Modification du relief montagneux pour prévenir des glissements de terrain

Hippolyte Boucherie n°16827

Session 2022

Introduction

Fig. : Glissement de terrain ayant détruit le village de Sant'Antonio Morignone en Italie

Objectif : uniformiser l'écoulement de l'eau sur le flanc montagneux

Plan

Plan

1er exemple

```
dictionnaire = {'0,0': 0,
                '1,0': 0,
                '2,0': 0,
                '0,1': 0,
                '1,1': 0,
                '2,1': 0,
                '0,2': 0,
                '1,2': 0,
                '2,2': 0}
```

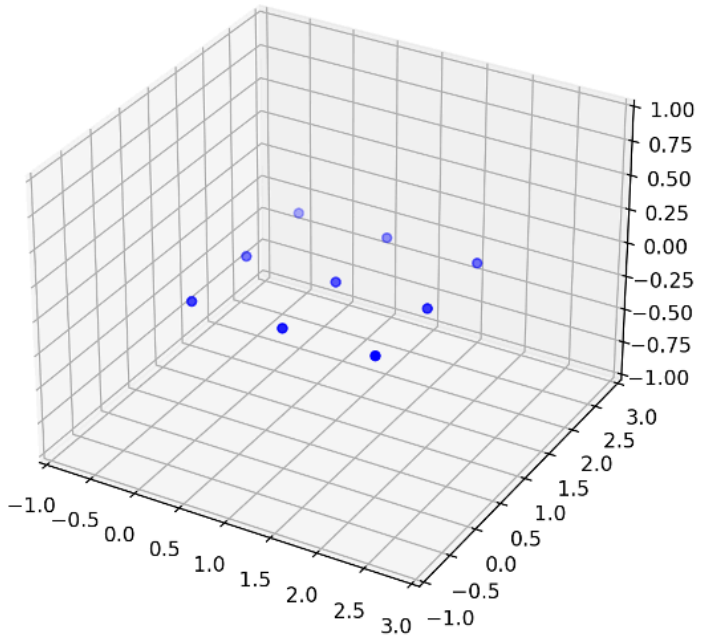

Fig. : Relief correspondant

1er exemple

```
dictionnaire = {'0,0': 1,
                '1,0': -1,
                '2,0': 0,
                '0,1': 0,
                '1,1': 0,
                '2,1': 0,
                '0,2': 1.5,
                '1,2': 0,
                '2,2': 0}
```

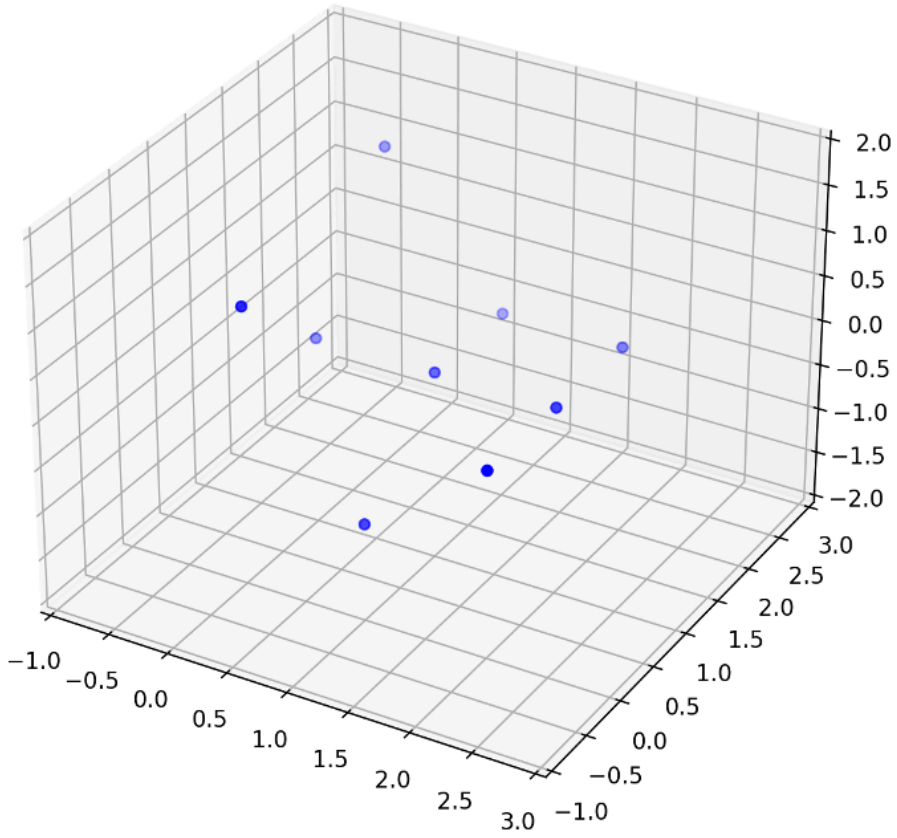

Fig. : Relief correspondant

1er exemple

```
dictionnaire = {'0,0': 1,
                '1,0': -1,
                '2,0': 0,
                '0,1': 0,
                '1,1': 0,
                '2,1': 0,
                '0,2': 1.5,
                '1,2': 0,
                '2,2': 0}
```

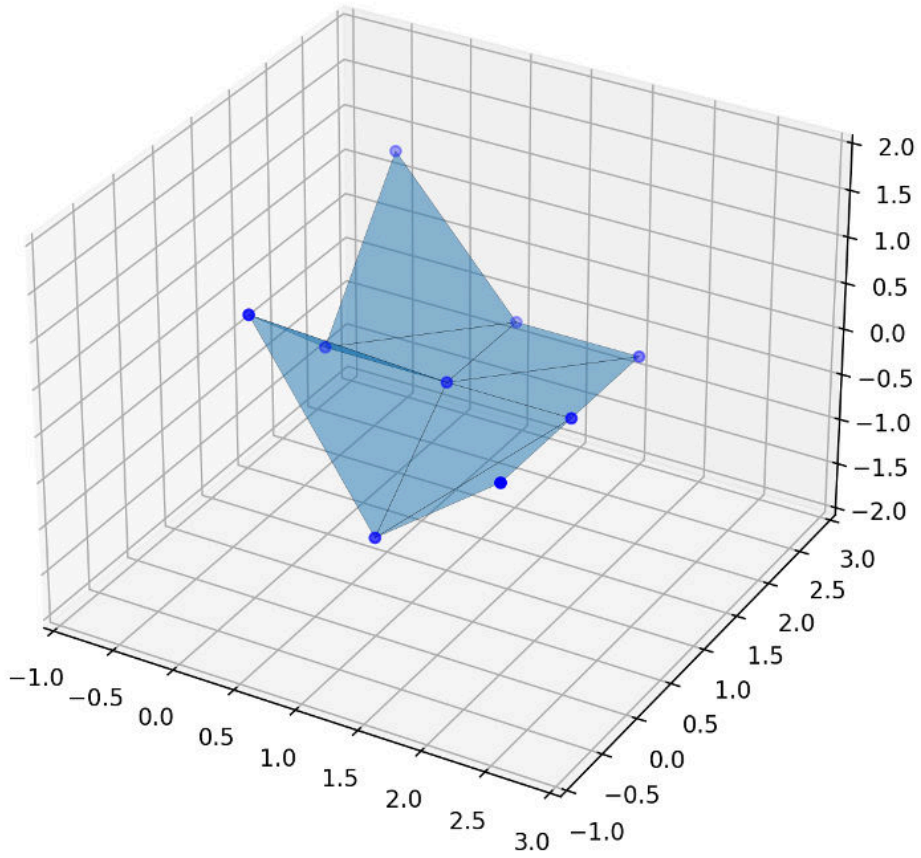

Fig. : Relief correspondant

Ajout du volume dans la modélisation

```
dictionnaire = {
        '0,0': [1, 0],
        '1,0': [-1, 0],
        '2,0': [0, 1],
        '0,1': [0, 1],
        '1,1': [0, 1],
        '2,1': [0, 0],
        '0,2': [1.5, 0],
        '1,2': [0, 0],
        '2,2': [0, 1]'
       }
```

hauteur

volume

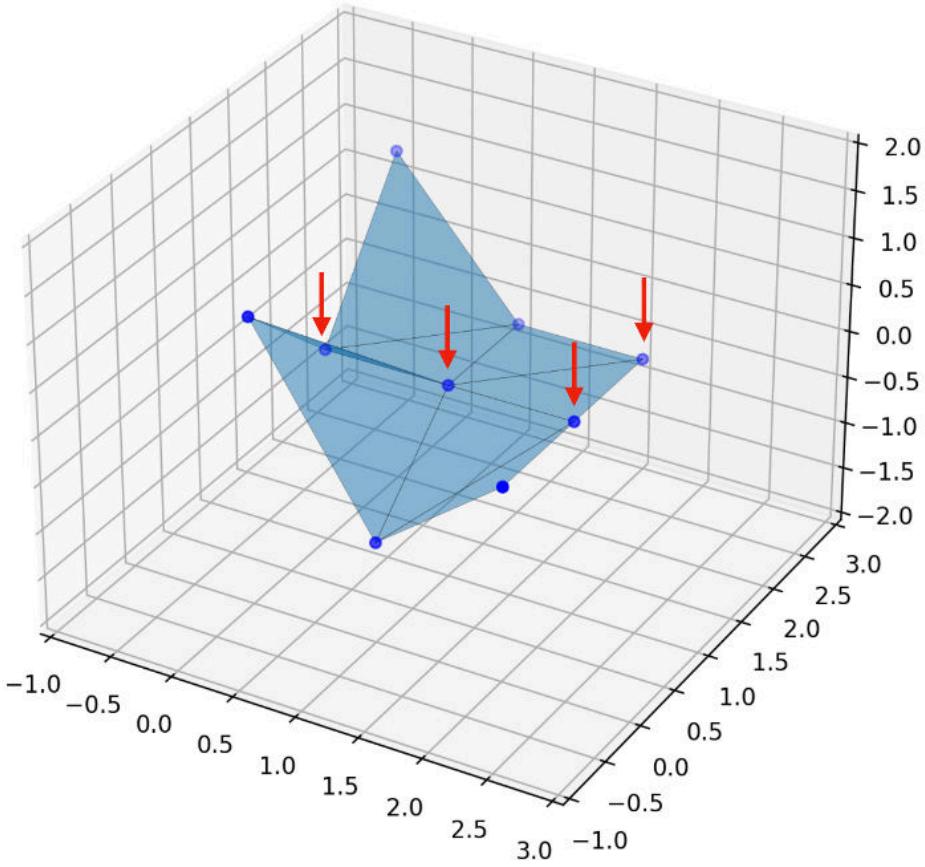

Fig. : Pluie tombant sur le relief

Récupération des altitudes d'un relief réel

Fig. : Site affilié à la NASA : https ://dwtkns.com/srtm30m/

- On récupère une carte des hauteurs
- On forme une matrice carrée à partir de la liste des hauteurs
- On associe à chaque coordonnée du dictionnaire la hauteur lui correspondant

Relief réel

Exemple de donnéees obtenues : 01d9 01d9 01d8 01d4 01d1 01cc 01c7 01c2
Hauteurs correspondantes : 473 473 472 468 465 460 455 450

