TP3 - Percolation

La percolation désigne le passage d'un fluide à travers un solide poreux. Ce concept a été formulé en 1957 par le mathématicien anglais J. M. Hammersley qui cherchait à comprendre comment les masques à gaz des soldats devenaient inefficaces. Il s'est ensuite peu à peu répandu dans de nombreux domaines : infiltration des eaux de pluie jusqu'aux nappes phréatiques, propagation d'une épidémie ou d'un feu de forêt...

Tous ces phénomènes physiques ont un point commun : ils présentent un comportement similaire à une transition de phase. Vous connaissez ce phénomène : quand on refroidit de l'eau liquide depuis la température ambiante, rien ne se passe en apparence jusqu'à ce que l'on atteigne la température de congélation de l'eau. Et là, très rapidement et sur une étroite bande de température, l'eau passe de l'état liquide à l'état solide. Cette notion de seuil, fondamentale en percolation, est précisément étudiée dans ce TP.

Objectifs

- Retravailler la manipulation des tableaux Numpy et des listes/piles ;
- Traduire un algorithme complexe dans le langage Python.
- Comprendre et utiliser la documentation d'une fonction en Python ;

1 Description du modèle

On considère dans tout ce TP une grille carrée de taille $n \times n$, chaque case pouvant être « ouverte » (avec une probabilité $p \in [0, 1[$) ou « fermée » (avec une probabilité $1 - p$). La première question à laquelle nous allons essayer de répondre est la suivante : est-il possible de joindre le haut et le bas de la grille par une succession de cases ouvertes adjacentes ? (on parle alors de réussite de la percolation). En observant la figure ci-dessous, on s'aperçoit que la réussite ou non de la percolation dépend beaucoup de p : plus celle-ci est grande, plus les chances de réussite sont importantes.

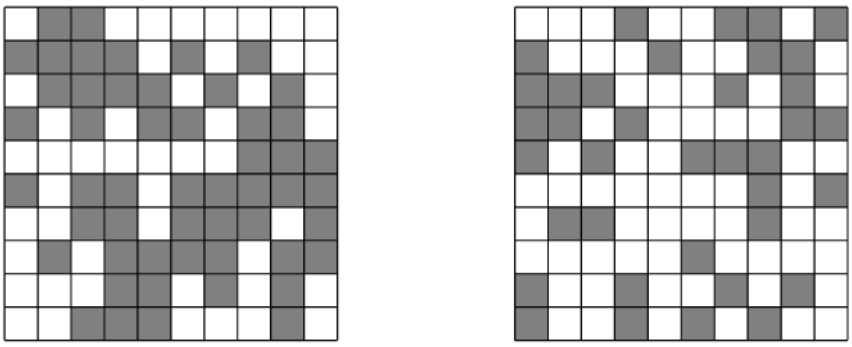


FIGURE 1 – Deux exemples de grilles 10×10 (les cases ouvertes sont les cases blanches). La percolation n'est possible que dans le second cas.

2 Création et visualisation de la grille

Travail préliminaire

Récupérer le fichier Python `TP3_percolation.py` mis à disposition par votre professeur, puis ouvrez-le dans Spyder. Il contient l'ensemble des programmes que vous devrez compléter pendant le TP.

La grille de percolation sera représentée par un tableau Numpy à deux dimensions (de type `np.array`).

Principales manipulations sur les tableaux Numpy

- la fonction `np.zeros((n, p))` renvoie un tableau de n lignes et p colonnes ne contenant que des `0.0` ;
- on accède (en lecture ou en écriture) à la case d'indice (i, j) d'un tableau `tab` par la syntaxe `tab[i][j]` ou `tab[i,j]` ;
- l'attribut `T.shape` retourne le couple (n, p) correspondant au nombre de lignes et de colonnes de `T`.

On rappelle que durant tout le TP, les grilles de percolation considérées seront de taille $n \times n$. Les cases fermées contiendront le flottant `0.0` et les cases ouvertes le flottant `1.0`.

Question 1. Compléter la fonction `creationGrille(p, n)` qui renvoie un tableau de taille $n \times n$ dans lequel chaque case sera ouverte avec la probabilité p et fermée sinon. Tester votre fonction avec $n = 10$. On utilisera la fonction `rand` (déjà importée) permettant de générer un flottant pseudo-aléatoire dans $[0, 1[$.

