LARROZE Chloé – MP Option SII
Candidat N°27871

TIPE : Description fractale des tissus urbains et du réseau viaire d’une ville

Session 2022−2023
Thème: La ville

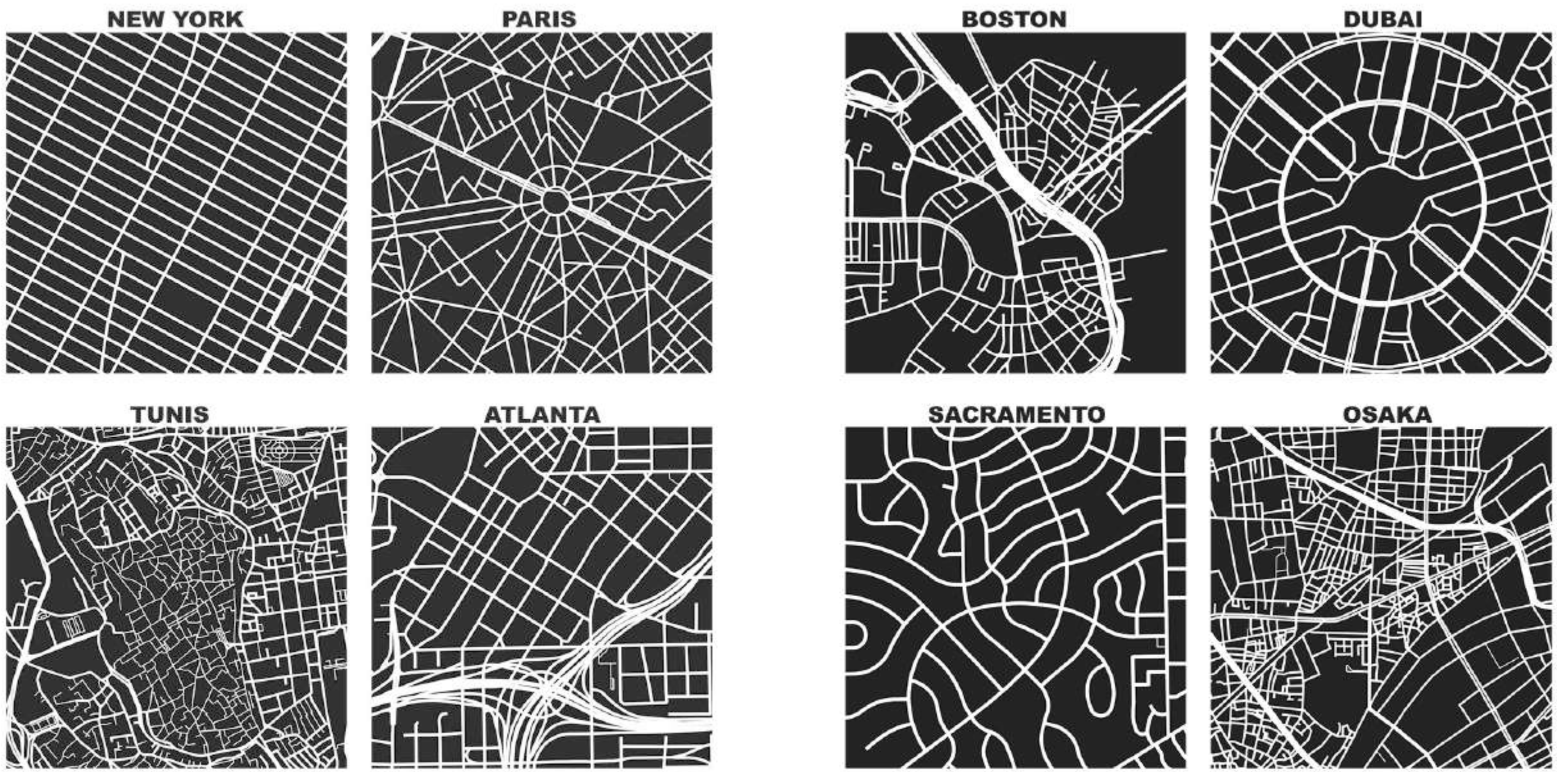
Sommaire

Problématique : **Comment modéliser et étudier le flux urbain par le prisme de la forme des villes ?**

Introduction

01.

Source :
Geoff Boeing



02. LES FRACTALES : DÉFINITIONS ET CARACTÉRISTIQUES

02.

Les fractales : définition et caractéristiques

Source : université de Laval

Définition 1.1: objet fractal

Un objet fractal est un objet qui respecte les trois propriétés suivantes :
- irrégulier à toutes les échelles ;
- auto-similaire;
- de dimension non-entière.

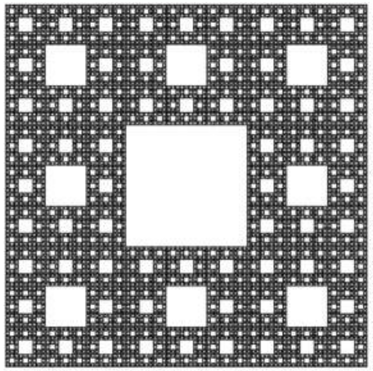


Schéma urbain fractal d'Istanbul

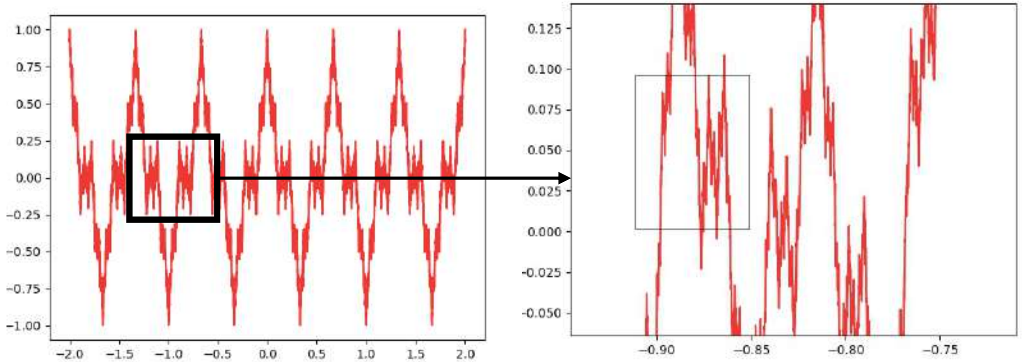
Les fractales : définition et caractéristiques

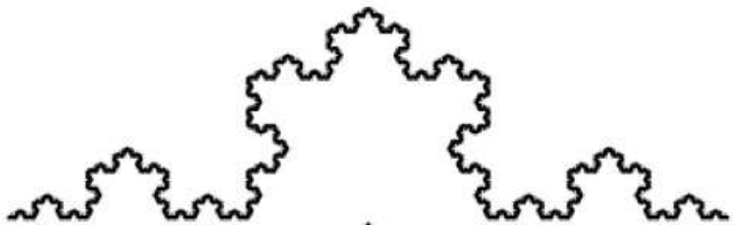
Source : université de Laval

Définition 1.2: Irrégularité

Un objet est irrégulier à toutes les échelles si, même en le regardant de plus en plus près (par exemple avec un zoom), il apparaît toujours irrégulier (non lisse).

$$w(x) = \sum_{n=0}^{\infty} b^n \cos(a^n \pi x),$$





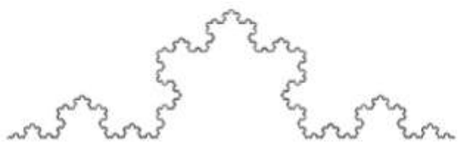
Les fractales : définition et caractéristiques

Source : université de Laval

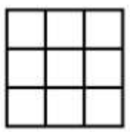
Définition 1.3: Auto-similarité

Un objet F est auto-similaire s'il se décompose en un nombre fini de parties $F_1, F_2, ..., F_N$ qui sont toutes similaires à l'objet entier F . Une partie F_i est similaire à F s'il existe un facteur de dilatation s tel que si l'on dilate F_i d'un facteur s, on retrouve F au complet.

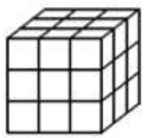
$$N = 3$$
$$s = 3$$



$$N = 4$$
$$s = 3$$



$$N = 9$$
$$s = 3$$



$$N = 27$$
$$s = 3$$

Les fractales : définition et caractéristiques

Source : université de Laval

Définition 1.4: Similitude

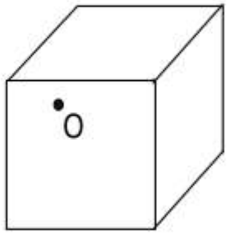
Une similitude du plan est une application w : $R^2 \to R^2$ telle que $\| w(P) - w(Q) \| = k \| P - Q \|$ pour tout P $\in R^2$ et tout Q $\in R^2$ où k est une constante appelée rapport de similitude w.

