TP07 Modèle d'avalanches

As usual, l'invitation GitHub pour votre dossier de travail vous a été envoyée par mail.

Partie I

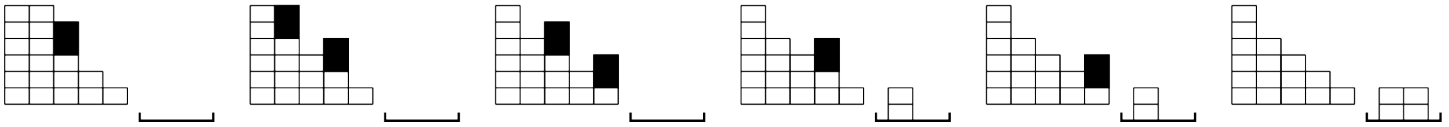
Introduction

Les avalanches générées dans une situation réelle présentent des tailles très variées. Certaines grosses avalanches entraînent vers le bas une grande quantité de matière tandis que d'autres, plus petites, ne font descendre que quelques granulés. Le modèle suivant dit « Modèle par automate cellulaire » (MAC) se propose d'éclaircir ces notions.

Les règles de fonctionnement de cet automate cellulaire unidimensionnel sont simples. On construit un empilement de carrés composé de colonnes juxtaposées et obéissant aux règles suivantes :

1. La différence de hauteurs entre deux colonnes adjacentes ne peut excéder deux unités. C'est l'image de l'angle de talus qui ne peut excéder une valeur limite sans que celui-ci ne s'écroule.
2. Lorsqu'une colonne se détruit parce qu'elle est trop élevée par rapport à ses voisines, elle entraîne deux unités dans sa chute. C'est l'image de l'effet domino ou effet d'entraînement dans une avalanche.

En partant d'un empilement quelconque, on laisse le système se relaxer selon les règles précédentes. On aboutit à un empilement stable tel que réprésenté sur la figure ci-dessous.



C'est la configuration initiale de notre édifice. On laisse ensuite tomber, un par un et au hasard, des petits carrés sur cet édifice. Chaque lâcher est éventuellement suivi d'un processus de relaxation obéissant aux règles précédentes. La surface du plan de base étant initialement limitée, on comptabilise le nombre de carrés qui sont éjectés de la plaque support après chaque lâcher d'un carré supplémentaire. On observe alors que se succèdent au rythme des lâchers successifs (c'est le temps de base) des « avalanches » de tailles variées. On remarque immédiatement qu'il existe un grand nombre de petites avalanches et que, en revanche, les avalanches de grande taille sont beaucoup plus rares.

Cette expérience, réalisée sur ordinateur dans le cas d'empilement importants permet d'obtenir la loi de distribution $D(S)$. On obtient le nombre D d'avalanches de taille S pour un grand nombre de lâchers successifs. Le but de ce TP est justement de tracer cette fonction de distribution.

Partie II

Spécifications

On va dans un premier temps se concentrer sur la partie difficile de l'algorithme, c'est-à-dire la gestion des avalanche après qu'il ait neigé un carré supplémentaire sur la montagne. Il peut donc se résumer de la façon suivante :

```
## Il neige un bloc sur la montagne
## Tant que la nouvelle montagne n'est pas stable
##     On fait une étape de relaxation
##     (déplacement des blocs instables d'une unité)
## On stocke le nombre total de blocs tombés après les avalanches pour étude
## ultérieure
```

On va représenter la montagne par une liste `montagne` contenant la hauteur de chacune des colonnes. Par exemple, la configuration initiale de l'exemple introductif serait `montagne = [6,6,5,2,1]`.

Il vous faut à présent écrire (et documenter !) les fonctions suivantes :

`il_neige(montagne,position)` : rajoute un bloc de neige à l'indice `position` de la liste `montagne`. Elle se contente de modifier `montagne` et ne renvoie rien (`None`).

Stop GitHub

Allez sur Github Desktop pour faire un commit. Choisissez vous-même (avec pertinence) le résumé. Pensez aussi à appuyer sur le bouton «Push origin» en haut à droite pour mettre à jour sur le web.

`est_instable(montagne)` : renvoie `True` s'il existe des colonnes instables (à droite ou à gauche) dans la liste `montagne` et `False` sinon. Une colonne est dite instable si sa taille excède strictement de deux unités celle d'une de ses voisines.

Stop GitHub

Allez sur Github Desktop pour faire un commit. Choisissez vous-même (avec pertinence) le résumé. Pensez aussi à appuyer sur le bouton «Push origin» en haut à droite pour mettre à jour sur le web.

