TP7 – MODELISATION NUMERIQUE DES MILIEUX GRANULAIRES

1. PRESENTATION - BDD

L'étude des écoulements granulaires est un domaine scientifique en pleine expansion. Assez solides pour soutenir le poids d'un immeuble, ils peuvent couler comme de l'eau dans un sablier ou être transportés par le vent pour sculpter les dunes et les déserts.

On appelle milieu granulaire une collection de particules solides macroscopiques, typiquer à 100 m. Les particules plus fines sont appelées poudres (entre 1 m et 100 m) ou colloïdes (

Pour étudier ces milieux, on néglige en général les interactions de van der Waals, les effets thermique. Une des principales motivations de l'étude des milieux granulaires est leur secteurs industriels. On estime en effet que plus de 50% des produits vendus dans le monde mettent en jeu des matériaux granulaires, soit dans leur élaboration, soit dans leur forme finale. Les milieux granulaires sont omniprésents dans l'activité minière (extraction des minerais, transport, …), le bâtiment et le génie civil (béton, ballast de train, …), l'industrie chimique et pharmaceutique, etc. Dans tous ces secteurs se posent des problèmes de stockage, de transport, d'écoulement, de mélange et de transformation, qu'on étudie à l'aide de simulations numériques.

Tous ces secteurs d'activités sont répertoriés dans une base de données appelée : **base_industries.s3db**. Une première table INDUSTRIE contient les différents attributs qui correspondent aux différents secteurs d'activités et chacun possède des types de matériaux (granulaires ou non) et des attributs de stockage, transport, écoulement, mélange …

activites	Type	stock	Trans	Ecoul	melange
Extraction charbon	granulaire	tas	camions	laminaire	…
Extraction pétrole	liquide	réservoir	bateaux	visqueux	…
béton	granulaire	sac	camions	fluide	…
Pharmacie (paracétamol)	granulaire	sachets	trains	fluide	…
Pharmacie (alcool)	liquide	flacons	camions	laminaire	…
…					

Q1. ***Définir*** *la requête SQL de type chaîne de caractère, notée* demande_liste_activites*, permettant d'obtenir les noms des activités qui concernent les matériaux granulaires uniquement.*

Définir également *la requête SQL de type chaîne de caractère, notée* demande_parametres*, qui à partir du nom d'une activité choisie, stockée dans la variable* nom_activite *de type chaîne de caractère, renvoie le stockage (nommée* ***stock*** *dans la base de données), le transport (nommée* ***Trans*** *dans la base de données) et l'écoulement (nommé* ***Ecoul*** *dans la base de données).*

Pour utiliser votre requête nous allons connecter la base de données à Python. Il faut donc suivre les étapes suivantes :

```
import os # Import des commandes permettant de gérer les répertoires de travail
os.chdir('C:/ ??? ') # on se place dans le répertoire ou se trouve la base de  donnée

import sqlite3 # Import des commandes permettant de manipuler la base de données
basesql = u"base_industries.s3db" # Base de données initiale
cnx = sqlite3.connect(basesql )
curseur = cnx.cursor ()
requete = "SELECT * FROM industrie"
curseur .execute(requete)
```

Le curseur est un objet contenant le résultat de la requête. Pour visualiser la première entité de la requête, la syntaxe est la suivante :

```
data = curseur.fetchone()
print (data)
```

Attention, à chaque utilisation de fetchone(), l'entité est supprimée du curseur. Pour parcourir chacune des entités, on peut utiliser la syntaxe suivante :

```
Tab= curseur.fetchall()
#ou
Tab=[]
for cur in curseur :
    Tab.append(cur)
```

Q2. *La vraie base contient-elle un élément indispensable en plus du tableau écrit dans ce sujet ?*

Q3. *Créer la fonction* interroge_bdd(requete) *qui permet de se connecter à la base, de passer les requêtes et de renvoyer la réponse sous la forme d'un* ***tableau resultat tel que*** resultat= interroge_bdd(requete)