Définition 1.5: Similitude et auto-similitude

Un ensemble F $\subset R^2$ est auto-similaire s'il existe des similitudes $w_1, ..., w_N$ telles que $F = w_1(F) \cup ... \cup w_N(F)$

Les fractales : définition et caractéristiques

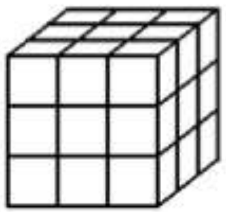
Cube



Dimension de similitude :

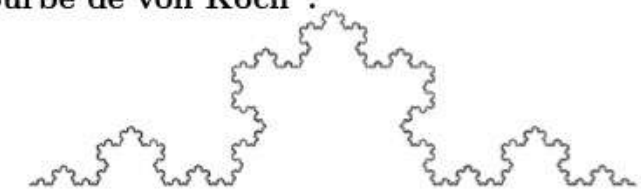
$N = s^d$, où d est la dimension. Ainsi, on a la dimension d'un objet se décomposant en N parties similaires de dilatation s est : $d = \frac{log(N)}{log(s)}$

Courbe différentiable



$$N = 27$$
$$s = 3$$
$$\boxed{N = s^3}$$

Dimension de la courbe de von Koch :



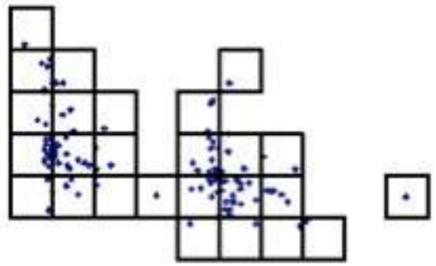
On a $N = 4$ et $s = 3$. Ainsi

$$d = \frac{\log 4}{\log 3} = 1.261859\ldots$$

Les fractales : définition et caractéristiques

Définition 1.6: Dimension de Box-counting

Soit K un sous-ensemble compact de R^2. Considérons un quadrillage du plan par des carrés de côtés ϵ et désignons par $N(\epsilon, K)$ le nombre de carrés de ce quadrillage qui contiennent des points de K.



Carrés d'un quadrillage rencontrant K

La **dimension fractale**, notée D_f , est définie par

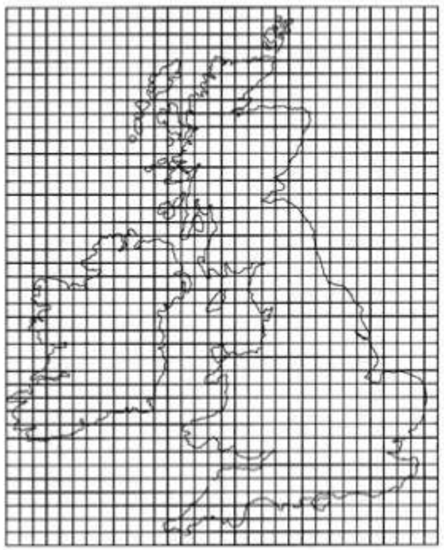
$$D_f = \lim_{\epsilon \to 0} \frac{log(N(\epsilon, K))}{log(\frac{1}{\epsilon})}$$

Les fractales : définition et caractéristiques

Exemple : longueur de la côte d'Angleterre

Image : Université de Laval

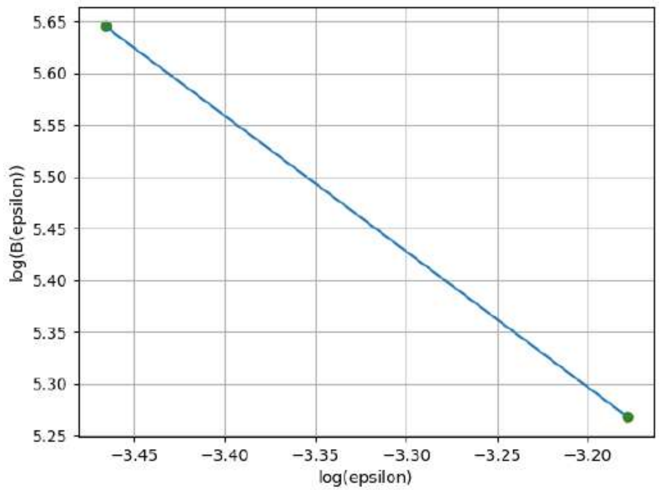
Grille avec des carrés de côtés $\frac{1}{24}$



Grille avec des carrés de côtés $\frac{1}{32}$

$\beta(\varepsilon_1, \Gamma) = 194$ et $\beta(\varepsilon_2, \Gamma) = 283$

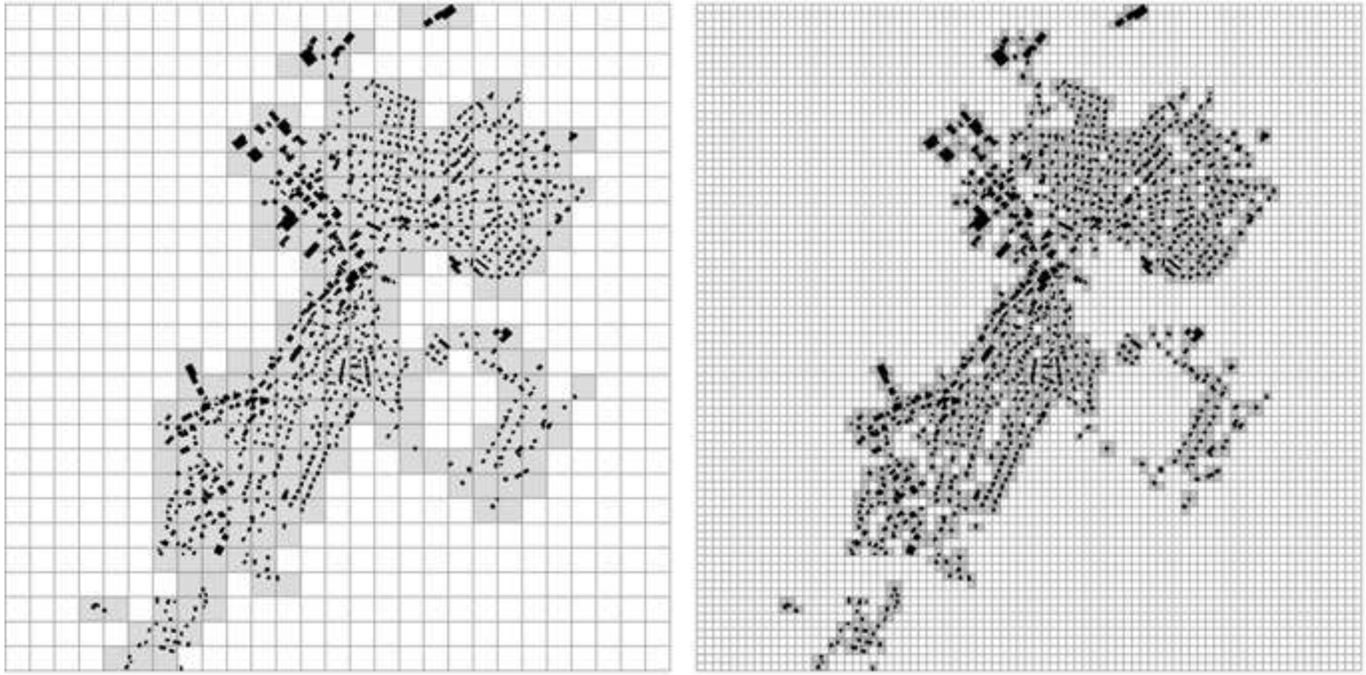
D_f est la pente de la droite reliant les points $(\log(\varepsilon), \log\beta(\varepsilon,\Gamma))$



$$D_f \simeq \frac{\log 283 - \log 194}{\log 32 - \log 24} \simeq 1.31.$$

03. LA VILLE COMME FRACTALE

La ville comme fractale



Troisième étape (maille d'ordre 3) Quatrième étape (maille d'ordre 4)