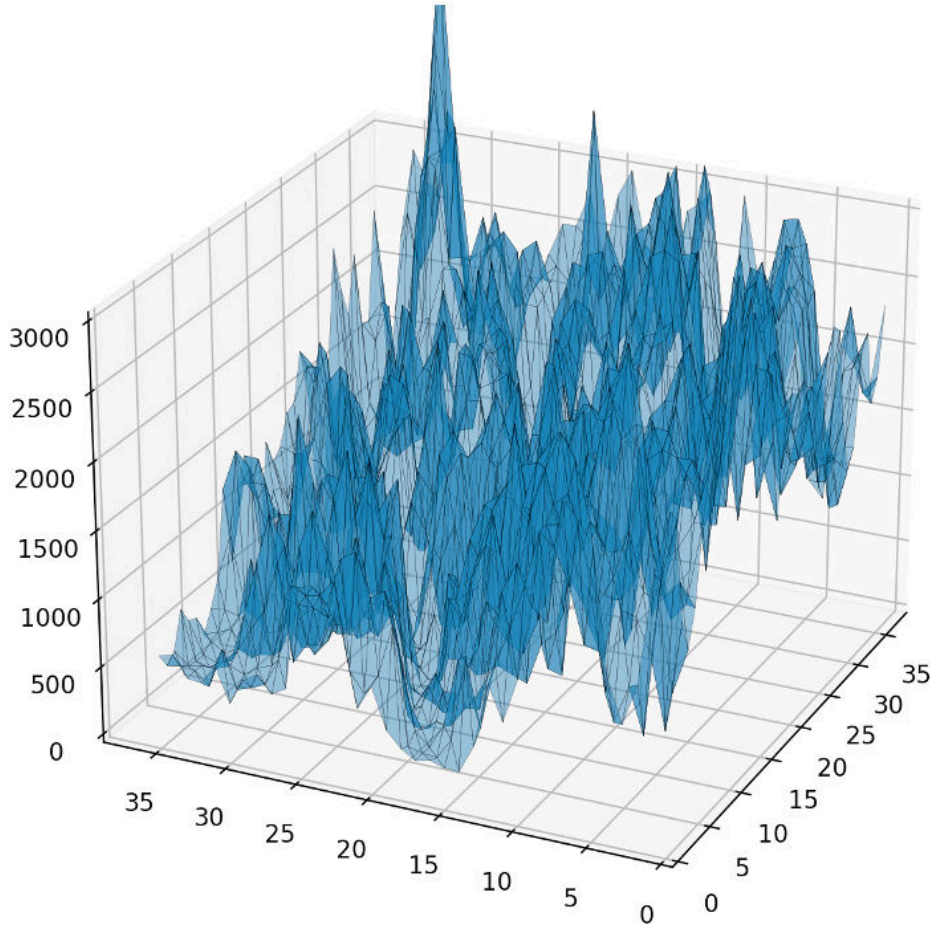

Fig. : Relief se trouvant à N45E006, précision 1/100

Algorithme de lissage

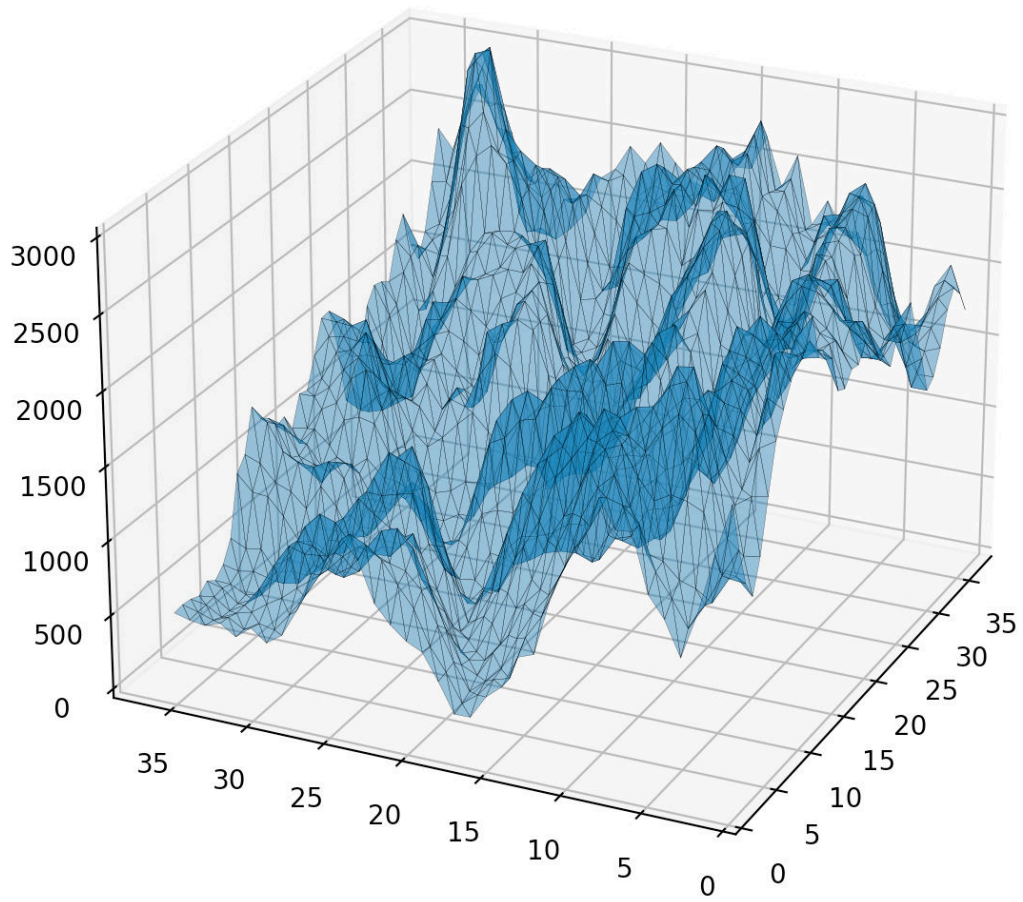

Fig. : Relief se trouvant à N45E006, précision 1/100, lissé

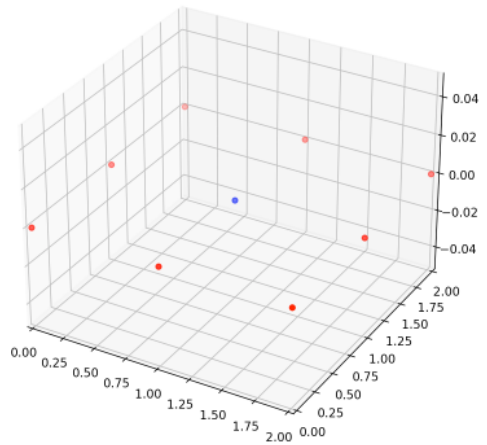

Fig. : Voisins du point bleu

$h_{x,y} = \textit{hauteur du point de coordonnes } (x, y)$

$$h'_{x,y} = \frac{\sum\limits_{\substack{-1 \leq i,j \leq 1 \\ (i,j) \neq (x,y)}} h_{\,x+i,\,y+j}}{8}$$

Ecoulement de l'eau, 1ère méthode

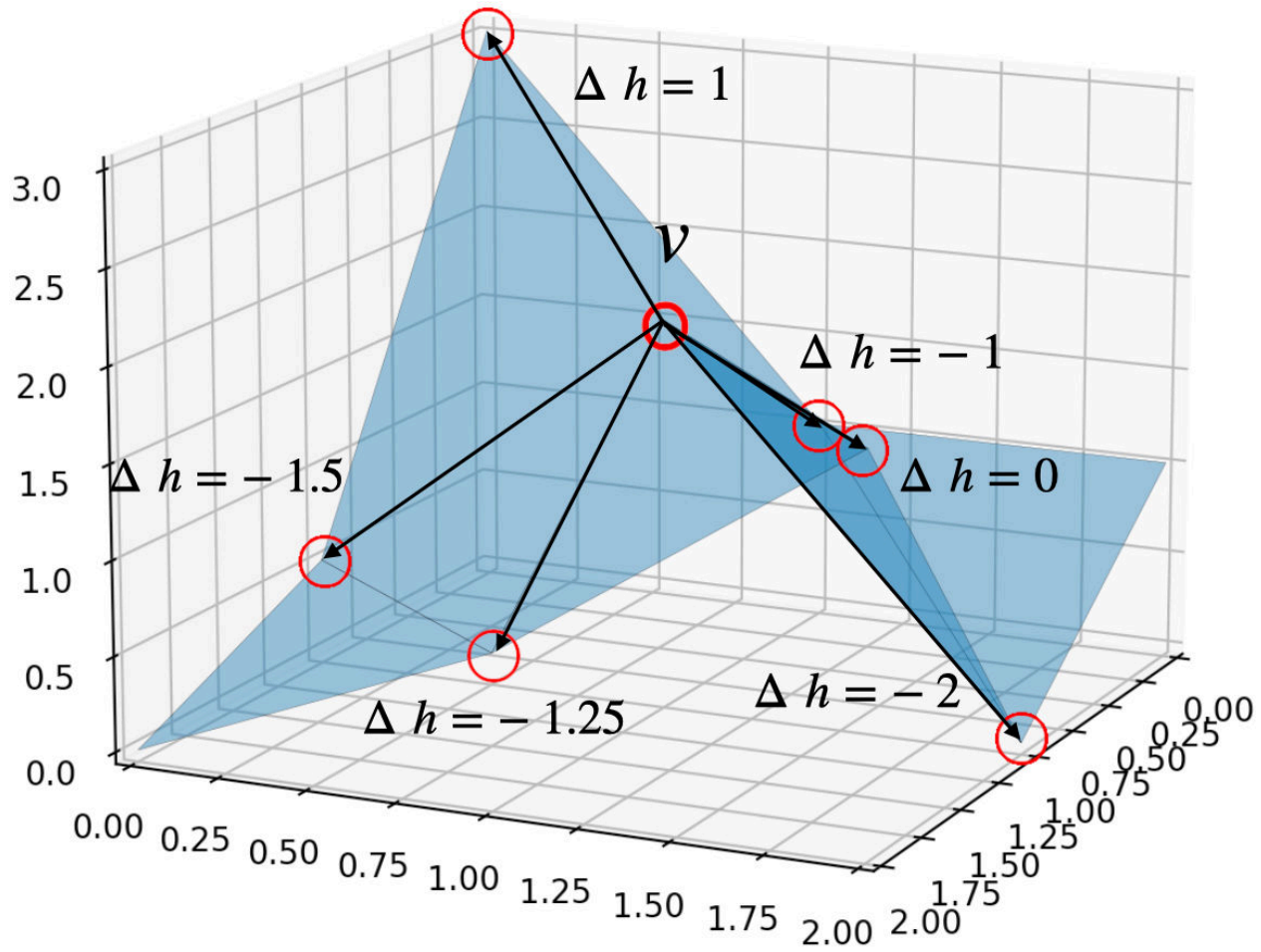

Fig. : Exemple sur un relief fictif

Ecoulement de l'eau, 1ère méthode

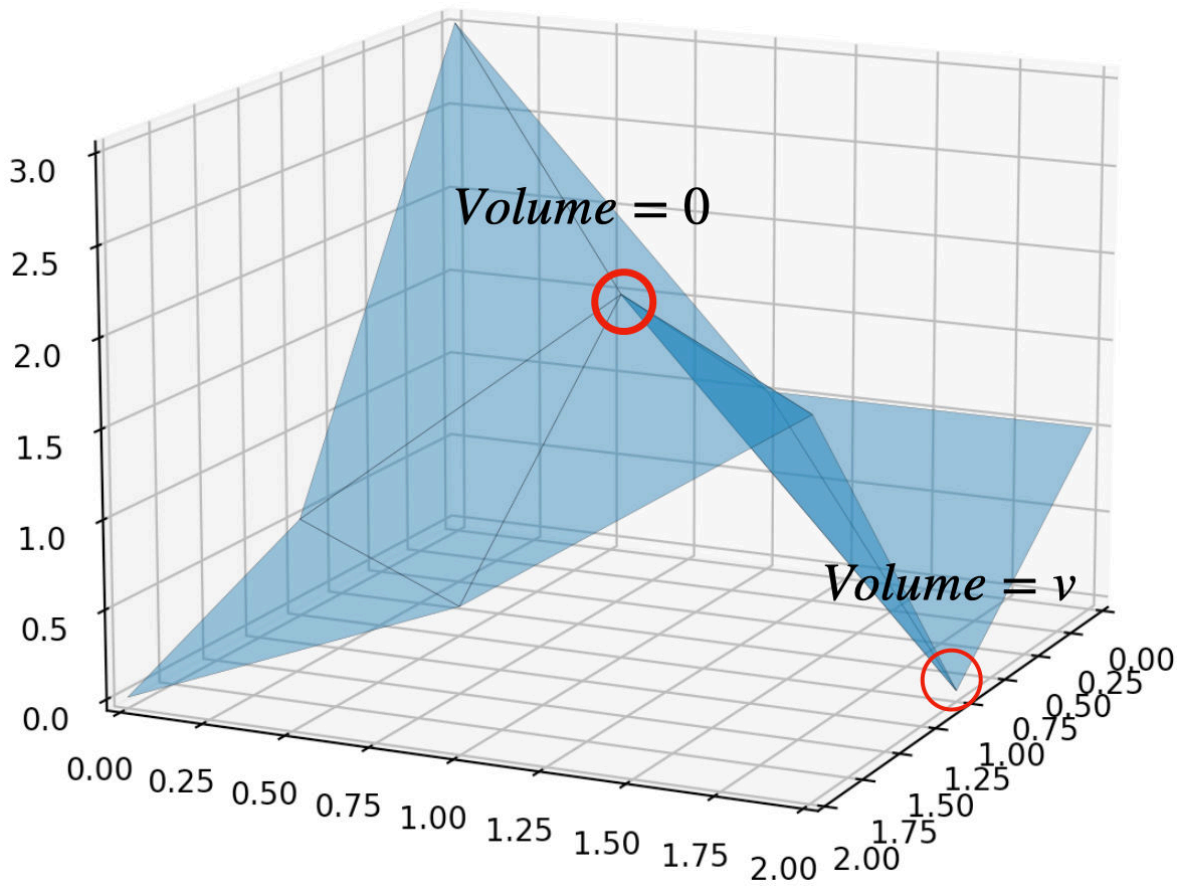

Fig. : Exemple sur un relief fictif

Tout le volume d'eau descend sur le point avec Δh minimal.