Observer que la première requête donne donc le tableau suivant :

Extraction charbon	granulaire	tas	camions	laminaire
béton	granulaire	sac	camions	fluide
Pharmacie (paracétamol)	granulaire	sachets	trains	fluide

Si votre fonction ne marche pas passer à la suite avec

```
# import numpy as np
# resultat1=np.array([['charbon','granulaire','tas','camions','laminaire'],
# ['beton','granulaire','sac','camions','fluide'],
# ['paracétamol','granulaire','sachets','trains','fluide']])
```

Q4. Ecrire ensuite une fonction nommée ***choix*** *Python qui permet à l'utilisateur de choisir l'activité (en demandant à l'utilisateur un numéro de matériau : charbon, béton ou paracétamol) et de renvoyer dans une liste le stockage, le transport et l'écoulement*

- *Entrée : tableau (identique à resultat1)*
- *Sortie : liste contenant 3 caractères (le stockage, le transport et l'écoulement)*

TESTER *en affectant choix(resultat1) à une variable resultat2.*

Une deuxième table est celle du prix associé aux compagnies d'extraction ou de stockage. Ce prix est lié aux cours de la bourse et fluctue en fonction de la date à laquelle on consulte la base.

Q5. ***Définir*** *la requête SQL de type chaîne de caractère, notée* demande_prix, *permettant d'obtenir le prix maximum des activités sélectionnées avec la première requête et* ***classer*** *dans l'ordre croissant des prix pour chacune des activités.*

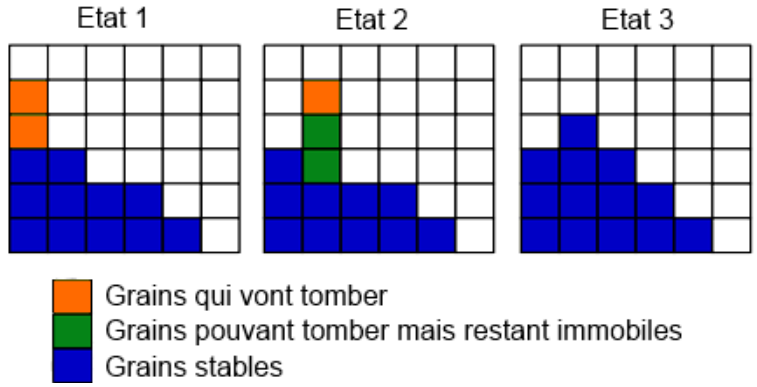
Vous afficherez le résultat avec Python.

compagnies	Prix_tonne	Id_activite	date
Vinci	10€	3	01/03/2016
Lafarge	9€	3	01/03/2016
TOTAL	0.1	2	01/03/2016
Mines d'Anzin	0.02	1	01/03/2016
Malet	0.019	1	01/03/2016
Malet	9.5	3	01/03/2016
Gabriel Larderet & Fils	0.021	1	01/03/2016
Sanofi-Aventis	150	4	01/03/2016
Rhodia	250	4	01/03/2014
Novacap	50	4	01/03/2016

2. FORME D'UN TAS PAR AUTOMATES CELLULAIRES

L'objet de cette partie est la simulation numérique par un modèle de type automate cellulaire. Un automate cellulaire se présente généralement sous la forme d'un quadrillage dont chaque case peut être occupée ou vide. Un grain y est symbolisé par une case occupée. La configuration des cases, qu'on appelle état de l'automate, évolue au cours du temps selon certaines règles très simples permettant de reproduire des comportements extrêmement complexes. La physique n'intervient pas directement mais les règles d'évolution sont choisies de façon à reproduire au mieux les lois naturelles.

Dans ce qui suit, nous allons simuler la formation d'un demi-tas de sable situé à droite de l'axe de symétrie vertical en appliquant les règles énoncées ci-après.



