

Le mouvement des dunes de sable

-

Modélisation

19532

LE GUILLOU

Auxence

En binôme avec
Bastien Allier

Motivation du sujet

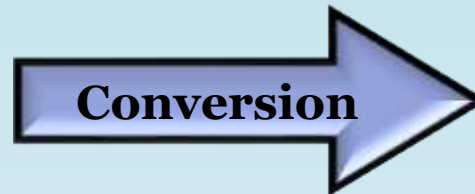
Pierre-Gilles de Gennes



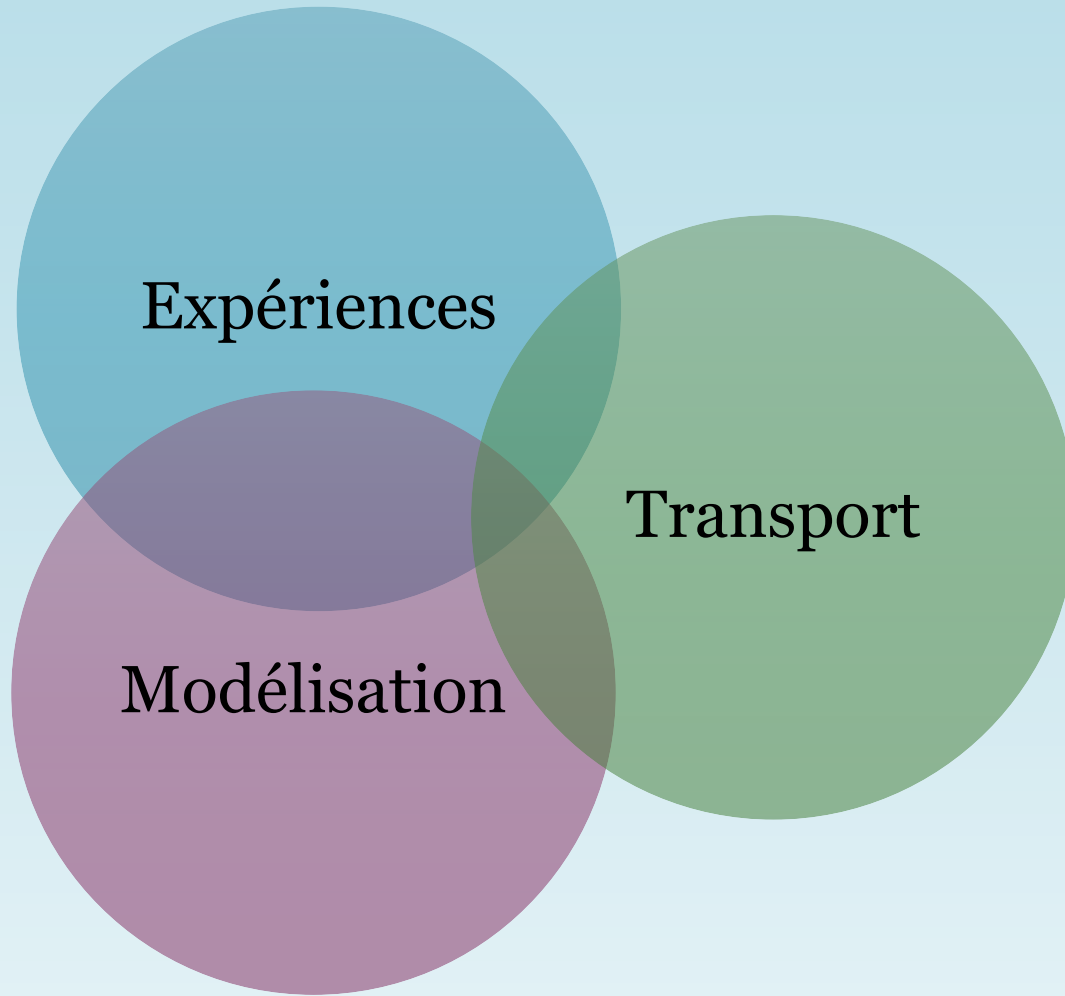
**« Si l'on comprenait parfaitement
les grains de sable, on
comprendrait une grande partie de
la physique »**



Lien avec le sujet 2025



L'articulation de notre TIPE



Mes objectifs annoncés

- Modéliser une dune de sable en Ocaml.
- Etudier ses déplacements en implémentant les différentes altérations possibles à notre modélisation.
- Aspects mathématiques des tas de sables.
- Conclure.



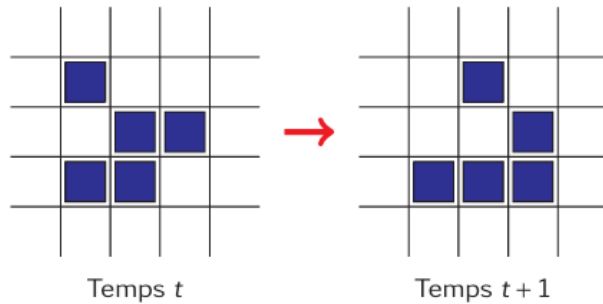
Stratégie pour la ☀️ modélisation ☀️



Automate
cellulaire

C'est quoi, un automate cellulaire ?

FIGURE 2 – Exemple d'évolution du jeu de la vie



➤ Grille

➤ Cellules

➤ Règles

Objet de modélisation

```
4  (*Initialisation du plateau recevant les grains de sables*)
5  let init_grille longueur largeur = Array.make_matrix largeur longueur 0;;
6
7
8  (*Affichage de la grille sous formes de cellules d'entiers*)
9  let affichage_grille grille =
10     Array.iter (fun ligne ->
11         Array.iter (fun colonne -> Printf.printf "%2d " colonne) ligne;
12         print_newline ()
13     ) grille;
14     print_newline ()
15 ;;
```

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Implémentation d'une règle d'écoulement simpliste

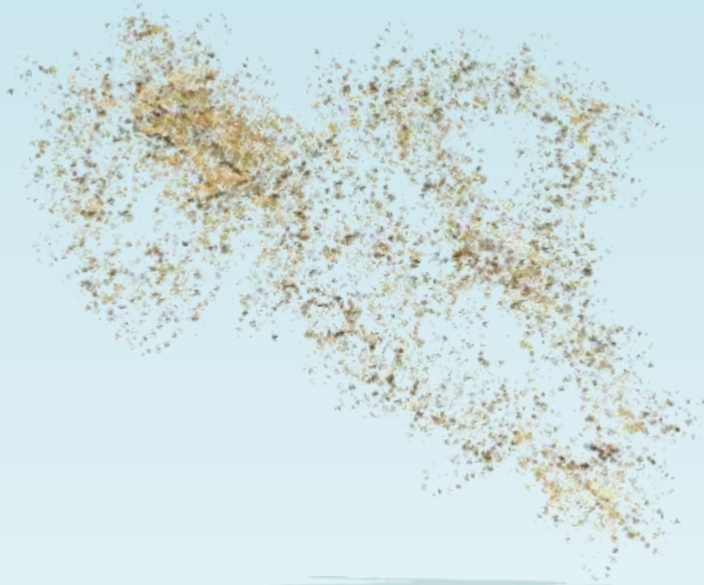
```
1 (*Seuil d'écoulement du sable*)
2 let seuil = 5;;
```

	0	
0	0	0
	0	

```
18 (*Règle d'écoulement en absence de contrainte sur notre automate*)
19 let appliquer_regle grille =
20   let largeur, longueur = (Array.length grille), (Array.length grille.(0)) in
21   let nouvelle_grille = Array.map Array.copy grille in
22   let modifie = ref false in
23   for i = 0 to (largeur-1) do
24     for j = 0 to (longueur-1) do
25       let voisins = [(i-1, j); (i+1, j); (i, j-1); (i, j+1)] in
26       List.iter (fun (vi, vj) ->
27         if vi >= 0 && vi < largeur && vj >= 0 && vj < longueur then
28           let ecart = grille.(i).(j) - grille.(vi).(vj) in
29           if ecart > seuil then (
30             nouvelle_grille.(i).(j) <- nouvelle_grille.(i).(j) - 1;
31             nouvelle_grille.(vi).(vj) <- nouvelle_grille.(vi).(vj) + 1;
32             modifie := true
33           )
34         ) voisins
35       done;
36     done;
37   if !modifie then Some nouvelle_grille else None
38 ;;
```

Implémentation d'une première dune simpliste

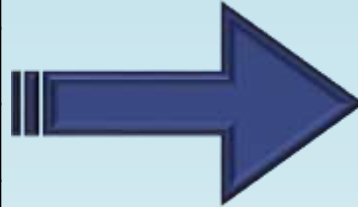
```
41 (* Simulation jusqu'à stabilisation *)
42 let rec simuler grille = match appliquer_regle grille with
43   | Some nouvelle_grille -> simuler nouvelle_grille
44   | None -> grille
45 ;;
```



```
47 (* Programme principal *)
48 let () =
49   let grille = init_grille 10 10 in
50   grille.(5).(5) <- 100;
51   Printf.printf "Grille initiale :\n";
52   affichage_grille grille;
53
54   let resultat = simuler grille in
55   Printf.printf "Grille après simulation :\n";
56   affichage_grille resultat
57 ;;
```

Implémentation d'une première dune simpliste

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	10 00	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0



0	0	0	0	4	8	5	1	0	0
0	0	0	4	9	13	8	4	2	0
0	0	4	9	14	17	13	9	5	2
0	4	9	14	19	22	18	14	9	5
4	9	14	19	24	27	23	18	13	10
8	13	17	22	27	32	27	23	18	14
5	8	13	18	23	27	24	19	14	9
1	4	9	14	18	23	19	14	9	4
0	2	5	9	13	18	14	9	4	0
0	0	2	5	10	14	9	4	0	0