Ecoulement de l'eau, 2ème méthode : de manière proportionnelle

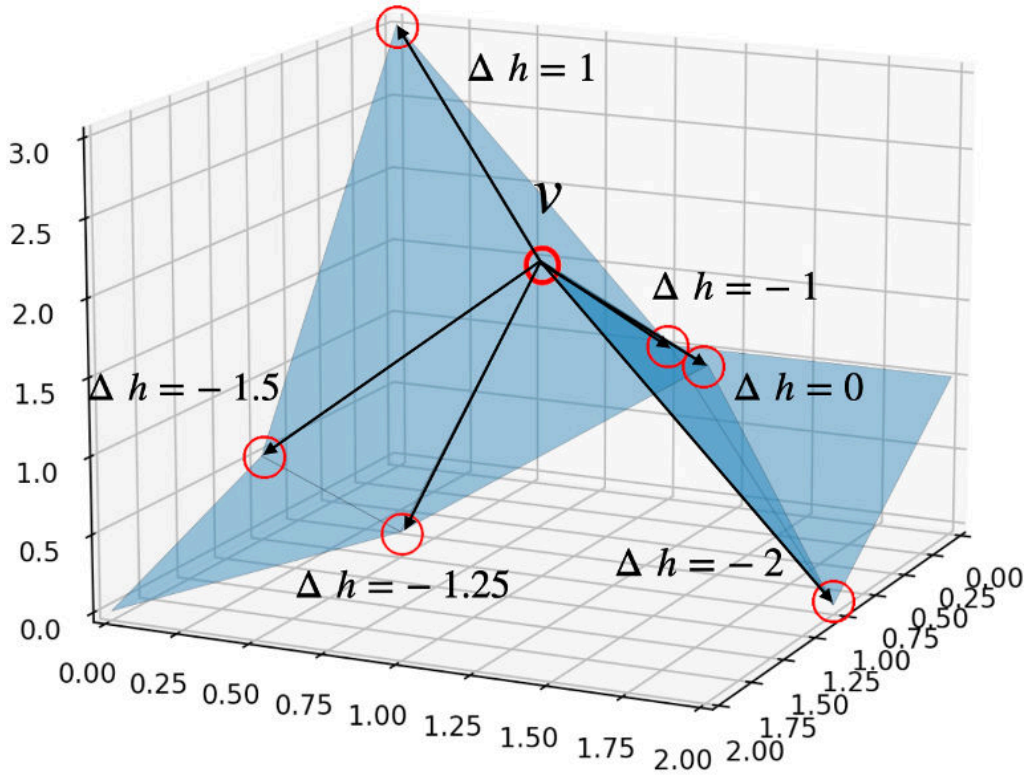

Fig. : exemple sur un relief exemple

- $v_{i,j}$ = volume du point (i,j)
- $\Delta h_{i,j}$ = différence d'altitude au point (i,j)

si $\Delta\, h_{i,j} < 0$: $$\boxed{v_{i,j} = \frac{v \cdot \sqrt{|\Delta\, h_{i,j}|}}{\sum_{\Delta\, h<0} \sqrt{|\Delta\, h|}}}$$

Modélisation de l'écoulement d'un pluie uniforme sur le relief

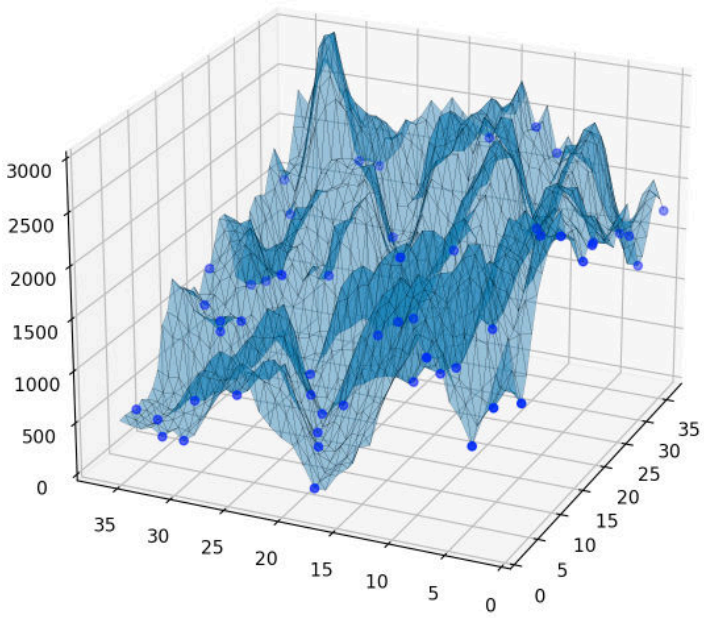

Fig. : Coulée de l'eau sur le relief

- L'eau à l'instant final se situe là où sont les points bleus
- On note le volume d'eau coulant sur chaque arête à chaque instant
- D'où le volume total ayant coulé sur chaque arête à la fin de la modélisation

Arêtes	...	4:18 / 3:17	4:18 / 5:19	4:19 / 5:19	...
Volume d'eau tombée au cours de la modélisation	...	24.028	15.103	14.989	...

Fig. : Volume d'eau ayant coulé sur chaque arête au cours de la modélisation

Plan

Modification du relief ?

Fig. : Canalisations

- Dans l'algorithme : modification de la hauteur du point
- Dans la réalité : création de canalisations

Choix des points à modifier

Arêtes	0:0 / 1:0	0:0 / 0:1	0:0 / 1:1	0:1 / 1:1	0:1 / 0:2	...
Volume d'eau écoulé au cours de la modélisation	0.28	0.29	0.43	0.24	0.53	...

Fig. : Volume d'eau écoulé sur les arêtes au cours de la modélisation

Arêtes	6:30 / 6:31	29:9 / 28:9	27:21 / 26:21	7:30 / 6:30	0:16 / 0:17	...
Volume d'eau écoulé au cours de la modélisation	43.87	41.57	40.85	40.17	39.13	...

Fig. : Volume d'eau trié par ordre décroissant

Arêtes	6:30 / 6:31	29:9 / 28:9	27:21 / 26:21	7:30 / 6:30	0:16 / 0:17	...
Volume d'eau écoulé au cours de la modélisation	43.87	41.57	40.85	40.17	39.13	...

Fig. : Sélection des points à modifier

Modification du relief : évaluation du relief

1. On note un relief de taille n :

$$r = \begin{pmatrix} h_{0,0} \\ h_{0,1} \\ \vdots \\ h_{n-1,n-1} \end{pmatrix}$$

2. On note $\mathcal{V} = \{v \mid v = \text{volume d'eau total tombé sur chaque arête}\}$
3. Pour évaluer r, on écrit la fonction

$$P : \left| \begin{array}{ccc} \mathbb{R}^{n^2} & \longrightarrow & \mathbb{R} \\ r & \longmapsto & P(r) \end{array} \right.$$

$$\text{Avec} \quad P(r) = \sqrt{\left(\sum_{v \in \mathcal{V}} v\right)^2 - \sum_{v \in \mathcal{V}} v^2}$$

Modification du relief : évaluation de la modification

On construit aléatoirement un vecteur M de modification :

- M est de longueur de X notée l
- Soit I un intervalle de $\mathbb{R}$ de hauteurs à ajouter pour la modification du relief
- D'ou $M = (\,i\,)_{i \in I \subset \mathbb{R}}$

Un exemple de M , avec $l = 10$:

$$M = \begin{pmatrix} -11.6 \\ -43.09 \\ \vdots \\ -3.4 \end{pmatrix}$$

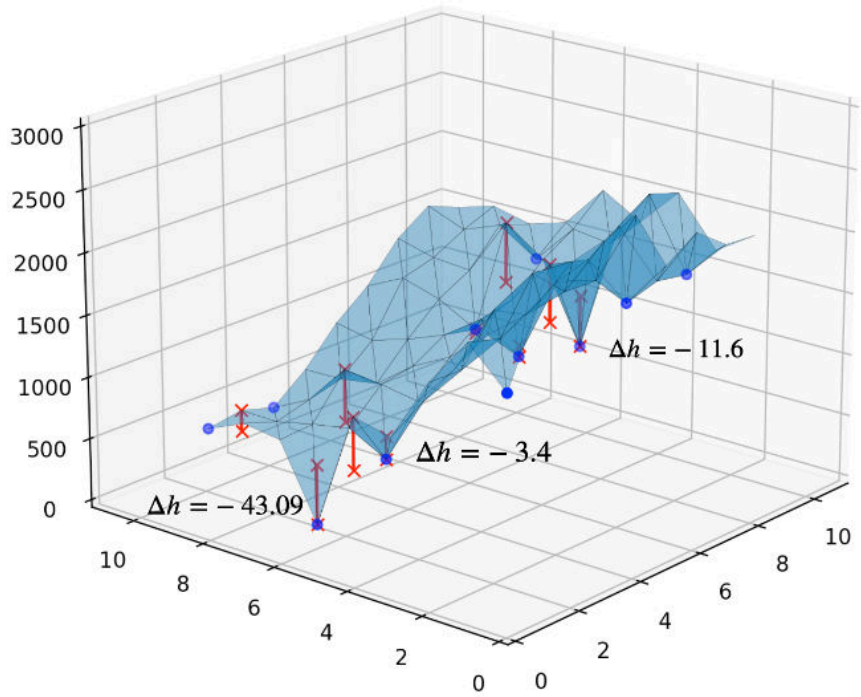

Fig. : Modification du relief

Modification du relief : évaluation de la modification

On dispose du vecteur X de taille l des points à modifier représenté comme : $X = (h_{x,y})_{(x,y)\in[0,n]^2}$

Un exemple de X avec $n = 10$: $\quad X = \begin{pmatrix} h_{3,7} \\ h_{8,6} \\ \vdots \\ h_{5,2} \end{pmatrix}$

$$X^{'} = X + M = \begin{pmatrix} h_{3,7} \\ h_{8,6} \\ \vdots \\ h_{5,2} \end{pmatrix} + \begin{pmatrix} -11.6 \\ -43.09 \\ \vdots \\ -3.4 \end{pmatrix} = \begin{pmatrix} h_{3,7} - 11.6 \\ h_{8,6} - 43.09 \\ \vdots \\ h_{5,2} - 3.4 \end{pmatrix}$$

Modification du relief : évaluation de la modification

D'où, avec $n = 10$ par exemple :

$$r' = \begin{pmatrix} h_{0,0} \\ h_{0,1} \\ \vdots \\ h_{3,7} - 11.6 \\ \vdots \\ h_{8,6} - 43.09 \\ \vdots \\ h_{5,2} - 3.4 \\ \vdots \\ h_{9,9} \end{pmatrix}$$

D'où la fonction d'évaluation

$$\sigma : \left| \begin{array}{ccccc} \mathbb{R}^l & \longrightarrow & \mathbb{R}^{n^2} & \longrightarrow & \mathbb{R} \\ M & \longmapsto & r' & \longmapsto & P(r') \end{array} \right.$$

Minimiser une fonction, analogie en dimension 2

Méthode de Newton : exemple avec $f(x) = x - 4 \cdot ln(x)$

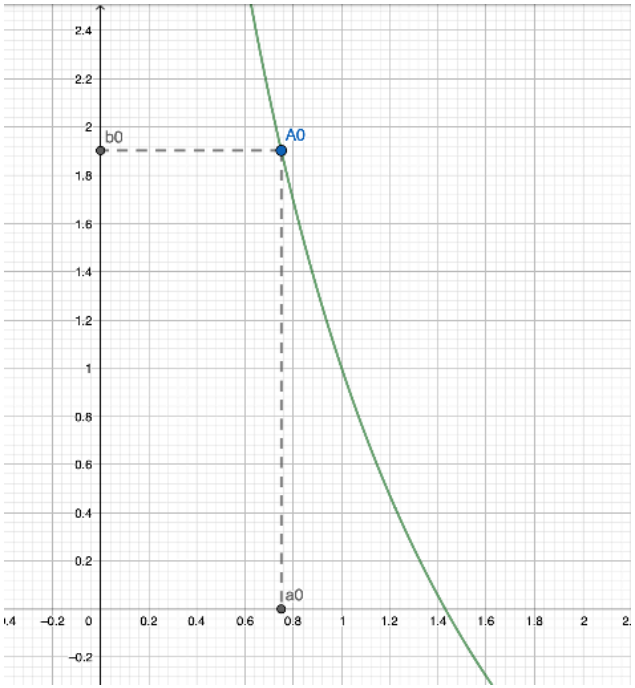

Fig. : 1ère itération

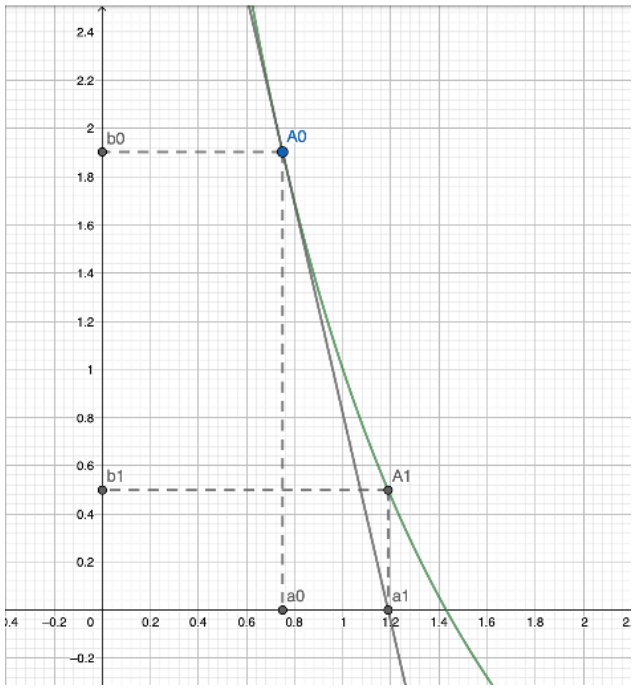

Fig. : 2ème itération

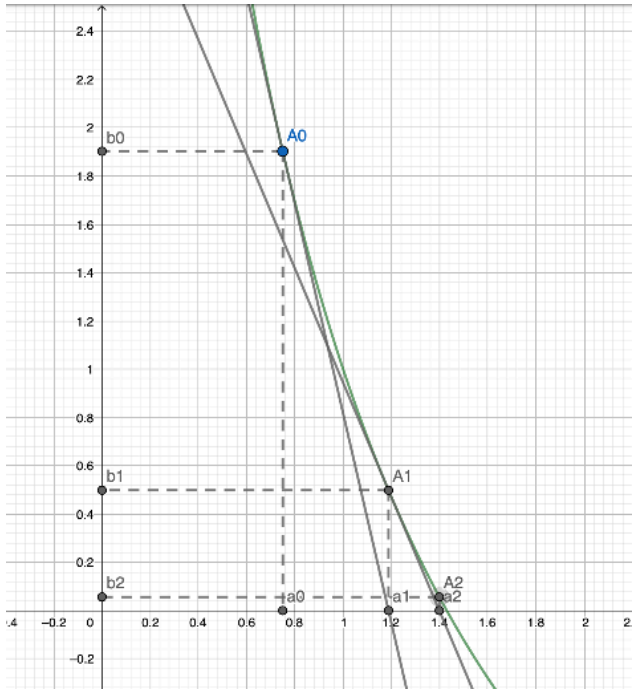

Fig. : 3ème itération

La pente de la tangente au point A_i vaut : $f'(a_i)$

Le point suivant est donc $\boxed{a_{i+1} = a_i - f'(a_i)}$

Minimiser une fonction, analogie en dimension 3

Descente de gradient en dimension 3 : exemple avec
$f(x, y) = 0.005 \cdot (x^3 + y^3)$
Ici le gradient vaut $\overrightarrow{\nabla f(x, y)} = 0.01 \cdot (x^2 \ \vec{e_x} + y^2 \ \vec{e_y})$