`relaxation(montagne)` : Fait *une* étape de relaxation, c'est-à-dire qu'on déplace deux blocs de toutes les colonnes instables vers les colonnes voisines (elle modifie donc la liste `montagne`). La montagne est supposée être adossée à un mur à gauche, mais être au bord d'un abysse à droite. La fonction renvoie le nombre de blocs qui tomberait dans l'abysse au cours de l'expérience. Exemple : `relaxation([6,6,5,2,1])` modifie la liste en `[6,6,3,4,1]` et renvoie `0` (car aucun bloc ne tombe dans l'abysse). De même `relaxation([6,4,5,2,3])` modifie la liste en `[6,4,3,4,1]` et renvoie `2` (deux blocs sont tombés dans l'abîme).

Remarque : On va supposer qu'un léger vent souffle vers l'abîme, de sorte que si une colonne est à la fois instable à droite *et* à gauche, elle privilégie toujours le côté droit.

Stop GitHub

Allez sur Github Desktop pour faire un commit. Choisissez vous-même (avec pertinence) le résumé. Pensez aussi à appuyer sur le bouton «Push origin» en haut à droite pour mettre à jour sur le web.

`relaxation_totale(montagne)` : Prend une montagne instable en entrée et procède à toutes les relaxations nécessaires tant que la montagne n'est pas devenue stable. Elle renvoie le nombre total de blocs tombés lors du processus. Exemple : `relaxation_totale([6,6,5,2,1])` modifie la liste en `[6,4,3,2,1]` et renvoie `4` (cf exemple introductif)

Stop GitHub

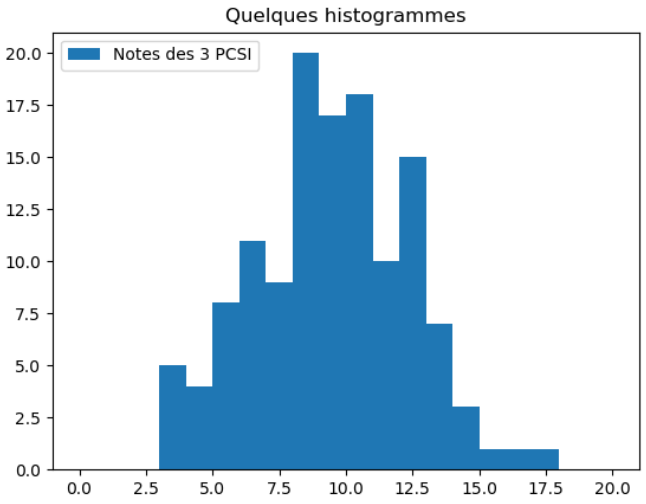
Allez sur Github Desktop pour faire un commit. Choisissez vous-même (avec pertinence) le résumé. Pensez aussi à appuyer sur le bouton «Push origin» en haut à droite pour mettre à jour sur le web.

Une fois ces fonctions écrites, il devrait être facile de les mettre ensemble pour répondre à la question posée, c'est-à-dire tracer $D(S)$ qui correspond au nombre D d'avalanches d'une taille S donnée en s'aidant de la fonction `plt.hist()` de matplotlib (cf.`help(plt.hist)` pour plus de détails). Exemple d'utilisation :

```
import matplotlib.pyplot as plt  # Pour les graphes
import random                    # Pour tirer des notes au hasard
# Gaussienne de moyenne 10 et écart-type 3 pour toutes les PCSI
notes = [random.gauss(10,3) for k in range(130)]
# Dessin de l'histogramme avec 20 bins
# (on suppose de ce fait que personne n'aura pile 20/20... [ou plus !])
plt.hist(notes,bins=20,range=(0,20),label="Notes des 3 PCSI")
plt.title('Quelques histogrammes') # Le titre
plt.legend(loc='upper left')       # Les légendes
plt.savefig('hist.png')            # Sauvegarde
plt.clf()                          # Nettoyage
```

Il s'agit en fait de faire une vraie simulation à partir de ce que l'on a concocté, à savoir :

