

TP 09

Tableau, complexité et tracé de fonctions

Sources : exercice 1 : A. Troesch, J.-P. Becirspahic

Proposition de corrigé

Activité 1 : Modélisation de la percolation

Q 1 : Définir une fonction Python, `creationgrille(p, n)` à deux paramètres : un nombre réel p (qu'on supposera dans l'intervalle $[0, 1[$ et un entier naturel n , qui renvoie un tableau (n, n) dans lequel chaque case sera ouverte avec la probabilité p et fermée sinon.

```
def creation_grille(p, n):
    grille = np.zeros((n, n))
    for i in range(n):
        for j in range(n):
            if rand() < p:
                grille[i][j] = 1.
    return grille
```

Q 2 : Écrire une fonction `afficher_grille(grille, nom_de_fichier)` qui prend en argument une variable grille qui correspond à une grille de percolation générée précédemment et ne renvoyant rien mais enregistrant dans `nom_de_fichier` le graphe obtenu. On pourra exporter une grille de 10×10 cases avec l'échelle suggérée précédemment, l'enregistrer sous le nom "tp09_q02_vos_noms.png" et l'envoyer à votre professeur.

```
echelle = ListedColormap(['black', 'aqua', 'white'])
def afficher_grille(grille, nom_de_fichier):
    plt.matshow(grille, cmap=echelle)
    plt.colorbar()
    plt.savefig(nom_de_fichier)
    return None
```

Q 3 : Écrire une fonction `percolation(grille)` qui prend en argument une grille et qui remplit de fluide celle-ci, en appliquant l'algorithme exposé 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 :
 - (a) extraire de cette liste les coordonnées d'une case quelconque;
 - (b) ajouter à la liste les coordonnées des cases voisines qui sont encore vides, et les remplir de liquide.

L'algorithme se termine quand la liste est vide.

```
def percolation(grille):
    n, p = grille.shape
    lst = []
    for j in range(p):
        if grille[0][j] == 1.: # les cases vides de la première ligne
```

```

        grille[0][j] = .5 # sont remplies et ajoutées à lst
        lst.append((0, j))
while len(lst) > 0:
    (i, j) = lst.pop() # une case est extraite de lst
    if i > 0 and grille[i-1][j] == 1.: # si le voisin haut est vide, il est ✓
        rempli
        grille[i-1][j] = .5
        lst.append((i-1, j))
    if i < n-1 and grille[i+1][j] == 1.: # si le voisin bas est vide, il est ✓
        rempli
        grille[i+1][j] = .5
        lst.append((i+1, j))
    if j > 0 and grille[i][j-1] == 1.: # si le voisin gauche est vide, il est ✓
        rempli
        grille[i][j-1] = .5
        lst.append((i, j-1))
    if j < p - 1 and grille[i][j+1] == 1.: # si le voisin droit est vide, il est ✓
        rempli
        grille[i][j+1] = .5
        lst.append((i, j+1))

```

Q 4 : Rédiger un script vous permettant de visualiser une grille avant et après remplissage, et 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 $n = 64$). On pourra exporter et l'enregistrer sous le nom "tp09_q04_vos_noms.png" et l'envoyer à votre professeur.

```

grille = creation_grille(.61, 64)
grille_percolee = grille.copy()
percolation(grille_percolee)
fig1 = plt.matshow(grille, cmap=echelle)
plt.axis('off')
fig2 = plt.matshow(grille_percolee, cmap=echelle)
plt.axis('off')
plt.savefig('tp09_Q04_durif.png')

```

Q 5 : Écrire une fonction `teste_percolation(p,n)` qui prend en argument un réel $p \in [0,1[$ et un entier $n \in \mathbb{N}^*$, crée une grille, effectue la percolation et retourne :

- **True** lorsque la percolation est réussie, c'est-à-dire lorsque le bas de la grille est atteint par le fluide;
- **False** dans le cas contraire.

```

def teste_percolation(p, n):
    grille = creation_grille(p, n)
    percolation(grille)
    for j in range(n):
        if grille[n-1][j] == .5:
            return(True)
    return(False)

```

Q 6 : Rédiger la fonction `proba(p,k,n)` qui prend en argument le nombre d'essai k , la variable p ainsi que le nombre de cases n sur la largeur de la grille et qui renvoie $P(p)$.

```

def proba(p, k=20, n=128):
    s = 0
    for i in range(k):
        if teste_percolation(p, n):
            s += 1
    return s/k

```

Q 7 : Ecrire une fonction `tracer_proba(n,nom_de_fichier)` qui prendre en argument une taille n ne renvoyant rien mais enregistrant dans `nom_de_fichier` le graphe obtenu. On pourra traiter le cas d'une grille de 128×128 cases et enregistrer la figure obtenue sous le nom "tp09_q07_vos_noms.png" et l'envoyer à votre professeur.

```
def tracer_proba(n,nom_de_fichier):  
    x=np.linspace(0,1,21)  
    y=[]  
    for p in x:  
        y.append(proba(p,20,n))  
    plt.clf()  
    plt.plot(x,y)  
    plt.savefig(nom_de_fichier)  
    return None
```