Pour visualiser simplement une grille `T`, nous allons utiliser la fonction `plt.imshow(T)` qui affiche cette dernière sous forme de cases colorées en fonction de leur valeur. L'argument par défaut `cmap` de cette fonction permet de modifier l'échelle de couleur utilisée.

Question 2. Observer, décommenter et exécuter les lignes de codes associées à cette question dans le fichier `TP3_percolation.py`.

3 Percolation

Une fois la grille créée, les cases ouvertes de la première ligne sont remplies par un fluide, qui sera représenté par la valeur `0.5` dans les cases correspondantes. Le fluide pourra ensuite être diffusé à chacune des cases ouvertes voisines d'une case contenant déjà le fluide jusqu'à remplir toutes les cases ouvertes possibles.

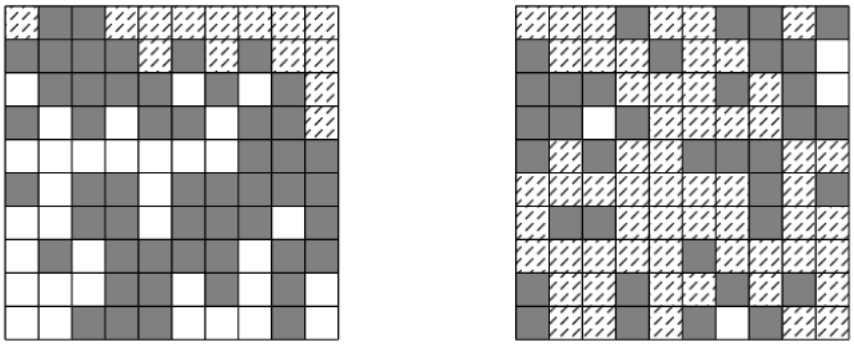


FIGURE 2 – Les deux grilles de la figure 1, une fois le processus de percolation terminé (pour plus de lisibilité, le fluide est représenté ici par des hachures).

Pour remplir la grille de fluide, on utilise l'algorithme ci-dessous :

1. Créer une liste contenant initialement les coordonnées des cases ouvertes de la première ligne de la grille et remplir ces cases de liquide.
2. Puis, tant que cette liste n'est pas vide, effectuer les opérations suivantes :
   → extraire de cette liste les coordonnées de la dernière case ;
   → ajouter à la liste les coordonnées des cases voisines de la grille qui sont encore vides, et les remplir de liquide.

   L'algorithme se termine quand la liste est vide.

Conseil

La question 4 a pour objectif l'implémentation cet algorithme plus complexe. Si vous souhaitez des indications y parvenir, n'hésitez pas à en demander !

Question 3. Lors de l'examen des cases voisines dans l'étape 2, il faut s'assurer qu'on ne sort pas de la grille. Compléter la fonction `estDansGrille(T, i, j)` qui renvoie `True` si la case de coordonnées (i, j) est bien dans la grille, et renvoie `False` sinon.

Question 4. Compléter la fonction `percolation(T)` qui prend en argument une grille `T` et qui remplit de fluide celle-ci en appliquant l'algorithme précédent. Votre fonction modifiera `T` et ne renverra rien.

Question 5. Compléter la fonction `visualise(p, n)` permettant de visualiser une grille avant et après remplissage **sur une même figure**. On utilisera pour cela la fonction `plt.subplot`. Par exemple, l'instruction `plt.subplot(3,1,2)` partage la figure en 3×1 emplacements de graphes (3 lignes et 1 colonne) et sélectionne le deuxième emplacement (au milieu) pour les instructions graphiques suivantes.

Question 6. Faire l'expérience avec quelques valeurs de p pour une grille de taille raisonnable : commencer avec $n = 10$ pour vérifier visuellement que votre algorithme est correct, puis augmenter la taille de la grille, par exemple avec $n = 500$.

On rappelle que la percolation est dite réussie lorsqu'à la fin du processus, au moins une des cases de la dernière ligne est remplie du fluide.

Question 7. Compléter la fonction `testePercolation(p, n)` qui crée une grille, effectue la percolation et renvoie `True` si la percolation est réussie, c'est-à-dire lorsque le bas de la grille est atteint par le fluide, et `False` sinon.