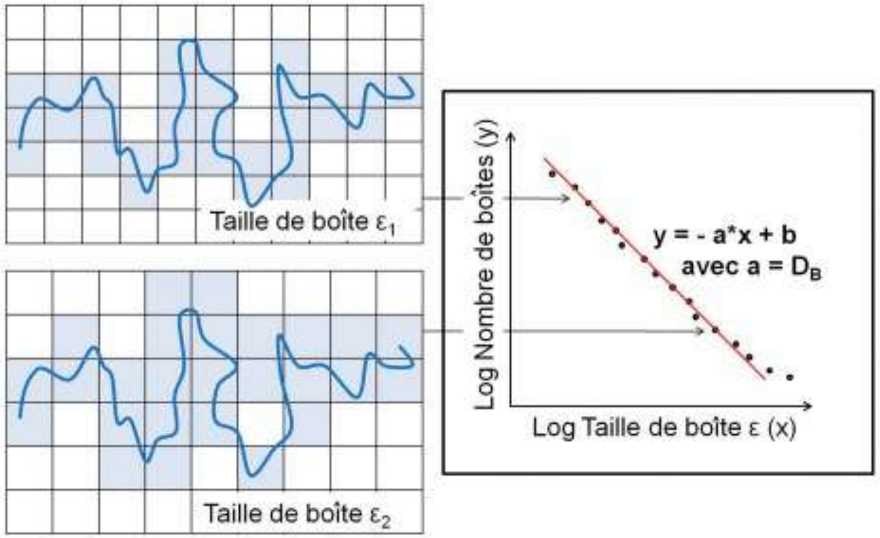
Image : Pierre Frankhauser

Valeurs remarquables:
*Milan : 1.85
*Stuttgart : 1.80

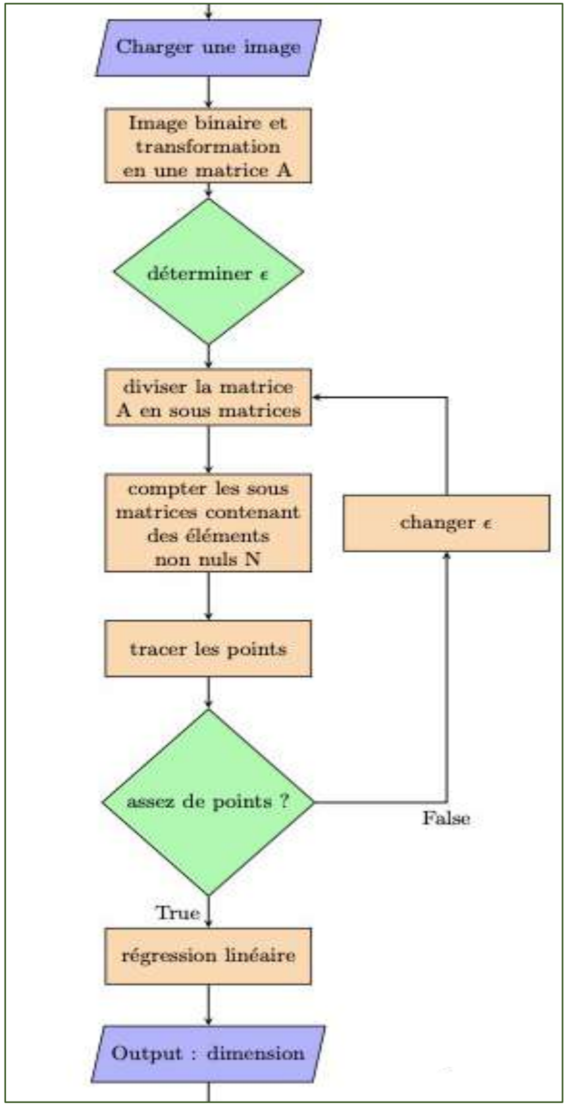
Matériel :
*processeur : 2.2 Ghz Intel i7
*RAM : 8Go
*Python 3.8.4 64 bit

La ville comme fractale

Principe de l'algorithme de Box counting



Gouyet, 1992



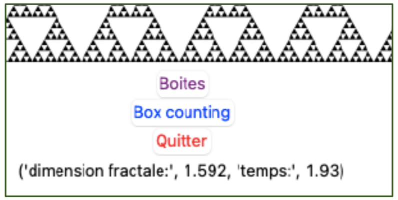
La ville comme fractale

Comparaison des algorithmes de Box counting

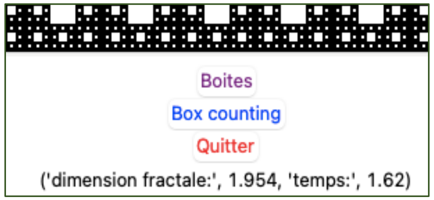
	Triangle de Sierpinski		Tapis de Sierpinski		Plan rempli	
	Dimension	Temps	Dimension	Temps	Dimension	Temps
Algorithme n°1	2.026	9.48 sec	1.843	7.21 sec	1.833	10.09 sec
Algorithme n°2	1.592	1.93 sec	1.954	1.62 sec	2.041	2.65 sec

La ville comme fractale

Résultats de la seconde version de l'algorithme



Triangle de Sierpinski
Dim. Théorique : 1.58



Tapis de Sierpinski
Dim. Théorique : 1.89

Plan rempli
Dim. Théorique : 2

écart : 0.008

écart : 0.034

écart : 0.02

Écart moyen : 0.02

La ville comme fractale

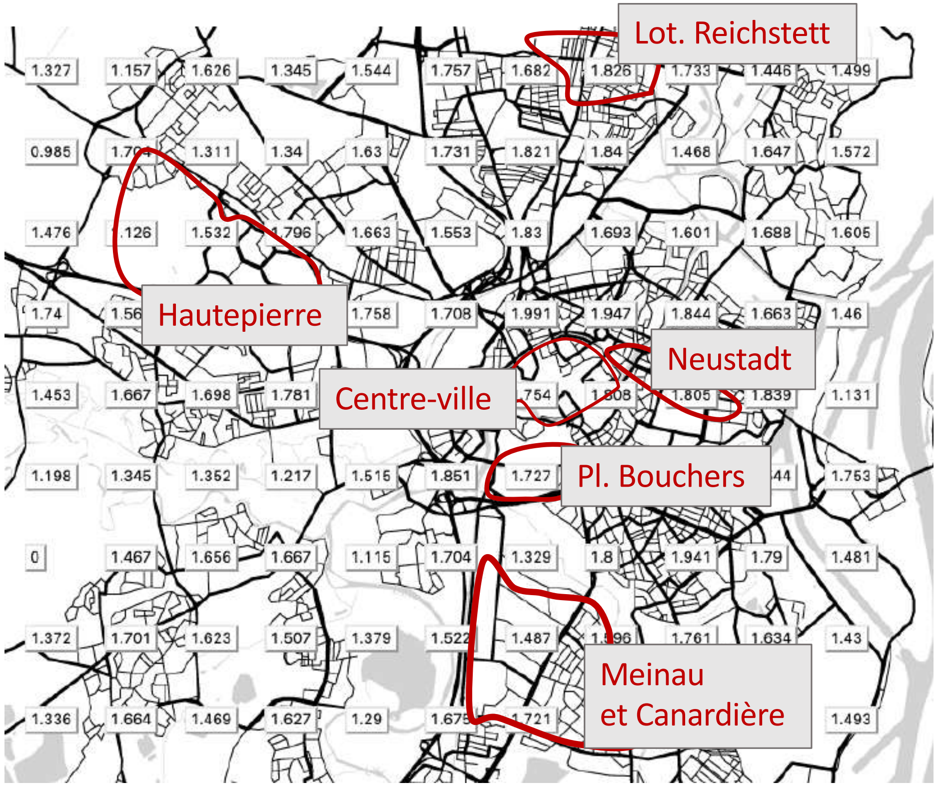
Source carte : *snazzymaps.com*

d = 70 (soit des boites de 900m environ)

Validation du modèle : Strasbourg Kehl

Type	Tissus	Lieu
Centre-ville	Centre ville	Strasbourg
Habitat	Haussmannien	Neustadt
	Lotissement	Reichstett
	Pavillonnaire	Meinau
	Grands ensembles	Canardière
Industriel et commercial	Industriel et commercial	Meinau-Bouchers
	Industriel et commercial	Kehl-Industriel
Mixte	Ville nouvelle	Hautepierre

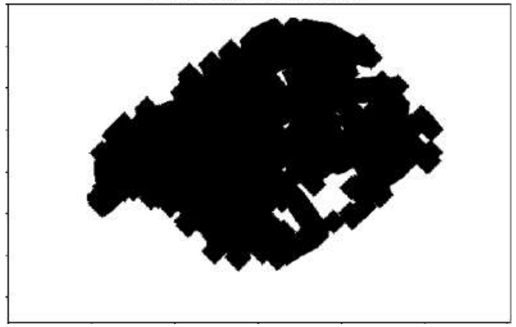
Tissus	D_{cor} surface	D_{cor} bordure	$D_{exp, surface}$
Pavillon. Meinau	1,882	1,676	1,721
Centre-ville	1,860	1,688	1,808
Lot. Reichstett	1,825	1,646	1,826
Cité Canardière	1,800	1,705	---
Ind. Pl. des Bouchers	1,707	1,795	1,727
Neustadt	1,666	1,542	1,808
Hautepierre	1,527	1,638	1,532
Vendenheim	1,415	1,831	-



