Lycée Louis-Le-Grand, Paris
MPSI 4 – Informatique pour tous
A. Troesch, J.-P. Becirspahic

2013/2014

# Corrigé du TP nº 9 bis

On débute par l'importation des fonctions nécessaires :

```
import matplotlib.pyplot as plt
import numpy as np
from numpy.random import rand
```

On crée une grille initiale fonction de la taille $n$ et de la probabilité $p$ à l'aide de la fonction :

```
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
```

Pour remplir par percolation une grille, on adopte la démarche suivante : on crée une liste contenant initialement les indices des cases ouvertes de la première ligne de la grille puis, tant que cette liste n'est pas vide, on effectue les opérations suivantes :

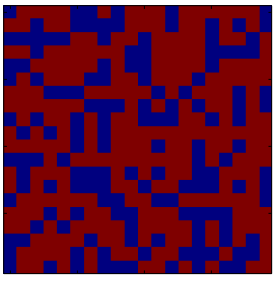
- on extrait de cette liste les indices d'une case que l'on remplit de liquide ;
- on ajoute à la liste les indices des cases voisines qui sont ouvertes.

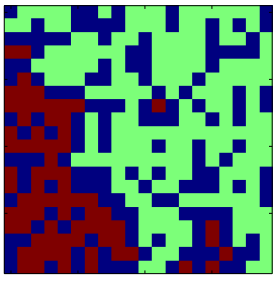
L'algorithme se termine quand la liste est vide.

```
def percolation(grille):
    n, _ = grille.shape
    lst = []
    for j in range(n):
        if grille[0][j] == 1.:
            lst.append((0, j))
    while len(lst) > 0:
        (i, j) = lst.pop()
        grille[i][j] = .5
        if i > 0 and grille[i-1][j] == 1.:
            lst.append((i-1, j))
        if i < n - 1 and grille[i+1][j] == 1.:
            lst.append((i+1, j))
        if j > 0 and grille[i][j-1] == 1.:
            lst.append((i, j-1))
        if j < n - 1 and grille[i][j+1] == 1.:
            lst.append((i, j+1))
```

Illustration avec un exemple :

```
grille = creation_grille(.6, 20)
grille_percolee = grille.copy()
percolation(grille_percolee)
plt.matshow(grille, fignum=1)
plt.matshow(grille_percolee, fignum=2)
plt.show()
```





On teste si une percolation traverse la grille à l'aide de la fonction suivante :

```
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)
```

Pour déterminer une valeur approchée de la probabilité de traverser la grille, on se contente d'effectuer $k$ essais (avec $k = 20$ par défaut, mais ceci peut être ajusté en fonction de la rapidité de la machine et du temps d'attente qu'on est disposé à perdre) avant de calculer la fréquence de la réussite de la percolation :

```
def proba(p, k=20, n=100):
    s = 0
    for i in range(k):
        if teste_percolation(p, n):
            s += 1
    return s/k
```

Pour déterminer le seuil critique $p_0$, on procède à une recherche dichotomique (assez grossière) en convenant que si $\mathrm{P}(p) < 0{,}4$ alors $p < p_0$ et si $\mathrm{P}(p) > 0{,}6$ alors $p > p_0$ :

```
def dicho(epsilon, e=20):
    x, y = 0., 1.
    while y - x > epsilon:
        p = (x + y) / 2
        prob = proba(p, k=e)
        if prob < 0.4:
            x = p
        elif prob > 0.6:
            y = p
    return (x + y) / 2
```

Pour une grille de taille infinie, le seuil critique vaut approximativement 0,593. On obtient des valeurs assez proches avec la fonction ci-dessus :

```
>>> dicho(1e-3)
0.59130859375
>>> dicho(1e-3)
0.58935546875
>>> dicho(1e-3)
0.59326171875
```

On calcule la densité de percolation d'une grille à l'aide de la fonction suivante :

```
def densite(grille):
    n, _ = grille.shape
    u, v = 0., 0.
    for i in range(n):
        for j in range(n):
            if grille[i][j] == .5:
                u += 1
            elif grille[i][j] == 1.:
                v += 1
    if u+v > 0:
        return u/(u+v)
    else:
        return 0
```

Cette fonction nous permet de calculer la densité moyenne en fonction de $p$ en la calculant sur un nombre $e$ (par défaut égal à 10) d'échantillons :

```
def d(p, e=10):
    s = 0.
    for i in range(e):
        grille = creation_grille(p, 100)
        percolation(grille)
        s += densite(grille)
    return s/e
```

Cette fonction permet de construire le graphe de densité suivant :

```
x = np.linspace(0, 1, num=100)
y = [d(p) for p in x]
plt.plot(x, y)
plt.show()
```