4 Détermination du seuil critique

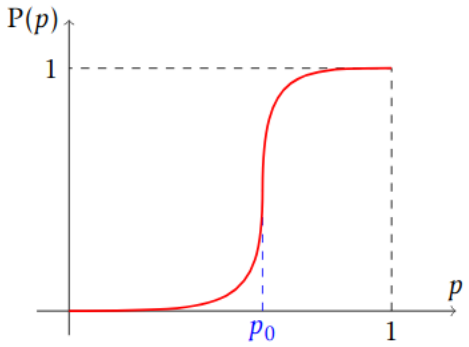
Question 8. Tester votre fonction `testePercolation` pour $n = 50$ et différentes valeurs de p. Observer qu'il semble exister un seuil p_0 en dessous duquel la percolation échoue presque à chaque fois, et au dessus duquel celle-ci réussit presque à chaque fois.

On souhaite à présent déterminer expérimentalement une valeur approchée de p_0. On suppose dans toute la suite que $n = 50$, et on note $P(p)$ la probabilité pour que le fluide traverse la grille.

Question 9. Proposer une démarche expérimentale simple pour estimer cette quantité en fonction de p, et rédiger la fonction `proba(p)` correspondante.

Question 10. Tracer la courbe de la fonction P à l'aide de `plt.plot`. On fera varier p entre 0 et 1 avec un pas de 0.01, et en faisant 50 essais pour chaque calcul de probabilité. Déterminer ensuite graphiquement une valeur approchée de p_0.

Vous devez retrouver l'allure théorique du graphe de la fonction P, représentée ci-dessous.



On suppose pour les questions suivantes que les valeurs de $P(p)$ et p utilisées pour tracer la courbe précédente ont été stockées dans deux listes (ou tableaux) de même longueur nommées `Lprob` et `Lp`.

Question 11. Compléter la fonction `seuilCritique(Lprob, Lp)` qui détermine en utilisant une recherche dichotomique un encadrement $[p_{\min}, p_{\max}]$ du seuil critique p_0 avec la plus grande précision possible.

La courbe théorique précédente est caractéristique d'une sigmoïde, que l'on retrouve dans la modélisation de nombreux systèmes biologiques. L'expression d'une telle fonction est de la forme :

$$f(x) = \frac{a}{1 + \mathrm{e}^{-k(x-x_0)}} + c$$

où a, k, x_0 et c sont des constantes. Plus précisément, x_0 représente l'abscisse du point d'inflexion de la courbe (où celle-ci change de concavité), correspondant exactement à la valeur du seuil critique recherché.

La fonction `curve_fit` du module `scipy.optimize` permet d'ajuster (en utilisant une méthode de moindres carrés non linéaire) les paramètres d'une courbe afin qu'elle passe au plus proche d'un certain nombre de points. Une adaptation de la documentation de cette fonction est fournie ci-dessous.

```
popt, pcov = scipy.optimize.curve_fit(f, xdata, ydata)
```

Paramètres

— `f` : callable
La fonction modèle, `f(x, ...)`. Elle doit prendre la variable indépendante comme premier argument et chaque paramètre à ajuster comme argument suivant.

— `xdata` : séquence de longueur `M`
La liste des valeurs de la variable indépendante correspondant aux différentes mesures.

— `ydata` : séquence de longueur `M`
Les mesures, typiquement `f(xdata, ...)`.

Résultat

— `popt` : tableau
Valeurs optimales des paramètres telles que la somme des carrés des écarts `f(xdata, *popt) - ydata` soit minimale.

— `pcov` : tableau à deux dimensions
Une estimation de la covariance de `popt`. Les termes diagonaux donnent la variance de l'estimateur du paramètre correspondant.
Pour estimer l'écart-type de l'erreur sur les paramètres, on peut utiliser `perr = np.sqrt(np.diag(pcov))`.

Question 12. Déterminer à l'aide de `curve_fit` les paramètres a, k, x_0 et c de la formule précédente pour qu'ils correspondent au mieux aux valeurs expérimentales obtenues pour $P(p)$. Commenter la valeur de x_0 obtenue.

Question 13. Tracer enfin la courbe de la sigmoïde associée à ces paramètres sur le même graphique que celui obtenu à la question 10.