Sujet 0 Mines Ponts Corrigé – Durée 1h30

Q1

n est du type integer.
random() ∈ [0, 1[donc (h + 2)/2 * random() ∈ [0, 1 + h/2[.
Si *h* est pair, $1 \leq n \leq 1 + h/2$
Si *h* est impair, $1 \leq n \leq 1 + (h+1)/2$

Q2

```
def calcul_n(h):
    if h > 1:
        return int((h+2.0)/2 * random()) + 1
    else:
        return 0
```

Q3

```
def initialisation(P):
    return [0]*P
```

Q4

Je ne comprends pas pourquoi perdus est un argument… à moins que les grains perdus s'accumulent… je vais donc l'interpréter ainsi… interprétation confirmée à la lecture de la question 5 !

```
def actualise(piles, perdus):
    for i in range(len(piles)-1):
        n = calcul_n(piles[i]-piles[i+1])
        piles[i] -= n
        piles[i+1] += n
    n = calcul_n(piles[-1])
    piles[-1] -= n
    return piles,perdus+n
```

Q5

```
# Programme principal
P=int(input("Entrer le nombre de piles P : "))
piles = initialisation(P)
perdus = 0
nb_ex_actu = 0
while perdus < 1000:
    if nb_ex_actu == 10:
        piles[0] += 1
        nb_ex_actu = 0
    piles,perdus = actualise(piles,perdus)
    nb_ex_actu += 1
```

Q6

```
import pylab as pl
# trace un diagramme en barre, le for i in range(2) permet de répéter deux fois chaque valeur
X = [0]+[k for k in range(1,P) for i in range(2)]+[P]
Y = [piles[k] for k in range(P) for i in range(2)]
pl.plot(X,Y)
```

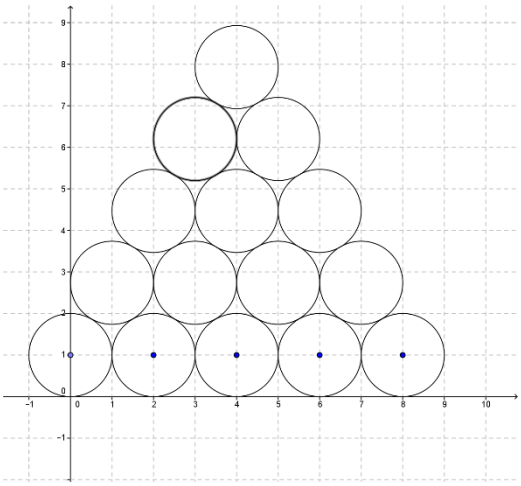
Q7

SELECT Densite, R, Kn, Gamma, Kt, Mu FROM granular_base WHERE Mat = ‘verre’ AND Note > 3 AND Votant >= 3

Q8

Programmation orientée objet (POO), hors programme !

On définit une nouvelle classe nommée grain. Les objets du type grain ont trois attributs pos, vit et force, qui représentent respectivement la position, la vitesse intermédiaire et la force d’interaction. La création d’une instance de cette classe demande deux paramètres x et y. L’attribut pos est alors initialisé à (x, y), les attributs vit et force sont initialisés à (0,0).



Q9

Les différents grains sont identifiés uniquement par leurs positions (X, Y).

Au niveau 0 (pour $i = 0$), X prend les valeurs 0, 2, 4, 6, 8 et $Y = 1$ pour les différents grains.

Au niveau 1 ($i = 1$), X prend les valeurs 1, 3, 5, 7 et $Y = 1 + \sqrt{3}$.

Au niveau 2 ($i = 2$), X prend les valeurs 2, 4, 6 et $Y = 1 + 2\sqrt{3}$.

Au niveau 3 ($i = 3$), X prend les valeurs 3, 5 et $Y = 1 + 3\sqrt{3}$.

Au niveau 4 ($i = 4$), X prend la valeur 4 et $Y = 1 + 4\sqrt{3}$.

Q10

Les grains sont empilés sans aucune interpénétration si on utilise des valeurs exactes. Numériquement, ce n’est pas possible puisque le programme va remplacer les différents multiples de $\sqrt{3}$ par des valeurs approchées.

Q11

```
l = sqrt((Xj-Xi)**2+(Yj-Yi)**2)      # Calcul de la distance CiCj
c = (Xj-Xi)/l                       # cosinus alpha
s = (Yj-Yi)/l                       # sinus alpha
```

Q12

La force d’interaction exercée sur le grain i par le grain j a pour composantes :

$F_x = \cos\alpha \times F_{jin} - \cos(\pi/2 - \alpha) \times F_{jit} = \cos\alpha \times F_{jin} - \sin\alpha \times F_{jit}$