La ville comme fractale

1ère étape de dilatation

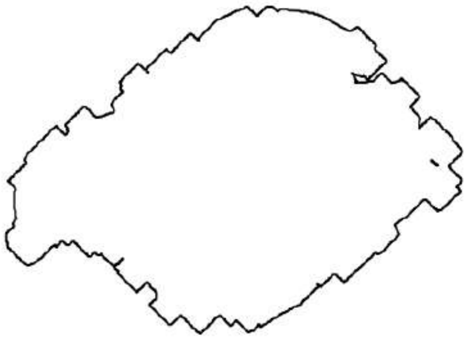
3ème étape de dilatation



7ème étape de dilatation

Exemple : Centre-ville

Bordure extraite

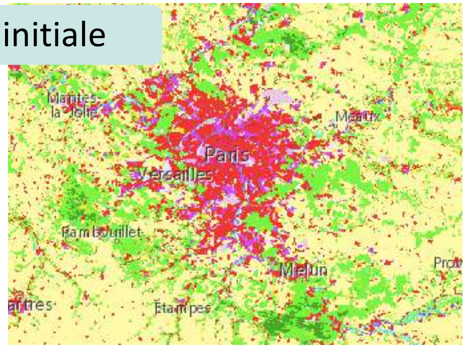


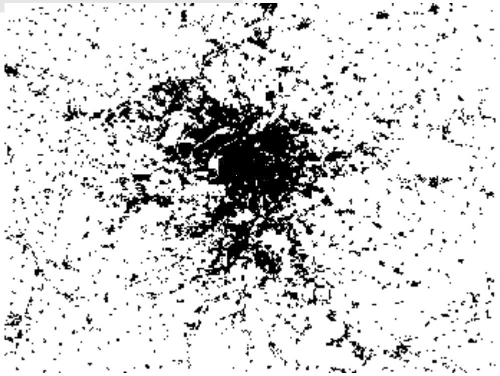
Tissus	D_{cor} surface	D_{cor} bordure	$D_{exp,\ bordure}$
Pavillon. Meinau	1,882	1,676	1,335
Centre-ville	1,860	1,688	1,355
Lot. Reichstett	1,825	1,646	1,458
Cité Canardière	1,800	1,705	---
Ind. Pl. des Bouchers	1,707	1,795	1,603
Neustadt	1,666	1,542	1,376
Hautepierre	1,527	1,638	1,432
Vendenheim	1,415	1,831	---

04. APPLICATION PRATIQUE À LA VILLE DE PARIS

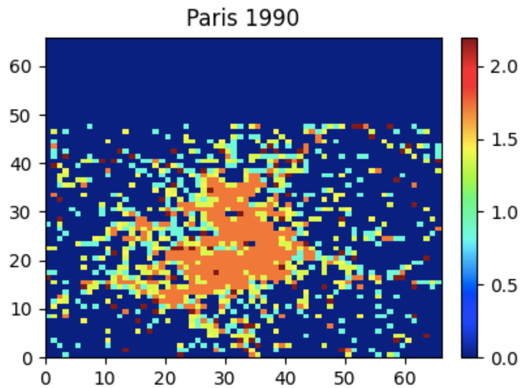
Application pratique

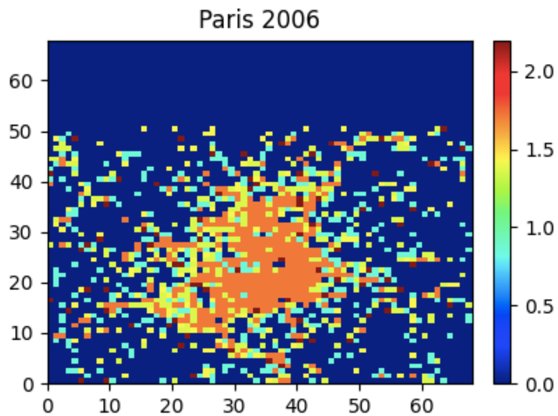
Carte initiale

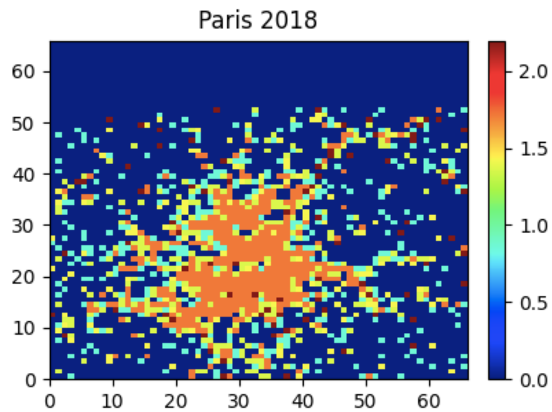




Source carte : **Copernicus Land Monitoring Service**



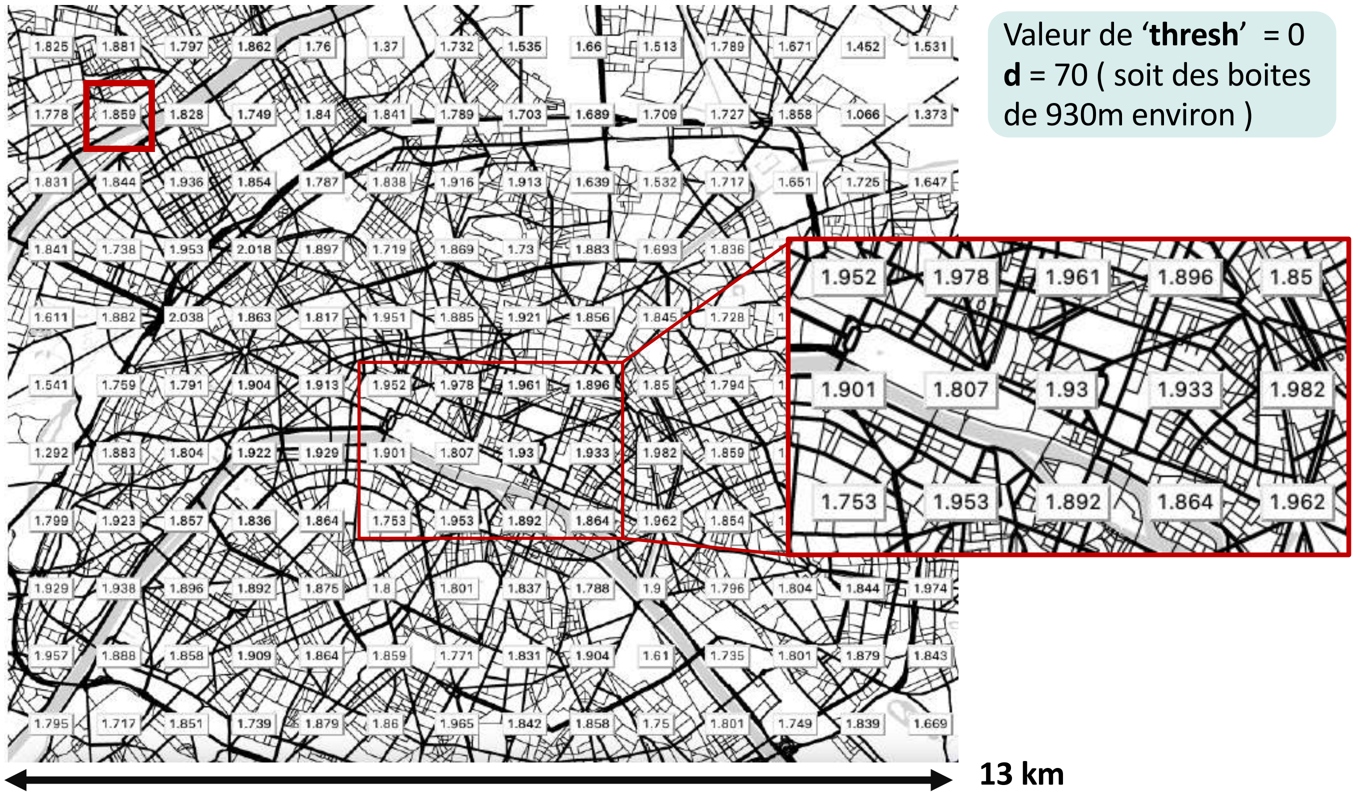




30 km

Échelle : 1 carré vaut 400m

Application pratique



13 km

Valeur de ‘**thresh**’ = 0
d = 70 (soit des boites de 930m environ)

Valeur moyenne : **1.81**

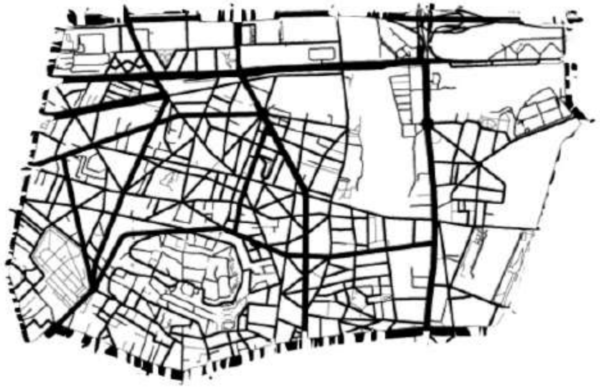
Valeurs remarquables:
*Milan : 1.85
*Stuttgart : 1.80

