

Prévention des risques relatifs à la foudre

TIPE de Florent Puy
Numéro 12174

La foudre en France c'est en moyenne chaque année:

**2 000 000
d'impacts**

La foudre en France c'est en moyenne chaque année:

**2 000 000
d'impacts**

**200
foudroyés
dont 20
morts**

**15 000 000€
de dégâts**

**15 000
départs
d'incendie**

**Première
cause de
coupure de
courant**

Objectifs

Caractérisation
du phénomène

Objectifs

Caractérisation
du phénomène



Modélisation
numérique

Objectifs

Caractérisation
du phénomène



Modélisation
numérique



Application
à l'efficacité des
paratonnerres

Objectifs

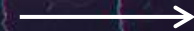
Caractérisation
du phénomène



Modélisation
numérique



Application
à l'efficacité des
paratonnerres

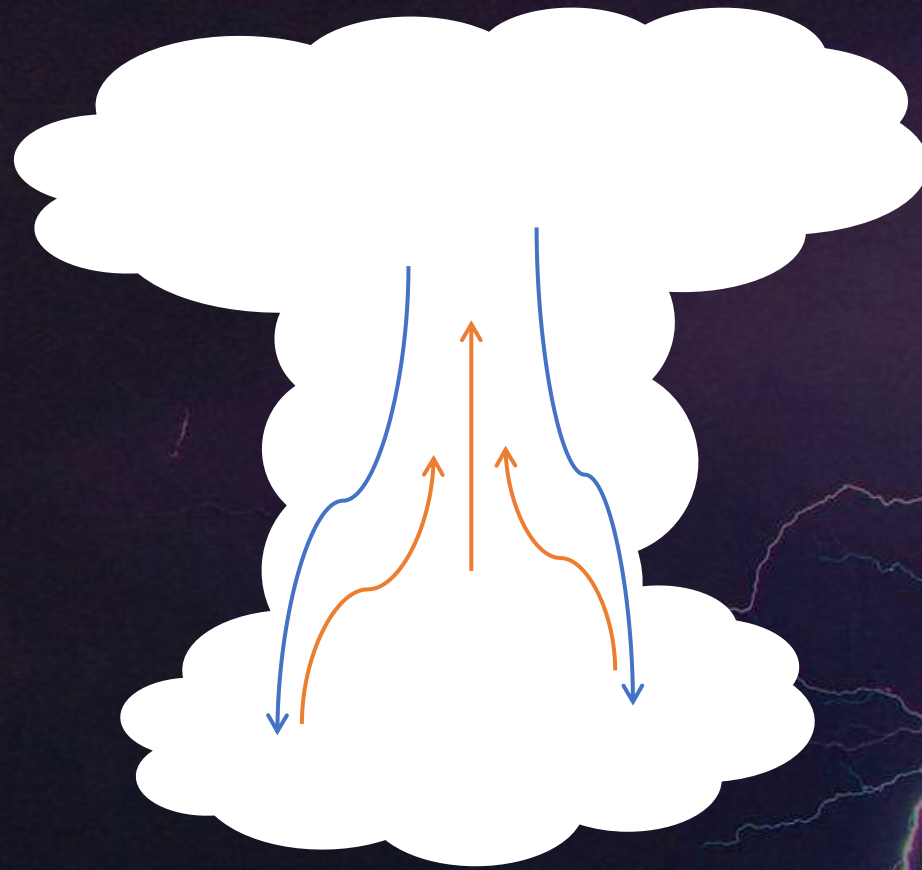


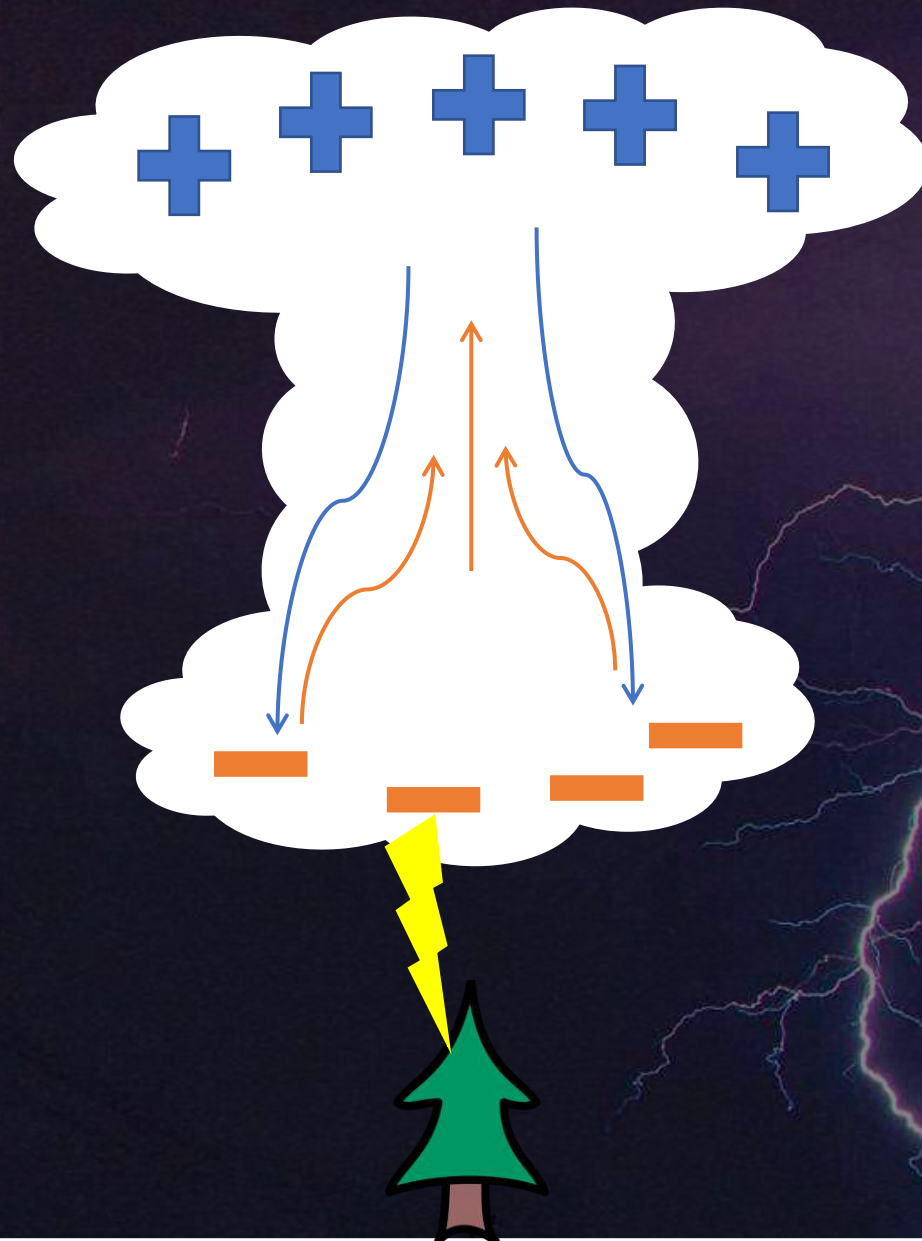
Protection
d'infrastructures
médicales

Caractérisation

du phénomène







2-Différentes phases



Traceurs



2-Différentes phases



Traceurs

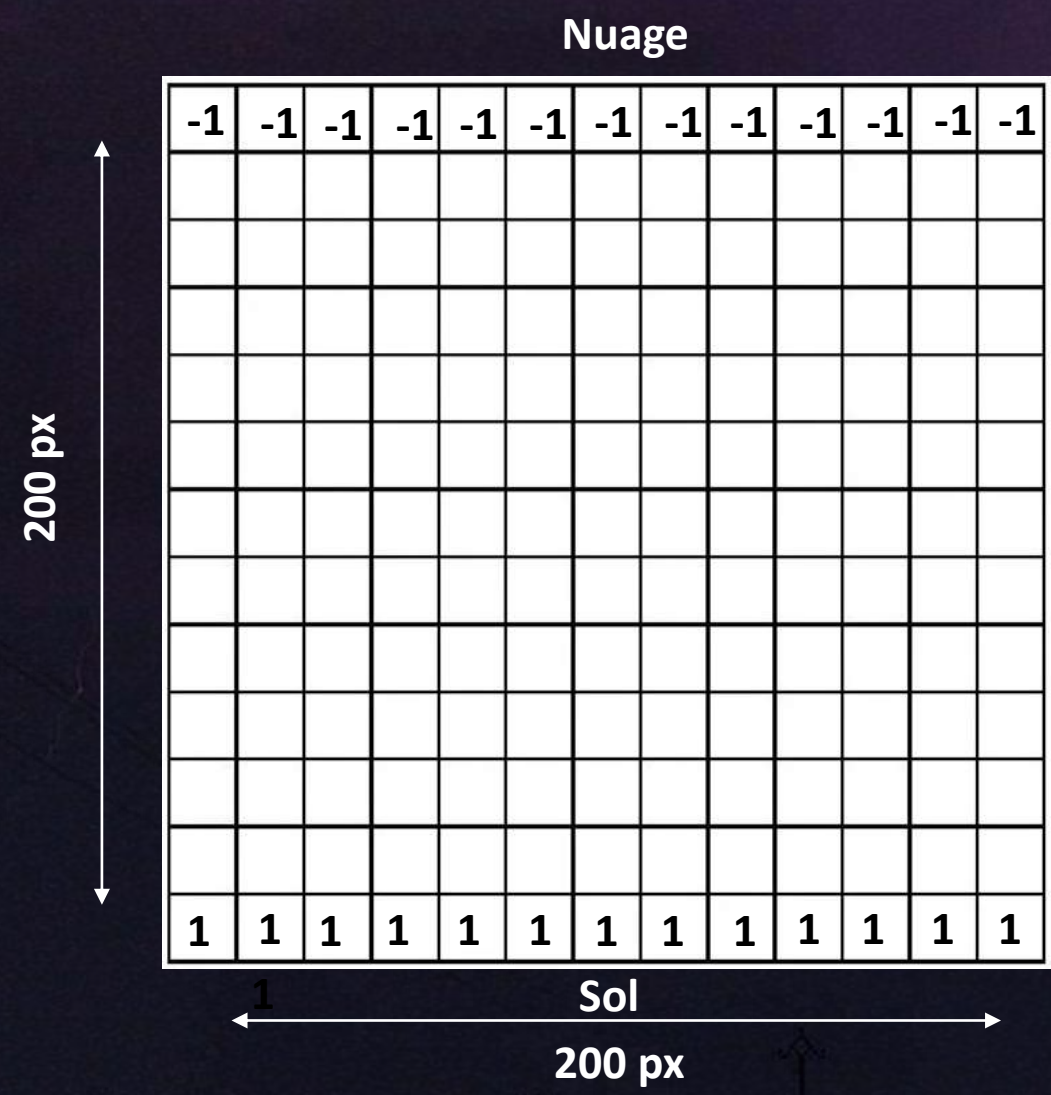


Décharge

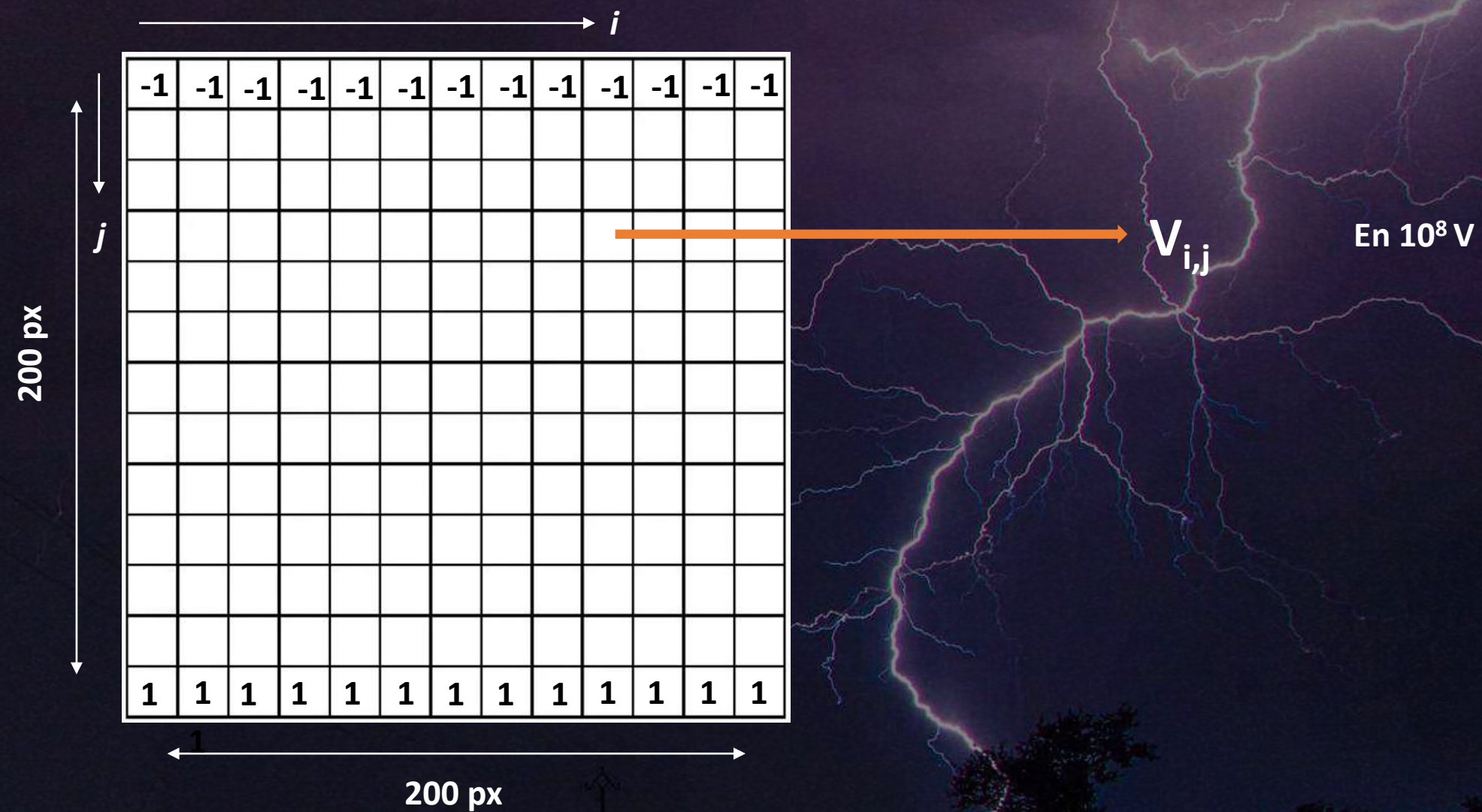
Modélisation

numérique de la foudre

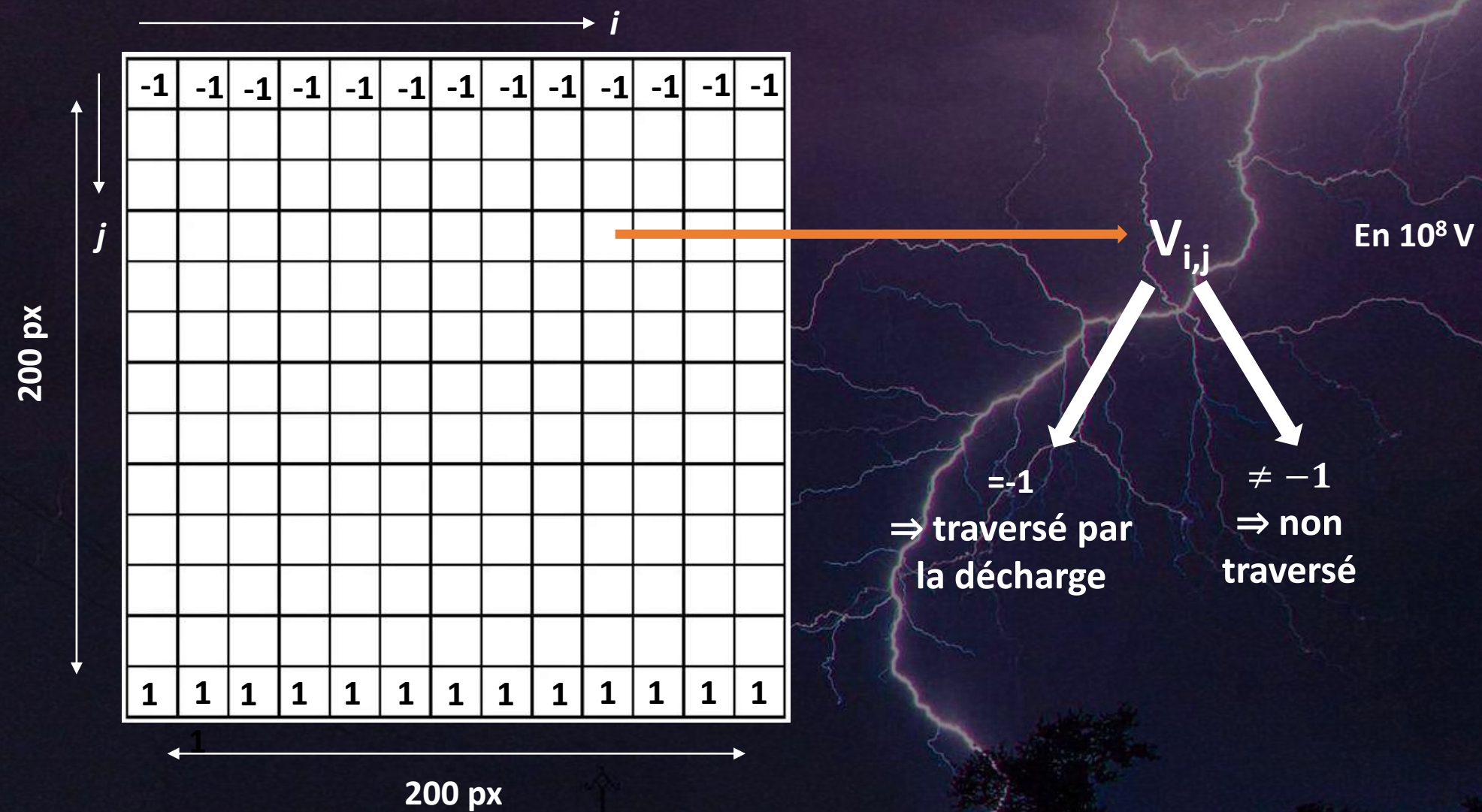
1-Obtention d'une
carte des potentiels



1-Obtention d'une
carte des potentiels



1-Obtention d'une
carte des potentiels



Equation de Laplace

$$\Delta v = 0$$

Discrétisation de l'équation

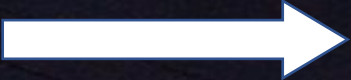
$$\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial z^2} = 0$$



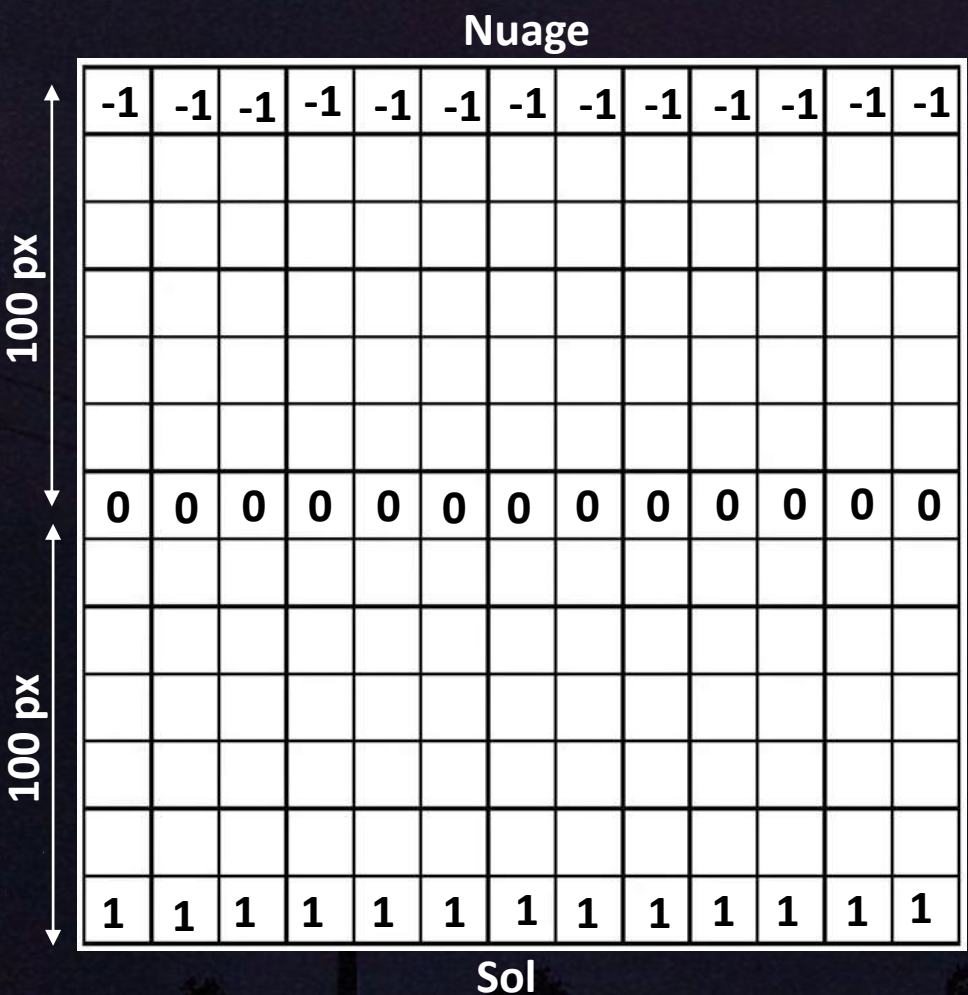
$$f''(a) = \lim_{h \rightarrow 0} \frac{f(a+2h) - 2f(a+h) + f(a)}{h^2}$$

$$\frac{v_{i+1,j} - 2v_{i,j} + v_{i-1,j}}{h^2} + \frac{v_{i,j+1} - 2v_{i,j} + v_{i,j-1}}{h^2} = 0$$

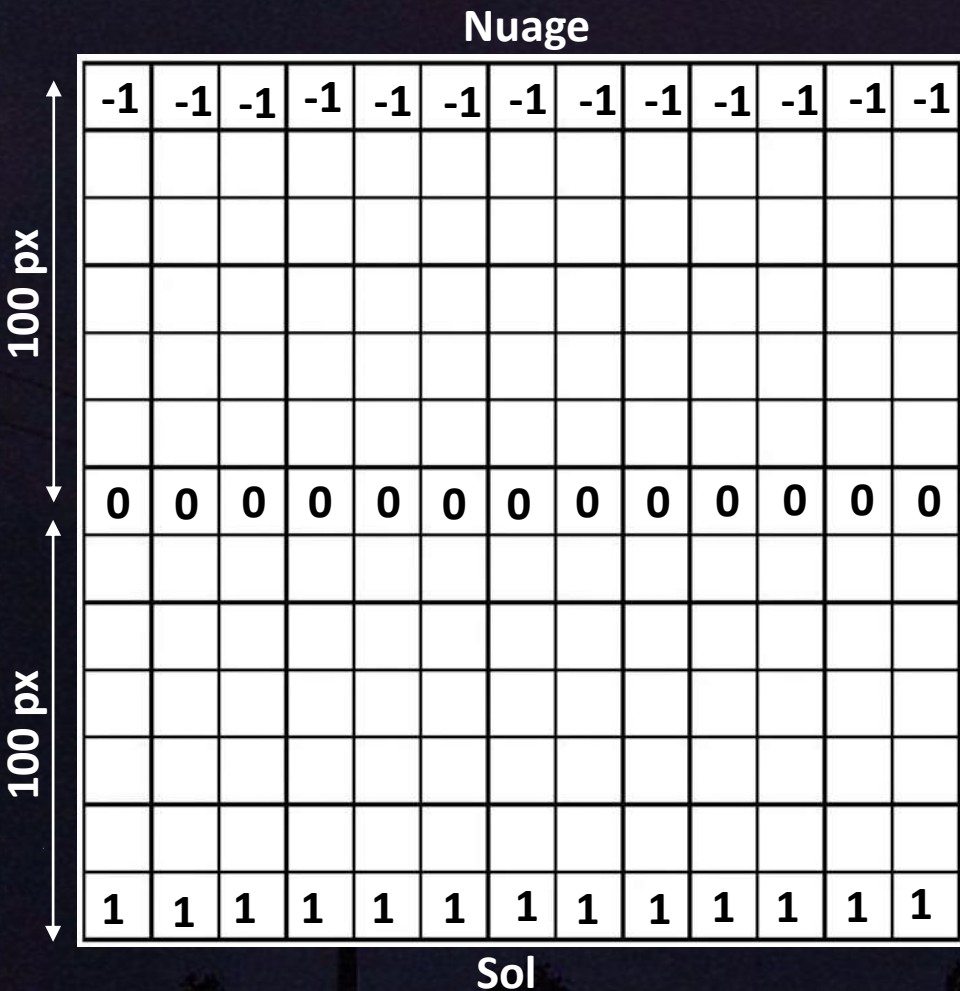