$F_y = \sin\alpha \times F_{jin} + \sin(\pi/2 - \alpha) \times F_{jit} = \sin\alpha \times F_{jin} + \cos\alpha \times F_{jit}$

```
def somme_int():
    for i in range(len(tas)):
        Fix = Fiy = 0
        Xi,Yi = tas[i].pos
        for j in range(len(tas)):
            if j != i:
                Fjin,Fjit=calcul_Fnt(i,j)
                Xj,Yj = tas[j].pos
                l = sqrt((Xj-Xi)**2+(Yj-Yi)**2)
                c = (Xj-Xi)/l
                s = (Yj-Yi)/l
                Fix += c*Fjin-s*Fjit
```

```
            Fiy += s*Fjin+c*Fjit
        tas[i].force = (Fix,Fiy)
```

Q13

$v_{k+1/2} = v_{k-1/2} + a_k \times \Delta t$

$x_{k+1} = x_k + v_{k+1/2} \times \Delta t$

Q14

On s'appuie sur la boucle de l'algorithme DEM présentée en page 3, les équations (1), la fonction somme_int() et les formules de la question 13 qui s'appliquent en x et en y.

On calcule la vitesse v_k en fonction de la vitesse précédente v_{k-1}.

Le premier intervalle de calcul de la vitesse est d'amplitude dt/2… ce qui oblige à traiter le premier pas à part… on pouvait aussi l'inclure dans la boucle principale en divisant vxi, vyi par 2 si k = 0.

```
# on considère connue M la masse des grains, Tstop le temps de simulation, inc le nombre d'incrémentations,
g la constante gravitationnelle et l'objet tas
dt = Tstop/inc
# premier pas traité séparément car la vitesse est à calculer sur un temps dt/2
somme_int()                          # calcul des forces d'interaction
for i in range(len(tas)):
   axi,ayi = tas[i].force            # récupère les forces d'interaction
   axi/=M                            # calcul de l'accélération
   ayi=ayi/M-g
   vxi,vyi = axi*dt/2,ayi*dt/2       # calcul de la vitesse
   tas[i].vit = vxi,vyi
   Xi,Yi = tas[i].pos
   tas[i].pos = Xi+vxi*dt,Yi+vyi*dt  # calcul de la nouvelle position
# boucle principale
for k in range(inc):
   somme_int()                       # calcul des forces d'interaction
   for i in range(len(tas)):         # pour chaque grain i
      axi,ayi = tas[i].force         # récupère les forces d'interaction
      axi /= M                       # calcul de l'accélération
      ayi = ayi/M - g
      vxi,vyi = tas[i].vit
      vxi,vyi = vxi+axi*dt,vyi+ayi*dt   # calcul de la nouvelle vitesse
      tas[i].vit = vxi,vyi
      Xi,Yi = tas[i].pos
      tas[i].pos = Xi+vxi*dt,Yi+vyi*dt  # calcul de la nouvelle position
```

Remarque : je n'ai pas collé complètement au schéma, puisque j'ai fait une boucle sur *inc* au lieu d'utiliser la variable *t*, et je n'ai pas séparé la sauvegarde de l'état du reste, mais mon algorithme est équivalent au schéma.

Q15

Si le pas est trop grand, les résultats ne seront pas précis. Si le pas est trop petit, l'algorithme sera trop lent.

Q16

L'accélération est une fonction de la vitesse… qui est elle-même une fonction de l'accélération ! De plus, l'accélération est calculée aux instants $k\Delta t$ et on ne dispose de la vitesse qu'aux instants $(k + ½)\Delta t$.

On peut prendre l'expression de la vitesse dont on dispose dans tas[i].vit, qui va être très proche de la vitesse à l'instant $(k + ½)\Delta t$ si Δt est petit, pour calculer l'accélération.

Q17

La partie la plus pénalisante est la fonction somme_int().
On l'exécute *inc* fois et elle contient deux boucles imbriquées sur la longueur du tas.
On évalue sa complexité en fonction du nombre d'appels de la fonction calcul_Fnt().
Un tas à n niveaux contient $\frac{n(n+1)}{2}$ grains, donc calcul_Fnt() est appelée $\frac{n(n+1)}{2}\left(\frac{n(n+1)}{2}-1\right)$ fois, soit une complexité en $O(n^4)$.

Q18

On effectue le calcul des forces d'interaction uniquement sur les grains voisins, qui sont au maximum 6. On obtient alors une complexité majorée par $6 \times \frac{n(n+1)}{2}$ soit $O(n^2)$.
Pour cela, il serait plus simple de repérer les grains avec deux paramètres (leur niveau et leur position dans le niveau) plutôt qu'uniquement par leur rang *i* dans le tas. On remplacerait alors la boucle sur i in range(len(tas)) par deux boucles imbriquées comme dans le code donné question 8.

Q19

On peut représenter les grains par des disques noirs sur fond blanc par exemple. On connait leurs centres et leur rayon. On rajoute deux dernières ligne dans la boucle **for** i **in** range(len(tas)): on trace en blanc le grain de coordonnées (Xi,Yi) et en noir le grain de coordonnées Xi+vxi*dt,Yi+vyi*dt.
Tous les grains vont être ainsi déplacés au fur et à mesure que leurs positions évoluent.