Application pratique

16e

4e

1er

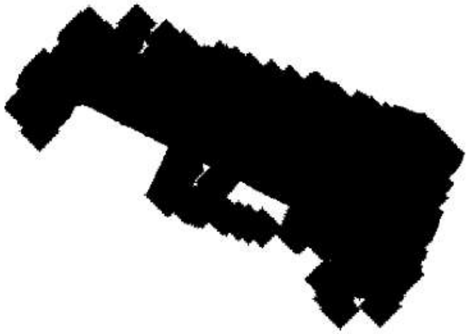


18e

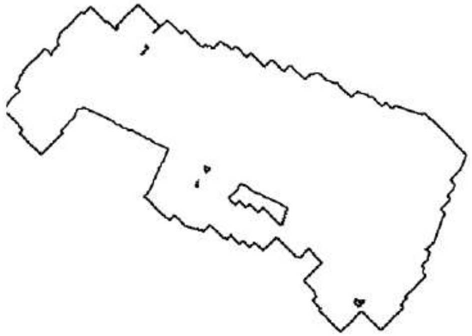
5e

Application pratique

1er arrondissement



9ème étape de dilatation



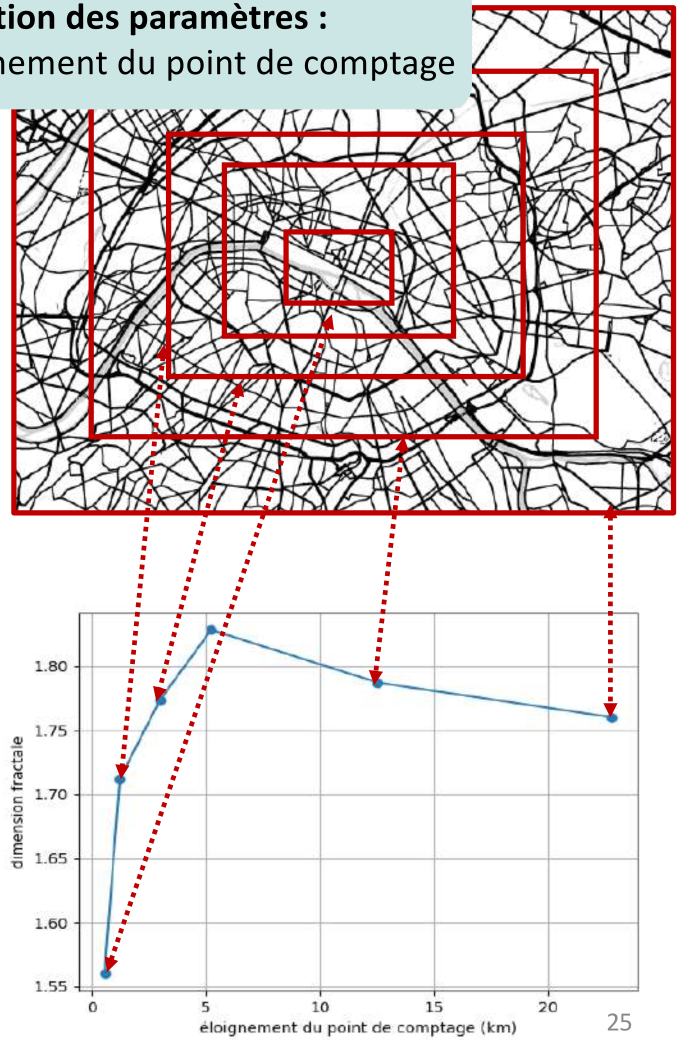
Bordure extraite

Arrondissement	$D_{surface}$	$D_{bordure}$	$D_{th\text{-}Fractalyse}$
1er	1,653	1,330	1,598
4ème	1,69	1,336	---
5ème	1,701	1,288	1,690
16ème	1,609	1,431	1,580
18ème	1,75	1,333	1,753

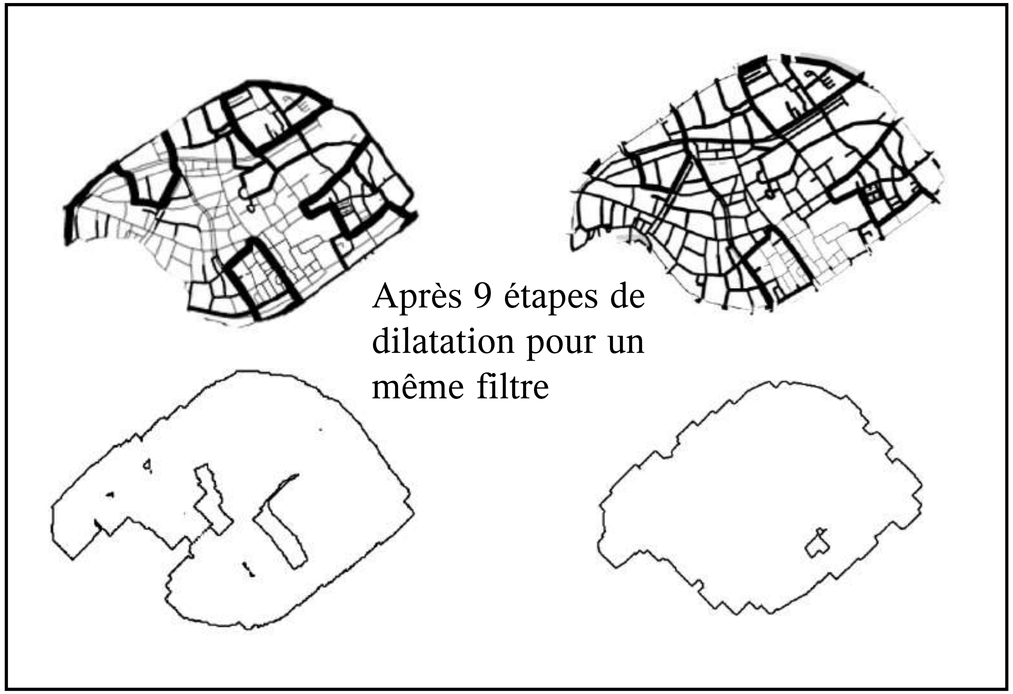
05. LIMITES ET PERSPECTIVES

Limites et perspectives

Variation des paramètres :
Éloignement du point de comptage



Variation des paramètres :
Dimension de surface



06. CONCLUSION

07. ANNEXES

Annexes

Complément courbes paramétrées

Source : université de Laval

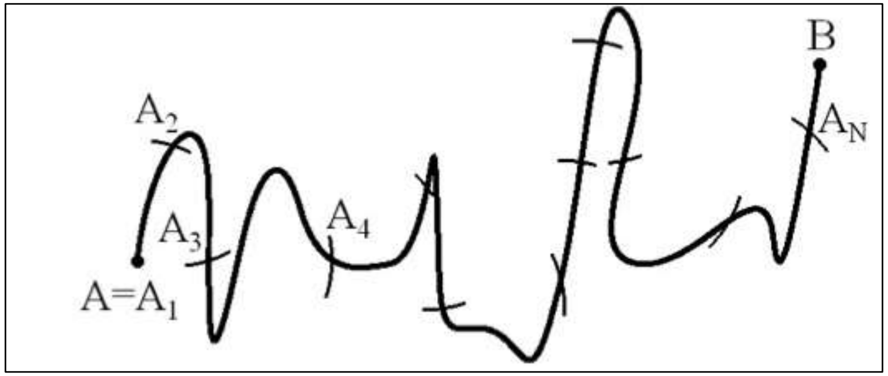
Définition 1.1: Courbes

Une courbe (bornée) Γ dans R^2 est l'image dans R^2 d'un intervalle [a, b] par une application continue γ.
γ est appelée une paramétrisation de la courbe Γ. Si $\gamma(a) = \gamma(b)$, on dit que la courbe est fermée.

Définition 1.3: Courbes

Une courbe Γ de paramétrisation γ : [a,b] $\to R^2$ est dite C^1 par morceaux si on peut décomposer [a, b] en sous-intervalles $[a_i, a_{i+1}], i \in 0, \ldots, p1$ où a0 $= a < a_1 < a_2 < \quad < a_p =$ b tels que la restriction de γ à chaque sous-intervalle soit de classe C^1.