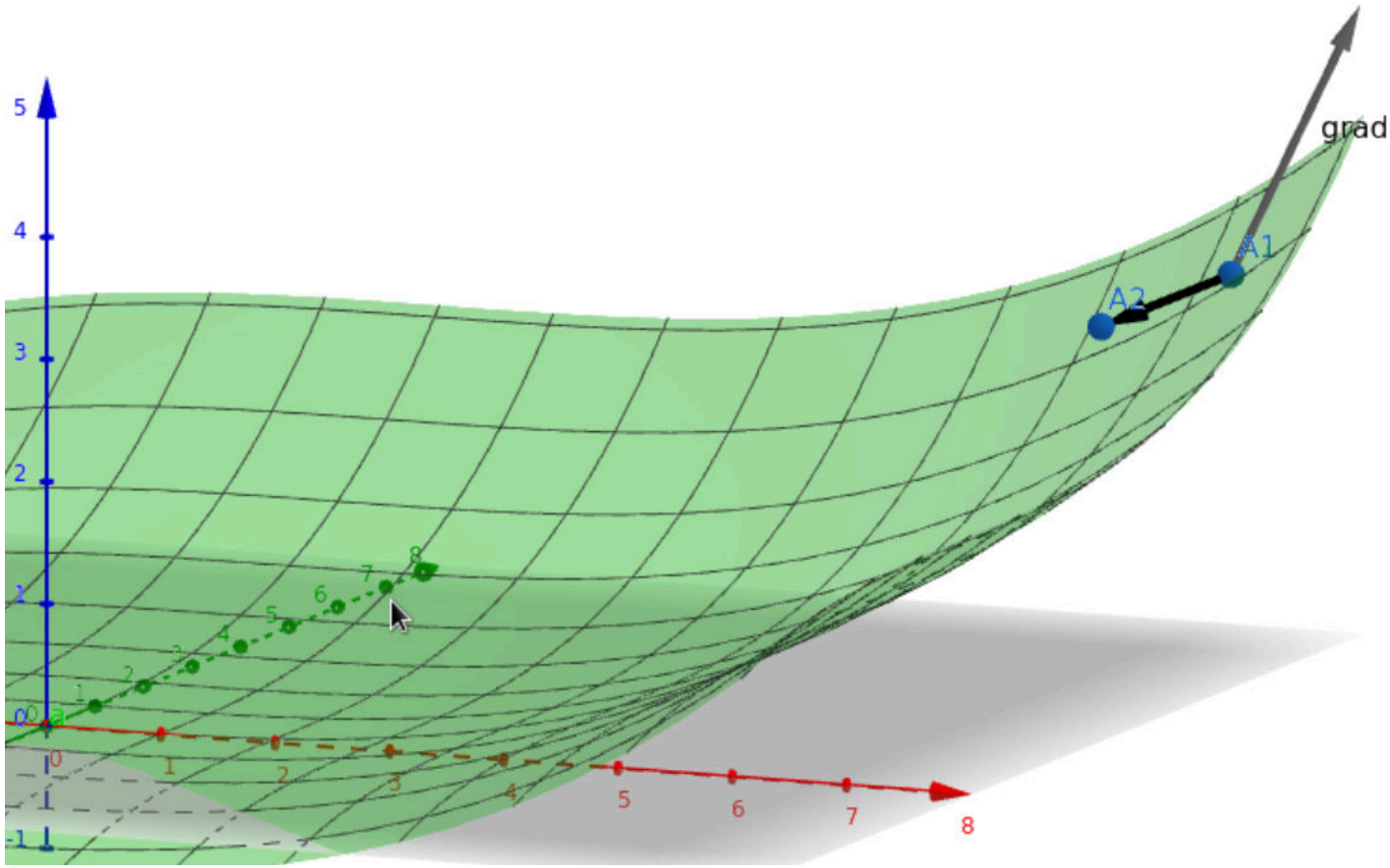

Fig. : 1ère itération

Minimiser une fonction, analogie en dimension 3

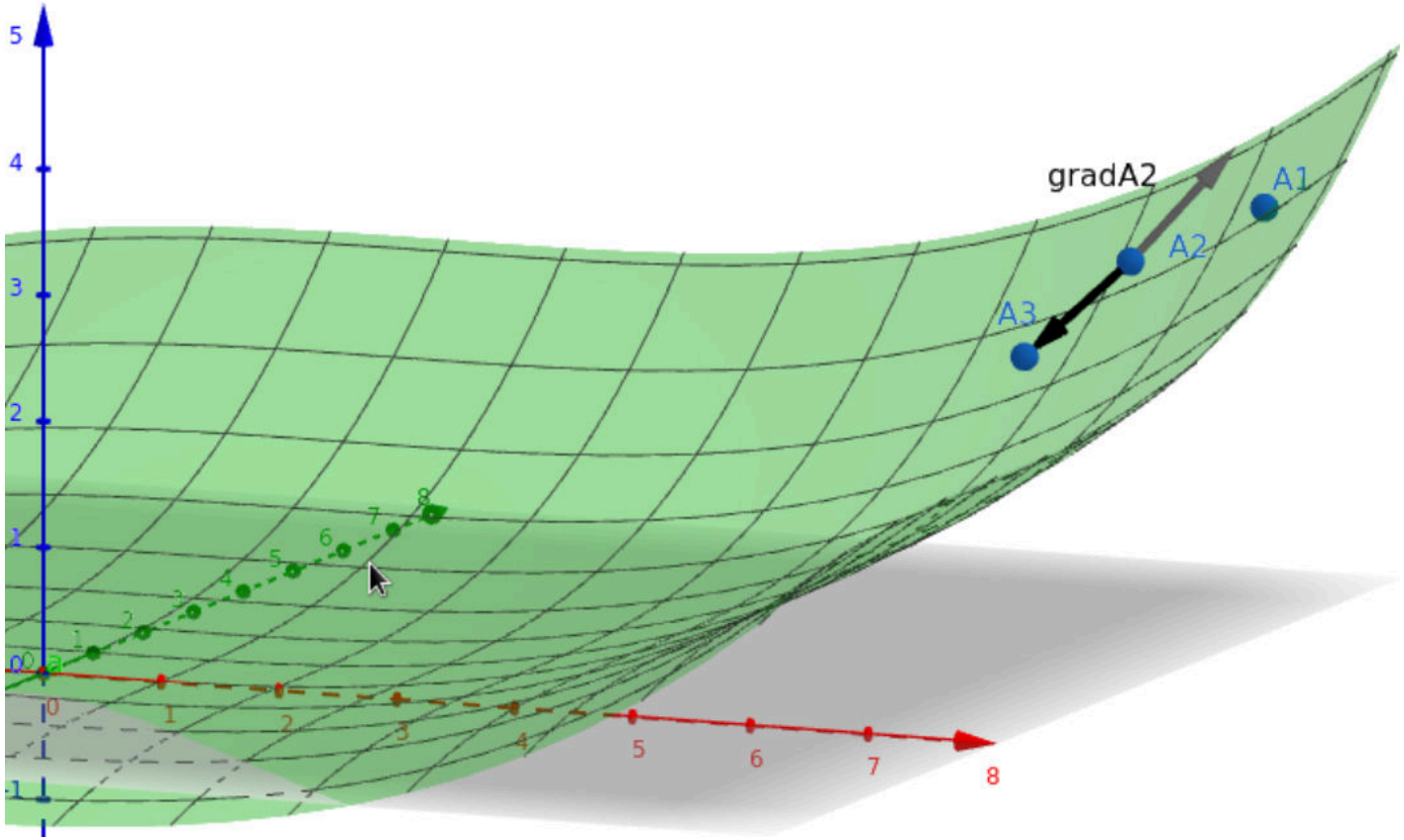

Fig. : 2ème itération

Minimiser une fonction, analogie en dimension 3

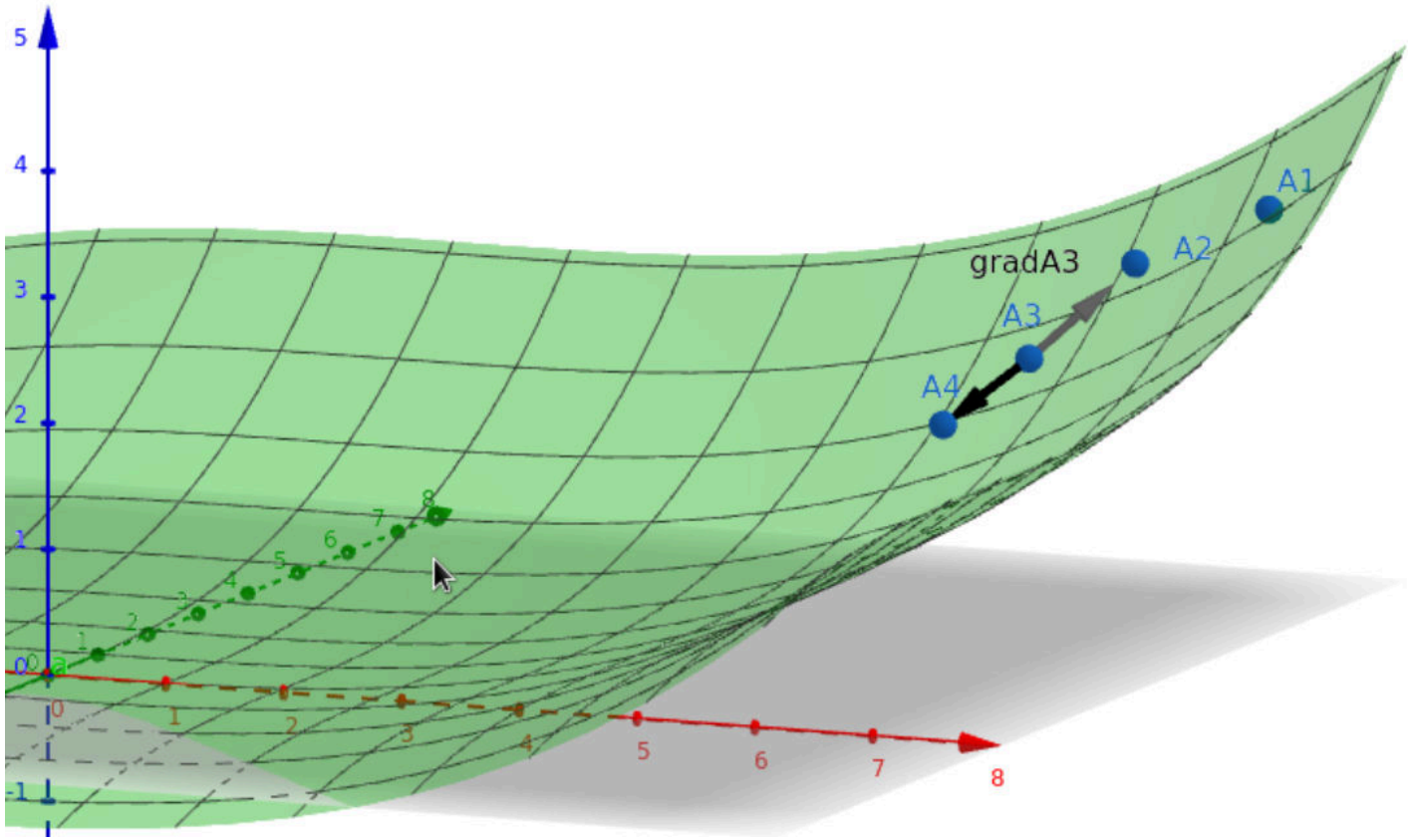

Fig. : 3ème itération

Minimiser une fonction, analogie en dimension 3

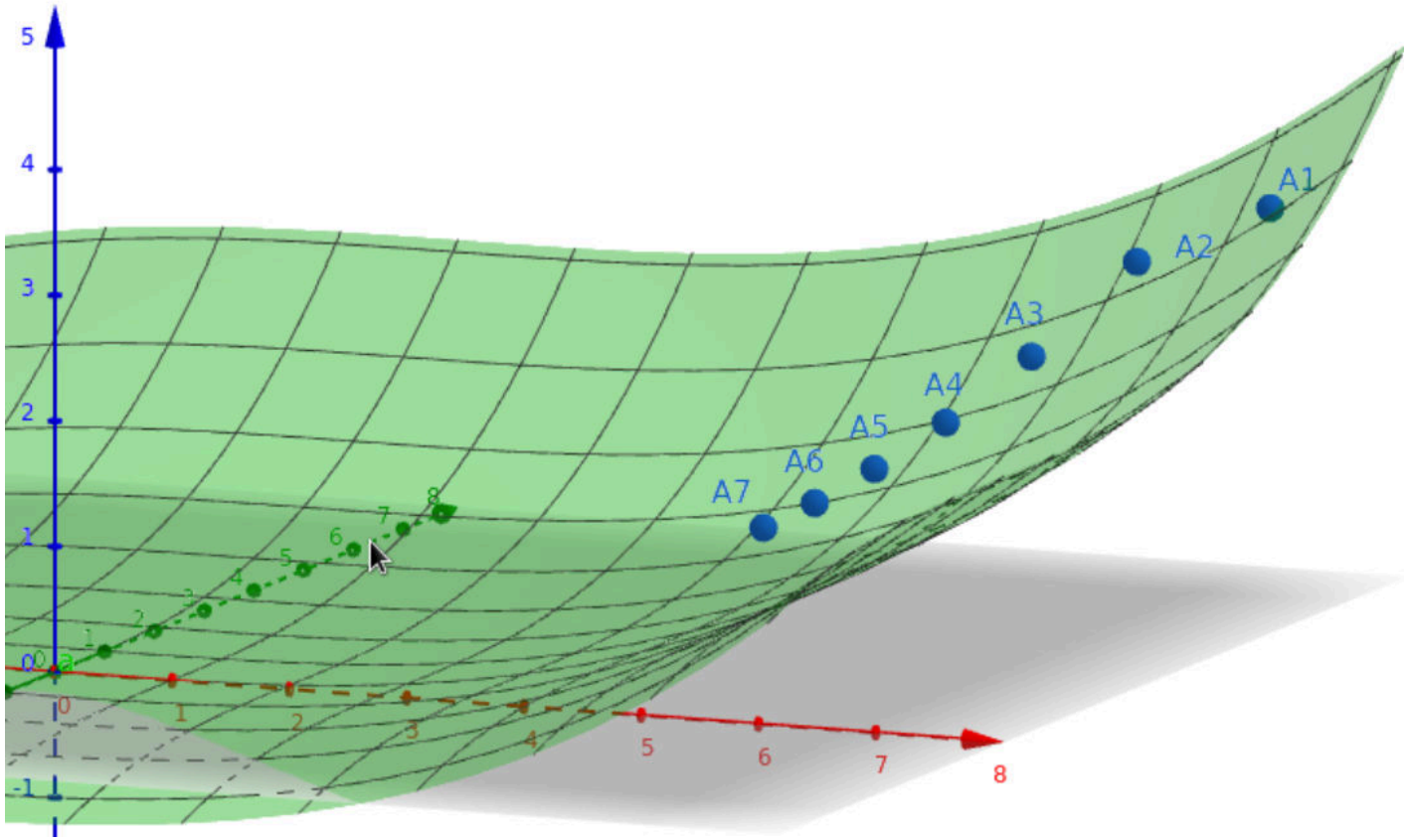

Fig. : 6ème itération

Importance des conditions initiales

Descente de gradient en dimension 3 : exemple avec $f(x,y) = e^{\frac{-x^2}{4}} + e^{\frac{-y^2}{10}}$

Ici $\overrightarrow{\nabla f(x,y)} = -\frac{x}{2} \cdot e^{\frac{-x^2}{4}} \overrightarrow{e_x} + -\frac{y}{5} \cdot e^{\frac{-y^2}{10}} \overrightarrow{e_y}$

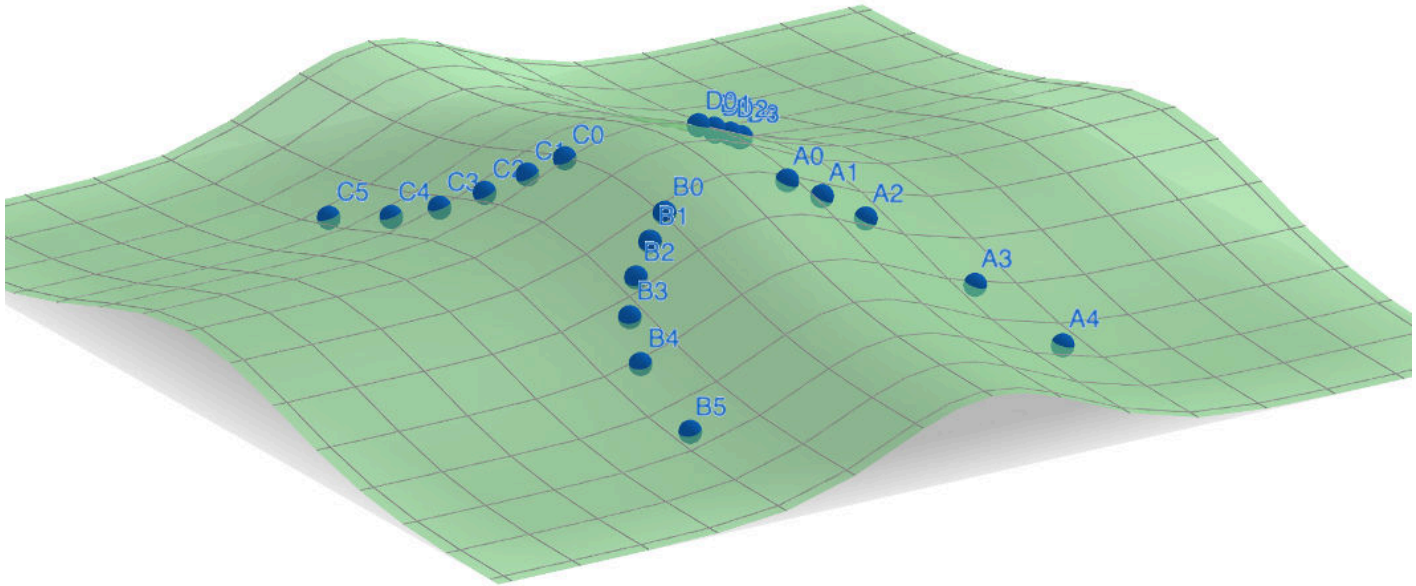

Fig. : Descente de gradient avec plusieurs points initiaux

Minimiser σ