Améliorons la visibilité des résultats

OCaml

Fichier txt

```
104 (* Transformation en fichier txt*)
105 let sauvegarder_grille grille nom_fichier =
106   let out = open_out nom_fichier in
107   Array.iter (fun ligne ->
108     Array.iter (fun valeur -> Printf.fprintf out "%d " valeur) ligne;
109     Printf.fprintf out "\n"
110   ) grille;
111   close_out out
112 ;;
113
114 let save_to_file filename matrix =
115   let oc = open_out filename in
116   Array.iter (fun row ->
117     Array.iteri (fun i v ->
118       output_string oc (string_of_int v);
119       if i < Array.length row - 1 then output_char oc ','
120     ) row;
121     output_char oc '\n'
122   ) matrix;
123   close_out oc
124 ;;
```

0	0	0	0	4	8	5	1	0	0
0	0	0	4	9	13	8	4	2	0
0	0	4	9	14	17	13	9	5	2
0	4	9	14	19	22	18	14	9	5
4	9	14	19	24	27	23	18	13	10
8	13	17	22	27	32	27	23	18	14
5	8	13	18	23	27	24	19	14	9
1	4	9	14	18	23	19	14	9	4
0	2	5	9	13	18	14	9	4	0
0	0	2	5	10	14	9	4	0	0

Améliorons la visibilité des résultats

Fichier txt



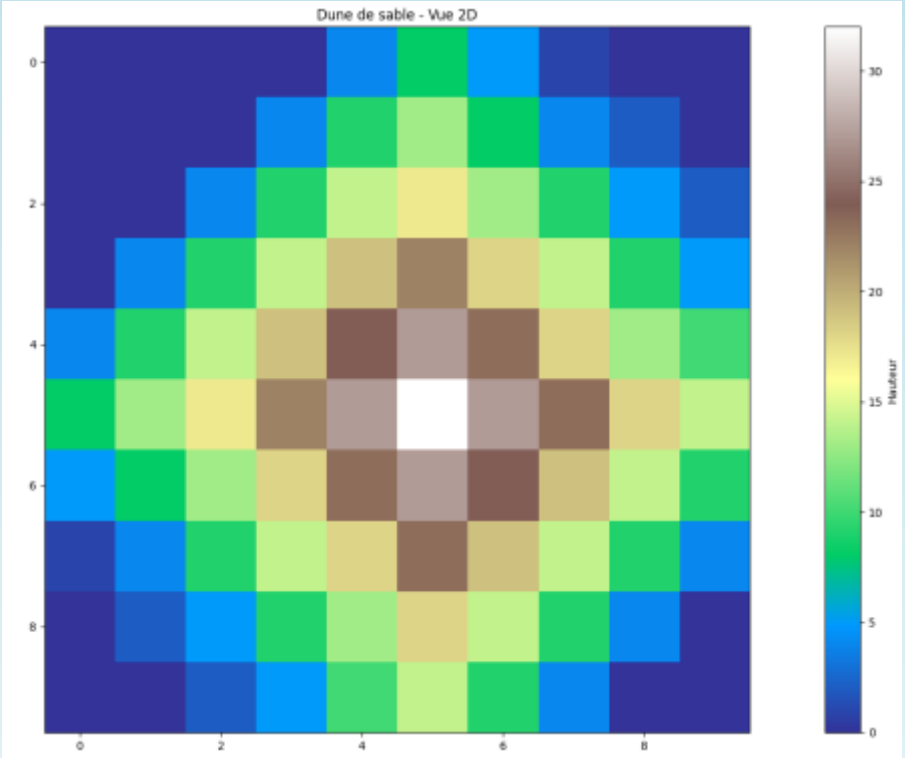
Python

0	0	0	0	4	8	5	1	0	0
0	0	0	4	9	13	8	4	2	0
0	0	4	9	14	17	13	9	5	2
0	4	9	14	19	22	18	14	9	5
4	9	14	19	24	27	23	18	13	10
8	13	17	22	27	32	27	23	18	14
5	8	13	18	23	27	24	19	14	9
1	4	9	14	18	23	19	14	9	4
0	2	5	9	13	18	14	9	4	0
0	0	2	5	10	14	9	4	0	0

```
12 #####
13
14 chemin = r"C:\Users\auxen\OneDrive\Documents\NP_etoile\TIPE\jpp\pilat_ocaml.txt"
15 print("Fichier existe :", os.path.isfile(chemin))
16
```

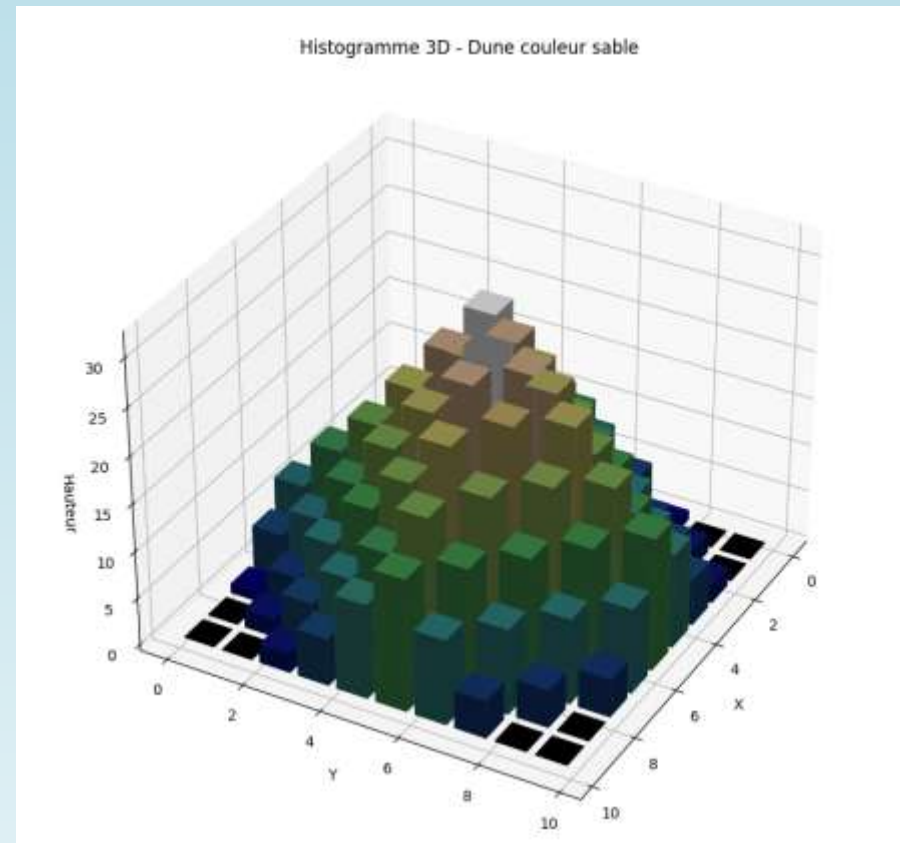
```
26 #####
27
28 ✓ def visualiser_2d(matrice, titre="Dune de sable - Vue 2D"):
29     """
30     Affiche une visualisation 2D de la matrice avec une échelle de couleur.
31     """
32     plt.figure(figsize=(10, 8))
33     im = plt.imshow(matrice, cmap=cm.terrain)
34     plt.colorbar(im, label="Hauteur")
35     plt.title(titre)
36     plt.tight_layout()
37     plt.savefig("dune_2d.png")
38     plt.show()
```

0	0	0	0	4	8	5	1	0	0
0	0	0	4	9	13	8	4	2	0
0	0	4	9	14	17	13	9	5	2
0	4	9	14	19	22	18	14	9	5
4	9	14	19	24	27	23	18	13	10
8	13	17	22	27	32	27	23	18	14
5	8	13	18	23	27	24	19	14	9
1	4	9	14	18	23	19	14	9	4
0	2	5	9	13	18	14	9	4	0
0	0	2	5	10	14	9	4	0	0



Trouvons un meilleur affichage

```
67 #####
68
69 # Ta matrice déjà chargée
70 Z = sortie_ocaml
71 nrows, ncols = Z.shape
72
73 fig = plt.figure()
74 ax = fig.add_subplot(111, projection='3d')
75
76 _x = np.arange(ncols)
77 _y = np.arange(nrows)
78 _xx, _yy = np.meshgrid(_x, _y)
79 x, y = _xx.ravel(), _yy.ravel()
80 top = Z.ravel()
81 bottom = np.zeros_like(top)
82 width = depth = 0.8
83
84 # Couleur sable : RGB (0.76, 0.70, 0.50)
85 colors = [(0.76, 0.70, 0.50)] * len(top)
86 colors = plt.cm.gist_earth((top - top.min()) / (top.max() - top.min()))
87
88 ax.bar3d(x, y, bottom, width, depth, top, shade=True, color=colors)
89
90 ax.set_title("Histogramme 3D - Dune couleur sable")
91 ax.set_xlabel("X")
92 ax.set_ylabel("Y")
93 ax.set_zlabel("Hauteur")
94
95 plt.show()
```