Longueur obtenue avec le compas



Soit P_ε l'approximation linéaire par morceaux passant par les points $A_1, A_2, \ldots, A_N, B$ et désignons par $N(\varepsilon)$ le nombre N, c'est-à-dire P_ε est constitué de $N(\varepsilon) - 1$ segments de longueur ε et d'un segment $([A_N, B])$ de longueur $\leq \varepsilon$. Ainsi,

$$\varepsilon(N(\varepsilon) - 1) \leq \ell(P_\varepsilon) \leq \varepsilon N(\varepsilon)$$

et par définition de la longueur $\ell(\Gamma)$ de Γ, on a

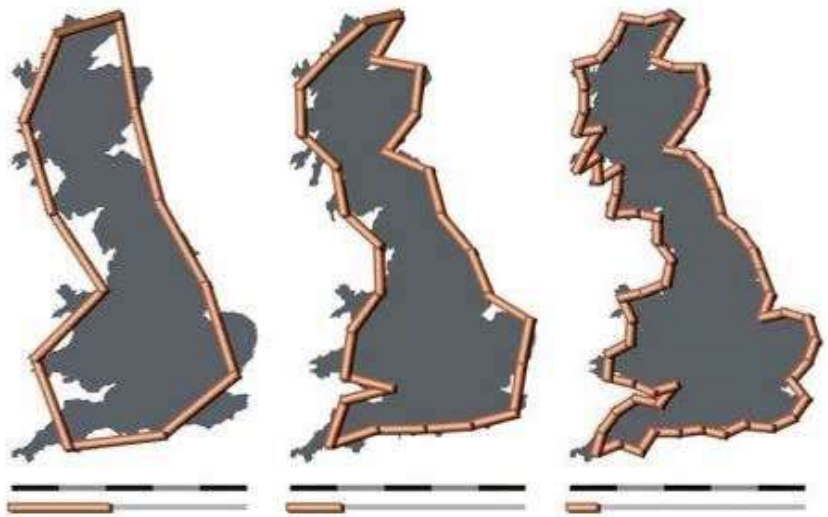
$$\begin{aligned}
\ell(\Gamma) &= \lim_{\varepsilon \to \infty} \ell(\Gamma_\varepsilon) \qquad \text{d'où} \\
\lim_{\varepsilon \to 0} \varepsilon(N(\varepsilon) - 1) &\leq \ell(\Gamma) \leq \lim_{\varepsilon \to 0} \varepsilon N(\varepsilon) \\
\lim_{\varepsilon \to 0} \varepsilon N(\varepsilon) - \lim_{\varepsilon \to 0} \varepsilon &\leq \ell(\Gamma) \leq \lim_{\varepsilon \to 0} \varepsilon N(\varepsilon) \\
\lim_{\varepsilon \to 0} \varepsilon N(\varepsilon) &\leq \ell(\Gamma) \leq \lim_{\varepsilon \to 0} \varepsilon N(\varepsilon) \qquad \text{et donc} \\
\ell(\Gamma) &= \lim_{\varepsilon \to 0} \varepsilon N(\varepsilon).
\end{aligned}$$

Annexes

Complément courbes paramétrées

Ouverture du compas (en km)	Longueur (en km)
500	2 600
100	3 800
54	4 770
17	8 640

Différentes longueurs de la côte de la Grande−Bretagne



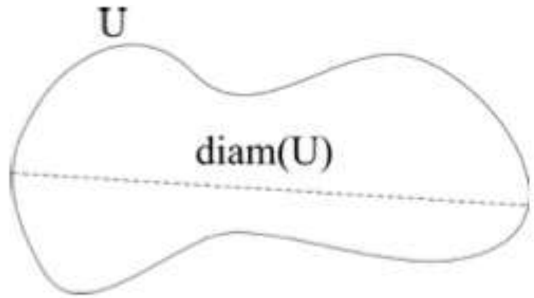
Source : Wikipédia

Annexes

Complément dimension de Hausdorff

Source : université de Laval

Cependant, la dimension de Hausdorff est en général très difficile (voire impossible) à calculer, mais elle possède les bonnes propriétés mathématiques et pour beaucoup de sous–ensembles intéressants, elle coïncide avec la dimension fractale. C'est la raison pour laquelle la dimension fractale (surtout la version «box–counting») est celle qui est la plus utilisée dans la pratique.



Définition 1.1: Diamètre

Soit $U \subset R^n$. Le diamètre de U est
$\text{diam}(U) = \sup d_2(P, Q) \mid P, Q \in U$.

Définition 1.2: Diamètre

Soit $U \subset R^n$. Un recouvrement ouvert de K est une famille dénombrable (ou finie) $U_1, U_2, U_3, \ldots, U_n, \ldots$ de sous-ensembles ouverts telle que

$$K \subset \bigcup_{i=1}^{\infty} U_i$$

Annexes

Complément dimension de Hausdorff

Proposition 1.1

Soient s et ϵ deux nombres positifs, on pose

$$h_\varepsilon^s(K) = \inf\left(\sum_{i=1}^{\infty}(\operatorname{diam}(U_i))^s\right)$$

où l'inf est pris sur tous les recouvrements ouverts $U_1, U_2, U_3, \ldots, U_n$ de K tels que diam (U_i) ¡ ϵ pour i = 1, 2, 3, ... Notons que cette quantité peut être infinie.

On a $h_{\varepsilon_1}^s(K) \leq h_{\varepsilon_2}^s(K)$ si $\varepsilon_2 < \varepsilon_1$ c'est-à-dire, avec s fixé, la fonction $\varepsilon \longmapsto h_\varepsilon^s(K)$ est décroissante. Évidemment, elle est aussi positive. Ainsi, $\lim\limits_{\varepsilon \to 0} h_\varepsilon^s(K)$ existe (elle peut être ∞).

Annexes

Complément dimension de Hausdorff

Définition 1.3: Dimension de Hausdorff

La mesure s-dimensionnelle de Hausdorff, notée h^s(K), est cette limite:

$$h^s(K) = \lim_{\epsilon \to 0} h^s_\epsilon(K) \in [0, \infty].$$

Définition 2.1: D_H

Soit K un sous-ensemble borné de R^n. Alors, il existe un unique nombre réel $D_H \in$ [0, n] tel que

$$h^s(K) = \begin{cases} \infty & si \quad s < D_H \\ 0 & si \quad s > D_H \end{cases}$$

D_H est appelé la dimension de Hausdorff de K que l'on désigne aussi $D_H(K)$.

- Si $K \subset \mathbb{R}^n$ est borné, on a toujours

$$0 \leq D_H(K) \leq D_f(K) \leq n.$$

- Si $K \subset \mathbb{R}^n$ est auto-similaire et se décompose en parties $K_1, \ldots, K_N$ similaires qui sont disjointes ou s'effleurent, on a

$$D_s(K) = D_f(K) = D_H(K).$$

Annexes

Complément dimension de similitude

Définition 3.1: Dimension similitude

Il y a un cas particulier intéressant, celui des compacts auto-similaires. de similitudes $s_1, \ldots, s_N$ de R^n,avec N¿1,de rapport $r_1, \ldots, r_N$ inférieur strict à 1, tels que

$$K = s_1(K) \cup \cdots s_N(K).$$

La dimension d'auto-similarité de K est alors l'unique réel s tel que :

$$r_1^s + \cdots + r_N^s = 1.$$

Annexes

1 Algorithme de Box counting - version n°1

```python
# Import bibliothèques
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image

def est_noir(image, epsilon):
    height, width = image.shape
    s = 0 # Nombre de boites contenant un pixel noir

    # Parcours de l'image avec une taille de boîte donnée
    for i in range(0, height - epsilon + 1, epsilon):
        for j in range(0, width - epsilon + 1, epsilon):
            box = image[i:i+epsilon, j:j+epsilon]
            if np.sum(box) > 0:
                s += 1
    return s

def box_counting(image, min_epsilon, max_epsilon, num_boites):
    boites = np.logspace(np.log10(min_epsilon), np.log10(max_epsilon),
    ↪  num_boites, dtype=int)
    boxes = []

    # Calcul du nombre de boîtes pour chaque taille de boîte
    for boite in boites:
        count = est_noir(image, boite)
        boxes.append(count)

    box_counts = np.array(boxes)
    boites = np.array(boites)

    # Régression linéaire pour estimer la dimension fractale
    coeffs = np.polyfit(np.log(boites), np.log(box_counts), 1)
    dimension = -coeffs[0]

    return dimension

# Utilisation
chemin_image = 'SIERPINSKI.png'  #fichier image
min_epsilon = 1
max_epsilon = 100
num_boites = 10

# Ouverture de l'image à l'aide de PIL
image = Image.open(chemin_image).convert('L')  # Conversion en niveaux de gris
image = np.array(image)

dimension = box_counting(image, min_epsilon, max_epsilon, num_boites)
print("Dimension fractale de "+chemin_image+':', dimension)
```