Méthode de Jacobi


$$v_{i,j} = \frac{v_{i+1,j} + v_{i-1,j} + v_{i,j+1} + v_{i,j-1}}{4}$$

Convergence de la méthode de Jacobi



Convergence de la méthode de Jacobi



$$\longrightarrow V_{i,j}(k+1) \leq V_{i,j}(k)$$

$$\longrightarrow V_{i,j}(k) \leq V_{i,j}(k+1)$$

Convergence de la méthode de Jacobi

Nuage

100 px

100 px

Sol

-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1	1	1	1	1

$$\longrightarrow V_{i,j}(k+1) \leq V_{i,j}(k)$$

$$\longrightarrow V_{i,j}(k) \leq V_{i,j}(k+1)$$

$$\forall i, j \in [|0, n - 1|], \forall k \in \mathbb{N}, \\ |V_{i,j}(k)| \leq 1$$

Convergence de la méthode de Jacobi

Nuage

100 px

100 px

Sol

-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1	1	1	1	1

$$\rightarrow V_{i,j}(k+1) \leq V_{i,j}(k)$$

$$\rightarrow V_{i,j}(k) \leq V_{i,j}(k+1)$$

$$\forall i, j \in [|0, n-1|], \forall k \in \mathbb{N}, \\ |V_{i,j}(k)| \leq 1$$

$\forall i, j, (V_{i,j})_n$ monotone et bornée

Convergence de la méthode de Jacobi

Nuage

100 px

100 px

Sol

-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1	1	1	1	1

$$\rightarrow V_{i,j}(k+1) \leq V_{i,j}(k)$$

$$\rightarrow V_{i,j}(k) \leq V_{i,j}(k+1)$$

$$\forall i, j \in [|0, n-1|], \forall k \in \mathbb{N}, \\ |V_{i,j}(k)| \leq 1$$

$\forall i, j, (V_{i,j})$ monotone et bornée

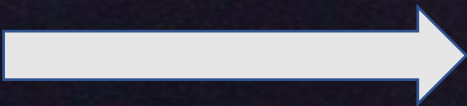
Convergence

Ecart quadratique moyen

$$e_k = \frac{1}{n^2} \sum_{i,j} [v_{i,j}(k) - v_{i,j}(k-1)]^2$$

1-Obtention d'une
carte des potentiels

Algorithme de calcul d'une carte de potentiels



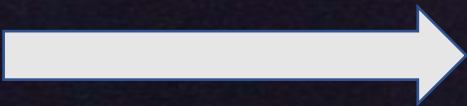
Conditions
initiales:

Nuage

-1	-1	-1	-1	-1
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
+1	+1	+1	+1	+1