$$\begin{array}{|l|ccccc|}\hline \sigma : & \mathbb{R}^l & \longrightarrow & \mathbb{R}^{n^2} & \longrightarrow & \mathbb{R} \\ & M & \longmapsto & r' & \longmapsto & P(r') \\ \hline \end{array}$$

- On effectue une descente de gradient sur σ , avec comme point initial M généré aléatoirement
- On introduit un coefficient η tel que plus η est grand, plus on suivra loin la direction indiquée par $\overrightarrow{\nabla\sigma(M)}$
- On note alors le prochain point M_0 :

$$M_0 = M - \eta \cdot \overrightarrow{\nabla\sigma(M)}$$

- En itérant :

$$\boxed{\forall i \in \mathbb{N} \quad M_{i+1} = M_i - \eta \cdot \overrightarrow{\nabla\sigma(M_i)}}$$

- On répète ce procédé pour différentes conditions initiales, *ie* différents vecteurs M

Plan

Représentation de la descente de gradient

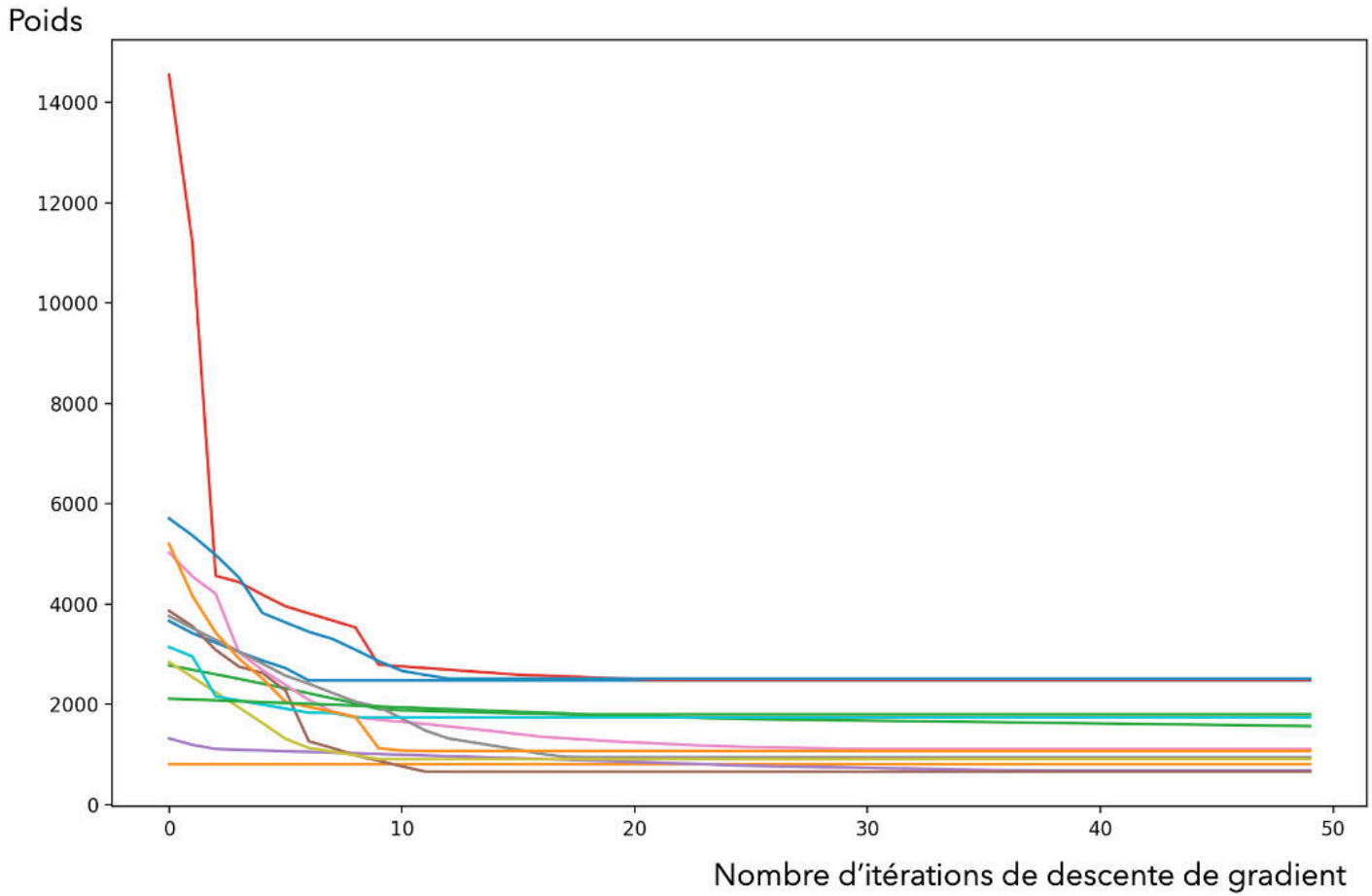

Fig. : Une descente de gradient, précision 1/300

Représentation du relief modifié

Relief affiché avec une précision de 1/300. Choix de 10 points à modifier.