Annexes

2 Algorithme de Box counting - version n°2

```python
# Import bibliothèque
from PIL import Image
import numpy as np
import pylab as pl
import time

def Box_counting(image_mat):
    # Chronomètre
    t_debut = time.time()

    image = Image.open(image_mat)  # Création d'une matrice
    image_gris = image.convert('L')   # Conversion en nuances de gris

    image_matrix = np.asmatrix(image_gris).copy()   # Adaptation numpy


    thresh = 128   # 0:pixel noir - 255:pixel blanc -> valeur du tresholding

    for i in range(image.size[1]): # Hauteur
        for j in range(image.size[0]): # Largeur
            if image_matrix[i,j] > thresh:
                image_matrix[i,j] = 255     # Blanc
            else:
                image_matrix[i,j] = 0    # Noir

    # Liste des coordonnées de pixel noirs
    liste_pixel=[]
    for i in range(image.size[1]):
        for j in range(image.size[0]):
            if image_matrix[i,j] == 0:  #  Le pixel est noir
                liste_pixel.append((i,j))    #  Ajout du pixel dans la matrice

    Lx=image_gris.size[0]
    Ly=image_gris.size[1]
    pixels_matrice=pl.array(liste_pixel)    # Conversion pylab array

    # On ne considère que l'échelle logarithmique
    echelles=np.logspace(1, 8, num=20, endpoint=False, base=2) #ln(2)

    Ns=[]

    # Boucle de 'plusieurs échelles'
    for echelle in echelles:
        print ('======= Échelle :', echelle)
        # Graphe
        H, edges=np.histogramdd(pixels_matrice,
        ↪   bins=(np.arange(0,Lx,echelle),np.arange(0,Ly,echelle)))
        Ns.append(np.sum(H > 0))

    # Regression linéaire
    coeffs=np.polyfit(np.log(echelles), np.log(Ns), 1)

    D = round(-coeffs[0], 3) # Opposé de la pente

    # Style tracé
    pl.plot(np.log(echelles),np.log(Ns), 'o', mfc='none', label='data')
    pl.plot(np.log(echelles), np.polyval(coeffs,np.log(echelles)),label='fit')
    ↪   #Régression
    pl.xlabel('log $\epsilon$') # Taille de la boîte considérée
    pl.ylabel('log N')
    pl.title( image_mat +' D = '+str(D))
    pl.show()

    # Stop chronomètre
    temps = time.time()-t_debut

    return (D, round(temps, 3) )

print(Box_counting('SIERPINSKI.png'))
```

Annexes

3 Algorithme Box counting Tkinter

```python
# Import bibliothèque
from PIL import Image
import numpy as np
import pylab as pl
import time

def Box_counting(image_mat):
    # Chronomètre
    t_debut = time.time()

    image = Image.open(image_mat)  # Création d'une matrice
    image_gris = image.convert('L')   # Conversion en nuances de gris

    image_matrix = np.asmatrix(image_gris).copy()  # Adaptation numpy


    thresh = 128  # 0:pixel noir - 255:pixel blanc -> valeur du tresholding

    for i in range(image.size[1]): # Hauteur
        for j in range(image.size[0]): # Largeur
            if image_matrix[i,j] > thresh:
                image_matrix[i,j] = 255     # Blanc
            else:
                image_matrix[i,j] = 0   # Noir

    # Liste des coordonnées de pixel noirs
    liste_pixel=[]
    for i in range(image.size[1]):
        for j in range(image.size[0]):
            if image_matrix[i,j] == 0:  #  Le pixel est noir
                liste_pixel.append((i,j))    #  Ajout du pixel dans la matrice

    Lx=image_gris.size[0]
```

```python
    Ly=image_gris.size[1]
    pixels_matrice=pl.array(liste_pixel)   # Conversion pylab array

    # On ne considère que l'échelle logarithmique
    echelles=np.logspace(1, 8, num=20, endpoint=False, base=2) #ln(2)

    Ns=[]

    # Boucle de 'plusieurs échelles'
    for echelle in echelles:
        print ('======= Échelle :', echelle)
        # Graphe
        H, edges=np.histogramdd(pixels_matrice, bins=(np.arange(0,Lx,echelle),np.arange(0,Ly
        Ns.append(np.sum(H > 0))

    # Regression linéaire
    coeffs=np.polyfit(np.log(echelles), np.log(Ns), 1)

    D = round(-coeffs[0], 3) # Opposé de la pente

    # Style tracé
    pl.plot(np.log(echelles),np.log(Ns), 'o', mfc='none', label='data')
    pl.plot(np.log(echelles), np.polyval(coeffs,np.log(echelles)),label='fit') #Régression
    pl.xlabel('log $\epsilon$') # Taille de la boîte considérée
    pl.ylabel('log N')
    pl.title( image_mat +' D = '+str(D))
    pl.show()

    # Stop chronomètre
    temps = time.time()-t_debut

    return (D, round(temps, 3) )

print(Box_counting('SIERPINSKI.png'))
```

Annexes

4 Algorithme de box counting appliqué à une subdivision d'image

```python
# Import bibli
from PIL import Image
from itertools import product
import numpy as np
import pylab as pl
import time
import os
'''version qui prend en entrée le format image PIL. Box counting renvoit la
↪   dimension fractale d'une image. tile() prend en entrée une image et subdivise
↪   cette image en images de coté d.  Puis, dans la même boucle, on applique
↪   l'algorithme de Box counting à chaque subdivision créée.

Le dictionnaire renvoyé contient les dimensions fractales de chaque boite, de bas
↪   en haut et de gauche à droite (l'origine se situant au coin supérieur gauche).
↪   '''


def Box_counting1(image):

    d = {}
    t_debut = time.time() #chronomètre

    image_grayscale = image.convert('L')   # on la convertit en nuances de gris

    image_matrix = np.asmatrix(image_grayscale).copy()   #adaptation numpy


    thresh = 0 # 0;black 255:white // valeur arbitrire du tresholding

    for i in range(image.size[1]): #hauteur
        for j in range(image.size[0]): #largeur
            if image_matrix[i,j] > thresh:
                image_matrix[i,j] = 255     # blanc
            else:
                image_matrix[i,j] = 0    # noir

    # liste des coordonnées de pixel noirs
    pixels=[]
    for i in range(image.size[1]):
        for j in range(image.size[0]):
            if image_matrix[i,j] == 0:   #  pixel is black
                pixels.append((i,j))     #  on le compte
```

```python
    Lx=image_grayscale.size[0]
    Ly=image_grayscale.size[1]

    pixels=pl.array(pixels)    # conversion pylab array

    if len(pixels)!=0:
        scales=np.logspace(1, 8, num=20, endpoint=False, base=2) #ln(2)

        Ns=[]

        #boucle de plusieurs echelles
        for scale in scales:
            #print ('======= Scale :', scale)  #vérif à virer

            #graphe
            H, edges=np.histogramdd(pixels,
            ↪   bins=(np.arange(0,Lx,scale),np.arange(0,Ly,scale)))
            Ns.append(np.sum(H > 0))

        a = np.log(scales)
        u = a.tolist()
    for i in range(len(Ns)):
        if 0 in Ns:
            Ns.remove(0)
            u.pop(-1)

    fin  = np.array(u)
    # regression linéaire
    coeffs=np.polyfit(fin, np.log(Ns), 1)

    D = round(-coeffs[0], 3) # oppose de la pente
    d['dim'] = D


else:
    d['dim']=0
temps = time.time()-t_debut #stop chrono
#affichage
d['temps'] = round(temps, 4)
return d

def tile(d, name):
    img = Image.open(name)
    w, h = img.size

    l = {}
    count  = 1
    grid = product(range(0, h-h%d, d), range(0, w-w%d, d))
    for i, j in grid:
        box = (j, i, j+d, i+d)
        a = img.crop(box) # On divise l'image en boites de coté d
        u = Box_counting1(a) # On applique un algorithme de box counting à
        ↪   chacune d'entre elles
        l[str(count)] = u # On ajoute la dim dans un dico (clef = indice)
        count+=1

    return l, w #renvoit le dico des dimensions
```

5 Affichage subdivision Tkinter

```
#BILBIOTHEQUES
from PIL import Image, ImageTk
import pylab as pl
import  tkinter as Tk
import numpy as np
import recup

#TKINTER
# Tkinter frame
win = Tk.Tk()

#IMAGE
picture = "Paris3.png"
im = Image.open(picture)
bg = Tk.PhotoImage(file = picture)
# Image sur le canevas
win.geometry(str(im.size[0])+'x'+str(im.size[1]))
label1 = Tk.Label(win, image = bg)
label1.place(x = 10, y = 0)


d = 70 # Taille des boites
u = recup.tableau_dimension(d, picture) # On récupère le tableau des dimensions
↪  obenu dans le module 'recup'

# Construction de la liste t
t = [] # On convertit la 'liste de liste' en une simple liste
for i in range(len(u)):
    t+=u[i]
t.pop(0)
t.append(0)

# Affichage des boites contenant la dimension fractale sur tkinter
for i in range(len(t)):
    _ = recup.numero_vers_coordonnees(i,int(im.size[0]/d)) # On convertit le
    ↪  numéro de la boite en coordonnées de la boite
    label2 = Tk.Label( win, text = str(t[i]), relief = "raised" )
    label2.place(x = _[0]*d+int(d/2) , y = _[1]*d+int(d/2)) # On place la boite

win.mainloop()
```