Essayons de se rapprocher de la dune du Pilat

- **Caractéristiques de cette dune**



▪ **Pente opposée au vent**

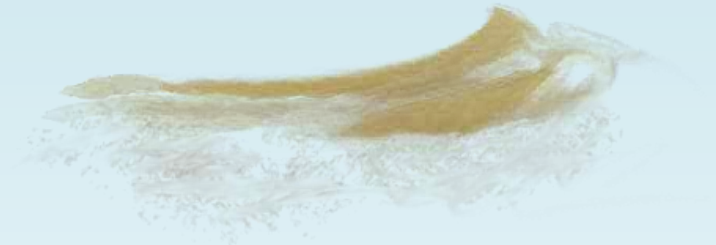
▪ **Pente face au vent**

▪ **Séparation par une ligne culminante**

Essayons de se rapprocher de la dune du Pilat

- **Implémentation du vent**

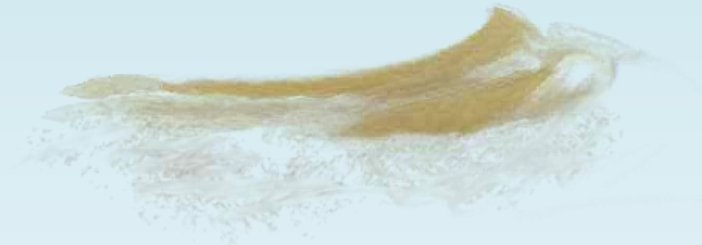
```
18  (*Implémentation du vent sous un module qui priorise les voisins*)
19  module Vent = struct
20      type direction = Nord | Sud | Est | Ouest
21
22      let voisins_selon_vent dir (i, j) =
23          match dir with
24          | Nord  -> [(i-1, j); (i, j+1); (i, j-1); (i+1, j)]
25          | Sud   -> [(i+1, j); (i, j+1); (i, j-1); (i-1, j)]
26          | Est   -> [(i, j+1); (i+1, j); (i-1, j); (i, j-1)]
27          | Ouest -> [(i, j-1); (i+1, j); (i-1, j); (i, j+1)]
28
29      let deplacement_vent dir =
30          match dir with
31          | Nord  -> (-1, 0)
32          | Sud   -> (1, 0)
33          | Est   -> (0, 2)
34          | Ouest -> (0, -1)
35
36      end
37  ;;
```



Essayons de se rapprocher de la dune du Pilat

- Implémentation du vent

```
39  let appliquer_regle_vent grille vent =
40      let largeur = Array.length grille in
41      let longueur = Array.length grille.(0) in
42      let nouvelle_grille = Array.map Array.copy grille in
43      let modifie = ref false in
44      for i = 0 to largeur - 1 do
45          for j = 0 to longueur - 1 do
46              let voisins = Vent.voisins_selon_vent vent (i, j) in
47              List.iter (fun (vi, vj) ->
48                  if vi >= 0 && vi < largeur && vj >= 0 && vj < longueur then
49                      let ecart = grille.(i).(j) - grille.(vi).(vj) in
50                      if ecart > seuil then (
51                          nouvelle_grille.(i).(j) <- nouvelle_grille.(i).(j) - 1;
52                          nouvelle_grille.(vi).(vj) <- nouvelle_grille.(vi).(vj) + 1;
53                          modifie := true
54                      )
55              ) voisins
56      done
57  done;
58  if !modifie then Some nouvelle_grille
59  else None
60  ;;
```

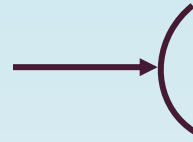


Essayons de se rapprocher de la dune du Pilat

- **Implémentation du vent**

```
120 (* Simulation avec vent jusqu'à stabilisation *)
121 let rec simuler_avec_vent grille vent =
122     match appliquer_regle_vent grille vent with
123     | Some nouvelle_grille -> simuler_avec_vent nouvelle_grille vent
124     | None -> grille
125 ;;
```

Ligne culminante



```
let grille = init_grille 30 30 in

(* Dépôt de sable *)
for i = 5 to 25 do
    grille.(i).(22) <- 1000;
done;

Printf.printf "Grille initiale :\n";
affichage_grille grille;

let vent = Vent.Est in
let resultat = simuler_dynamique_avec_vent grille vent
```

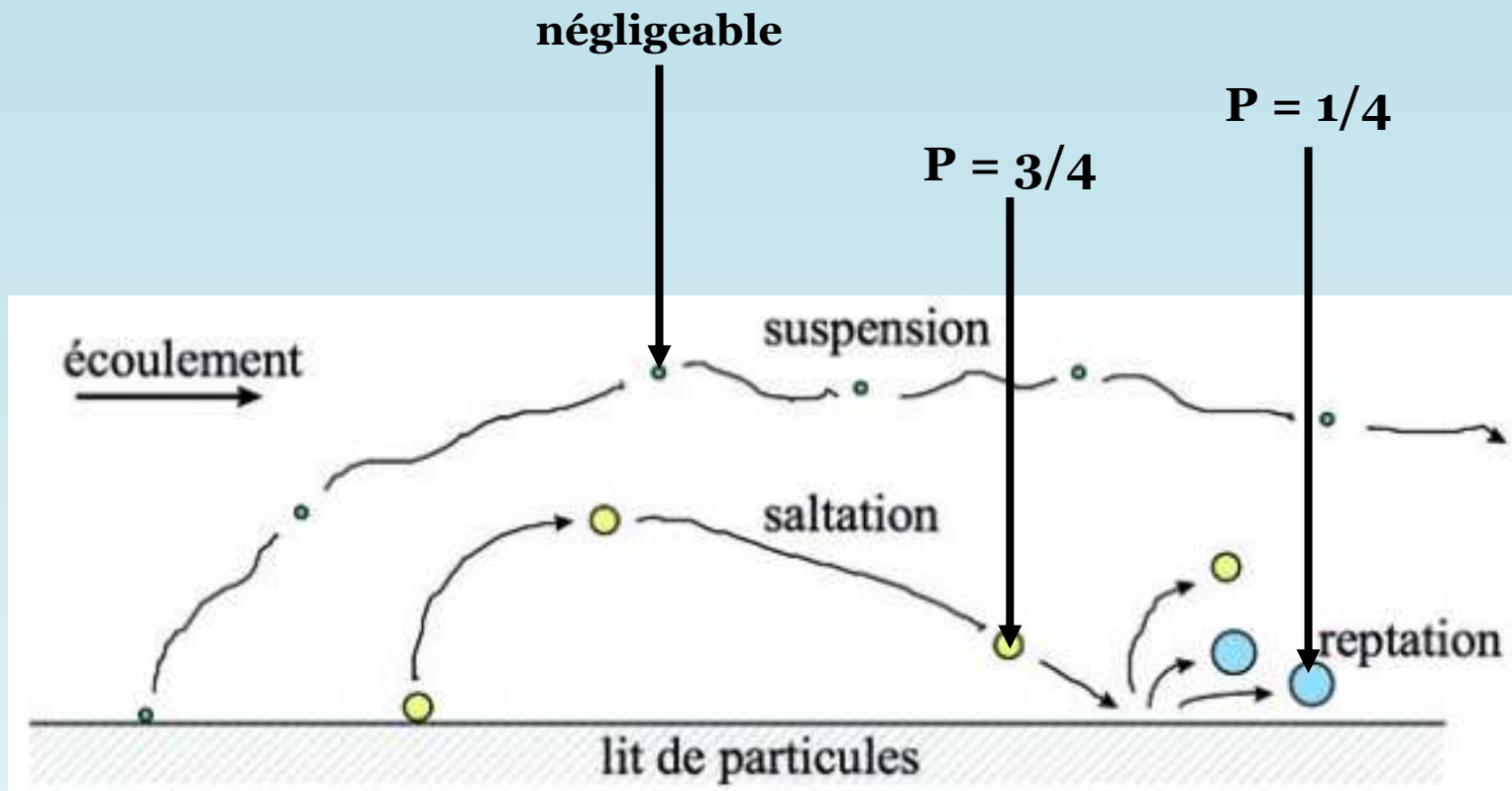
Essayons de se rapprocher de la dune du Pilat

- **Caractéristiques de cette dune**



La pente n'est pas lisse

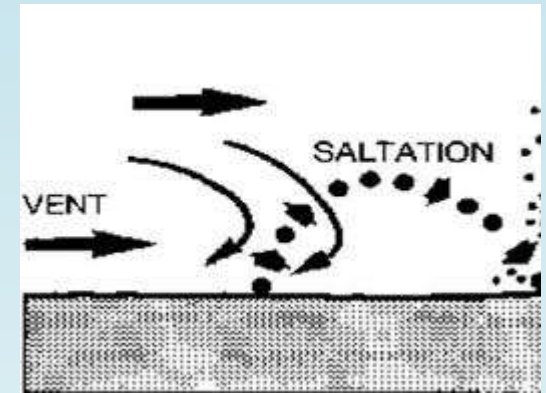
Mouvements de la dune



Essayons de se rapprocher de la dune du Pilat

- **Implémentation de la saltation**

```
62 (* Implémentation de la saltation sur chacune des cellules de la dune *)
63
64 let appliquer_saltation grille x y =
65   let (_,vent_dir) = Vent.deplacement_vent Vent.Est in
66   let proba_saut = 0.75 in
67   match grille.(x).(y) with
68   | qte when qte > 0 && Random.float 1.0 < proba_saut ->
69     let dist = 2 + Random.int 10 in (* Saut entre 1 et 10 cases *)
70     let x' = x + (vent_dir * dist) in
71     if x' < Array.length grille then
72       ( match grille.(x').(y) with
73       | 0 ->
74         grille.(x').(y) <- 3; (* dépose dans la cellule cible entre 1 et 5 grains *)
75         let nouvelle_qte = qte - 3 in
76         grille.(x).(y) <- nouvelle_qte
77       | qte' ->
78         (* ajoute un grain à la cellule cible *)
79         grille.(x').(y) <- qte' + 2;
80         let nouvelle_qte = qte - 2 in
81         grille.(x).(y) <- nouvelle_qte
82       )
83   | _ -> ()
84 ;;
```



Essayons de se rapprocher de la dune du Pilat

- **Implémentation de la reptation**

```
86 (* Implémentation du charriage sur chacune des cellules recevant la saltation *)
87
88 let appliquer_charriage grille x y force_charriage =
89   let proba_charriage = 0.25 in
90   match grille.(x).(y) with
91   | qte when qte >= force_charriage && Random.float 1.0 < proba_charriage ->
92     let hauteur = Array.length grille.(0) in
93     if y + 1 < hauteur then
94       ( match grille.(x).(y+1) with
95       | 0 ->
96         grille.(x).(y+1) <- force_charriage;
97         grille.(x).(y) <- qte - force_charriage
98       | qte_bas ->
99         grille.(x).(y+1) <- qte_bas + force_charriage;
100        grille.(x).(y) <- qte - force_charriage
101       )
102   | _ -> ()
103 ;;
```