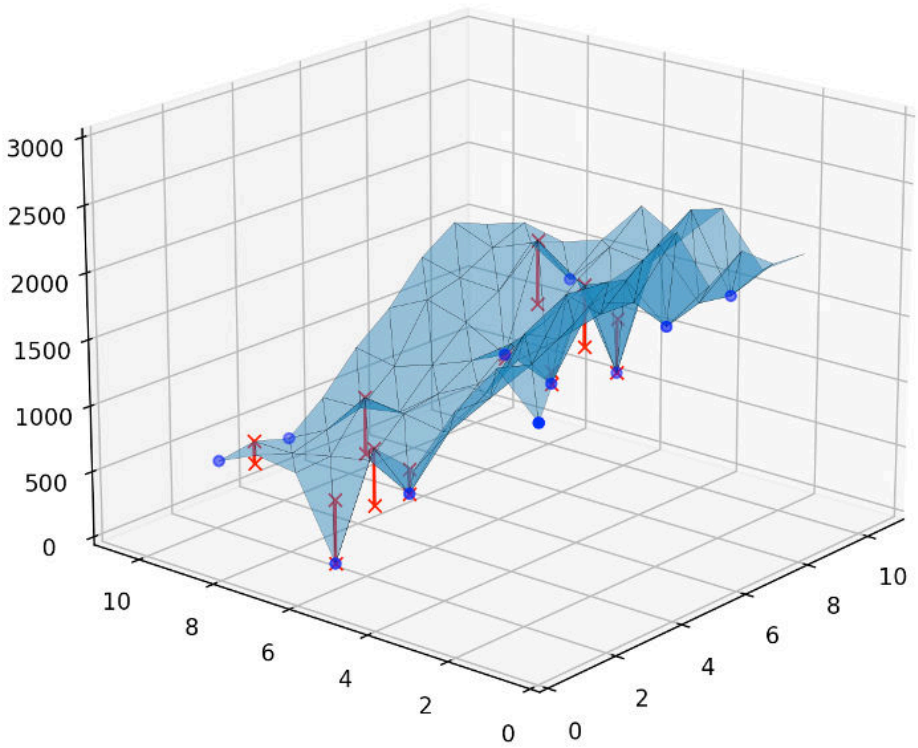

Fig. : Modification du relief

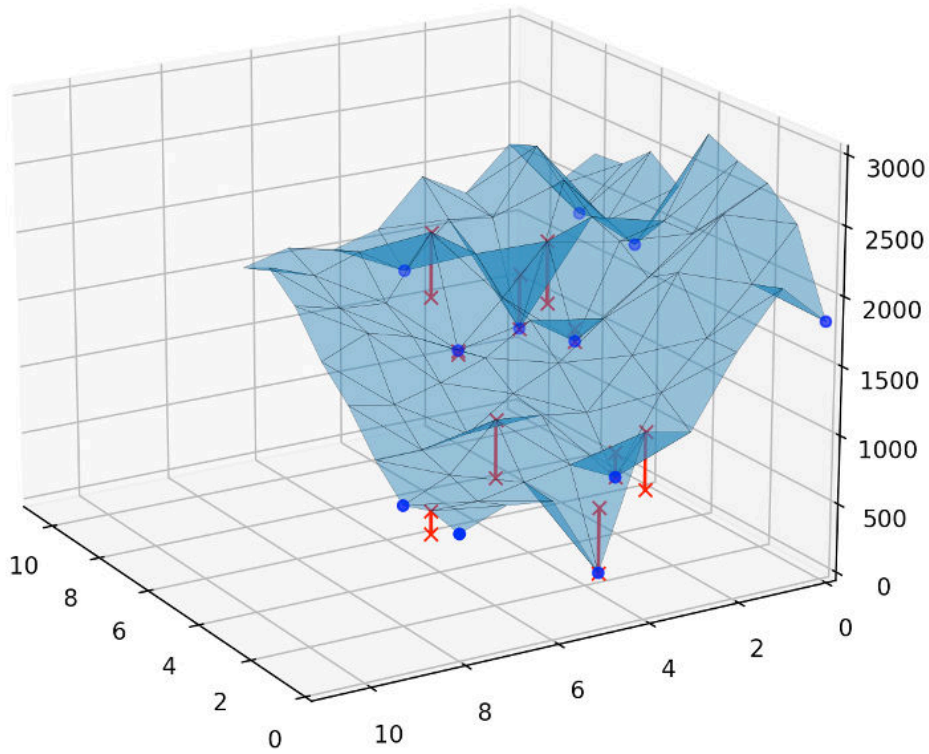

Fig. : Modification du relief

Analyse et limite

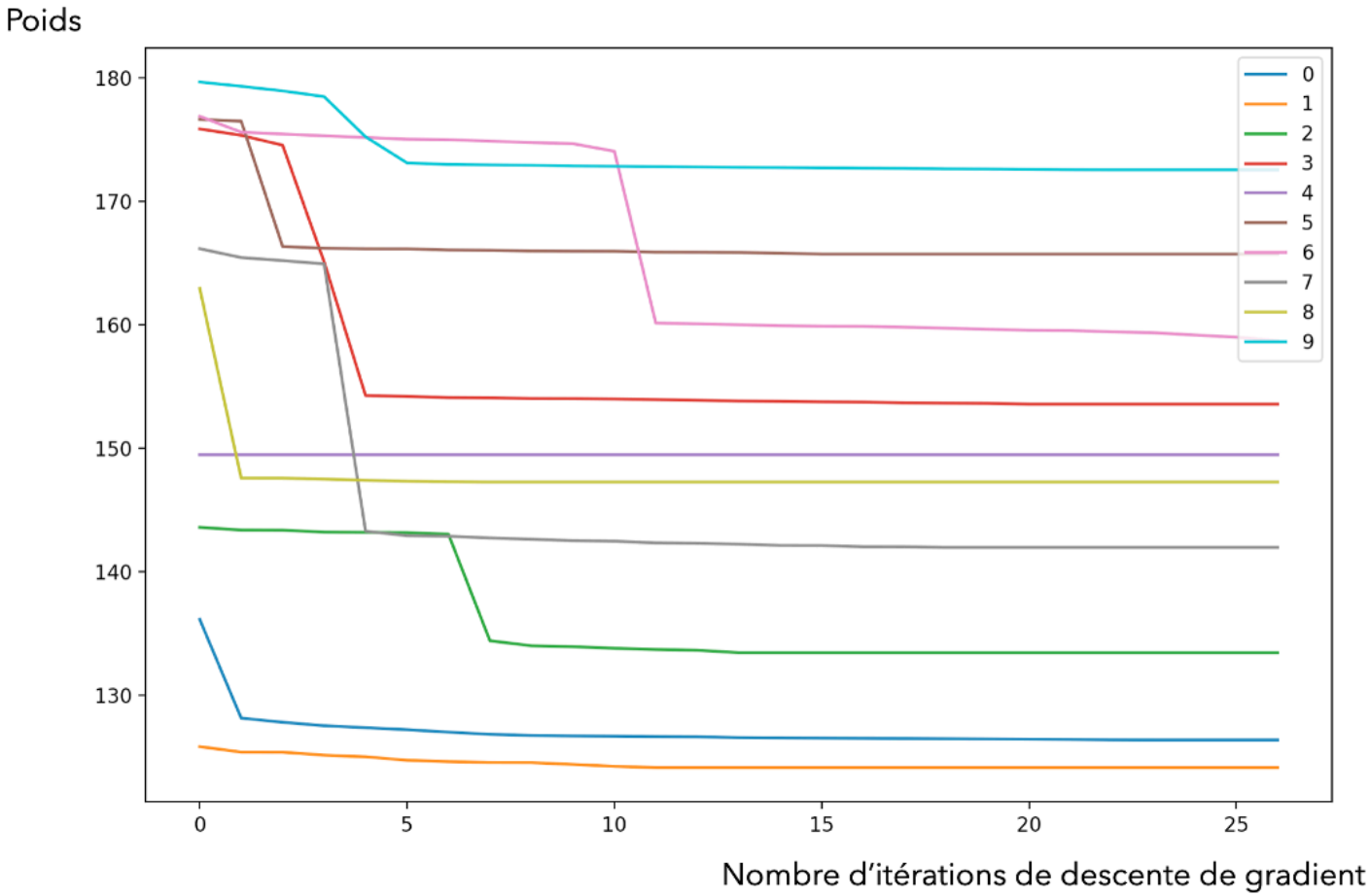

Fig. : Descente de gradient, précision : 1/400

Analyse et limite

Exemple avec $f(x) = cos(x) \cdot sin(x^2)^2 + 1$

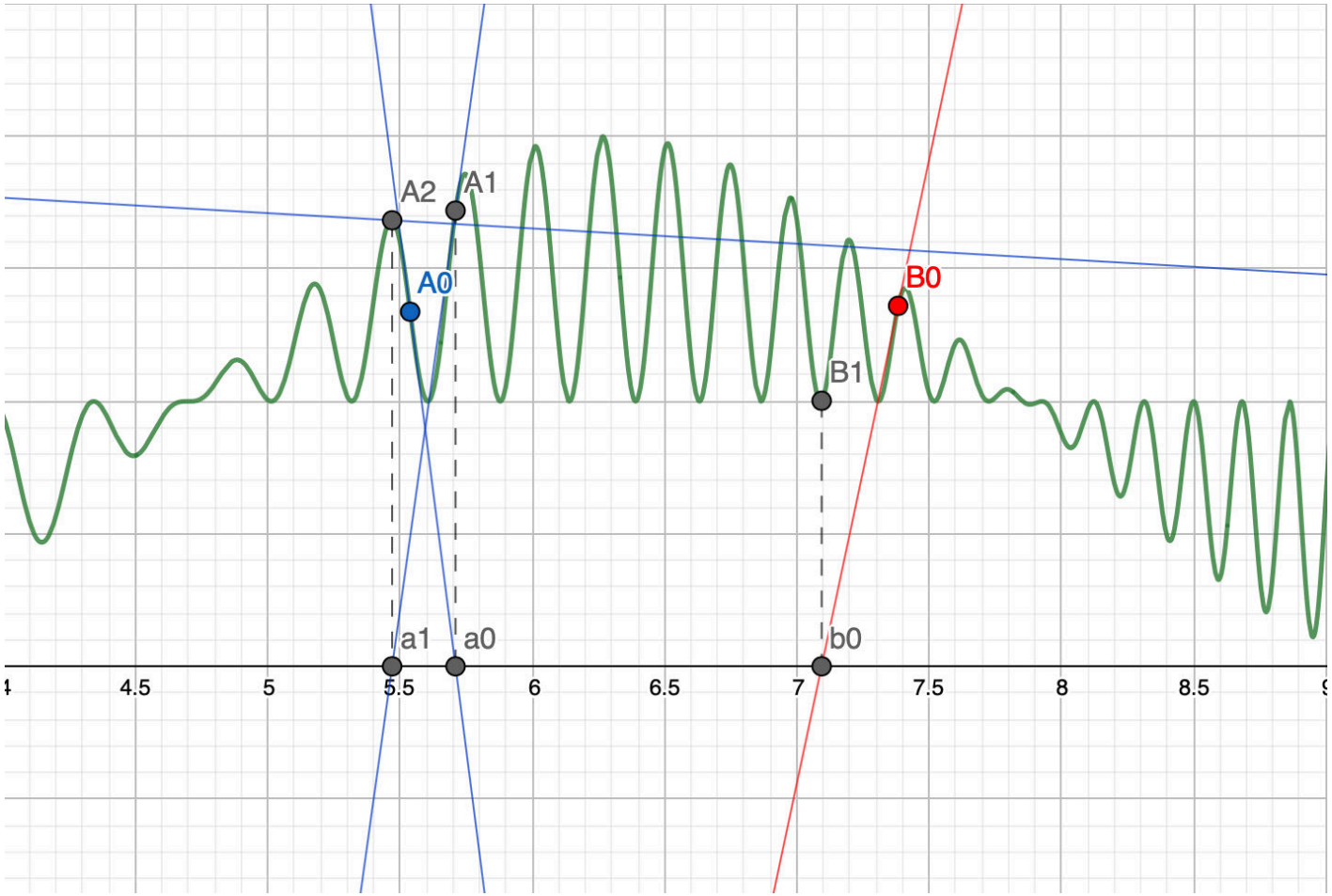

Fig. : Divergence et multiplicité des minima

Représentation de la descente de gradient

Descente de gradient effectuée sur un relief affiché avec une précision de 1/200

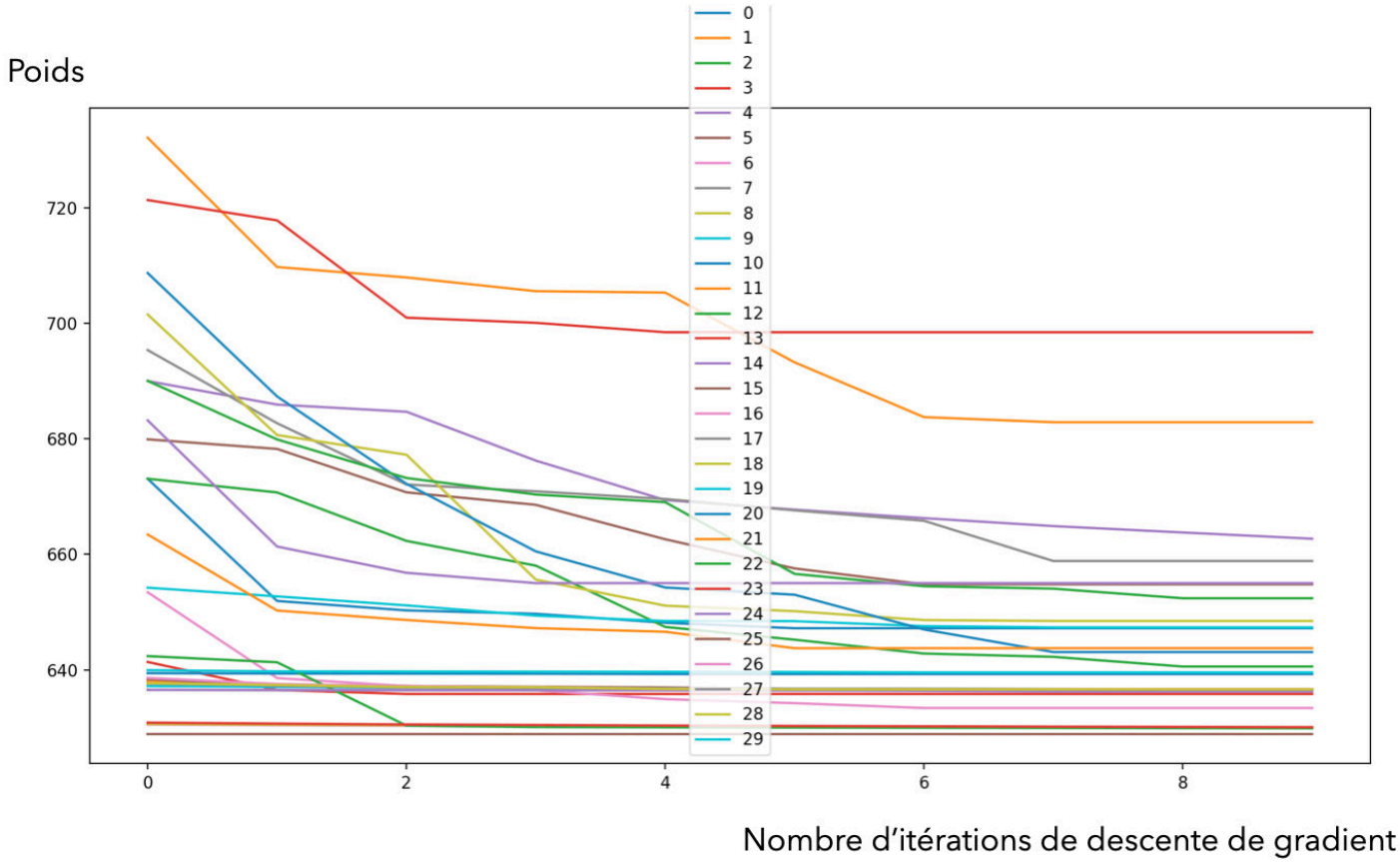

Fig. : Une descente de gradient

Distribution de l'eau

$$\text{Théorème de Bernoulli : } \frac{1}{2}v^2 + gz + \frac{P}{\rho} = cste \quad \Rightarrow \Delta z \propto \Delta(v^2)$$