Annexes

6 Histogramme dimension fractale

```
#BILI
import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mpl
from matplotlib.colors import ListedColormap, LinearSegmentedColormap
import numpy as np
import recup

d = 5 # Taille des 'sous-images'

#issu de :
↪ https://matplotlib.org/stable/tutorials/colors/colormap-manipulation.html#sphx-glr-tutorials-colors-colormap-m

'''Prend en entrée colormaps: une palette de couleur dans le module cmp'''

def plot_examples(colormaps):
    name = 'Paris.png'
    data = recup.tableau_dimension(d, name) # Nombre de carrés
    n = len(colormaps)
    fig, axs = plt.subplots(1, n, figsize=(n * 2 + 2, 3),
                            constrained_layout=True, squeeze=False)
    for [ax, cmap] in zip(axs.flat, colormaps):
        psm = ax.pcolormesh(data, cmap=cmap, rasterized=True, vmin=0, vmax=2.2) #
        ↪ Plage de variation des couleurs

          fig.colorbar(psm, ax=ax)
      plt.title('Paris et sa banlieue')
      plt.show()

  viridis = mpl.colormaps['jet'].resampled(256)

  plot_examples([viridis])
```

Annexes

7 Algorithme de dilatation d'une image

```python
#Ce programme permet la dilatation d'une image
# adapté de:
↪ https://openclassrooms.com/fr/courses/5060661-initiez-vous-aux-traitements-de-base-des-
# images-numeriques/5217276-maitrisez-les-operations-morphologiques-de-base

# Import bibli
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt

# Fonctions ------------------------------------------
def convolution (image,filtre,function):
    new_image = np.zeros(image.shape, int)
    for i in range(0,image.shape[0]):
        for j in range(0,image.shape[1]):
            new_image[i,j] = function(image,i,j,filtre)
    return new_image

# Dilatation
def pixel_erosion(image, ligne, colonne, elmt_structurant):

    width = len(elmt_structurant) // 2
    height = len(elmt_structurant[0]) // 2
    pixel_value = True;

    for i in range(0,len(elmt_structurant)):
        for j in range(0,len(elmt_structurant[0])):
            x_image = ligne + i - width
            y_image = colonne + j - height
            if((x_image >= 0) and (x_image < image.shape[0]) and (y_image>=0) and
            ↪ (y_image<image.shape[1])):
                if (elmt_structurant[i,j] and not(image[x_image,y_image])):
                    pixel_value = False

    return pixel_value

def erosion(image, elmt_structurant):
    return convolution(image,elmt_structurant,pixel_erosion)

def ero(image):
    return erosion(image, filtre)

#  Image et filtre
image = cv.imread('18e.png',0)

filtre = np.array([[0, 0, 1, 0, 0], [0, 0, 1, 0, 0], [1, 1, 1, 1, 1], [0, 0, 1,
↪ 0, 0], [0, 0, 1, 0, 0]])

'''
Filtre de la forme :
[[0 0 1 0 0]
 [0 0 1 0 0]
 [1 1 1 1 1]
 [0 0 1 0 0]
 [0 0 1 0 0]]

'''

# Affichage matplotlib
objet  = ero(image)

plt.figure()
plt.imshow(objet,cmap='gray')
plt.show()
```

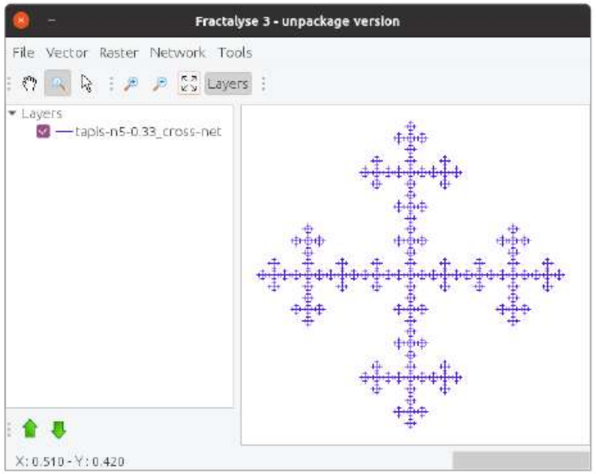
Annexes

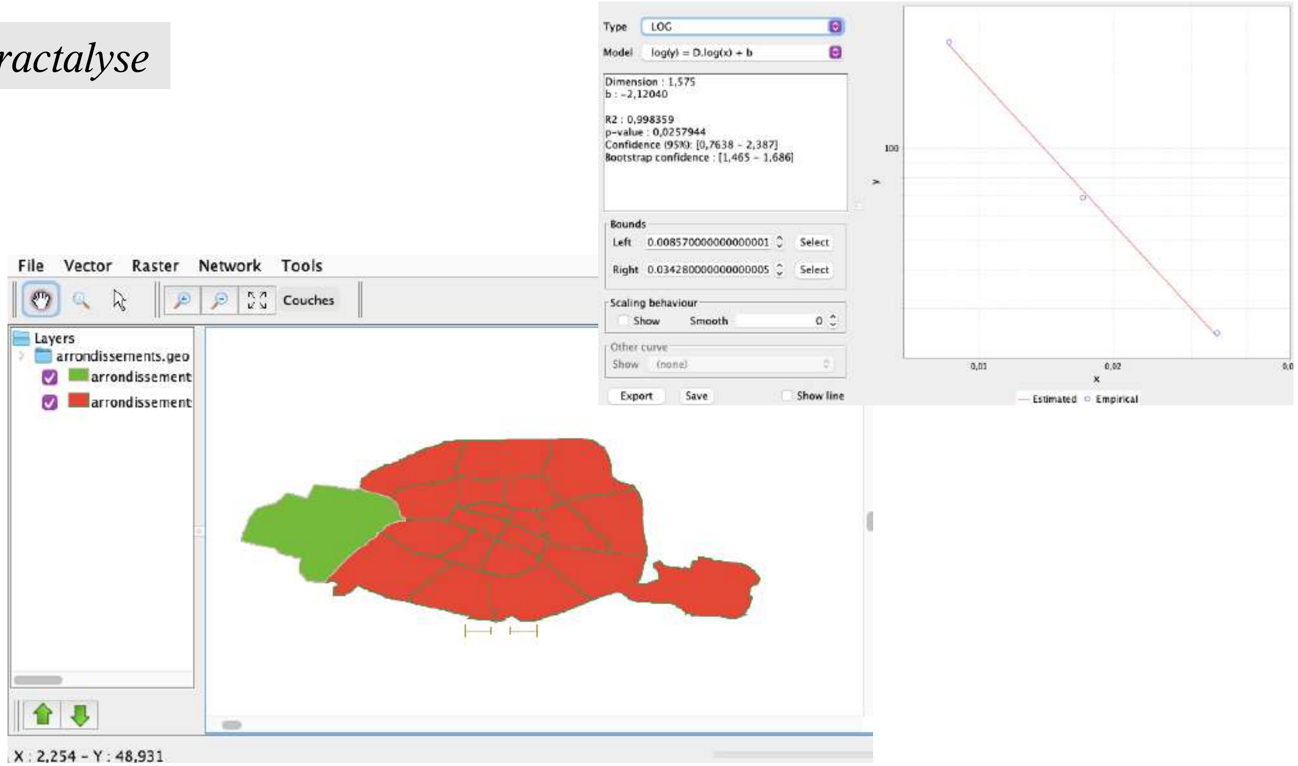
8 Fonction de Weierstraß

```
import numpy as np
import matplotlib.pyplot as plt

M = 100000

def weierstrass(x, N):
        y = np.zeros((1,M))
        for n in range(1,N):
                y = y + np.cos(3**n*np.pi*x)/2**n
        return y

x = np.linspace(-2,2,M)
y = np.reshape(weierstrass(x,500),(M,))

plt.plot(x, y, 'r-')
plt.show()
```

Annexes

Logiciel Fractalyse





Fractalyse est un logiciel dédié à l'analyse fractale de modèles 2D (ensembles de points). Cette version a été écrite en langage Java, à partir de laquelle résulte l'amélioration de la gestion des données avec SIG (Systèmes d'Information Géographique), interface utilisateur graphique et la performance avec le parallélisme.

Fractalyse a été développée par Gilles Vuidel au laboratoire ThéMA (CNRS – Université de Franche-Comté, Besançon, France).