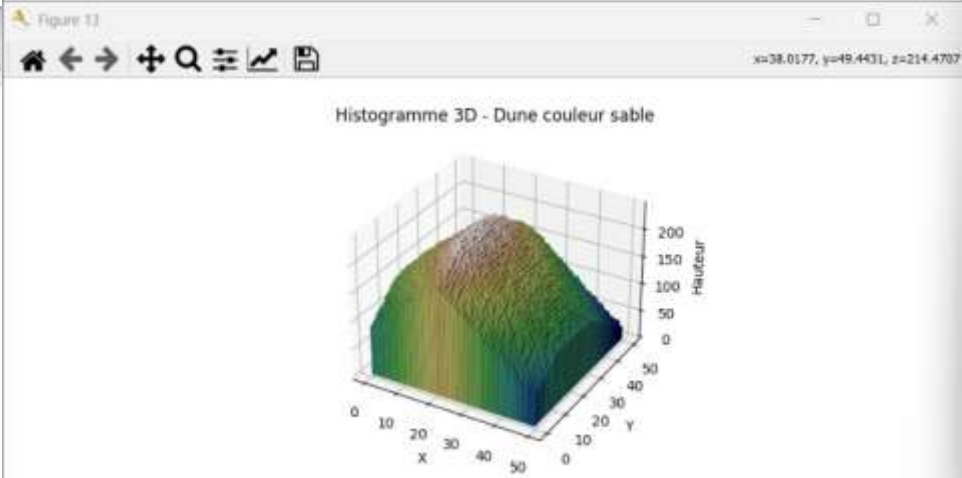
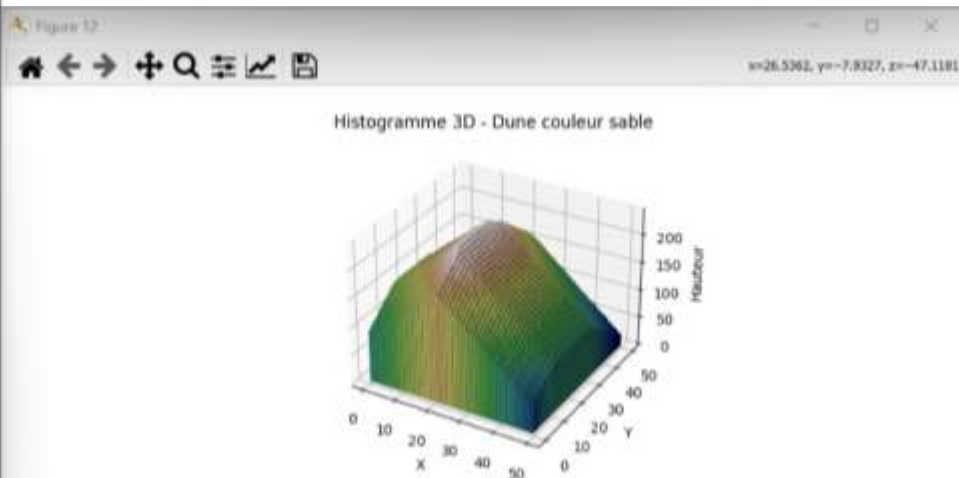
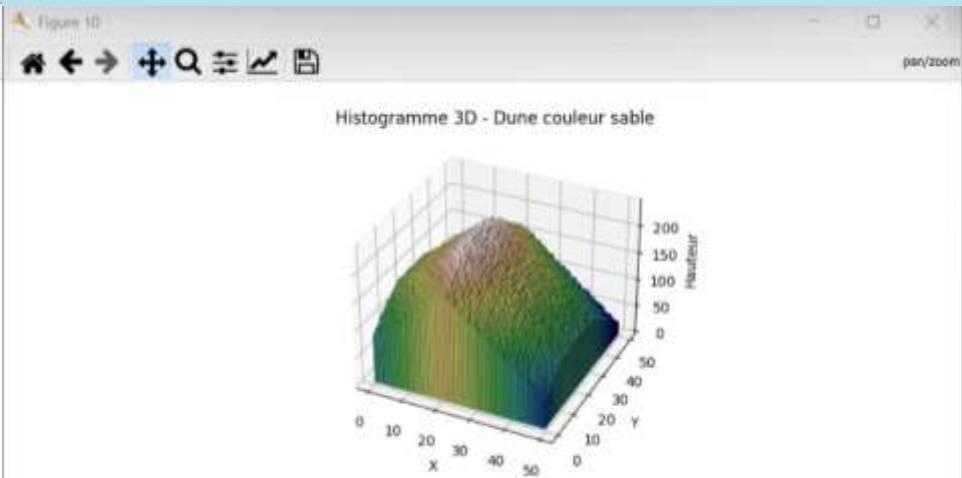
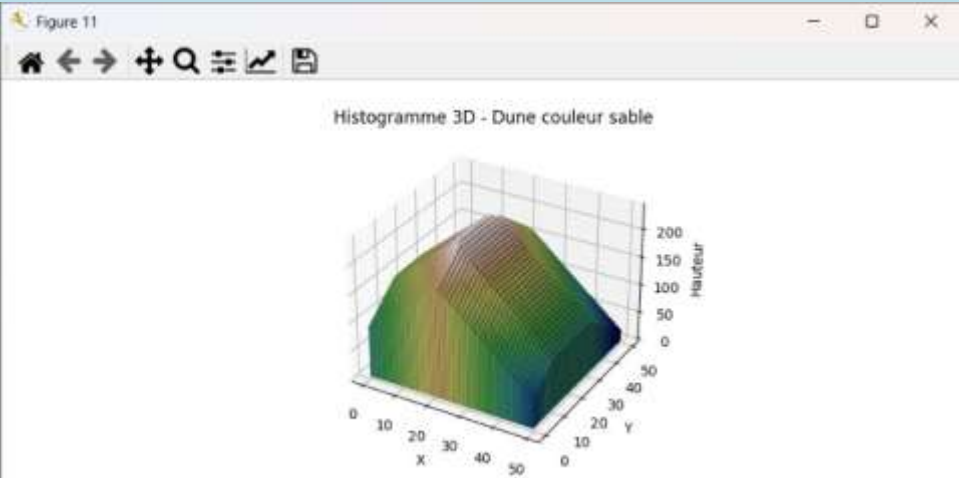
Essayons de se rapprocher de la dune du Pilat

- **Modification du programme principal**

Boucles

```
182 (* Programme principal *)
183 let () =
184   let grille = init_grille 30 30 in
185
186   (* Dépôt de sable *)
187   for i = 5 to 25 do
188     grille.(i).(22) <- 1000;
189   done;
190
191   Printf.printf "Grille initiale :\n";
192   affichage_grille grille;
193
194   let vent = Vent.Est in
195   let resultat = simuler_dynamique_avec_vent grille vent in
196
197   for x = 0 to (Array.length resultat - 1) do
198     for y = 0 to (Array.length resultat.(0) - 1) do
199       appliquer_saltation resultat x y
200     done;
201   done;
202
203   for x = 0 to (Array.length resultat - 1) do
204     for y = 0 to (Array.length resultat.(0) - 2) do (* on s'arrête avant que l'on ne puisse plus transférer plus bas *)
205       appliquer_charriage resultat x y 2
206     done;
207   done;
208
209   Printf.printf "Grille après simulation :\n";
210   affichage_grille resultat;
211   sauvegarder_grille resultat "pilat_ocaml.txt";
212   save_to_file "pilat_ocaml.txt" resultat
213 ;;
```


Modélisation de la dune du Pilat



Et une modélisation dynamique ?



modification

```
73 (* Simulation jusqu'à stabilisation *)
74 let rec simuler grille = match appliquer_regle grille with
75   | Some nouvelle_grille -> simuler nouvelle_grille
76   | None -> grille
77 ;;
```

```
81 (* Simulation dynamique jusqu'à stabilisation *)
82 let simuler_dynamique grille sommets =
83   let grille_courante = ref grille in
84   let continuer = ref true in
85   while !continuer do
86     match appliquer_regle !grille_courante with
87     | Some nouvelle ->
88       if sommet_n_est_plus_maximum nouvelle sommets then (
89         print_endline "Un des sommets initiaux est dépassé. Arrêt.";
90         continuer := false
91       ) else (
92         print_string "\027[2J"; (* Efface écran *)
93         print_string "\027[H"; (* Curseur en haut *)
94         Printf.printf "Étape suivante :\n";
95         affichage_grille nouvelle;
96         grille_courante := nouvelle
97       )
98     | None -> continuer := false
99   done;
100   !grille_courante
101 ;;
```

On modifie également pour le vent



modification

```
127 let rec simuler_dynamique_avec_vent grille vent =
128     match appliquer_regle_vent grille vent with
129     | Some nouvelle_grille ->
130         (* Effacer l'écran *)
131         print_string "\027[2J"; (* code ANSI pour effacer l'écran *)
132         print_string "\027[H";  (* se replacer en haut à gauche *)
133         affichage_grille nouvelle_grille;
134         simuler_dynamique_avec_vent nouvelle_grille vent
135     | None -> grille
136     ;;
```

```
120 (* Simulation avec vent jusqu'à stabilisation *)
121 let rec simuler_avec_vent grille vent =
122     match appliquer_regle_vent grille vent with
123     | Some nouvelle_grille -> simuler_avec_vent nouvelle_grille vent
124     | None -> grille
125     ;;
```

Et la complexité dans tout ça ?

```
160 let sauvegarder_etape_cumulative oc etape grille =  
161   Printf.fprintf oc "Étape %d:\n" etape;  
162   Array.iter (fun ligne ->  
163     Array.iter (fun valeur -> Printf.fprintf oc "%d " valeur) ligne;  
164     Printf.fprintf oc "\n"  
165   ) grille;  
166   Printf.fprintf oc "\n%!";  
167   ;;
```

```
(* Ouverture du fichier de sortie *)  
let out = open_out "historique_simulation.txt" in  
  
(* Initialisation de la simulation *)  
let etape = ref 0 in  
let g = ref (Array.map Array.copy grille) in  
  
(* Simulation dynamique avec export à chaque étape *)  
while true do  
  sauvegarder_etape_cumul   ative out !etape !g;  
  
  match appliquer_regle_vent !g vent with  
  | Some nouvelle ->  
    incr etape;  
    g := nouvelle;  
  | None ->  
    Printf.fprintf out "Simulation terminée à l'étape %d\n" !etape;  
    close_out out;  
    exit 0  
done;
```

**Temporelle
inchangée**

$O(n \times m)$  $O(n \times m)$

Spatiale ...