$$\text{Or} \quad D_m = \iint_{section\ droite} \rho \vec{v} \cdot \vec{dS} = \rho v S$$

$$\Rightarrow \boxed{V \propto \sqrt{|\Delta z|}}$$

Calcul du gradient

```python
def descente_gradient(f, dico: dict, points_a_modifier: list[tuple[int, int]], point: list[int], j, eps=5, maxIter=50) -> \
        tuple[list[int], list]:
    eta = 20000
    norme_grad = 2 * eps + 1
    i = 0
    n = len(point)
    notes = []
    while norme_grad > 0.003 and i < maxIter:
        grad = [0. for _ in range(n)]
        note = f(dico, point, points_a_modifier)
        for k in range(n):
            liste_eps = np.array([0. for _ in range(n)])
            liste_eps[k] = eps
            p1 = plus(point, liste_eps)
            p2 = moins(point, liste_eps)
            grad[k] = (f(dico, p1, points_a_modifier) - f(dico, p2, points_a_modifier)) / 2 * eps
        point = moins(point, fois(grad, eta))
        norme_grad = norme(grad)
        note = f(dico, point, points_a_modifier)
        # éviter d'avoir un gradient qui explose la note
        notes.append(note)
        i += 1
    return point, notes
```

$$\epsilon_k = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

$$p1_k = M + \epsilon_k$$

$$p2_k = M - \epsilon_k$$

$$grad(\sigma(M))_k = \frac{\partial \sigma(M)_k}{\partial k}$$

$$\approx \frac{\sigma(p1) - \sigma(p2)}{2\epsilon}$$

Code : getset

```python
def get_height(a: int, b: int, dico: dict) -> int:
    return dico[str(a) + "," + str(b)][0]


def get_volume(a: int, b: int, dico: dict) -> int:
    return dico[str(a) + "," + str(b)][1]


def get_index(point: str) -> tuple[int, int]:
    liste = point.split(",")
    return int(liste[0]), int(liste[1])


def get_coords(dico: dict, point: str) -> list:
    x, y = get_index(point)
    return [x, y, get_height(x, y, dico)]


def get_indice(x, liste: list) -> int or str:
    for i in range(len(liste)):
        if liste[i] == x:
            return i
    return "l'objet n'est pas dans la liste"


def set_to_string(coord: tuple[int, int]) -> str:
    return str(coord[0]) + ',' + str(coord[1])


def set_volume(a: int, b: int, v: int, dico: dict) -> None:
    dico[str(a) + "," + str(b)][1] = v

```

Code : initialisation du relief

```python
from getset import *
import random
import math as m
from read_files import read_elevation_from_file
import numpy as np


lon = 45
lat = 6
file = f"N{lon}E00{lat}.hgt"
SAMPLES = 3601
precision = 1 / 100
taille = m.floor(SAMPLES * precision)
hauteur = 3000


def creer_sommets(nb: int) -> dict[str, list[int]]:
    """Créer le dictionnaire de points consitituant le flanc de montagne simplifié ( une pente constante )"""
    sommets = {}
    inf, sup = nb, nb + 5
    for i in range(nb):
        inf -= 1
        sup -= 1
        for j in range(nb):
            z = random.randint(inf, sup)
            sommets[str(i) + "," + str(j)] = [z, 1]
    return sommets


def creer_sommets_reels(nb: int, v: int) -> dict:
    """Créer le dictionnaire de points consitituant le flanc de montagne à partir du fichier de la NASA"""
    sommets = {}
    pas = np.floor(SAMPLES / precision)
    for i in range(nb):
        for j in range(nb):
            z = read_elevation_from_file(file, lon + pas * i / 3600, lat + pas * j / 3600)
            sommets[str(i) + "," + str(j)] = [z, v]
    return sommets


def lissage(dico: dict) -> dict[str, list[int]]:
    """Lisser le relief"""
    nb = int(np.sqrt(len(dico))) - 1
    nouveau_dico = {}
    for i in range(0, nb + 1):
        for j in range(0, nb + 1):

            if i != 0 and i != nb and j != 0 and j != nb:  # points quelconques
                pt1 = get_height(i + 1, j, dico)
                pt2 = get_height(i - 1, j, dico)
                pt3 = get_height(i, j + 1, dico)
                pt4 = get_height(i, j - 1, dico)
                pt5 = get_height(i + 1, j + 1, dico)
                pt6 = get_height(i + 1, j - 1, dico)
                pt7 = get_height(i - 1, j + 1, dico)
                pt8 = get_height(i - 1, j - 1, dico)
                moyenne = (pt1 + pt2 + pt3 + pt4 + pt5 + pt6 + pt7 + pt8) // 8
                nouveau_dico[str(i) + "," + str(j)] = [moyenne, 1]
```

Code : initialisation du relief

```
59
60          elif i == 0 and j == 0:  # coin haut gauche
61              pt1 = get_height(i + 1, j, dico)
62              pt2 = get_height(i, j + 1, dico)
63              pt3 = get_height(i + 1, j + 1, dico)
64              moyenne = (pt1 + pt2 + pt3) // 3
65              nouveau_dico[str(i) + "," + str(j)] = [moyenne, 1]
66
67          elif i == 0 and j == nb:  # coin bas gauche
68              pt1 = get_height(i + 1, j, dico)
69              pt2 = get_height(i, j - 1, dico)
70              pt3 = get_height(i + 1, j - 1, dico)
71              moyenne = (pt1 + pt2 + pt3) // 3
72              nouveau_dico[str(i) + "," + str(j)] = [moyenne, 1]
73
74          elif i == nb and j == 0:  # coin haut droite
75              pt1 = get_height(i - 1, j, dico)
76              pt2 = get_height(i, j + 1, dico)
77              pt3 = get_height(i - 1, j + 1, dico)
78              moyenne = (pt1 + pt2 + pt3) // 3
79              nouveau_dico[str(i) + "," + str(j)] = [moyenne, 1]
80
81          elif i == nb and j == nb:  # coin bas droite
82              pt1 = get_height(i - 1, j, dico)
83              pt2 = get_height(i, j - 1, dico)
84              pt3 = get_height(i - 1, j - 1, dico)
85              moyenne = (pt1 + pt2 + pt3) // 3
86              nouveau_dico[str(i) + "," + str(j)] = [moyenne, 1]
87
88          elif i == 0 and j != nb and j != 0:  # côté gauche
89              pt1 = get_height(i + 1, j, dico)
90              pt2 = get_height(i, j + 1, dico)
91              pt3 = get_height(i, j - 1, dico)
92              pt4 = get_height(i + 1, j + 1, dico)
93              pt5 = get_height(i + 1, j - 1, dico)
94              moyenne = (pt1 + pt2 + pt3 + pt4 + pt5) // 5
95              nouveau_dico[str(i) + "," + str(j)] = [moyenne, 1]
96
97          elif i == nb and j != nb and j != 0:  # côté droit
98              pt1 = get_height(i - 1, j, dico)
99              pt2 = get_height(i, j + 1, dico)
100             pt3 = get_height(i, j - 1, dico)
101             pt4 = get_height(i - 1, j + 1, dico)
102             pt5 = get_height(i - 1, j - 1, dico)
103             moyenne = (pt1 + pt2 + pt3 + pt4 + pt5) // 5
104             nouveau_dico[str(i) + "," + str(j)] = [moyenne, 1]
105
106         elif j == 0 and i != 0 and i != nb:  # côté supérieur
107             pt1 = get_height(i + 1, j, dico)
108             pt2 = get_height(i - 1, j, dico)
109             pt3 = get_height(i, j + 1, dico)
110             pt4 = get_height(i + 1, j + 1, dico)
111             pt5 = get_height(i - 1, j + 1, dico)
112             moyenne = (pt1 + pt2 + pt3 + pt4 + pt5) // 5
113             nouveau_dico[str(i) + "," + str(j)] = [moyenne, 1]
114
115         elif j == nb and i != 0 and i != nb:  # côté inférieur
116             pt1 = get_height(i + 1, j, dico)
117             pt2 = get_height(i - 1, j, dico)
118             pt3 = get_height(i, j - 1, dico)
119             pt4 = get_height(i + 1, j - 1, dico)
120             pt5 = get_height(i - 1, j - 1, dico)
121             moyenne = (pt1 + pt2 + pt3 + pt4 + pt5) // 5
122             nouveau_dico[str(i) + "," + str(j)] = [moyenne, 1]
123
124     return nouveau_dico
125
```

Code : initialisation du relief

```
125
126
127 def creer_triangles(dico: dict) -> list[list[tuple[int, int, int]]]:
128     """Créer les triangles à partir du dictionnaire de points"""
129     triangles = []
130     nb = int(np.sqrt(len(dico))) - 1
131     for i in range(nb):
132         for j in range(nb):
133             triangles.append([(i, j, get_height(i, j, dico)),
134                               (i + 1, j, get_height(i + 1, j, dico)),
135                               (i + 1, j + 1, get_height(i + 1, j + 1, dico))])
136             triangles.append([(i, j, get_height(i, j, dico)),
137                               (i, j + 1, get_height(i, j + 1, dico)),
138                               (i + 1, j + 1, get_height(i + 1, j + 1, dico))])
139     return triangles
140
```

Code : affichage

```python
from initialisation import *
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d.art3d import Poly3DCollection


def trouver_coords(pts: dict) -> list[list[int]]:  # utile pour scatter les points de la montagne
    """Transformer les coordonées des points du dictionnaire en une forme adaptée à matplotlib pour les afficher"""
    X, Y, Z, V = [], [], [], []
    for i in pts.keys():
        x, y = get_index(i)
        z = get_height(x, y, pts)
        v = get_volume(x, y, pts)
        X.append(x)
        Y.append(y)
        Z.append(z)
        V.append(v)
    return [X, Y, Z, V]


def sequence(V):
    """former les couleurs des points en fonction du volume"""
    L = []
    for v in V:
        if v > 10e-2:
            L.append((0, 0, 1))
        else:
            L.append((1, 0, 0, 0))
    return L


def afficher(dico: dict, dico_ref, points_a_modifier):
    montagnes = creer_triangles(dico)
    # création du repère
    fig = plt.figure()
    ax = plt.axes(projection='3d')
    ax.set_xlim([0, taille + 2])
    ax.set_ylim([0, taille + 2])
    ax.set_zlim([0, hauteur])
    ax.set_title('Flanc de montagne')

    # affichage flanc de montagne
    ax.add_collection(Poly3DCollection(montagnes, edgecolors="black", linewidths=0.1, alpha=0.5))

    # afficher tous les points
    X_pts, Y_pts, Z_pts, V_pts = trouver_coords(dico)
    ax.scatter(X_pts, Y_pts, Z_pts, c=sequence(V_pts))

    for point in points_a_modifier:
        x, y = point
        z_new = get_height(x, y, dico)
        z_old = get_height(x, y, dico_ref)
        plt.plot([x, x], [y, y], [z_old, z_new], color="red")
        plt.plot(x, y, z_old, marker='x', color="red")
        plt.plot(x, y, z_new, marker='x', color="red")