Sol

Algorithme de calcul d'une carte de potentiels

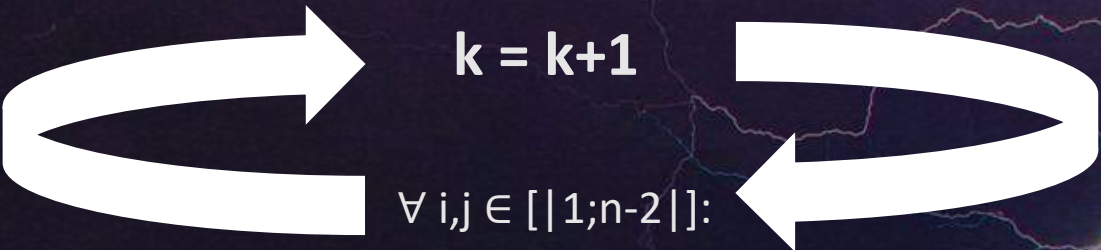


Conditions
initiales:

Nuage

-1	-1	-1	-1	-1
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
+1	+1	+1	+1	+1

Sol

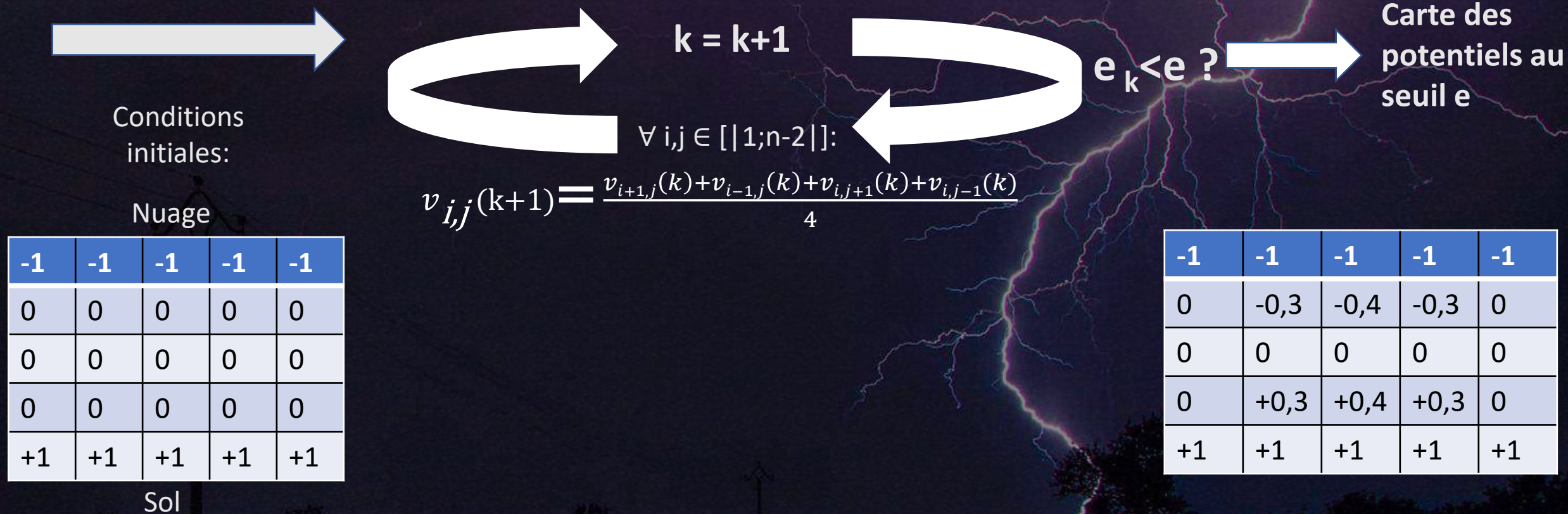


$k = k+1$

$\forall i,j \in [|1;n-2|]:$

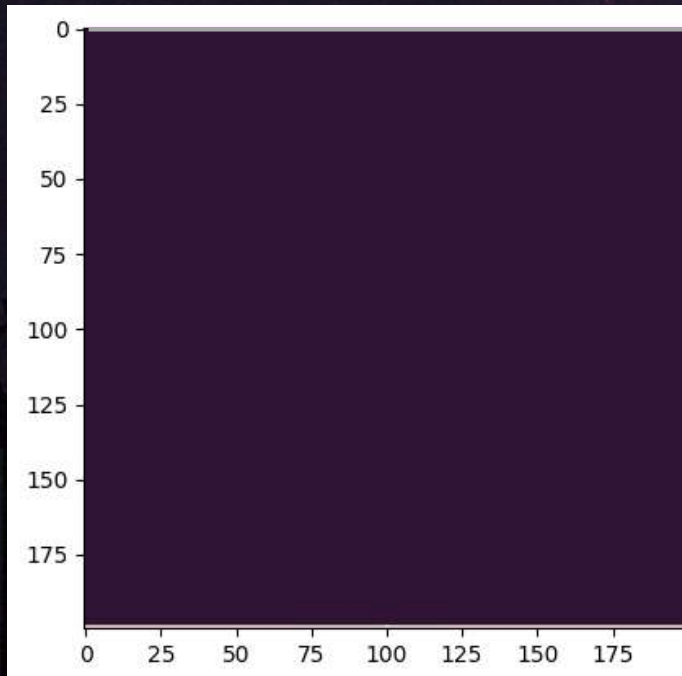
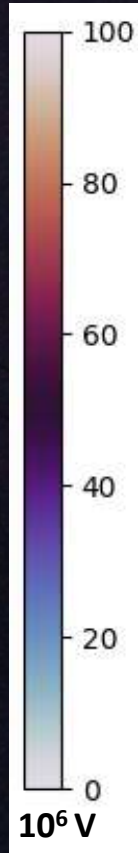
$$v_{i,j}(k+1) = \frac{v_{i+1,j}(k) + v_{i-1,j}(k) + v_{i,j+1}(k) + v_{i,j-1}(k)}{4}$$

Algorithme de calcul d'une carte de potentiels



1-Obtention d'une carte des potentiels

Exemple en 200x200

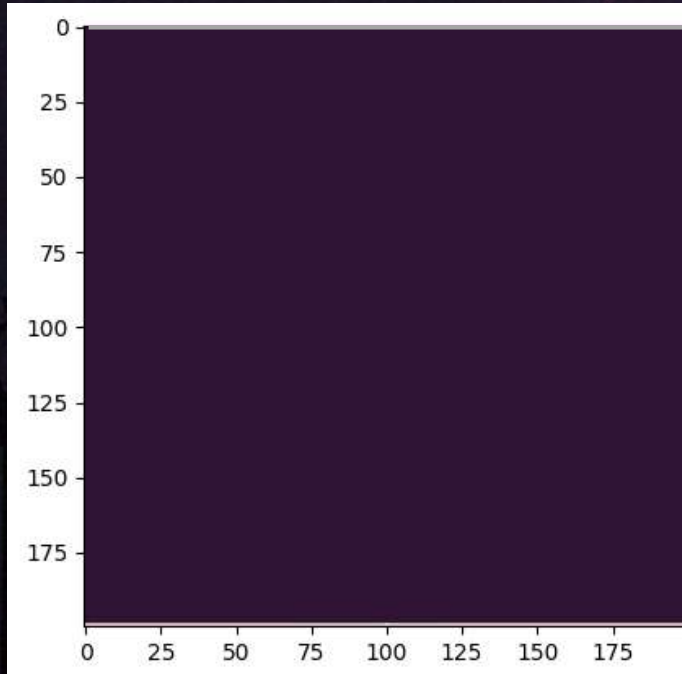
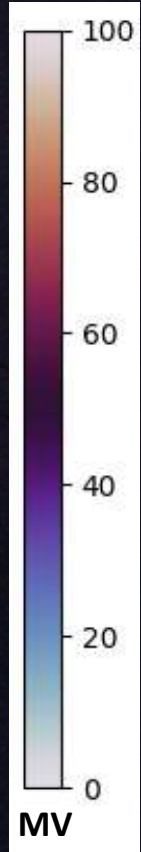


→ $v = 0 \text{ V}$

→ $v = 10^8 \text{ V}$

1-Obtention d'une carte des potentiels

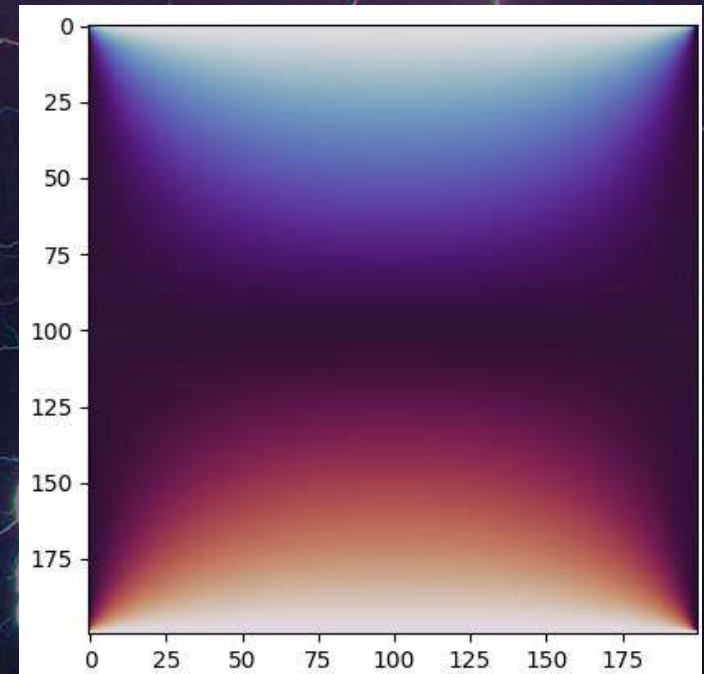
Exemple en 200x200



→ $v = 0 \text{ V}$

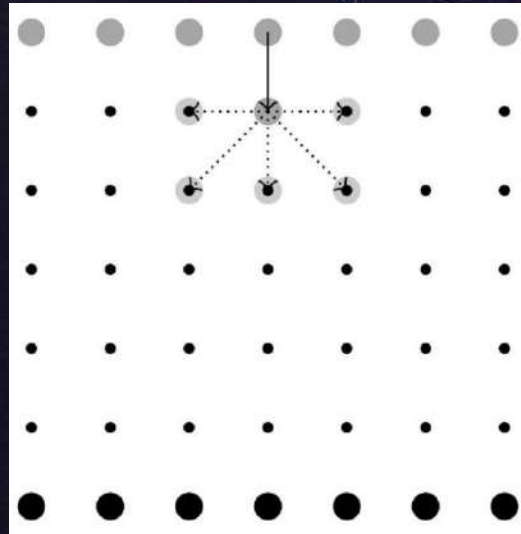


→ $v = 10^8 \text{ V}$



Choix du nouveau point de propagation:

Point éligible = Point dont un unique voisin est traversé par la décharge



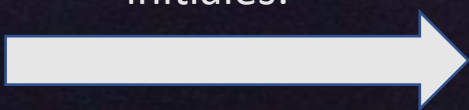
Probabilité de propagation de la décharge en un point (i,j):

$$P((i,j)) = \frac{v_{i,j}^{\eta}}{\sum_k v_k^{\eta}}$$

Avec les v_k représentant l'ensemble des points éligibles

Exemple en taille 5x5

Conditions
initiales:



Nuage

-1	-1	-1	-1	-1
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
+1	+1	+1	+1	+1

Sol

Exemple en taille 5x5

Conditions
initiales

Nuage

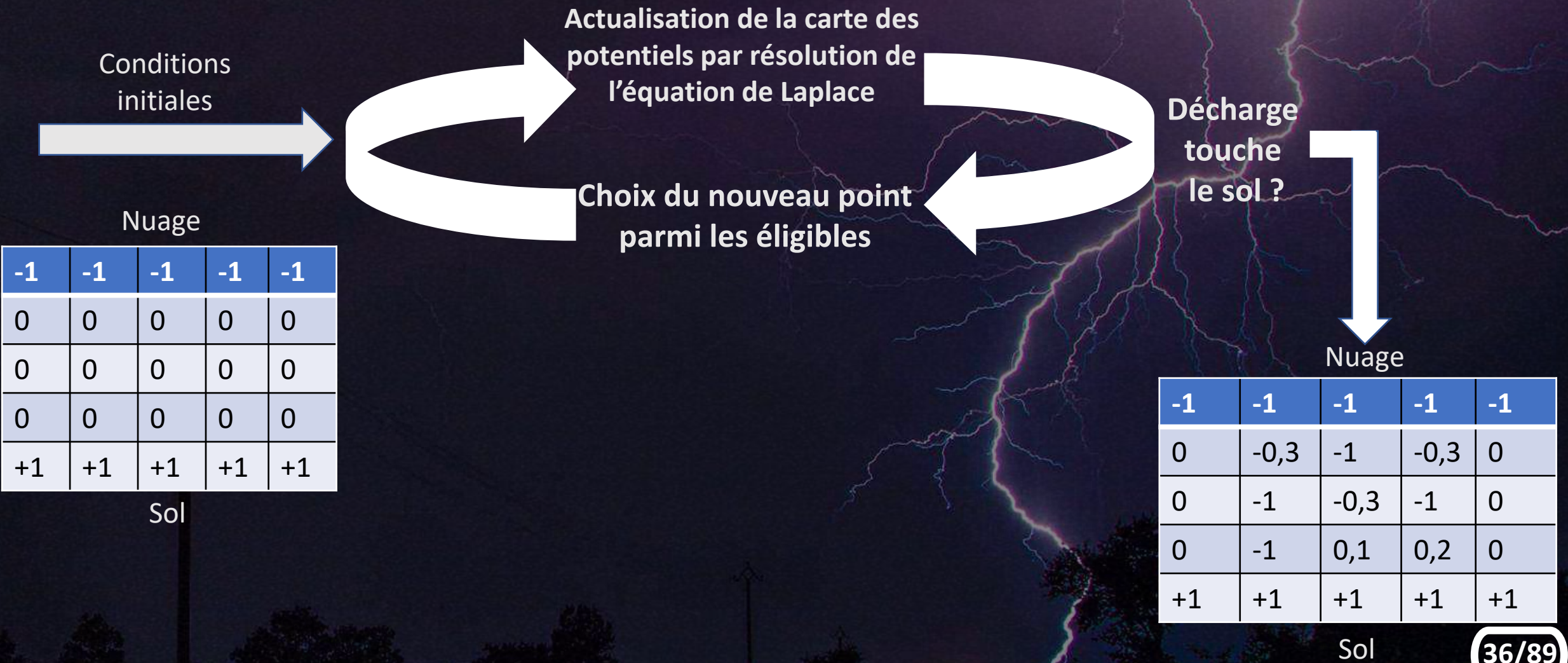
-1	-1	-1	-1	-1
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
+1	+1	+1	+1	+1