1 matrice



**500
matrices**

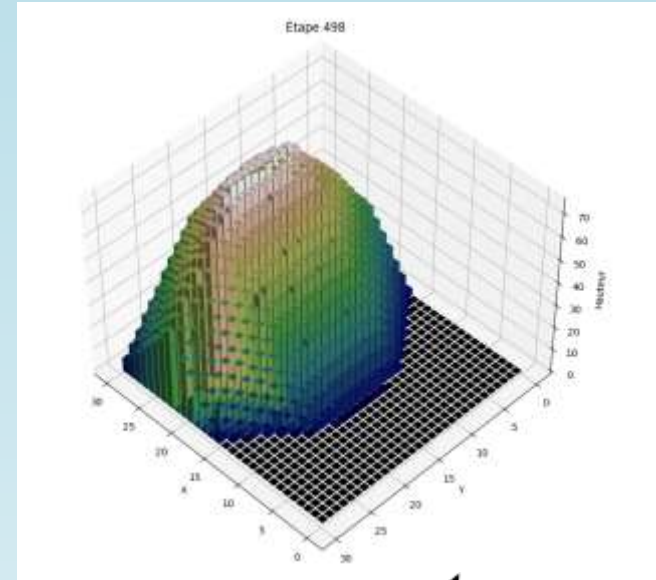
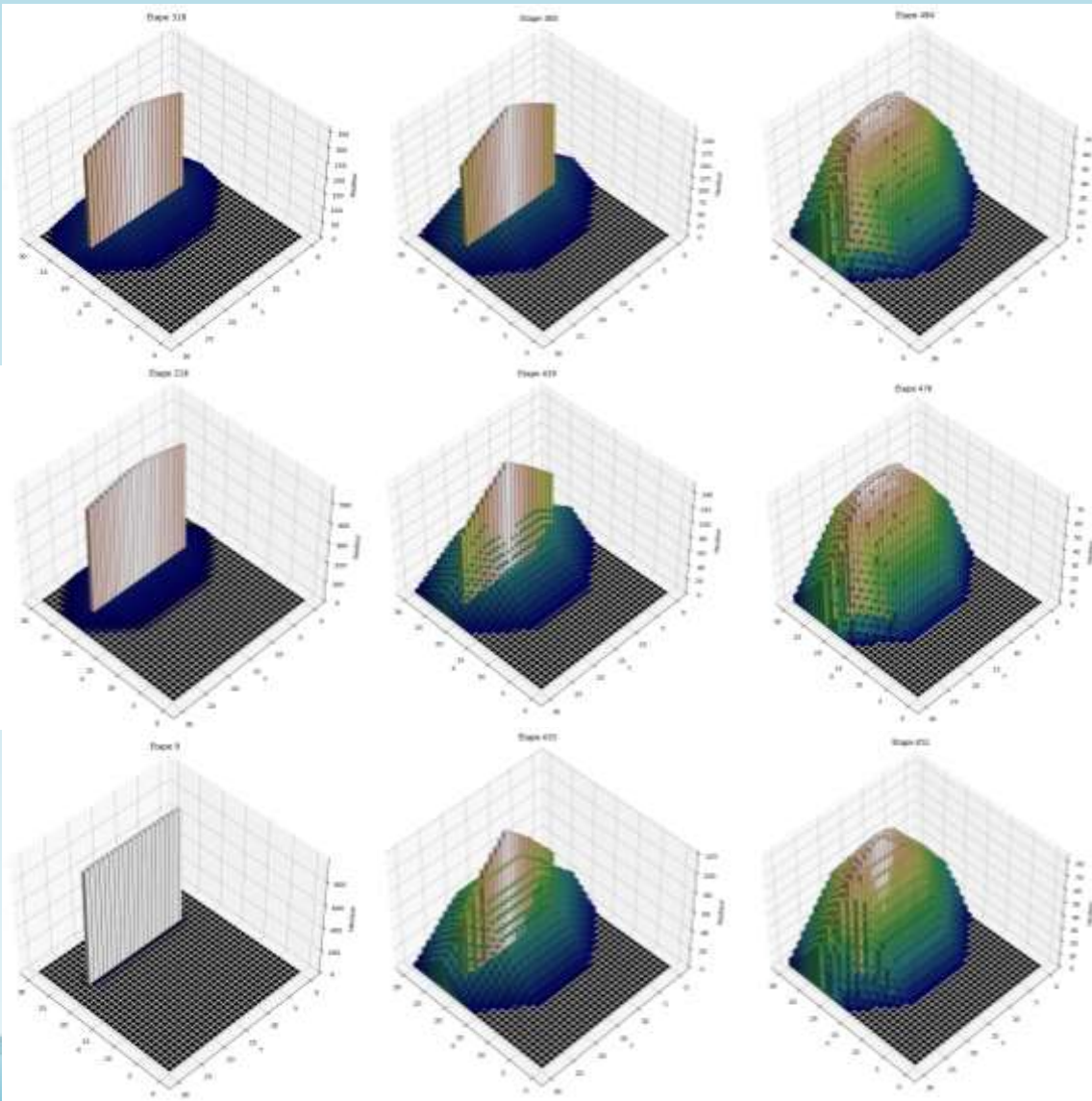
Importations dans python

```
16  def charger_matrices_depuis_fichier(nom_fichier):
17      with open(nom_fichier, 'r') as f:
18          lignes = f.readlines()
19
20      matrices = []
21      matrice_courante = []
22
23      for ligne in lignes:
24          ligne = ligne.strip()
25          if ligne.startswith("Étape"):
26              if matrice_courante:
27                  matrices.append(np.array(matrice_courante, dtype=int))
28                  matrice_courante = []
29              elif ligne and all(c.isdigit() or c.isspace() for c in ligne):
30                  valeurs = list(map(int, ligne.split()))
31                  matrice_courante.append(valeurs)
32
33      if matrice_courante:
34          matrices.append(np.array(matrice_courante, dtype=int))
35
36      return matrices
```

Fonction de vérification

```
105  (* Retourne true si un des sommets initiaux n'est plus un maximum local *)
106  let sommet_n_est_plus_maximum grille positions =
107      let hauteur i j = grille.(i).(j) in
108      let largeur = Array.length grille
109      and longueur = Array.length grille.(0) in
110      List.exists (fun (i,j) ->
111          let h = hauteur i j in
112          let voisins = [(i-1,j); (i+1,j); (i,j-1); (i,j+1)] in
113          List.exists (fun (vi,vj) ->
114              vi >= 0 && vi < largeur && vj >= 0 && vj < longueur &&
115              grille.(vi).(vj) > h
116          ) voisins
117      ) positions
118  ;;
```

