Corrigé

Percolation

Question 1. 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
```

Question 2. La démarche suggérée par l'énoncé correspond à un algorithme classique de la théorie des graphes. Si chaque case vide de la grille est un sommet du graphe et si les arêtes relient deux cases vides voisines, alors l'algorithme proposé correspond à un *parcours en profondeur* du graphe (cet algorithme est étudié en détail en option informatique de seconde année). Pour l'appliquer ici, on utilise la méthode `list.pop()` qui supprime et retourne le dernier élément d'une liste, ainsi que la méthode `list.append(x)` qui ajoute l'élément `x` en fin de liste.

```
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))
```

Le script qui suit trace dans deux figures distinctes : d'abord la grille avant percolation, puis la grille après percolation.

```
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.show()
```

Question 3. On teste si une percolation traverse la grille à l'aide de la fonction suivante :

Figure 1 – Une grille avant et après percolation pour $p = 0{,}61$.

```
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)
```

Remarque. On pourrait être plus efficace et ne pas poursuivre la percolation dès lors que le bord opposé est atteint, mais cela demanderait de réécrire la fonction percolation. C'est cependant ce que j'ai fait pour accélérer les calculs à la question suivante.

Seuil critique

Question 4.

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

b) 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
```

Propriétés macroscopiques

Question 5.

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

```
def densite(grille):
    n, p = grille.shape
    u, v = 0, 0
    for i in range(n):
        for j in range(p):
            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
```

b) Cette fonction nous permet de calculer la densité médiane 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, 128)
        percolation(grille)
        s += densite(grille)
    return s/e
```

Nous pouvons finalement visualiser le graphe des densités moyennes en fonction de la probabilité p :

```
x = np.linspace(0, 1, num=128)
y = [d(p) for p in x]
plt.plot(x, y)
plt.show()
```