Sol

Actualisation de la carte des
potentiels par résolution de
l'équation de Laplace

Choix du nouveau point
parmi les éligibles

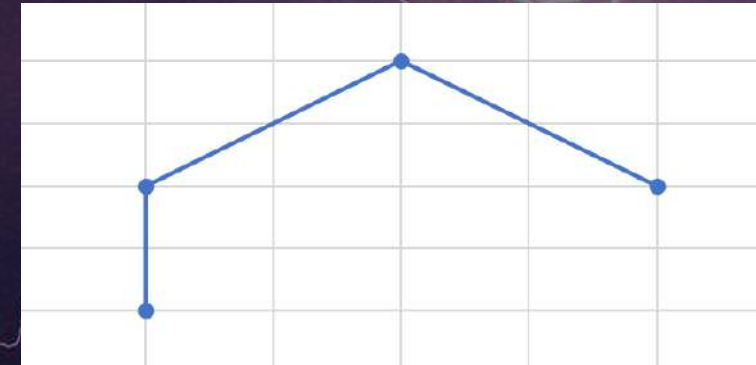
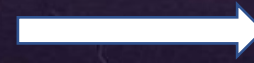
Exemple en taille 5x5



Obtention du tracé final de la décharge

-1	-1	-1	-1	-1
0	-0,3	-1	-0,3	0
0	-1	-0,3	-1	0
0	-1	0,1	0,2	0
+1	+1	+1	+1	+1

Carte des potentiels finale

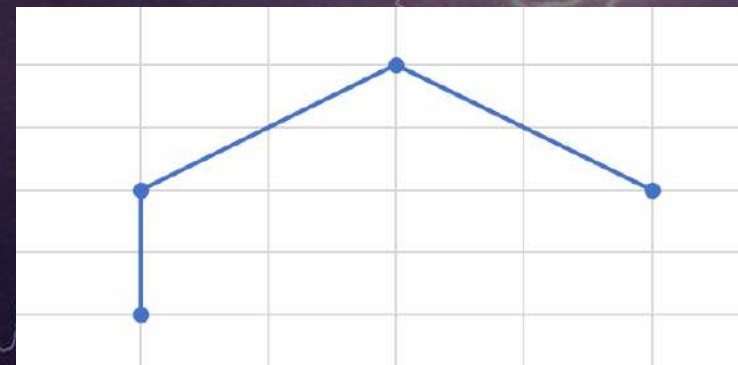
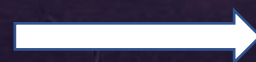


Points de potentiel -1 = Points des traceurs

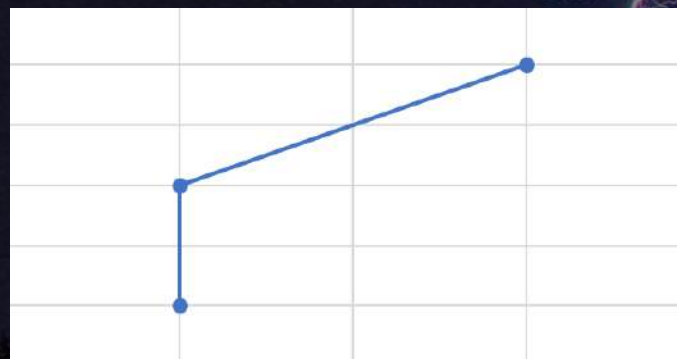
Obtention du tracé final de la décharge

-1	-1	-1	-1	-1
0	-0,3	-1	-0,3	0
0	-1	-0,3	-1	0
0	-1	0,1	0,2	0
+1	+1	+1	+1	+1

Carte des potentiels finale

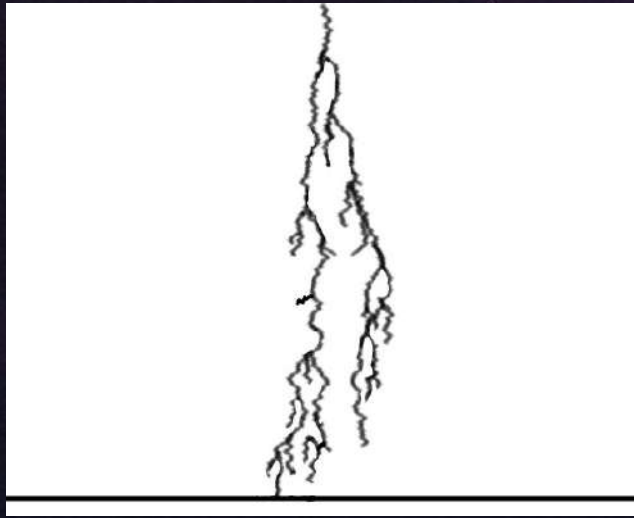


Points de potentiel -1 = Points des traceurs

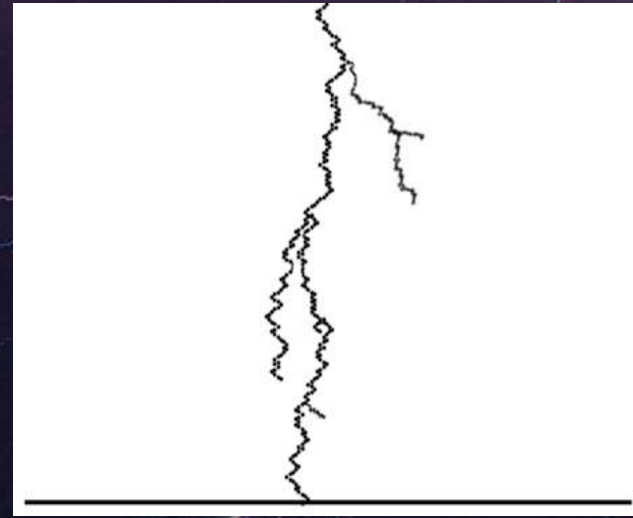


Suppression des points n'intervenant pas dans la décharge

Premiers résultats sur une grille 200x200



$\eta=2$



$\eta=4$

Améliorations de l'algorithme

Fixer le paramètre η par box-counting



Calcul de la dimension
fractale selon η

Fixer le paramètre η par box-counting



Calcul de la dimension
fractale selon η

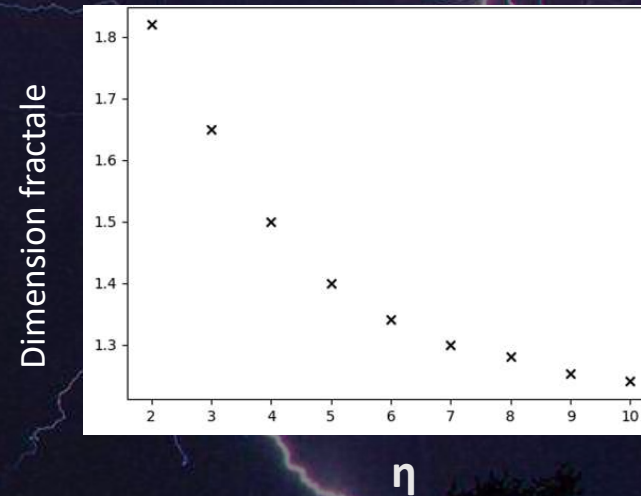


$$\delta = \lim_{\varepsilon \rightarrow 0} \frac{\ln(N(\varepsilon))}{\ln \frac{1}{\varepsilon}}$$

Fixer le paramètre η par box-counting

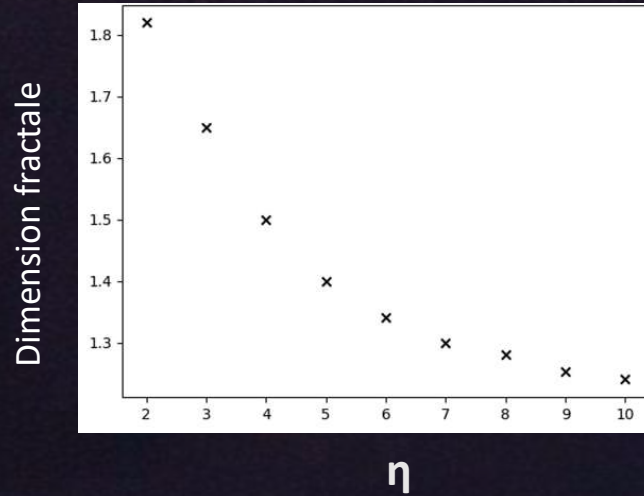
➔ Calcul de la dimension fractale selon η

➔
$$\delta = \lim_{\varepsilon \rightarrow 0} \frac{\ln(N(\varepsilon))}{\ln \frac{1}{\varepsilon}}$$



1-Paramètre η

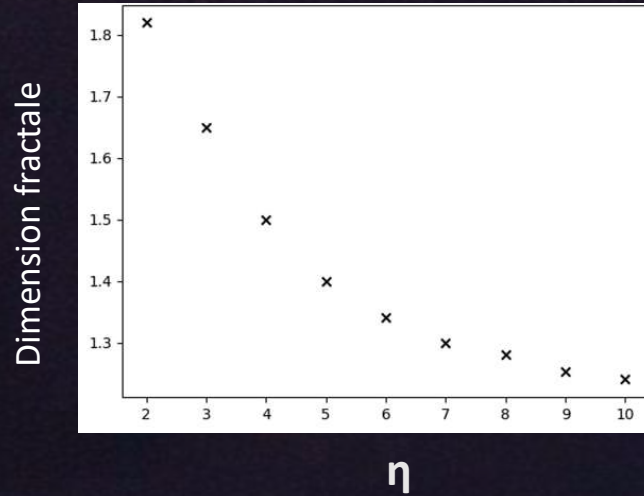
Fixer le paramètre η par box-counting



Calcul de la dimension fractale
moyenne d'un éclair réel

1-Paramètre η

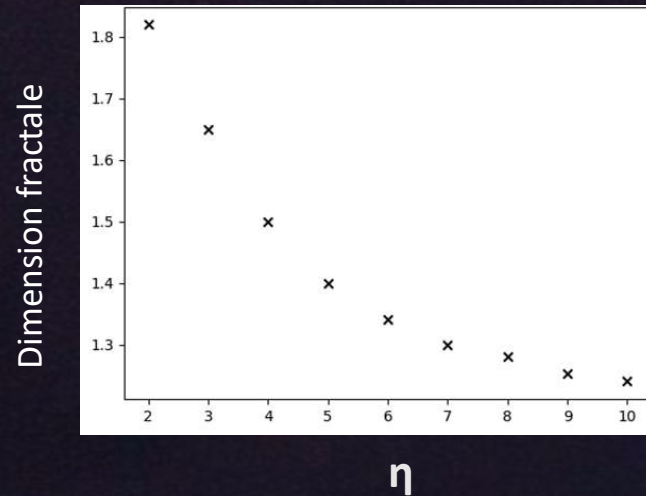
Fixer le paramètre η par box-counting



Calcul de la dimension fractale
moyenne d'un éclair réel

$\delta_{th} = 1,38$

Fixer le paramètre η par box-counting



Calcul de la dimension fractale
moyenne d'un éclair réel

$\eta=6$

$\delta_{th}=1,38$

Améliorer le temps d'exécution

Résolution de
l'équation de
Laplace



$O(n^3)$

Améliorer le temps d'exécution

Résolution de
l'équation de
Laplace



$O(n^3)$

Algorithme final



$O(n^4)$

Méthode de Gauss-Seidel

Jacobi

$$v_{i,j}(k+1) = \frac{v_{i+1,j}(k) + v_{i-1,j}(k) + v_{i,j+1}(k) + v_{i,j-1}(k)}{4}$$

Méthode de Gauss-Seidel

Jacobi

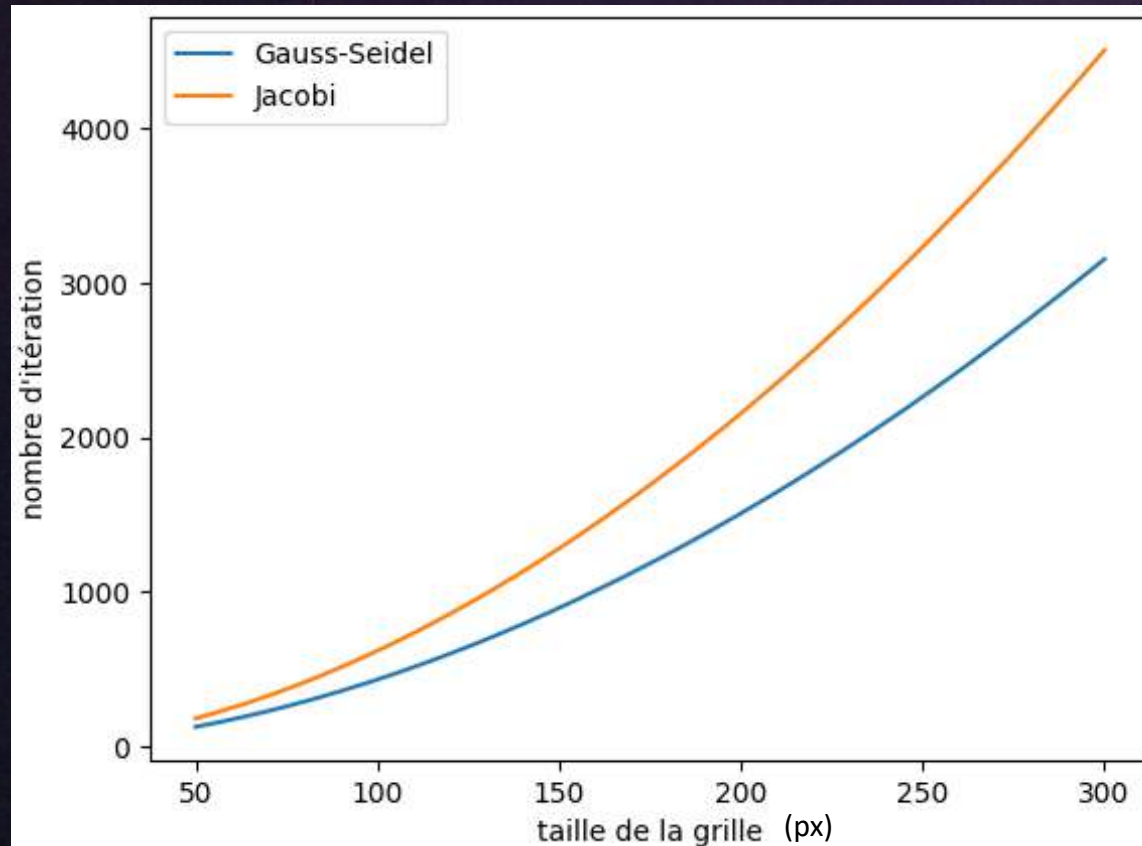
$$v_{i,j}(k+1) = \frac{v_{i+1,j}(k) + v_{i-1,j}(k) + v_{i,j+1}(k) + v_{i,j-1}(k)}{4}$$