Un exemple en image

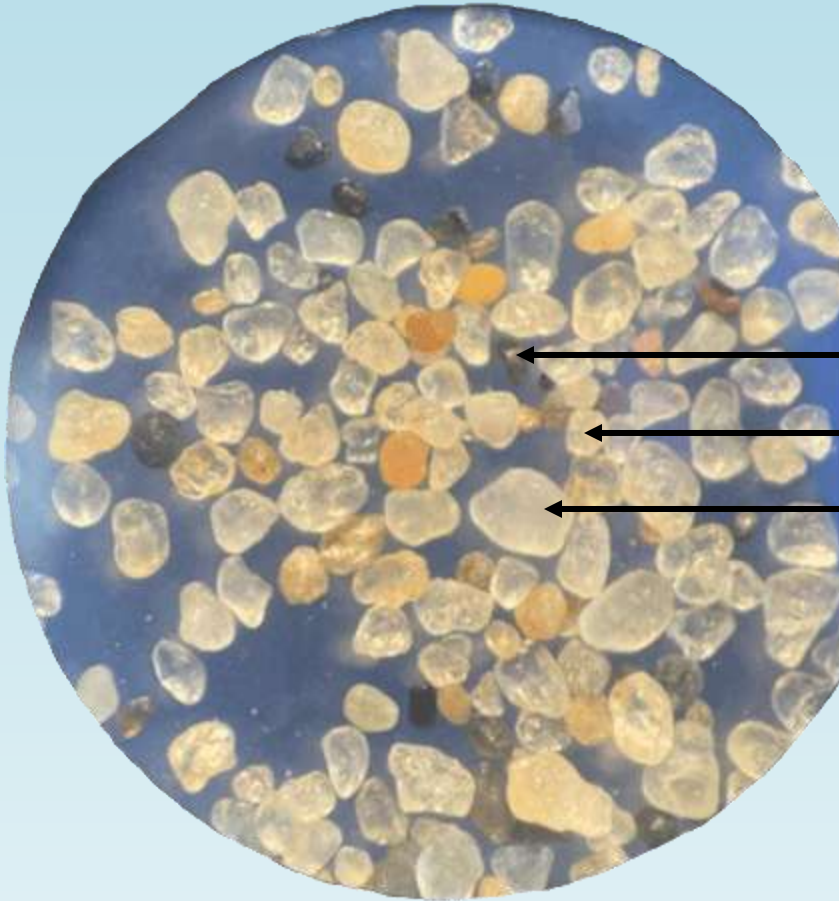


Pendant ce temps, mon
partenaire

Calcul coefficient de frottements



Zoom sur les grains de sables



Proportion de grain fin
($D < 0,2\mu m$) : 6%

**Différentes
tailles de
grains**

Mouvement d'une dune à échelle réduite

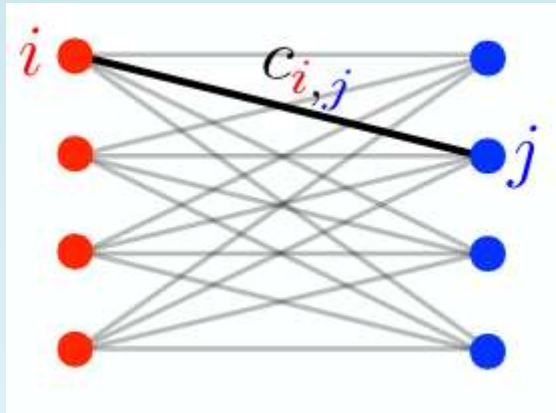


Le Transport Optimal

Gaspard Monge,
XVIII-XIX



$$\min_{\sigma \in \Sigma_n} \sum_{i=1}^n c_{i, \sigma(i)}$$



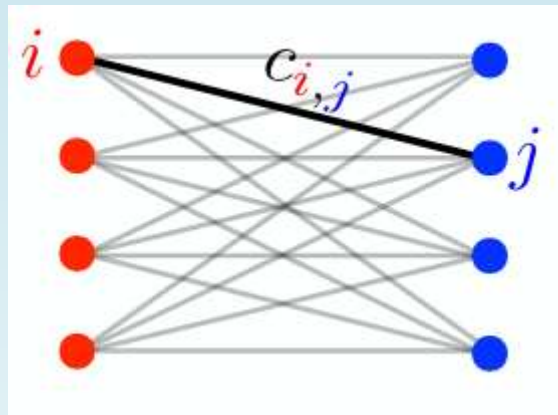
Le Transport Optimal

Gaspard Monge,
XVIII-XIX



$$\min_{\sigma \in \Sigma_n} \sum_{i=1}^n c_{i, \sigma(i)}$$

Coût



Reformulation de Kantorovich

Leonid Kantorovich,
XX

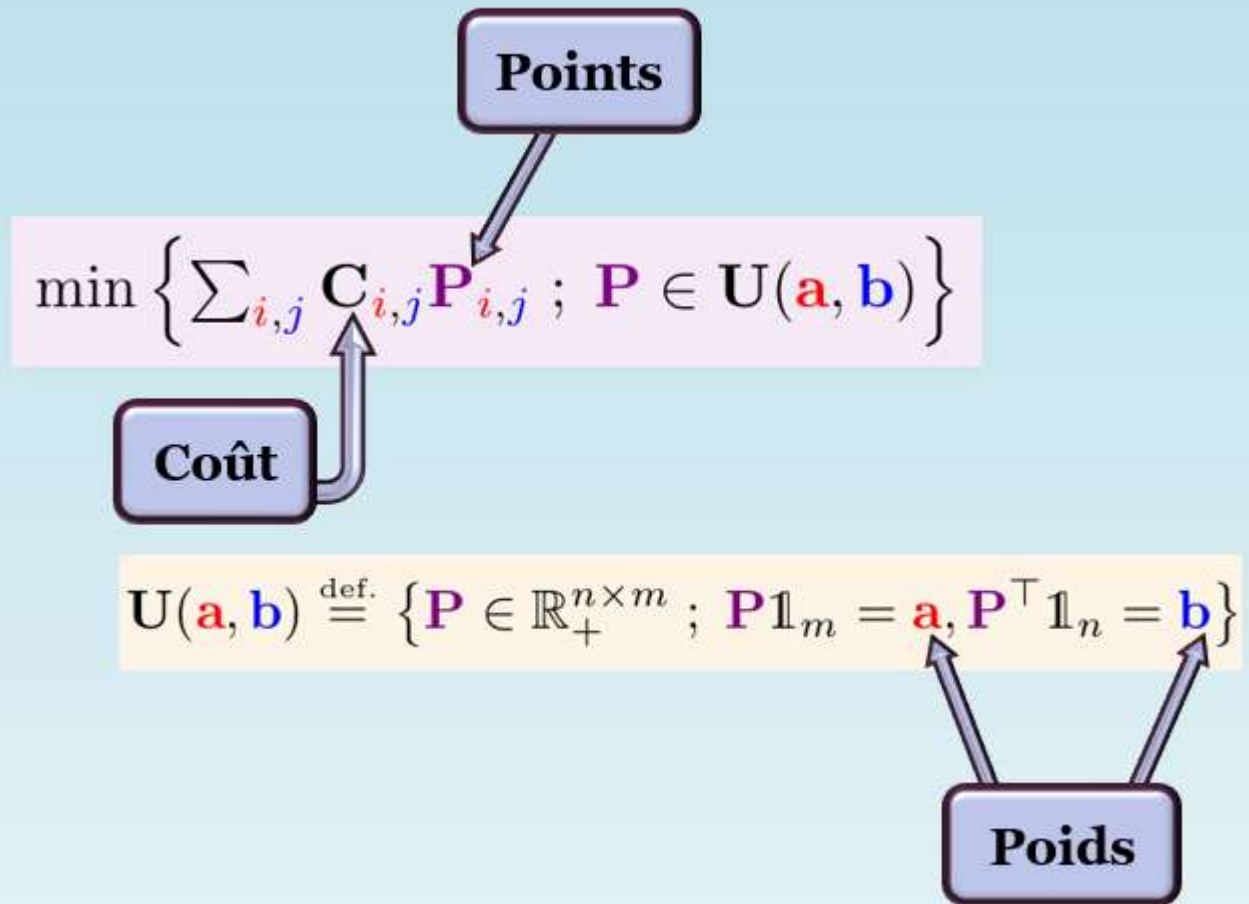


$$\min \left\{ \sum_{i,j} \mathbf{C}_{i,j} \mathbf{P}_{i,j} ; \mathbf{P} \in \mathbf{U}(\mathbf{a}, \mathbf{b}) \right\}$$

$$\mathbf{U}(\mathbf{a}, \mathbf{b}) \stackrel{\text{def.}}{=} \left\{ \mathbf{P} \in \mathbb{R}_+^{n \times m} ; \mathbf{P} \mathbf{1}_m = \mathbf{a}, \mathbf{P}^\top \mathbf{1}_n = \mathbf{b} \right\}$$

Reformulation de Kantorovich

Leonid Kantorovich,
XX



Pour notre dune de sable

- Poids → Masse du sable à conserver
- Coût → Distance euclidienne entre deux cellules
- Points → Nombre de grains à transporter d'une cellule à une autre



Problème d'optimisation linéaire

Idées algorithmiques

- La solution du problème d'optimisation se trouve sur un sommet du polytope
- Une solution de base admissible correspond à un sommet
- Une solution de base admissible est obtenue en choisissant quelles variables sont en bases
- Proposition d'algorithme :
 1. Enumérer les solutions de base admissible
 2. Pour chacune d'elle, calculer la fonction objectif $c^T x$
 3. Identifier celle qui correspond à la plus petite valeur
- Nécessite de considérer

$$\binom{n}{m} = \frac{n!}{m!(n-m)!}$$



Distribution Initiale - p. 46/54



Complexité
temporelle élevée

Exemple simplifié : 3 x 3

a

<u>Cellules</u>	<u>Masse</u>
A	3
B	4
C	2

b

<u>Cellules</u>	<u>Masse</u>
X	2
Y	5
Z	2

c

	X	Y	Z
A	1	3	5
B	2	1	4
C	3	2	1

Minimisation

Contraintes :

- $\forall i \in \{1, 2, 3\},$
 $P_{i,1} + P_{i,2} + P_{i,3} = a_i$
- $\forall j \in \{1, 2, 3\},$
 $P_{1,j} + P_{2,j} + P_{3,j} = b_j$
- $\sum_{i=1}^3 a_i = \sum_{j=1}^3 b_j$

Minimisation

Bases réalisables : 5

Contraintes :

- $\forall i \in \{1, 2, 3\},$
 $P_{i,1} + P_{i,2} + P_{i,3} = a_i$
- $\forall j \in \{1, 2, 3\},$
 $P_{1,j} + P_{2,j} + P_{3,j} = b_j$
- $\sum_{i=1}^3 a_i = \sum_{j=1}^3 b_j$

Minimisation

Contraintes :

- $\forall i \in \{1, 2, 3\},$
 $P_{i,1} + P_{i,2} + P_{i,3} = a_i$
- $\forall j \in \{1, 2, 3\},$
 $P_{1,j} + P_{2,j} + P_{3,j} = b_j$
- $\sum_{i=1}^3 a_i = \sum_{j=1}^3 b_j$

Bases réalisables : 5

$\binom{9}{5} = 126$ *bases*
admissibles possibles

Choix du plan optimal

Calcul de $c^T x$ pour
chaque base :