```

Code : fichier principal

```python
from copy import deepcopy
import time
from read_files import *
from traitement_des_donees import *
from affichage import *

t1 = time.time()


def norme(vect):
    return np.sqrt(sum([coord ** 2 for coord in vect]))


def moins(l1, l2):
    liste = []
    # fais la difference de deux listes de memes longueurs
    for k in range(len(l1)):
        liste.append(l1[k] - l2[k])
    return liste


def plus(l1, l2):
    liste = []
    # fais la somme de deux listes de memes longueurs
    for k in range(len(l1)):
        liste.append(l1[k] + l2[k])
    return liste


def fois(l0: list, k):
    liste = []
    # multiplie un scalaire à une liste
    for i in range(len(l0)):
        liste.append(l0[i] * k)
    return liste


def plus_dico(dico, points_a_modifier, point):
    # modifie les altitudes du dictionnaire en place
    n = len(point)
    # ajouter les modifications du point
    for i in range(n):
        dico[set_to_string(tuple(points_a_modifier[i]))][0] += point[i]

```

Code : fichier principal

```
45
46 def moins_dico(dico, points_a_modifier, point):
47     plus_dico(dico, points_a_modifier, fois(point, -1))
48
49
50 def reset_eau(dico):
51     for k in dico.keys():
52         x, y = get_index(k)
53         set_volume(x, y, 1, dico)
54
55
56 def voisins(point: str) -> list[tuple[int, int]]:
57     """Trouver les voisins d'un point"""
58     nb = taille - 1
59     x, y = get_index(point)
60     listel = [(x + 1, y), (x - 1, y), (x, y + 1), (x, y - 1), (x - 1, y - 1), (x + 1, y + 1)]
61     liste = []
62     for i in range(len(listel)):
63         if not (listel[i][0] > nb or listel[i][0] < 0 or listel[i][1] > nb or listel[i][1] < 0):
64             liste.append(listel[i])
65     return liste
66
67
68 def trouver_volume(dico: dict, point: str) -> list[list[[tuple[int, int], float]]]:
69     """En partant d'un point, répartir le volume de manière proportionnelle à la racine des altitudes sur les
   points adjacents"""
70     sortie = []
71     somme = 0
72     x, y = get_index(point)
73     z = get_height(x, y, dico)
74     v = get_volume(x, y, dico)
75     for pt in voisins(point):
76         a, b = pt[0], pt[1]
77         delta = z - get_height(a, b, dico)
78         if delta >= 0 and v > 0:
79             sortie.append([pt, np.sqrt((delta))])
80             somme += np.sqrt(delta)
81         else:
82             sortie.append([pt, None])
83     for i in range(len(sortie)):
84         if sortie[i][1] is not None:
85             sortie[i][1] = v * sortie[i][1] / somme
86     return sortie
87
88
```

Code : fichier principal

```python
def liste_aretes(dico):
    # renvoie la liste de reference des aretes
    aretes_fichier = []
    aretes_code = []
    for pt in dico.keys():
        x, y = get_index(pt)
        distribution = trouver_volume(dico, pt)
        for i in range(len(distribution)):
            a, b = distribution[i][0]
            aretes_fichier.append(str(x) + str(y) + " / " + str(a) + str(b))
            aretes_code.append((x, y, a, b))
    return aretes_fichier, aretes_code


def incrementation(dico: dict) -> tuple[[list], bool]:
    # faire glisser l'eau de chaque point une fois
    aretes = []
    fin = True
    for pt in dico.keys():
        v_somme = 0
        distribution = trouver_volume(dico, pt)
        a, b = get_index(pt)
        v0 = get_volume(a, b, dico)
        distrib_test = [distribution[i][1] for i in range(len(distribution))]
        bool2 = distrib_test == [None for _ in range(len(distribution))]
        fin = fin and bool2
        for i in range(len(distribution)):
            if distribution[i][1] is not None:
                x, y = distribution[i][0][0], distribution[i][0][1]
                v = get_volume(x, y, dico)
                v_ajout = distribution[i][1]
                v_somme += v_ajout
                aretes.append(v_ajout)
                v += v_ajout
                set_volume(x, y, v, dico)
            else:
                aretes.append(None)
        if v0 - v_somme > 0.5:
            set_volume(a, b, v0 - v_somme, dico)
        else:
            set_volume(a, b, 0, dico)
    return aretes, fin


# boucle d'incrémentation
def modelisation(dico: dict) -> list[list[int]]:
    termine = False
    historique_aretes = []
    while not termine:
        arete, termine = incrementation(dico)
        historique_aretes.append(arete)
    return historique_aretes


```

Code : fichier principal

```python
def points_selectionnes(dico: dict, historique_aretes: list[list[int]], proportion: float) -> list[tuple[int, int]]:
    aretes = liste_aretes(dico)[1]
    volume_aretes = volume_par_aretes(historique_aretes)
    volume_aretes_triee = [v for v in volume_aretes]
    volume_aretes_triee.sort(reverse=True)
    nb = int(proportion * len(volume_aretes))
    points_a_modifer = []
    for i in range(nb):
        pt = volume_aretes.index(volume_aretes_triee[i])
        pt_modif = (aretes[pt][2], aretes[pt][3])
        if pt_modif not in points_a_modifer:
            points_a_modifer.append(pt_modif)
    return points_a_modifer


def population_intitale(points_a_modifer: list, intervalle: int) -> list[[list[int]]]:
    n = len(points_a_modifer)
    liste_points = []
    for i in range(n):
        liste_points.append([float(random.randint(-intervalle, intervalle)) for _ in range(n)])
    return liste_points


def ecart_type(historique_aretes: list[list[int]]) -> float:
    volume_aretes = volume_par_aretes(historique_aretes)
    liste_carre = [vol ** 2 for vol in volume_aretes]
    return np.sqrt(-sum(liste_carre) + sum(volume_aretes) ** 2)  # peut etre diviser par la longeur


def poids(dico: dict, point: list, points_a_modifier: list[tuple[int, int]]) -> float:
    # ajouter les modifications du point
    plus_dico(dico, points_a_modifier, point)
    historique_aretes = modelisation(dico)
    reset_eau(dico)
    moins_dico(dico, points_a_modifier, point)
    return ecart_type(historique_aretes)


```

Code : fichier principal

```
181 def descente_gradient(f, dico: dict, points_a_modifier: list[tuple[int, int]], point: list[int], j, eps=5,
    maxIter=50) -> \
182         tuple[list[int], list]:
183     eta = 20000
184     norme_grad = 2 * eps + 1
185     i = 0
186     n = len(point)
187     notes = []
188     while norme_grad > 0.003 and i < maxIter:
189         grad = [0. for _ in range(n)]
190         note = f(dico, point, points_a_modifier)
191         for k in range(n):
192             liste_eps = np.array([0. for _ in range(n)])
193             liste_eps[k] = eps
194             p1 = plus(point, liste_eps)
195             p2 = moins(point, liste_eps)
196             grad[k] = (f(dico, p1, points_a_modifier) - f(dico, p2, points_a_modifier)) / 2 * eps
197         point = moins(point, fois(grad, eta))
198         norme_grad = norme(grad)
199         note = f(dico, point, points_a_modifier)
200         # éviter d'avoir un gradient qui explose la note
201         notes.append(note)
202         i += 1
203     return point, notes
204
205
206 def mutation(dico: dict, points_a_modifier: list[tuple[int, int]], intervalle: int) -> tuple[list[dict],
    list[list[list[int]]], list[float], list[list]]:
207     liste_points = population_intitale(points_a_modifier, intervalle)
208     nouveaux_points = []
209     notes_par_dico = []
210     notes_par_point = []
211     liste_dico = []
212     liste_histo = []
213     j = 0
214     for point in liste_points:
215         j += 1
216         nouveau_point, notes_point = descente_gradient(poids, dico, points_a_modifier, point, j)
217         nouveaux_points.append(nouveau_point)
218         notes_par_point.append(notes_point)
219         # on applique maintenant la modification calculée par le gradient au dictionnaire pour récupérer les
    informations voulues
220         # il s'agit d'une fonction poids "améliorée" avec plus d'information en retour
```

Code : fichier principal

```
221         plus_dico(dico, points_a_modifier, point)
222         historique_aretes = modelisation(dico)
223         liste_histo.append(historique_aretes)
224         notes_par_dico.append(ecart_type(historique_aretes))  # liste des notes finales pour voir l'amélioration
    sans modification et avec
225         # elle ne contient pas les notes successives calculées lors des descentes de gradient
226         liste_dico.append({k: [x for x in dico[k]] for k in dico.keys()})
227         reset_eau(dico)
228         moins_dico(dico, points_a_modifier, point)
229     return liste_dico, liste_histo, notes_par_dico, notes_par_point
230
231
232 # création du flanc de montagne
233 points_initiaux = lissage(creer_sommets_reels(taille, 1))
234 dico_ref = lissage(creer_sommets_reels(taille, 0))
235 points = deepcopy(points_initiaux)
236 histo_aretes = modelisation(points)
237
238 # création de la liste des points à modifier
239 pts_a_modifier = points_selectionnes(points, histo_aretes, 0.05)
240 pts_a_modifier = pts_a_modifier[:10]
241
242 # application de la fonction de modification
243 nouveaux_dico, nouveaux_histo, liste_notes_dico, listes_notes_point = mutation(points_initiaux, pts_a_modifier,
    500)
244
245 # afficher le meilleur
246 prems = liste_notes_dico.index(min(liste_notes_dico))
247 afficher(nouveaux_dico[prems], points, pts_a_modifier)
248
249 n_listes_notes_point = mise_en_forme_liste_notes_point(listes_notes_point)
250
251
252 write_liste_point(n_listes_notes_point)
253 t2 = time.time()
254 print("t = ", t2 - t1)
255
256 plt.show()
257
258 evolution_des_points()
```

Code : lecture de fichier

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


SAMPLES = 3601


def read_elevation_from_file(hgt_file, lon, lat):
    with open(hgt_file, 'rb') as hgt_data:
        # Each data is 16 bit signed integer ( i2 ) - big endian ( > )

        elevations = np.fromfile(hgt_data, np.dtype('>i2'), SAMPLES*SAMPLES).reshape((SAMPLES, SAMPLES))
        lat_row = int(round((lat - int(lat)) * (SAMPLES - 1), 0))
        lon_row = int(round((lon - int(lon)) * (SAMPLES - 1), 0))

        # SAMPLES - lat_row car les valeurs commencent en bas à gauche ( au lieu de en haut à gauche )
        # .astype(int) car sinon le type est : <class 'numpy.int16'>
        return elevations[SAMPLES - 1 - lat_row, lon_row].astype(int)


def evolution_des_points():
    fichier = open("liste_points.txt")
    lignes = fichier.readlines()
    liste = []
    listedeliste = []
    for ligne in lignes:
        for i in ligne.split(','):
            if i != '\n':
                liste.append(float(i))
            else:
                listedeliste.append(liste)
        liste = []
    fichier.close()

    X = [[i for i in range(len(listedeliste[k]))] for k in range(len(listedeliste))]

    for i in range(len(X)):
        if max(listedeliste[i]) < 100000:
            plt.plot(X[i], listedeliste[i], label=str(i))
            plt.legend()
    plt.show()

```

Code : traitement des données

```python
def write_edges_data(dico: dict, liste_aretes, historique_aretes: list[list[int]]):
    # ecrit la quantité d'eau par aretes tombéee au cours de la modélisation
    fichier_ = open("valeurs.csv", "w")
    fichier_.write(str(liste_aretes(dico)[0]) + "\n")
    for liste_arete in historique_aretes:
        fichier_.write(str(liste_arete) + "\n")
    fichier_.close()


def colonnes_aretes(historique_aretes: list[list[int]]):
    # transforme l'écriture en ligne en colonne = transposition de matrice
    n = len(historique_aretes)
    N = len(historique_aretes[0])
    colonnes = [[] for _ in range(N)]
    for j in range(n):
        for i in range(N):
            valeur = historique_aretes[j][i]
            if valeur is not None:
                colonnes[i].append(valeur)
    return colonnes


def volume_par_aretes(historique_aretes: list[list[int]]):
    # utilise l'écriture en colonne pour obtenir le volume total d'eau tombée par arete au cours de la modélisation
    colonnes = colonnes_aretes(historique_aretes)
    volume_par_arete = [sum(colonne) for colonne in colonnes]
    return volume_par_arete


def mise_en_forme_liste_notes_point(liste):
    # prolonge la liste des notes avec la dernière note ( pour un meilleur affichage )
    if not liste:
        return []
    maxi = max([len(notes) for notes in liste])
    for i in range(len(liste)):
        k = len(liste[i])
        element = liste[i][-1]
        if k < maxi:
            for _ in range(maxi - k):
                liste[i].append(element)
    return liste


def write_liste_point(liste):
    # utile pour afficher le graphe des notes en fonction des conditions initiales
    fichier = open("liste_points.txt", 'w')
    for liste_notes_point in liste:
        for np in liste_notes_point:
            fichier.write(str(np) + ',')
        fichier.write('\n')
    fichier.close()

```

Code : évolution des points

```python
import matplotlib.pyplot as plt


def evolution_des_points():
    fichier = open("liste_points.txt")
    lignes = fichier.readlines()
    liste = []
    listedeliste = []
    for ligne in lignes:
        for i in ligne.split(','):
            if i != '\n':
                liste.append(float(i))
            else:
                listedeliste.append(liste)
        liste = []
    fichier.close()

    X = [[i for i in range(len(listedeliste[k]))] for k in range(len(listedeliste))]

    for i in range(len(X)):
        if max(listedeliste[i]) < 100000:
            plt.plot(X[i], listedeliste[i], label=str(i))
            plt.legend()
    plt.show()

```