Gauss-Seidel

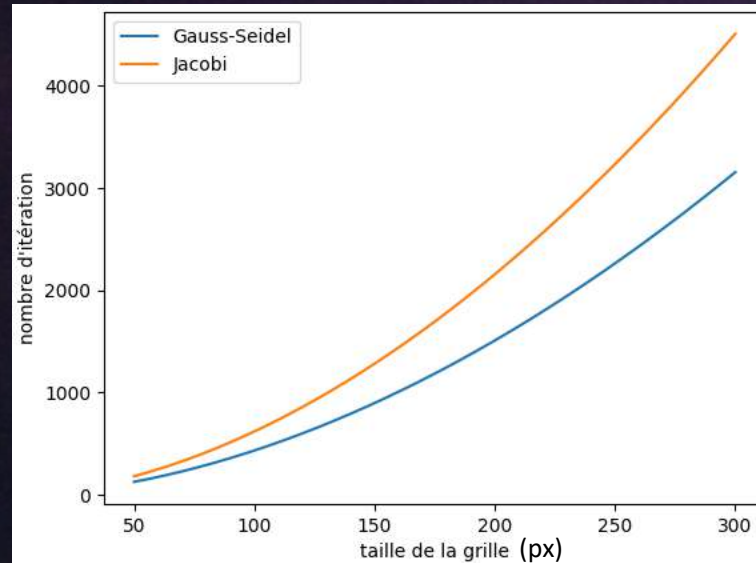
$$v_{i,j}(k+1) = (1-\omega)v_{i,j}(k) + \omega \frac{v_{i-1,j}(k+1) + v_{i,j-1}(k+1) + v_{i,j+1}(k) + v_{i+1,j}(k)}{4}$$

avec $\omega_{\text{idéal}} = \frac{2}{1+\frac{\pi}{n}}$

Comparaison des deux méthodes



Comparaison des deux méthodes



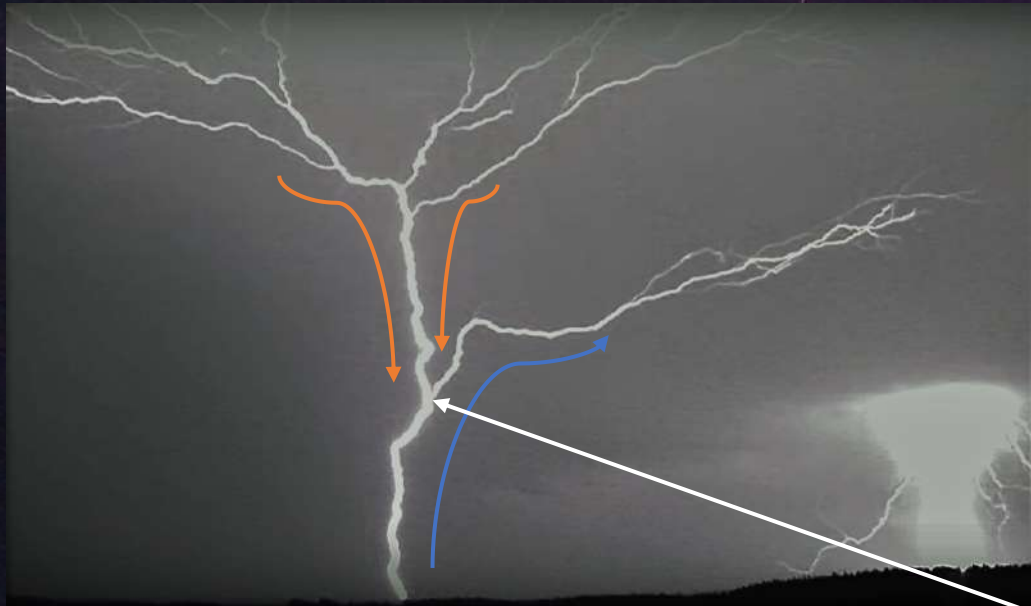
Jacobi



Gauss-Seidel

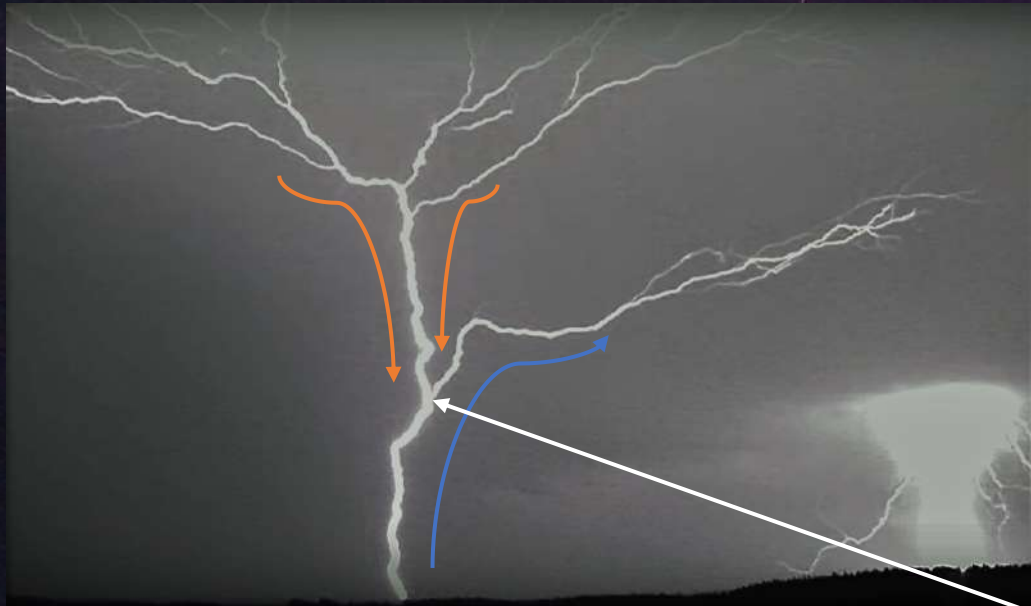


Prise en compte des traceurs ascendants



Point de contact
entre descendant
et ascendant

Prise en compte des traceurs ascendants

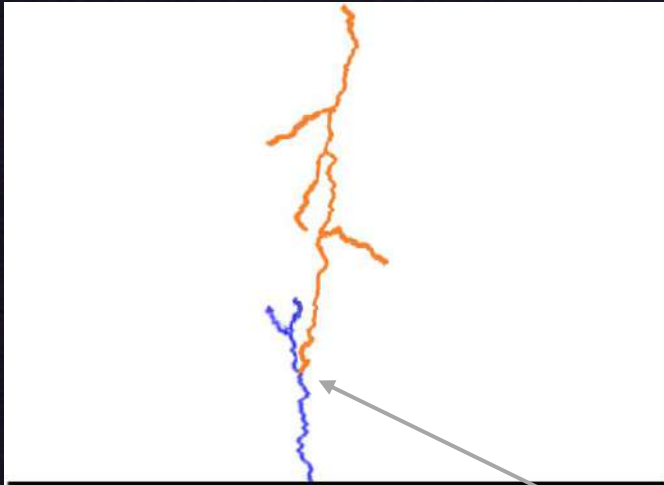


Point de contact
entre **descendant**
et **ascendant**



Décharge passant par le
point de contact

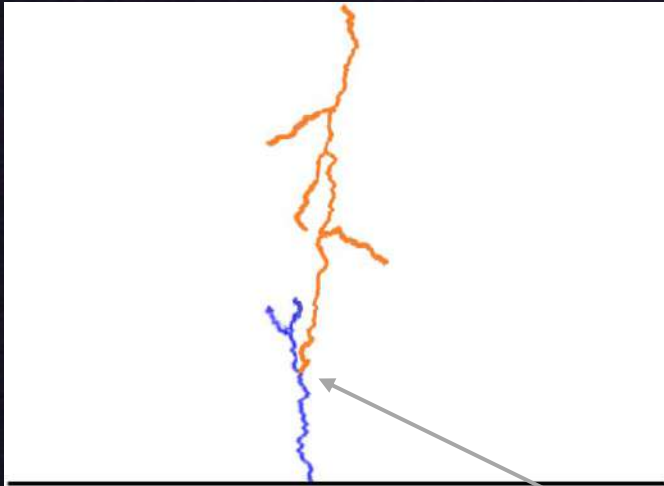
Prise en compte des traceurs ascendants



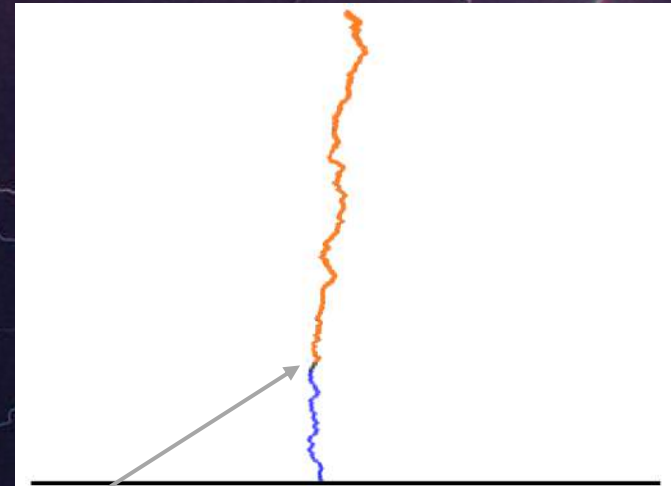
Déclenchement de l'ascendant
lorsque la tension entre le sol
et l'air est suffisante

Condition d'arrêt:
contact entre
descendant et
ascendant

Prise en compte des traceurs ascendants



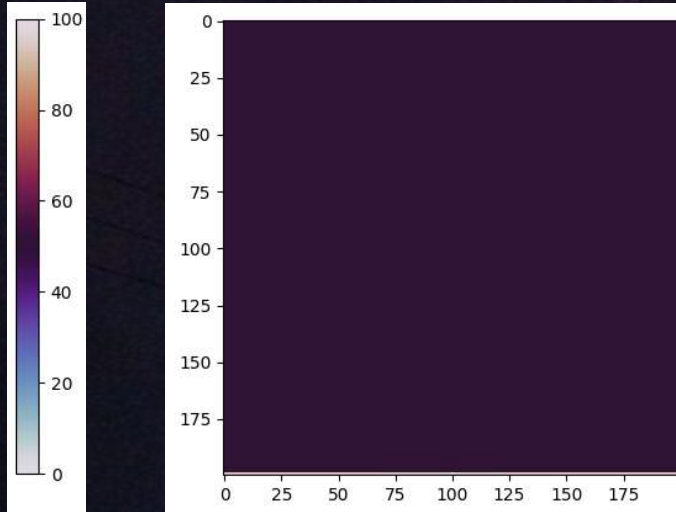
Déclenchement de l'ascendant
lorsque la tension entre le sol
et l'air est suffisante



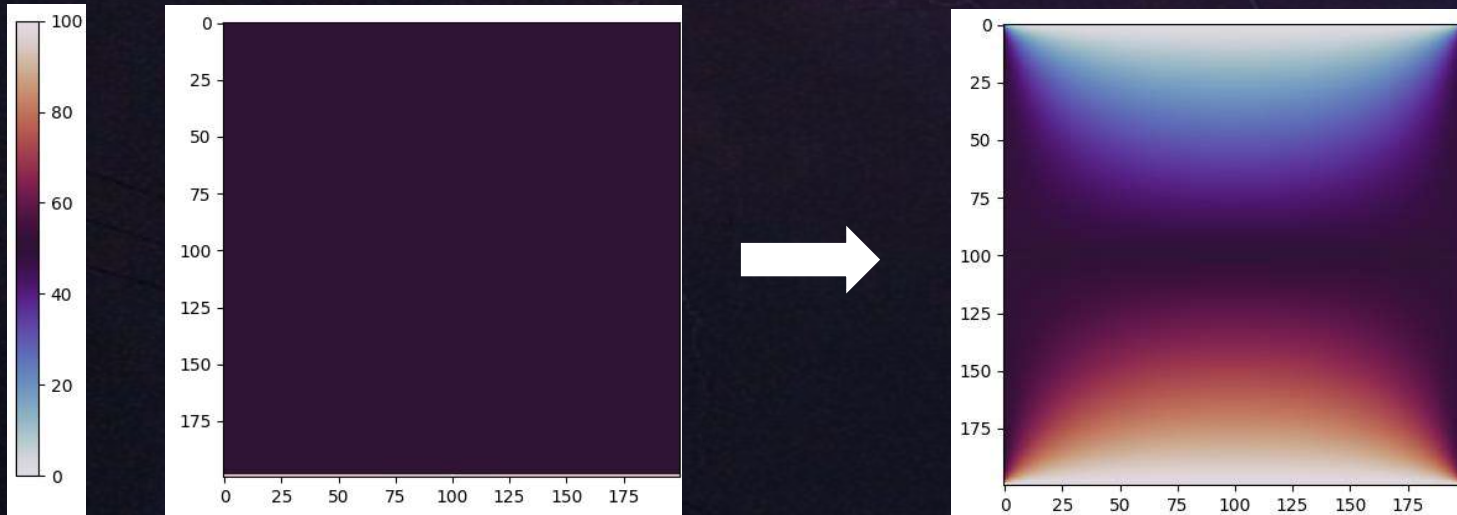
Condition d'arrêt:
contact entre
descendant et
ascendant