1. Résoudre le système lié à la base active
2. S'assurer de la positivité
3. Calculer $c^T P$

Plan optimal

=

$$\min_{P \in U(\textcolor{red}{a}, \textcolor{blue}{b})} c^T P$$

D'un point de vue physique



Comparons ces 2 points de vues

Conclusion



➤ **Comprendre**

➤ **Tirer profit**

➤ **Trouver des solutions durables**

Annexe

dune- simulation.ml

```

1 (*Seuil d'écoulement du sable*)
2 let seuil = 5;;
3
4 (*Initialisation du plateau recevant les grains de sables*)
5 let init_grille longueur largeur = Array.make_matrix largeur longueur 0;;
6
7
8 (*Affichage de la grille sous formes de cellules d'entiers*)
9 let affichage_grille grille =
10   Array.iter (fun ligne ->
11     Array.iter (fun colonne -> Printf.printf "%2d " colonne) ligne;
12     print_newline ()
13   ) grille;
14   print_newline ()
15 ;;
16
17 (*Règle d'écoulement en absence de contrainte sur notre automate*)
18 let appliquer_regle grille =
19   let largeur, longueur = (Array.length grille), (Array.length grille.(0)) in
20   let nouvelle_grille = Array.map Array.copy grille in
21   let modifie = ref false in
22   for i = 0 to (largeur-1) do
23     for j = 0 to (longueur-1) do
24       let voisins = [(i-1, j); (i+1, j); (i, j-1); (i, j+1)] in
25       List.iter (fun (vi, vj) ->
26         if vi >= 0 && vi < largeur && vj >= 0 && vj < longueur then
27           let ecart = grille.(i).(j) - grille.(vi).(vj) in
28           if ecart > seuil then (
29             nouvelle_grille.(i).(j) <- nouvelle_grille.(i).(j) - 1;
30             nouvelle_grille.(vi).(vj) <- nouvelle_grille.(vi).(vj) + 1;
31             modifie := true
32           )
33       ) voisins
34     done;
35   done;
36   if !modifie then Some nouvelle_grille else None
37 ;;

```

```

41 (* Simulation jusqu'à stabilisation *)
42 let rec simuler grille = match appliquer_regle grille with
43   | Some nouvelle_grille -> simuler nouvelle_grille
44   | None -> grille
45 ;;
46
47 (* Programme principal *)
48 let () =
49   let grille = init_grille 10 10 in
50   grille.(5).(5) <- 100;
51   Printf.printf "Grille initiale :\n";
52   affichage_grille grille;
53
54   let resultat = simuler grille in
55   Printf.printf "Grille après simulation :\n";
56   affichage_grille resultat
57 ;;

```

```

1 (*Seuil d'écoulement du sable*)
2 let seuil = 5;;
3
4 (*Initialisation du plateau recevant les grains de sables*)
5 let init_grille longueur largeur = Array.make_matrix largeur longueur 0;;
6
7
8 (*Affichage de la grille sous formes de cellules d'entiers*)
9 let affichage_grille grille =
10   Array.iter (fun ligne ->
11     Array.iter (fun colonne -> Printf.printf "%2d " colonne) ligne;
12     print_newline ()
13   ) grille;
14   print_newline ()
15 ;;
16
17 (*Implémentation du vent sous un module qui priorise les voisins*)
18 module Vent = struct
19   type direction = Nord | Sud | Est | Ouest
20
21   let voisins_selon_vent dir (i, j) =
22     match dir with
23     | Nord -> [(i-1, j); (i, j+1); (i, j-1); (i+1, j)]
24     | Sud -> [(i+1, j); (i, j+1); (i, j-1); (i-1, j)]
25     | Est -> [(i, j+1); (i+1, j); (i-1, j); (i, j-1)]
26     | Ouest -> [(i, j-1); (i+1, j); (i-1, j); (i, j+1)]
27
28   let deplacement_vent dir =
29     match dir with
30     | Nord -> (-1, 0)
31     | Sud -> (1, 0)
32     | Est -> (0, 2)
33     | Ouest -> (0, -1)
34
35   end
36 end
37 ;;

```

dune du pilat_ ocaml.ml

```

39 let appliquer_regle_vent grille vent =
40   let largeur = Array.length grille in
41   let longueur = Array.length grille.(0) in
42   let nouvelle_grille = Array.map Array.copy grille in
43   let modifie = ref false in
44   for i = 0 to largeur - 1 do
45     for j = 0 to longueur - 1 do
46       let voisins = Vent.voisins_selon_vent vent (i, j) in
47       List.iter (fun (vi, vj) ->
48         if vi >= 0 && vi < largeur && vj >= 0 && vj < longueur then
49           let ecart = grille.(i).(j) - grille.(vi).(vj) in
50           if ecart > seuil then (
51             nouvelle_grille.(i).(j) <- nouvelle_grille.(i).(j) - 1;
52             nouvelle_grille.(vi).(vj) <- nouvelle_grille.(vi).(vj) + 1;
53             modifie := true
54           )
55       ) voisins
56     done
57   done;
58   if !modifie then Some nouvelle_grille
59   else None
60 ;;

```



dune du pilat_ ocaml.ml

```
62 (* Implémentation de la saltation sur chacune des cellules de la dune *)
63
64 let appliquer_saltation grille x y =
65   let (_,vent_dir) = Vent.deplacement_vent Vent.Est in
66   let proba_saut = 0.75 in
67   match grille.(x).(y) with
68   | qte when qte > 0 && Random.float 1.0 < proba_saut ->
69     let dist = 2 + Random.int 10 in (* Saut entre 1 et 10 cases *)
70     let x' = x + (vent_dir * dist) in
71     if x' < Array.length grille then
72       ( match grille.(x').(y) with
73       | 0 ->
74         grille.(x').(y) <- 3; (* dépose dans la cellule cible entre 1 et 5 grains *)
75         let nouvelle_qte = qte - 3 in
76         grille.(x).(y) <- nouvelle_qte
77       | qte' ->
78         (* ajoute un grain à la cellule cible *)
79         grille.(x').(y) <- qte' + 2;
80         let nouvelle_qte = qte - 2 in
81         grille.(x).(y) <- nouvelle_qte
82       )
83   | _ -> ()
84 ;;
85
86 (* Implémentation du charriage sur chacune des cellules recevant la saltation *)
87
88 let appliquer_charriage grille x y force_charriage =
89   let proba_charriage = 0.25 in
90   match grille.(x).(y) with
91   | qte when qte >= force_charriage && Random.float 1.0 < proba_charriage ->
92     let hauteur = Array.length grille.(0) in
93     if y + 1 < hauteur then
94       ( match grille.(x).(y+1) with
95       | 0 ->
96         grille.(x).(y+1) <- force_charriage;
97         grille.(x).(y) <- qte - force_charriage
98       | qte_bas ->
99         grille.(x).(y+1) <- qte_bas + force_charriage;
100        grille.(x).(y) <- qte - force_charriage
101       )
102   | _ -> ()
103 ;;
```

```
105 (* Retourne true si un des sommets initiaux n'est plus un maximum local *)
106 let sommet_n_est_plus_maximum grille positions =
107   let hauteur i j = grille.(i).(j) in
108   let largeur = Array.length grille
109   and longueur = Array.length grille.(0) in
110   List.exists (fun (i,j) ->
111     let h = hauteur i j in
112     let voisins = [(i-1,j); (i+1,j); (i,j-1); (i,j+1)] in
113     List.exists (fun (vi,vj) ->
114       vi >= 0 && vi < largeur && vj >= 0 && vj < longueur &&
115       grille.(vi).(vj) > h
116     ) voisins
117   ) positions
118 ;;
```

dune du pilat_ ocaml.ml

```

120 (* Simulation avec vent jusqu'à stabilisation *)
121 let rec simuler_avec_vent grille vent =
122     match appliquer_regle_vent grille vent with
123     | Some nouvelle_grille -> simuler_avec_vent nouvelle_grille vent
124     | None -> grille
125 ;;
126
127 let rec simuler_dynamique_avec_vent grille vent =
128     match appliquer_regle_vent grille vent with
129     | Some nouvelle_grille ->
130         (* Effacer l'écran *)
131         print_string "\027[2J"; (* code ANSI pour effacer l'écran *)
132         print_string "\027[H"; (* se replacer en haut à gauche *)
133         affichage_grille nouvelle_grille;
134         simuler_dynamique_avec_vent nouvelle_grille vent
135     | None -> grille
136 ;;
137
138 (* Transformation en fichier txt*)
139 let sauvegarder_grille grille nom_fichier =
140     let out = open_out nom_fichier in
141     Array.iter (fun ligne ->
142         Array.iter (fun valeur -> Printf.fprintf out "%d " valeur) ligne;
143         Printf.fprintf out "\n"
144     ) grille;
145     close_out out
146 ;;
147
148 let save_to_file filename matrix =
149     let oc = open_out filename in
150     Array.iter (fun row ->
151         Array.iteri (fun i v ->
152             output_string oc (string_of_int v);
153             if i < Array.length row - 1 then output_char oc ','
154         ) row;
155         output_char oc '\n'
156     ) matrix;
157     close_out oc
158 ;;

```

```

160 let sauvegarder_etape_cumulative oc etape grille =
161     Printf.fprintf oc "Étape %d:\n" etape;
162     Array.iter (fun ligne ->
163         Array.iter (fun valeur -> Printf.fprintf oc "%d " valeur) ligne;
164         Printf.fprintf oc "\n"
165     ) grille;
166     Printf.fprintf oc "\n%!";
167     ;;
168
169 (* Programme principal *)
170 let () =
171     let grille = init_grille 30 30 in
172
173     (* Dépôt de sable *)
174     for i = 5 to 25 do
175         grille.(1).(22) <- 1000;
176     done;
177
178     Printf.printf "Grille initiale :\n";
179     affichage_grille grille;
180
181     let vent = Vent.Est in
182     let resultat = simuler_dynamique_avec_vent grille vent in
183
184     for x = 0 to (Array.length resultat - 1) do
185         for y = 0 to (Array.length resultat.(0) - 1) do
186             appliquer_saltation resultat x y
187         done;
188     done;
189
190     for x = 0 to (Array.length resultat - 1) do
191         for y = 0 to (Array.length resultat.(0) - 2) do (* on s'arrête avant que l'on ne puisse plus transférer plus bas *)
192             appliquer_charriage resultat x y 2
193         done;
194     done;
195
196     Printf.printf "Grille après simulation :\n";
197     affichage_grille resultat;
198     sauvegarder_grille resultat "pilat_ocaml.txt";
199     save_to_file "pilat_ocaml.txt" resultat
200 ;;