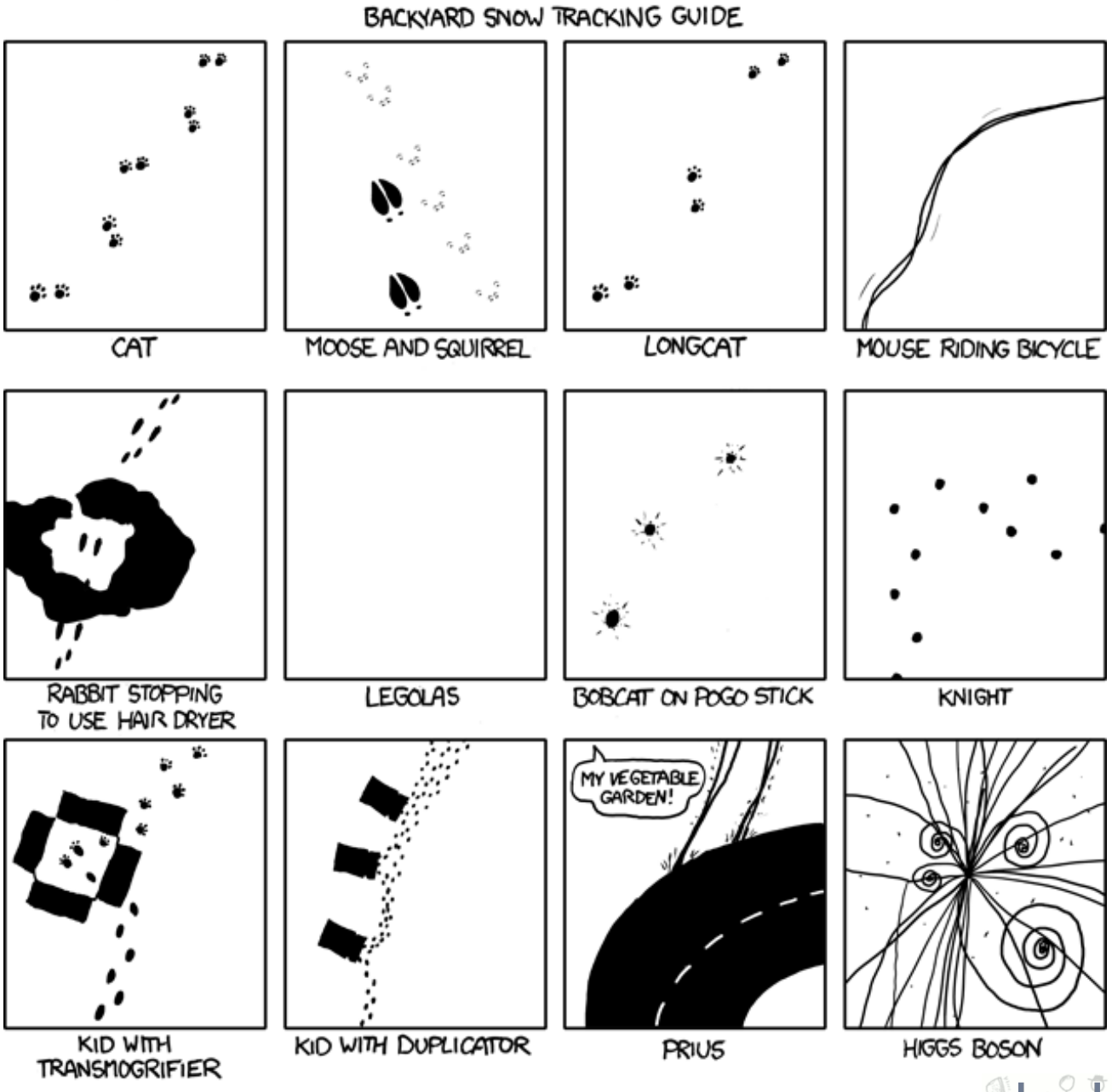
- Créer une montagne (disons de 50 colonne de large) avec des hauteurs aléatoires (utilisez `random.randrange`, voir `help(random.randrange)` après avoir importé le module `random`).
- Relaxer totalement une première fois, juste histoire de partir d'un état stable.



- Faire neiger (disons 100 000 fois) et après *chaque* ajout de neige faire une relaxation totale et noter dans une liste le nombre de blocq tombés par avalanche (vous voudrez peut-être ne rien noter si jamais il y a 0 blocs qui tombent car c'est un cas somme toute très courant et pas très intéressant)
- Faire un histogramme à partir de la liste des avalanches notées précédemment et le mettre en échelle logarithmique verticalement à l'aide de la commande
  `plt.hist(avalanches,bins=range(2,100,2),log=True)` où l'on prend en compte le fait que l'on ne veut pas les avalanches nulles (d'où un démarrage à 2), que chaque avalanche fait tomber un nombre pair de blocs (d'où le pas de 2) et que si on a 50 colonnes, au pire il y a 100 blocs qui tombent.
- Stocker le graphique résultant sous le nom `TP07_avalanche_VotreNom.png` (en remplaçant bien sur `VotreNom` par votre propre nom). Pensez à aussi mettre votre nom dans le titre du graphique.

Stop GitHub

Allez sur Github Desktop pour faire un commit. Choisissez vous-même (avec pertinence) le résumé. Pensez aussi à appuyer sur le bouton «Push origin» en haut à droite pour mettre à jour sur le web.



I suppose that's more accurately a hare dryer.