Trajectoire de la
décharge

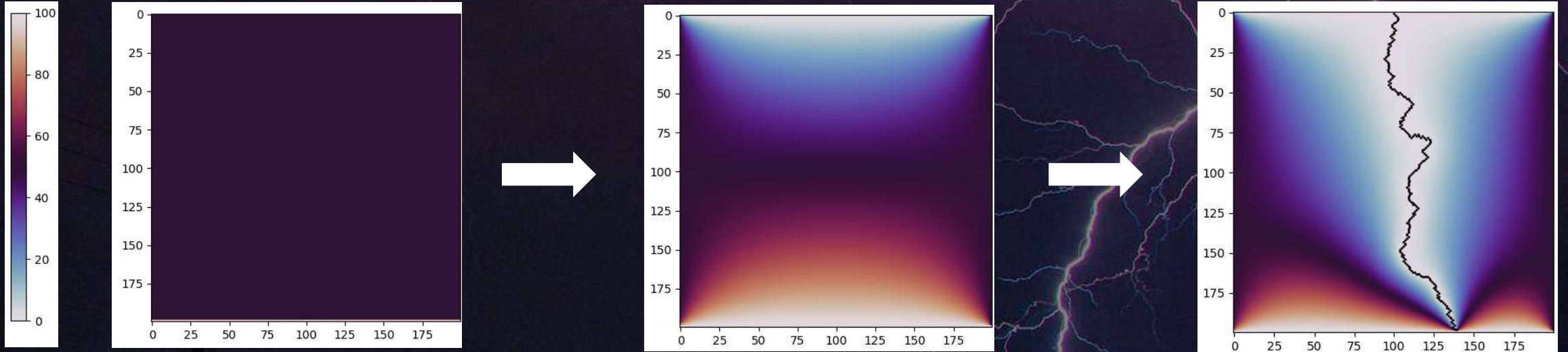
Résultats sur une grille 200x200



Résultats sur une grille 200x200



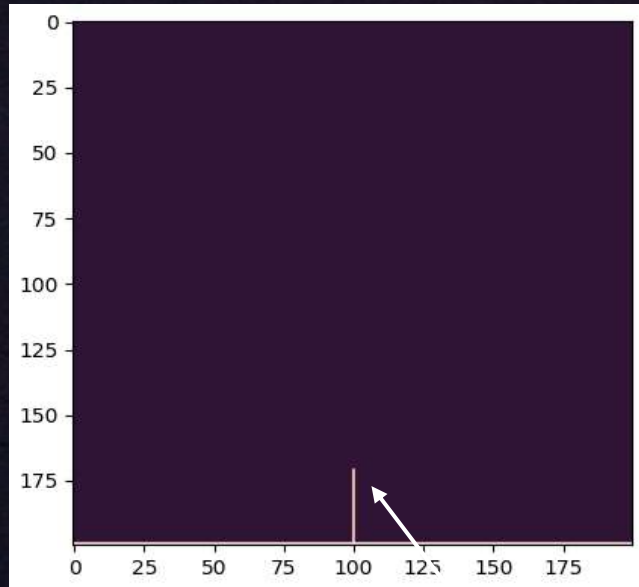
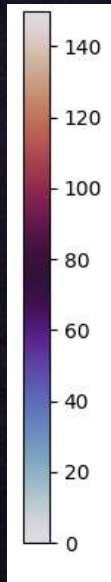
Résultats sur une grille 200x200



Application

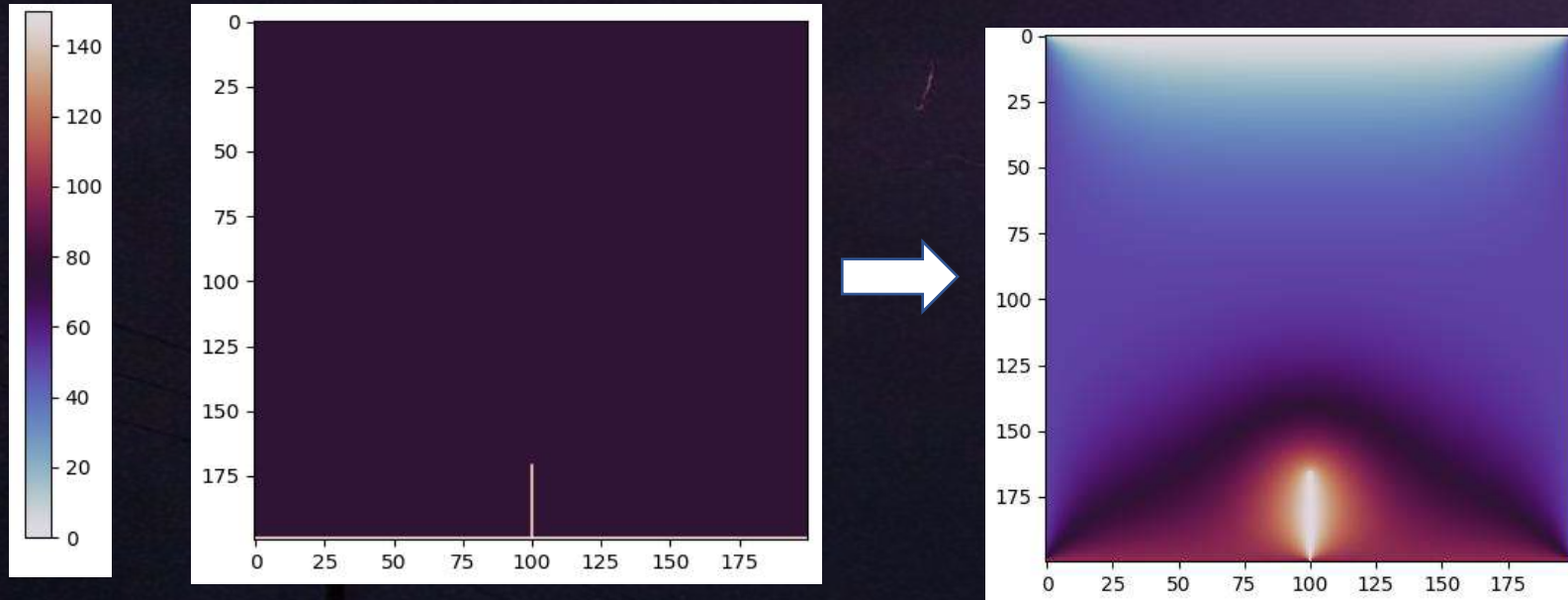
à l'efficacité des paratonnerres

1-Présence d'un paratonnerre

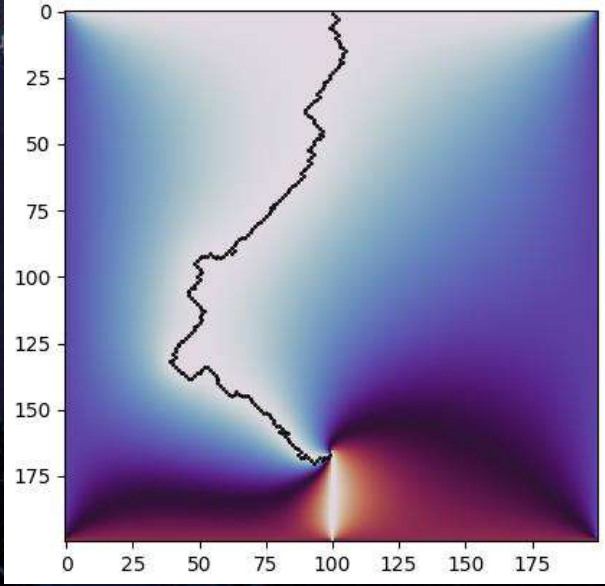
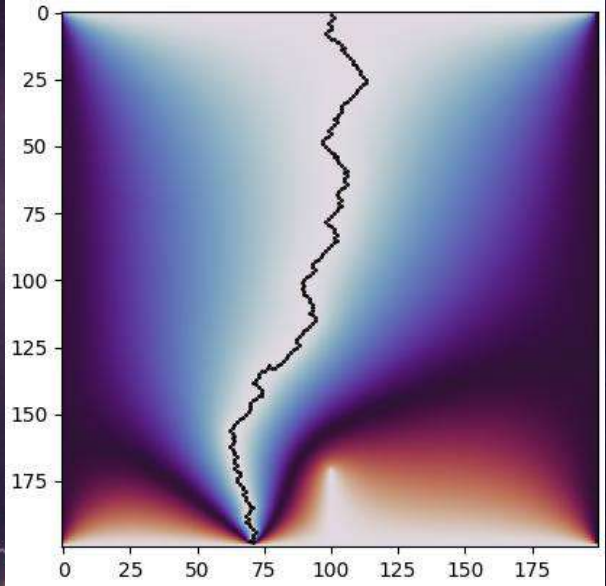
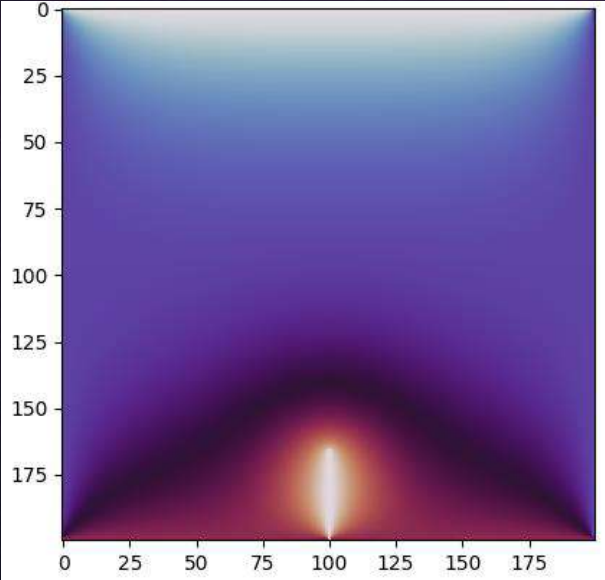
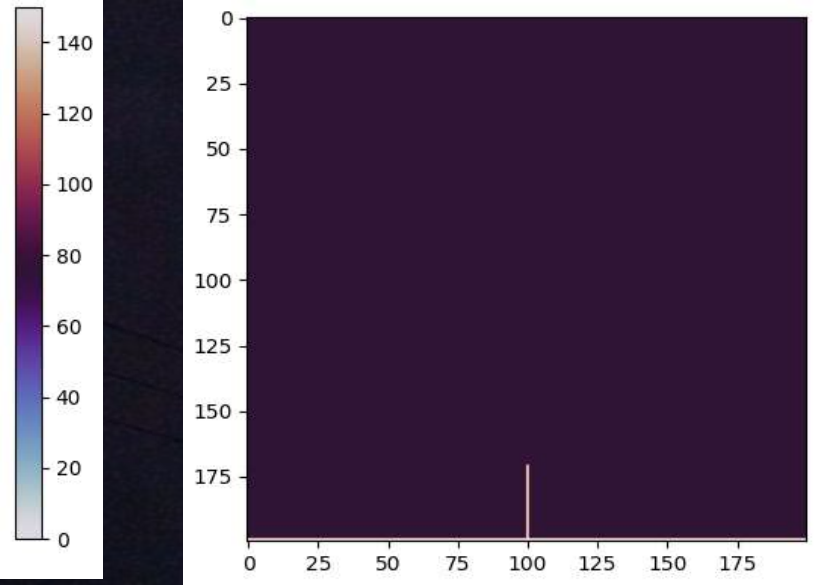


Potentiel plus élevé au niveau de la tige pour simuler l'effet de pointe

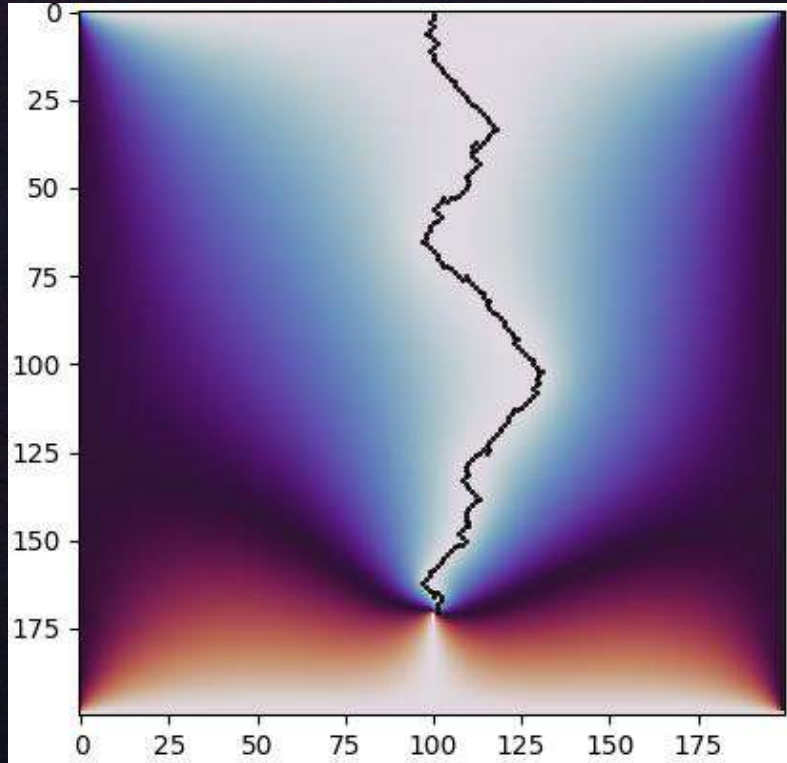
1-Présence d'un paratonnerre



1-Présence d'un
paratonnerre



2-Analogie de longueur

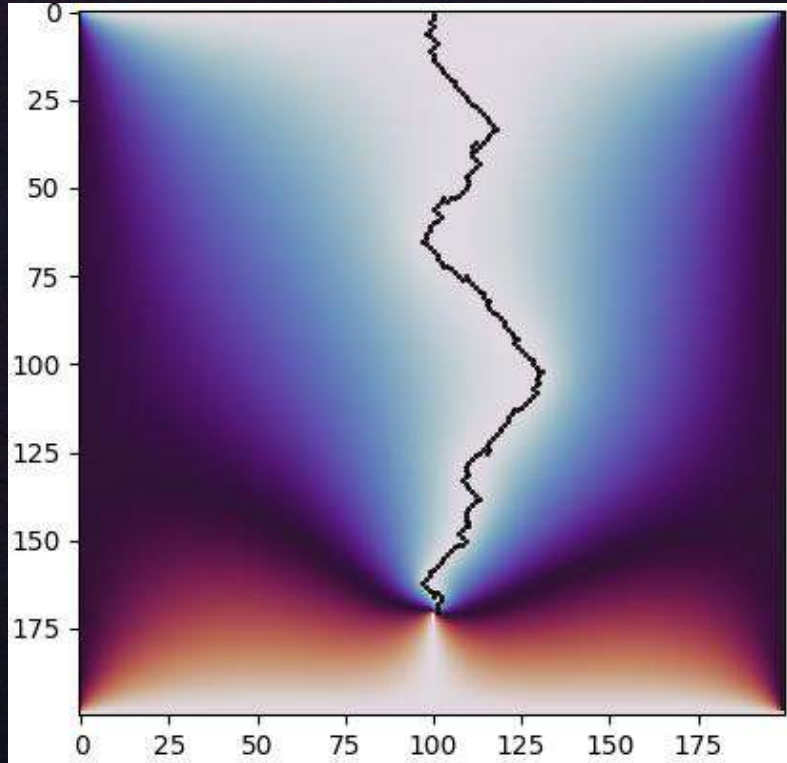


Bas du cumulonimbus: 1000m

200 pixels

Sol: 0m

2-Analogie de longueur



200 pixels



1 pixel = 5m

Bas du cumulonimbus: 1000m

Sol: 0m

Rayon de protection à 95% en fonction de la taille de la tige

Avec 50 essais par taille, on obtient:

