie = [(frac[0][i]*a,frac[1][i]*a) for i in range(0,len(frac[0]))]
    return dimensioncomplexereel(toutcomplexe(fracagrandie))
```

```python
# Dragon de Heighway
def brancheH(X,Y) :
    ListerevX = [-y for y in Y]
    ListerevY = [x for x in X]
    ListerevX.reverse()
    ListerevY.reverse()
    accx = X[0]- ListerevX.pop()
    accy = Y[0] -ListerevY.pop()
    for i in range(0,len(ListerevX)):
        ListerevX[i] += accx
        ListerevY[i] += accy
    Xfin = ListerevX + X
    Yfin = ListerevY + Y
    return Xfin, Yfin

def dragonHeighway(n) :
    X = [0,1]
    Y = [0,0]
    for i in range(0,n) :
        X,Y = brancheH(X,Y)
    plt.plot(X,Y,'k',linewidth=0.5)
    plt.axis('equal')
    plt.show()

# Triangle de Sierpinski
def triangleSierpinski(n) :
    a, b, c = (-0.5,0), (0.5,0), (0,0.75)
    x, y = [], []
    i = (choice(np.linspace(-0.5,0,1000)),choice(np.linspace(0,0.75,1000)))
    def f(x):
        return 1.5*x + 0.75
    if i[1] > f(i[0]) :
        i = (i[0] + 0.5, 0.75 - i[1])
    for j in range(0,n) :
        coin = choice([a,b,c])
        i = ((coin[0]+i[0])/2, (coin[1]+i[1])/2 )
        x.append(i[0])
        y.append(i[1])
    plt.scatter(x,y,s=1, marker=".", color="black", linewidth = 0.5)
    plt.axis("equal")
    plt.show()
```

Programmes - MALLET Grégoire

```
from PIL import Image as img
import numpy as np
from PIL import ImageFilter
import matplotlib.pyplot as plt
image = img.open('greyscale.png')


imageWithEdges = image.filter(ImageFilter.FIND_EDGES)
matrice = np.array(imageWithEdges)
matrice[matrice>=240]=255
matrice[matrice< 240] = 0




print('_____')
print(matrice)
print ('nombre de pixels : ',len(matrice),len(matrice[0]))
pil_image=img.fromarray(matrice)
pil_image.show()
 #L'image en sortie a été : passée en Noir//Blanc : les contours seuls sont en
blanc.


from itertools import product
image = img.open('Greg1.png')

###Pré-traitement
imageWithEdges = image.filter(ImageFilter.FIND_EDGES)
matrice = np.array(imageWithEdges)

###paramètres

X = len(matrice) #Taille des colonnes
Y = len(matrice[0]) #Taille des lignes

###Variables et algo
```

```
#Boucle sur l'ensemble de la matrice
def calcul_dim_eps(eps) :
    compteur = 0


    haut_gauche = [0,0]#Cette variable définit le coin haut/gauche du carré en
cours de traîtement.
    limite_x = mth.floor(X/eps) #Calcul du nombre de carrés de taille eps par
ligne/colonne
    limite_y = mth.floor(Y/eps)
    for (i,j) in product(range(0,limite_x ),range(0,limite_y)) :
            #Création d'une liste contenant les éléments du carré, de coin
haut_gauche [',']
        B =matrice[haut_gauche[0] : haut_gauche[0]+eps,haut_gauche[1] :
haut_gauche[1]+eps] #Sélection d'un carré de taille eps
        maxi = B.max() - B.min()
        compteur += (maxi/255) #Le compteur, contrairement à la méthode
précédente, n'ajoute pas un ou 0, mais
            #des valeurs intermédiaires, en fonction des échelles de gris.

        haut_gauche = [haut_gauche[0], haut_gauche[1]+eps]

        if haut_gauche[1] >= Y :
            haut_gauche = [haut_gauche[0]+eps,0]
            if haut_gauche[0] >= Y :
                break

    return([1/eps,compteur])

def calcul_dim() : #Cette fonction renvoie la dimension fractale de l'image
ouverte.
    Reg = []
    leps = [k for k in range(1,mth.floor(len(matrice)/2))]
```

```
    for eps in leps :
        ne = calcul_dim_eps(eps) #Dans cette méthode, ne est une veriable
correspondant à la somme des probabilités de chaque
        #boite, calculé par différence entre les niveaux max et min de gris
        A = X/(2*(eps+1))
        for i in range(0,mth.floor(A)+1) :

            Reg.append(ne)
    Reg1 = np.log(np.array(Reg))
    plt.scatter([i[0] for i in Reg1], [i[1] for i in Reg1],s = 2)
    plt.show()
    Reg3 = Reg1.tolist()
    Dim = regLin(Reg3)
    return Dim
for i in range(1,300) :
    a = rd.choice(['0','-1','+1'])
    if a == '0' :
        L.append(L[-1])
    elif a == '+1' :
        L.append(L[-1] + 1.5)

    elif a == '-1' :
        L.append(L[-1] - 1.5)
Liste = L
print(L)
###Hurst
def calcul_Q(Liste) :
    mini = max(Liste)
    maxi = 0
    for i in range(1, len(Liste))  :
        moyenne = (sum(Liste)/len(Liste))
        Liste2 = Liste[0:i]
        Liste2 = [i-moyenne for i in Liste2]
        somme = sum(Liste2)
        if somme > maxi :
            maxi = somme
        if somme < mini :
            mini = somme
    print(Liste2)
    Liste_2 = [i**2 for i in Liste2]
    Q = maxi - mini
    Q = Q/(np.sqrt(sum(Liste_2)/len(Liste_2)))
    H = np.log(Q)/np.log(len(Liste))
    return H
def Hurst(Liste) :
    H = calcul_Q(Liste)
    print (H,2-H)
```