```

Aggrégement

```

79      (* ou *)
80
81      (* Simulation dynamique jusqu'à stabilisation *)
82      let simuler_dynamique grille sommets =
83          let grille_courante = ref grille in
84          let continuer = ref true in
85          while !continuer do
86              match appliquer_regle !grille_courante with
87              | Some nouvelle ->
88                  if sommet_n_est_plus_maximum nouvelle sommets then (
89                      print_endline "Un des sommets initiaux est dépassé. Arrêt.";
90                      continuer := false
91                  ) else (
92                      print_string "\027[2J"; (* Efface écran *)
93                      print_string "\027[H"; (* Curseur en haut *)
94                      Printf.printf "Étape suivante :\n";
95                      affichage_grille nouvelle;
96                      grille_courante := nouvelle
97                  )
98              | None -> continuer := false
99          done;
100          !grille_courante
101      ;;
102      (* Vérification du programme *)
103
104      let compteur_grille =
105          let cpt = ref 0 in
106          let n,m = (Array.length grille, Array.length grille.(0)) in
107          for i = 0 to (n-1) do
108              for j = 0 to (m-1) do
109                  cpt := !cpt + grille.(i).(j)
110              done;
111          done;
112          !cpt;
113      ;;
114
115      ;;

```

```

46      if !modifie then (
47          let nouvelle_grille = Array.map Array.copy grille in
48          for i = 0 to largeur - 1 do
49              for j = 0 to longueur - 1 do
50                  nouvelle_grille.(i).(j) <- grille.(i).(j) + delta.(i).(j)
51              done
52          done;
53          Some nouvelle_grille
54      ) else None
55      ;;

```

Quelques tests OCaml

```

100 (* Programme principal *)
101 let () =
102     let grille = init_grille 50 50 in
103
104     (* Dépôt de sable *)
105     for i = 2 to 44 do
106         grille.(i).(22) <- 9000;
107     done;
108
109     Printf.printf "Grille initiale :\n";
110     affichage_grille grille;
111
112     let vent = Vent.Est in
113     let resultat = simuler_avec_vent grille vent in
114
115     Printf.printf "Grille après simulation :\n";
116     affichage_grille resultat;
117     sauvegarder_grille resultat "pilat_ocaml.txt";
118     save_to_file "pilat_ocaml.txt" resultat
119 ;;

```

```

126 (* Programme principal*)
127 let () =
128     let grille = init_grille 30 30 in
129     let sommets = ref [] in
130     for i = 0 to 29 do
131         grille.(10).(i) <- 10000;
132         sommets := (i,7)::(!sommets);
133     done;
134     grille.(25).(25) <- 50000;
135     Printf.printf "Grille initiale :\n";
136     affichage_grille grille;
137
138     let resultat = simuler(*_dynamique*) grille (* !voisins *) in
139     Printf.printf "Grille après simulation :\n";
140     affichage_grille resultat;
141     sauvegarder_grille resultat "mat_ocaml.txt";
142     save_to_file "mat_ocaml.txt" resultat
143 ;;

```

Modélisation_ dynamique_dune.py

1

```
1  # -*- coding: utf-8 -*-
2  """
3  Created on Wed May 21 09:57:28 2025
4
5  @author: auxen
6  """
7
8  import numpy as np
9  import matplotlib.pyplot as plt
10 from matplotlib import cm
11 from matplotlib.animation import FuncAnimation
12
13 #####
14 # Fonction pour obtenir le fichier OCaml
15
16 def charger_matrices_depuis_fichier(nom_fichier):
17     with open(nom_fichier, 'r') as f:
18         lignes = f.readlines()
19
20     matrices = []
21     matrice_courante = []
22
23     for ligne in lignes:
24         ligne = ligne.strip()
25         if ligne.startswith("Étape"):
26             if matrice_courante:
27                 matrices.append(np.array(matrice_courante, dtype=int))
28                 matrice_courante = []
29             elif ligne and all(c.isdigit() or c.isspace() for c in ligne):
30                 valeurs = list(map(int, ligne.split()))
31                 matrice_courante.append(valeurs)
32
33     if matrice_courante:
34         matrices.append(np.array(matrice_courante, dtype=int))
35
36     return matrices
37
```

```
39 # Charger les données souhaitées
40 chemin = r"C:\Users\auxen\OneDrive\Documents\MP_etoile\TIPE\jpp\historique_simulation.txt"
41 matrices = charger_matrices_depuis_fichier(chemin)
42
43 # Initialisation de la figure
44 fig = plt.figure()
45 ax = fig.add_subplot(111, projection='3d')
46
47 nrows, ncols = matrices[0].shape
48 _x = np.arange(ncols)
49 _y = np.arange(nrows)
50 _xx, _yy = np.meshgrid(_x, _y)
51 x, y = _xx.ravel(), _yy.ravel()
52 bottom = np.zeros_like(x)
53 width = depth = 0.8
54 bars = None
55 # Fonction d'animation
56
57 def update(frame):
58     global bars
59     ax.clear()
60
61     Z = matrices[frame]
62     top = Z.ravel()
63
64     colors = cm.gist_earth((top - top.min()) / (top.max() - top.min()))
65     bars = ax.bar3d(x, y, bottom, width, depth, top, color=colors, shade=True)
66
67     ax.set_title(f"Étape {frame}")
68     ax.set_xlabel("X")
69     ax.set_ylabel("Y")
70     ax.set_zlabel("Hauteur")
71     ax.view_init(elev=45, azimuth=135)
72
73 # Lancer l'animation
74 ani = FuncAnimation(fig, update, frames=len(matrices), interval=200)
75
76 plt.show()
```


modélisation_ nimporte_quelle_ dune.py

1

```
1  # -*- coding: utf-8 -*-
2  """
3  Created on Wed May 21 09:57:28 2025
4
5  @author: auxen
6  """
7
8  import numpy as np
9  import matplotlib.pyplot as plt
10 from matplotlib import cm
11 from matplotlib.animation import FuncAnimation
12
13 #####
14 chemin = r"C:\Users\auxen\OneDrive\Documents\MP_etoile\TIPE\jpp\pilat_ocaml.txt"
15 print("Fichier existe :", os.path.isfile(chemin))
16
17 try:
18     sortie_ocaml = np.loadtxt(chemin, delimiter=",", dtype=int)
19     print("Matrice chargée !")
20     # print(sortie_ocaml)
21 except Exception as e:
22     print("Erreur :", e)
23
24 ▼ def visualiser_2d(matrice, titre="Dune de sable - Vue 2D"):
25     """
26     Affiche une visualisation 2D de la matrice avec une échelle de couleur.
27     """
28     plt.figure(figsize=(10, 8))
29     im = plt.imshow(matrice, cmap=cm.terrain)
30     plt.colorbar(im, label="Hauteur")
31     plt.title(titre)
32     plt.tight_layout()
33     plt.savefig("dune_2d.png")
34     plt.show()
```

```
42 ▼ def visualiser_3d(matrice, titre="Dune de sable - Vue 3D"):
43     """
44     Crée une représentation 3D de la dune de sable.
45     """
46     fig = plt.figure(figsize=(12, 10))
47     ax = fig.add_subplot(111, projection='3d')
48
49     # Créer des coordonnées x, y pour chaque point de la matrice
50     y, x = np.meshgrid(range(matrice.shape[1]), range(matrice.shape[0]))
51
52     # Tracer la surface
53     surf = ax.plot_surface(x, y, matrice, cmap=cm.terrain,
54                           linewidth=0, antialiased=True)
55
56     # Ajouter une barre de couleur
57     fig.colorbar(surf, ax=ax, shrink=0.5, aspect=5, label="Hauteur")
58
59     ax.set_xlabel('X')
60     ax.set_ylabel('Y')
61     ax.set_zlabel('Hauteur')
62     ax.set_title(titre)
63
64     plt.savefig("dune_3d.png")
65     plt.show()
```

modélisation_ nimporte_quelle_ dune.py

2

```
# Ta matrice déjà chargée
70 Z = sortie_ocaml
71 nrows, ncols = Z.shape
72
73 fig = plt.figure()
74 ax = fig.add_subplot(111, projection='3d')
75
76 _x = np.arange(ncols)
77 _y = np.arange(nrows)
78 _xx, _yy = np.meshgrid(_x, _y)
79 x, y = _xx.ravel(), _yy.ravel()
80 top = Z.ravel()
81 bottom = np.zeros_like(top)
82 width = depth = 0.8
83
84 # Couleur sable : RGB (0.76, 0.70, 0.50)
85 colors = [(0.76, 0.70, 0.50)] * len(top)
86 colors = plt.cm.gist_earth((top - top.min()) / (top.max() - top.min()))
87
88 ax.bar3d(x, y, bottom, width, depth, top, shade=True, color=colors)
89
90 ax.set_title("Histogramme 3D - Dune couleur sable")
91 ax.set_xlabel("X")
92 ax.set_ylabel("Y")
93 ax.set_zlabel("Hauteur")
94
95 plt.show()
100 #####
101
102 # Afficher la matrice dans la console
103 #print("Matrice importée:")
104 #for ligne in sortie_ocaml:
105 #    print(" ".join(f"{val:2d}" for val in ligne))
106
107 # Visualisations
108 #if __name__ == "__main__":
109 #    visualiser_2d(sortie_ocaml)
110 #    visualiser_3d(sortie_ocaml)
```