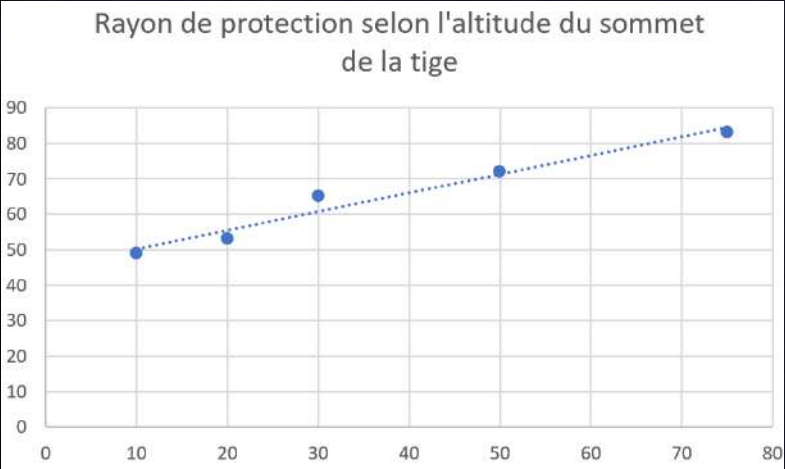
Altitude sommet de la tige (m)	10	20	30	50	75
Rayon de protection (m)	49	53	65	72	83

Rayon de protection à 95% en fonction de la taille de la tige

Avec 50 essais par taille, on obtient:

Altitude sommet de la tige (m)	10	20	30	50	75
Rayon de protection (m)	49	53	65	72	83

Ce qui correspond au graphe:

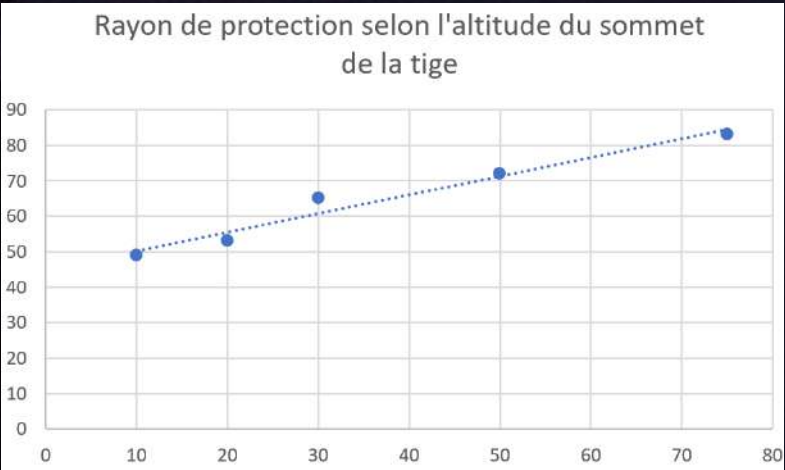


Rayon de protection à 95% en fonction de la taille de la tige

Avec 50 essais par taille, on obtient:

Altitude sommet de la tige (m)	10	20	30	50	75
Rayon de protection (m)	49	53	65	72	83

Ce qui correspond au graphe:



De la forme $y=ax+b$ avec
 $a \approx 0,58$ et $b \approx 46m$

Surface de protection annoncée par la société Schirtec

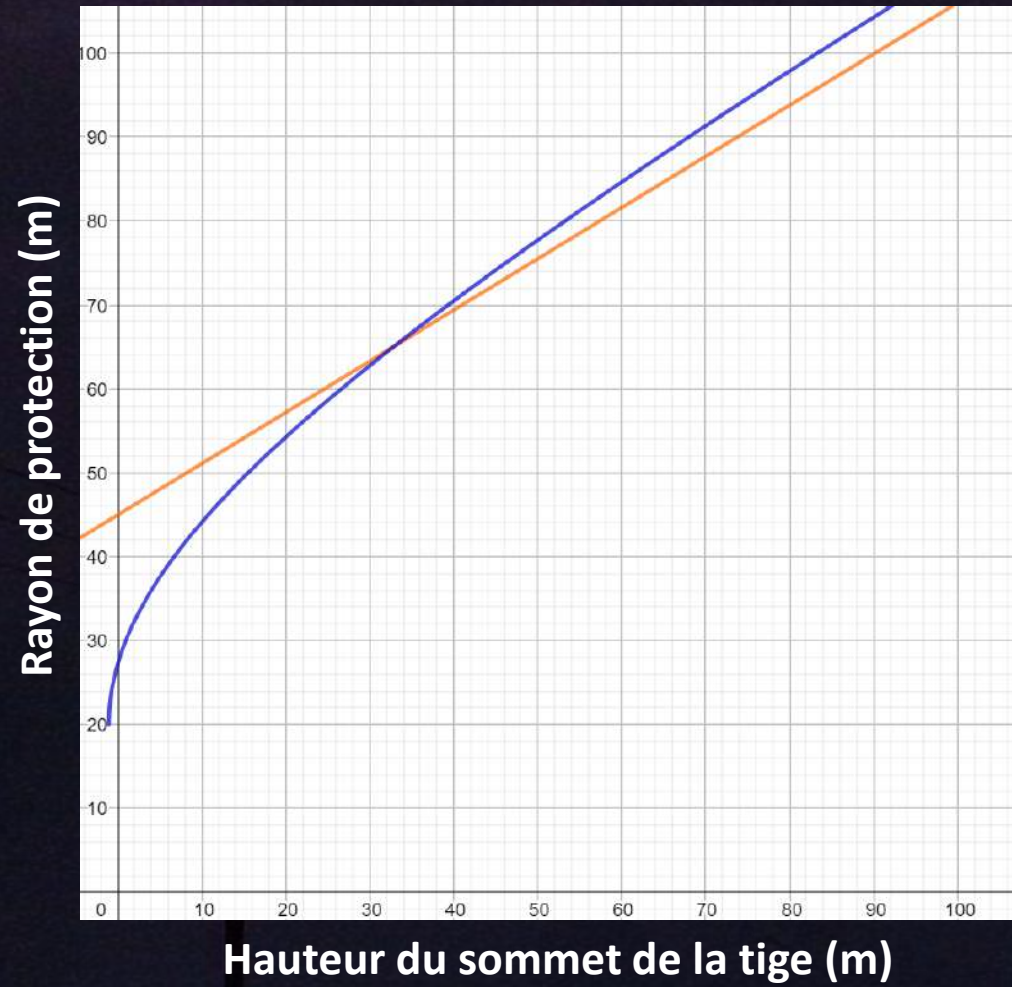
$$A_p(h)=h^2+12Q h+Q^2$$

Surface de protection annoncée par la société Schirtec

$$A_p(h)=h^2+12Q h+Q^2$$

$$\longrightarrow R_p(h)=\sqrt{\frac{h^2+12 Q h+Q^2}{\pi}}$$

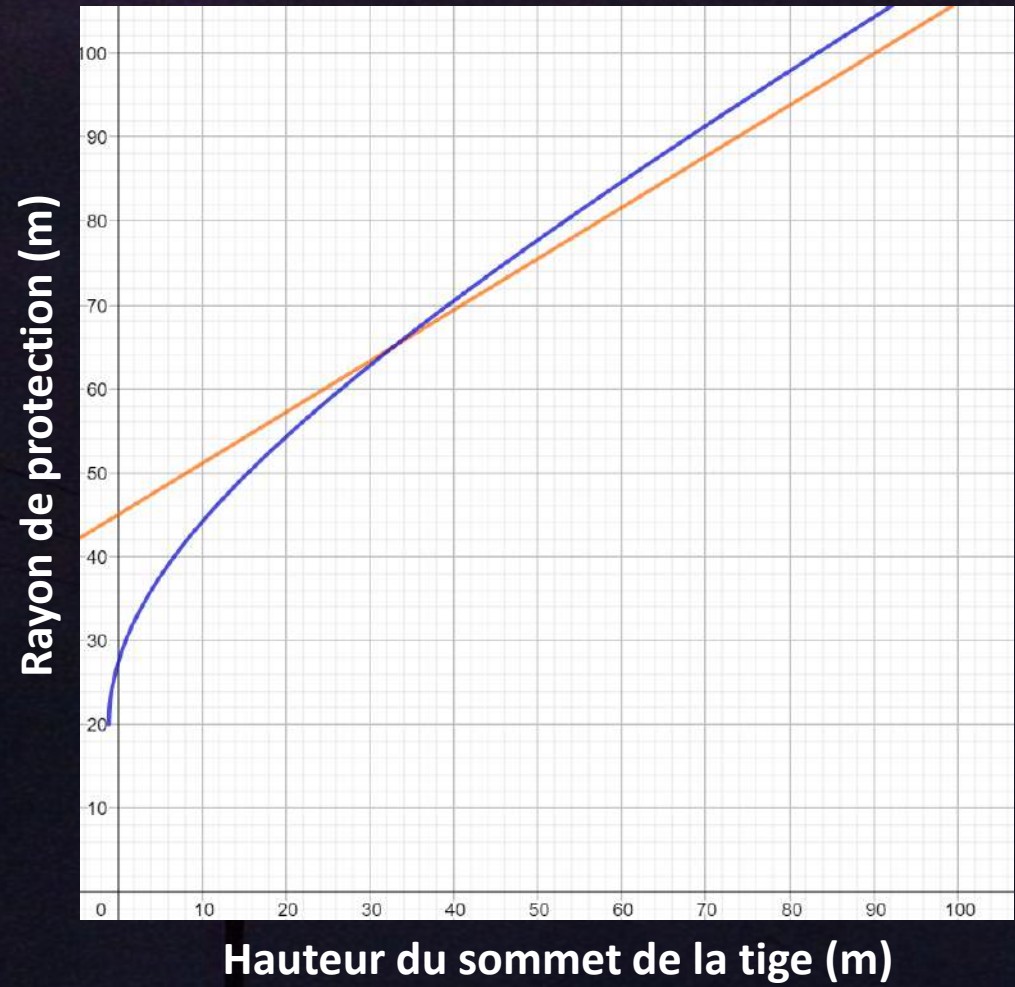
Confrontation des résultats



$R_{théorique}$

R_{obtenu}

Confrontation des résultats



Asymptote en

$$y=\frac{1}{\sqrt{\pi}}x+49$$

$$\frac{1}{\sqrt{\pi}} \approx 0,56$$

R_{théorique} R_{obtenu}

Protection d'infrastructures médicales

Application à l'hôpital Lapeyronie de Montpellier

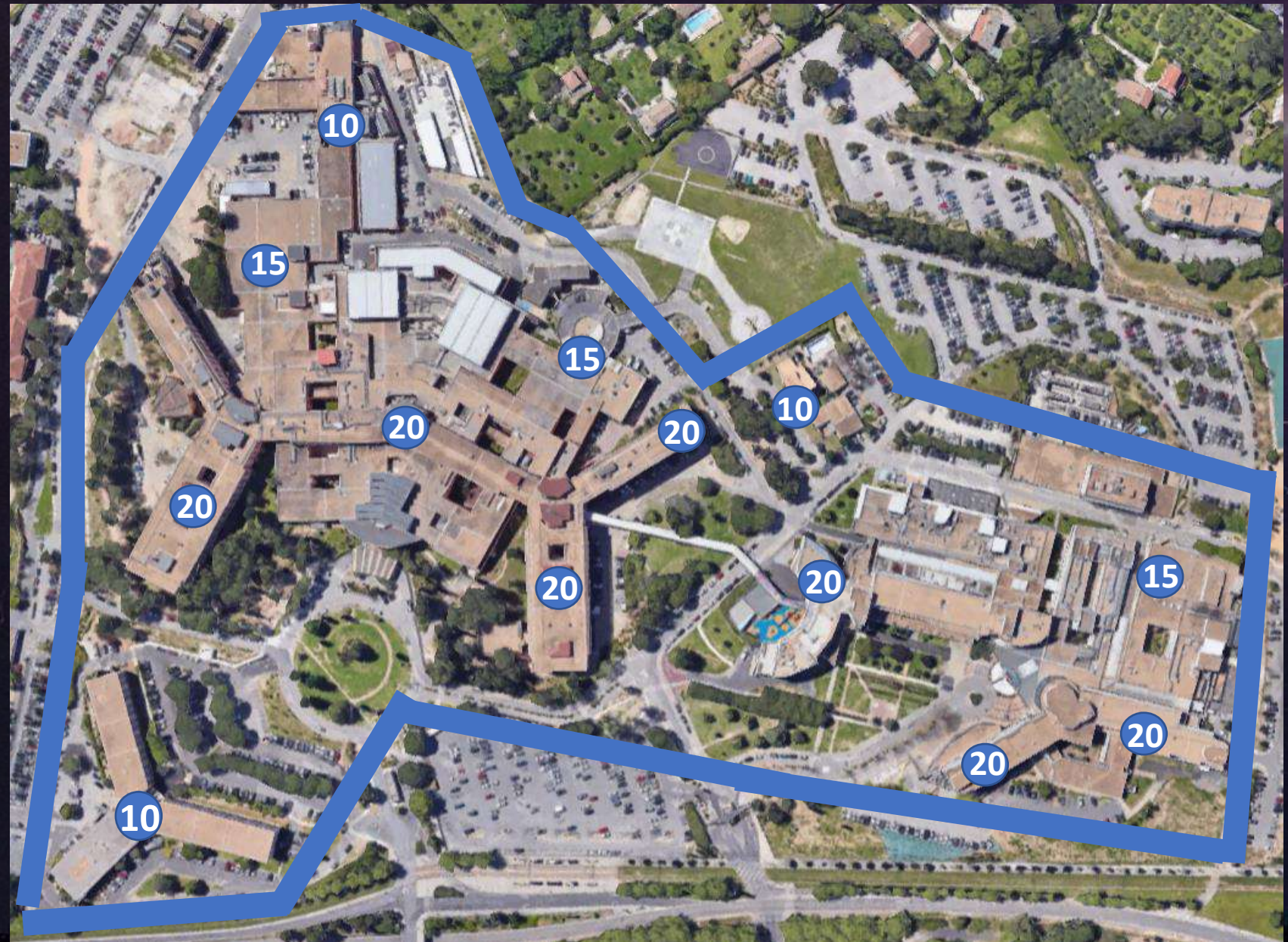


1-Hôpital Lapeyronie



1-Hôpital Lapeyronie

Hauteur des
bâtiments
+
Paratonnerre
(m)



1-Hôpital Lapeyronie



Zones protégés



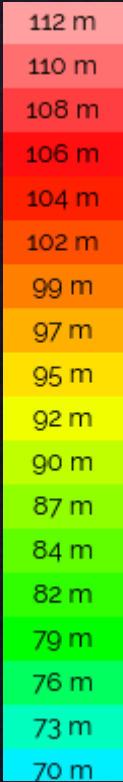
Application à l'hôpital Carémeau de Nîmes



2-Hôpital Carémeau



2-Hôpital Carémeau



Altitude



100 m

2-Hôpital Carémeau

Altitude rapport
au 0 fixé
+
Hauteur des
bâtiments
+
Paratonnerre
(m)



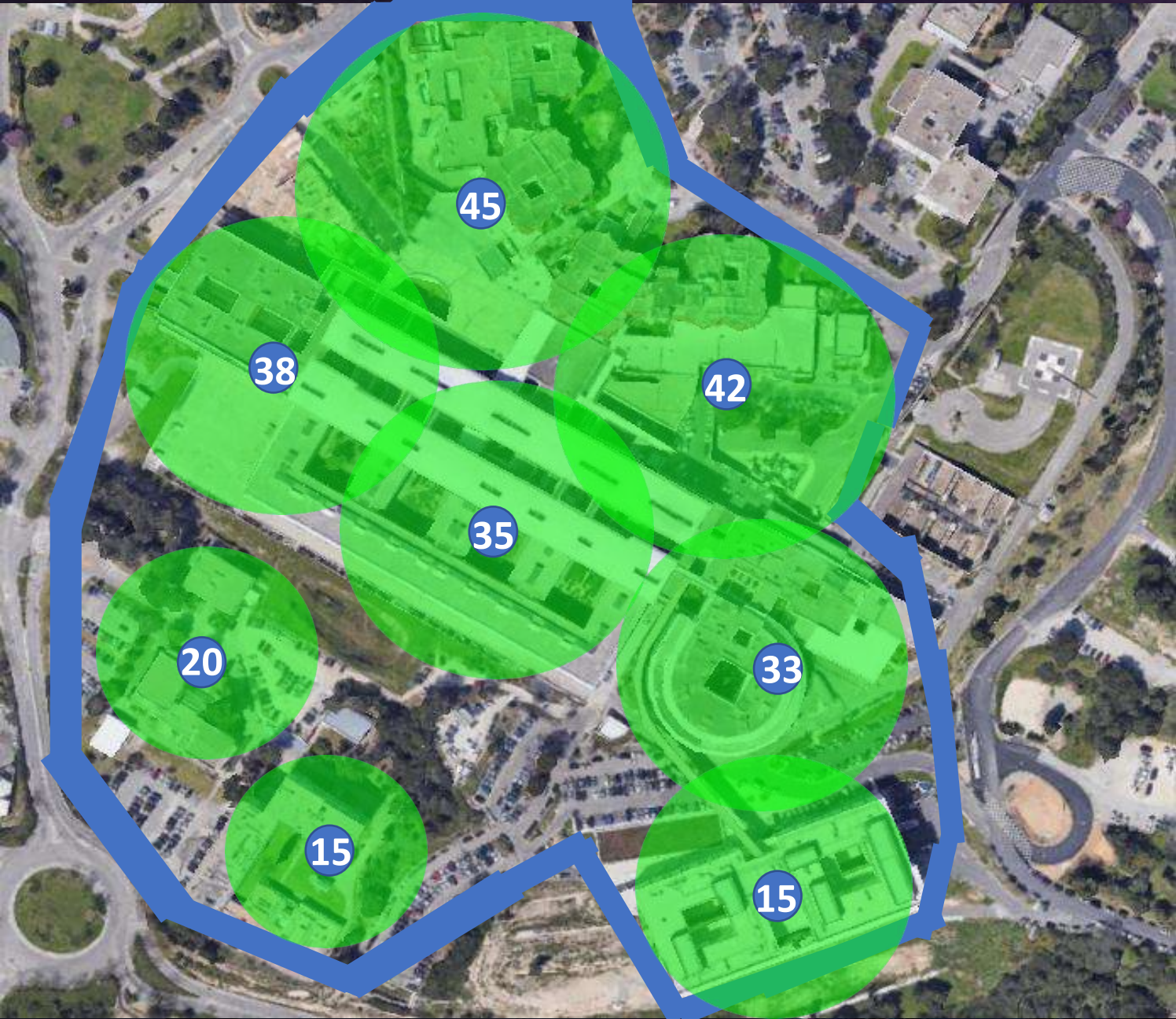
Altitude



2-Hôpital Carémeau



Zones protégées





Conclusion



Annexe

codes V3.py

```
001 | import numpy as np
002 | import matplotlib.pyplot as plt
003 | from math import *
004 | from random import *
005 |
006 |
007 | #Taille de la grille
008 | N = 200
009 |
010 |
011 | #On fixe eta=6
012 | eta=6
013 |
014 |
015 | # définition des paramètres physiques de l'expérience
016 | # potentiels >0 pour s'affranchir des erreurs de signe
017 | # tout en MV
018 | Vsol = 100          # du sol
019 | Vair = 50.0         # de l'air
020 | Vnuage= 0           # potentiel du nuage
021 |
022 |
023 | # création de la carte des potentiels
024 | V = np.zeros([N,N])
025 |
026 |
027 | # définition des conditions initiales
028 | V[0,:] = Vnuage     # bord supérieur
029 | V[-1,:] = Vsol      # bord inférieur
030 | V[1:N-1,0:N] = Vair # intérieur de la grille
031 |
032 | # facteur w
033 | w=2/(1+pi/N)
034 |
035 |
036 | def ecart_quadratique_moyen(V1,V2):
037 |     e=0
038 |     for i in range(N):
039 |         for j in range(N):
040 |             e+=(V1[i,j]-V2[i,j])**2
041 |     return e
042 |
```



```

043 |
044 | # résolution de l'équation de Laplace par méthode de Jacobi
045 | def jacobi(V):
046 |     ecart = 1.0
047 |     iteration = 0
048 |     e=10e-3 #seuil de résolution
049 |     while ecart > e:
050 |         # sauvegarde de la grille courante pour calculer l'écart
051 |         V1 = V.copy()
052 |
053 |         # méthode de Jacobi
054 |         for j in range (N-1):
055 |             for i in range (N-1):
056 |                 if V[i,j]!=0 and V[i,j]<Vsol:
057 |                     V[i,j]=0.25*(V[i-1,j]+V[i,j-1]+V[i,j+1]+V[i+1,j])
058 |
059 |         # critère de convergence
060 |         ecart = ecart_quadratique_moyen(V,V1)
061 |         iteration += 1
062 |     return V
063 |
064 |
065 | # résolution de l'équation de Laplace par méthode de Gauss-Seidel
066 | def gauss_seidel(V):
067 |     ecart = 1.0
068 |     iteration = 0
069 |     e=10e-3 #seuil de résolution
070 |     while ecart > e:
071 |         # sauvegarde de la grille courante pour calculer l'écart
072 |         V1 = V.copy()
073 |
074 |         # méthode de Gauss-Seidel
075 |         for j in range (N-1):
076 |             for i in range (N-1):
077 |                 if V[i,j]!=0 and V[i,j]<Vsol:
078 |                     V[i,j]=(1-w)*V[i,j]+w*0.25*(V[i-1,j]+V[i,j-1]+V[i,j+1]+V[i+1,j])
079 |
080 |         # critère de convergence
081 |         ecart = ecart_quadratique_moyen(V,V1)
082 |         iteration += 1
083 |     return V
084 |
085 |
086 | #renvoie la liste des points éligibles à être traversés par la décharge

```



```

087| def eligible(V):
088|     L=[]
089|     for i in range(1,N-1):
090|         for j in range (1,N-1):
091|             if V[i,j]==Vnuage:
092|                 #eligible=unique voisin traversé par la décharge
093|                 if (V[i+1,j]==Vnuage and V[i-1,j]<Vnuage and V[i,j+1]<Vnuage and V[i,j-1]<Vnuage) or (V[i-1,j]==Vnuage and
V[i+1,j]<Vnuage and V[i,j+1]<Vnuage and V[i,j-1]<Vnuage) or (V[i,j+1]==Vnuage and V[i-1,j]<Vnuage and V[i,j-1]<Vnuage and
V[i+1,j]<Vnuage) or (V[i,j-1]==Vnuage and V[i-1,j]<Vnuage and V[i+1,j]<Vnuage and V[i,j+1]<Vnuage):
094|                     L.append((j,i))
095|     return L
096|
097|
098|
099| # fonction principale
100| def foudre(point_depart, V, bool=False):
101|
102|     (j,i)=point_depart
103|
104|     #cas d'arret: touche le sol ou recontre l'ascendant
105|     if j>=N-1 or j<0:
106|         return
107|
108|     #tracé point et mise au potentiel du nuage
109|     plt.scatter([i],[j], s=1, color='black')
110|     V[j,i]=0
111|
112|     # actualisation de la carte de potentiels
113|     V=gauss_seidel(V)
114|
115|     liste_points_éligibles=eligible(V)
116|
117|     #somme des probas des points éligibles puissance eta
118|     sum=0
119|     for k in liste_points_éligibles:
120|         k=(j,i)
121|         sum+=V[i,j]**eta
122|
123|     #liste des probas
124|     liste_probas=[]
125|     for k in liste_points_éligibles:
126|         k=(j,i)
127|         liste_probas.append((V[i,j]**eta)/sum)
128|

```



```

129|
130|     #choix du prochain point
131|     new=choices(liste_points_éligibles,liste_probas)
132|
133|     #initiation de l'ascendant
134|     for k in range(N):
135|         if V[1,k]<Vsol-0.1:
136|             bool=True
137|             #appel récursif sur l'ascendant
138|             foudre((k,1),V,bool)
139|             #Sortie de boucle forcée pour garder un seul ascendant
140|             break
141|
142|     #appel récursif pour sur nouveau point élu
143|     foudre(new,V,bool)
144|
145|
146|
147| #Affichage 1
148| plt.imshow(V, cmap='twilight')
149| plt.colorbar()
150| plt.show()
151|
152|
153| #Initiation du descendant au milieu de la 1ère ligne
154| foudre((1,int(N/2)),V)
155|
156|
157| #Affichage 2
158| plt.imshow(V, cmap='twilight')
159| plt.colorbar()
160| plt.show()
161|
162| ##
163| ##
164|
165|
166|
167| def boxcount(Z, k):
168|     S = np.add.reduce(
169|         np.add.reduce(Z, np.arange(0, Z.shape[0], k), axis=0),
170|         np.arange(0, Z.shape[1], k), axis=1)
171|
172|     # Calcul du nombre de carrés de taille k nécessaire pour recouvrir la figure

```



```

173 |     return len(np.where((S > 0) & (S < k*k))[0])
174 |
175 |
176 | def dim_fractale(Z, seuil=0.9):
177 |
178 |     # Transforme Z en tableau numpy de booléens
179 |     Z = (Z < seuil)
180 |
181 |     # Dimension de l'image rognée en carré
182 |     p = min(Z.shape)
183 |
184 |     # Plus grande entier n tel que de 2^n <= p
185 |     n = int(np.log(p)/np.log(2))
186 |
187 |     # definition des tailles de carrés
188 |     # de 2 à 2^n
189 |     liste_tailles = 2**np.arange(n, 1, -1)
190 |
191 |     # Compte de nombre de carrés nécessaires selon la taille des carrés avec l'appel de boxcount
192 |     c = []
193 |     for taille in liste_tailles:
194 |         c.append(boxcount(Z, taille))
195 |
196 |     # regression linéaire de la décroissance des tailles de carrés avec polyfit
197 |     coeffs = np.polyfit(np.log(liste_tailles), np.log(c), 1)
198 |     return -coeffs[0]#droite décroissante, on renvoie l'opposé du coef directeur
199 |
200 |
201 |
202 | I = plt.imread("C://Users//florent//Documents//MP//tipe//box_counting//eclair.png")
203 |
204 | dimension_fractale(I/n)
205 |
206 |

```