Une tour de hauteur h est une pile de cases pleines consécutives dont h voisines de droite sont vides. Si h > 1, on détermine arbitrairement le nombre n de grains du sommet de la tour qui vont tomber avec l'instruction python :

```
n = int((h+2.0)/2 * random()) + 1
```

Dans l'exemple figure 3, lors du passage de l'état 1 à l'état 2, le hasard introduit par la fonction random() (qui renvoie de manière aléatoire un flottant dans l'intervalle semi-ouvert [0.0; 1.0[ ) fait que toute la tour se décale apparemment vers la droite.

Ensuite, pour le passage à l'état 3 seul un grain sur les trois possibles tombe. L'état 3 est stable, plus aucun grain ne tombe.

*Q6. Ecrire une fonction nommée **calcul_n** qui prend la hauteur h comme argument et qui renvoie le nombre n de grains qui vont tomber sur la pile suivante. (penser à importer la bibliothèque adéquate !)*

***TESTER** cette fonction sur h=10*

La représentation graphique est une image en deux dimensions mais l'automate est à une dimension car on considère le tas comme un ensemble de piles dont la hauteur future dépend uniquement des piles adjacentes. Le tas est complètement défini avec la variable piles qui permet de stocker la hauteur des différentes piles.

Au départ le support est vide et on vient déposer périodiquement un grain sur la première pile (à gauche) qui correspondra au sommet du demi-tas. Le support peut recevoir P piles et à son extrémité droite il n'y a rien, les grains tombent et sont perdus. La pile P+1 est donc toujours vide.

*Q7. Définir la fonction **initialisation(P)** renvoyant une variable piles de type liste contenant P piles de hauteur 0.*

***TESTER** avec une variable P=20 en affichant le résultat.*

*Q8. Définir une fonction **actualise(piles,perdus)** qui va parcourir les piles de gauche à droite et les faire évoluer en utilisant les règles, fonctions et variables définies précédemment.*

Cette fonction doit renvoyer la variable piles actualisée ainsi que le nombre de grains perdus (en prenant en compte ceux qui seront tombés de la dernière pile P).

***TESTER** avec une variable piles=[50,0,0,0,0] en affichant le résultat de actualise(piles,0)*

*Q9. Écrire une fonction principale **calcul_piles(P, seuil_perte,nb_actu)** qui permet d'ajouter 1 grain à la première pile après chaque dizaine d'exécutions de la fonction actualise() et qui s'arrêtera lorsqu'au moins 1000 grains seront sortis du support.*

***TESTER** la fonction calcul_piles(15, seuil_perte, nb_actu) avec seuil_perte et nb_actu bien choisis et en affichant le résultat.*

Représentation graphique : Utiliser matplolib pour tracer les tas .. pour 1 calcul donné

Q10. construire un vecteur X constitué de [0,1,...,P-1] avec P=15

construire un vecteur (tableau) Y constitué de [piles[0],[piles[1],...,piles[P-1]]
mettre des titres aux axes et au graphique.
***MONTRER** la figure au professeur avec P=15 ; seuil_perte=1000 ; nb_actu=10*

3. METHODE DES ELEMENTS DISTINCTS (DEM)

Pour remédier à certaines limitations des modèles par automates cellulaires, il est possible de modéliser les milieux granulaires par la méthode des éléments distincts DEM issue des modèles moléculaires et présentée par Cundall en 1971.

Un des principaux inconvénients de la méthode est la détermination des paramètres physiques qui permettront une bonne corrélation entre la simulation et l'expérimentation. Différents laboratoires de recherche ont créé une base de données commune qui leur permet de partager des jeux de paramètres et de les noter de 1 à 5 selon les résultats obtenus lors des simulations. Une fonction d'agrégation permet de calculer la note moyenne. Il est donc possible d'effectuer des requêtes de recherche dans la relation « granular_base » définie par la table ci-après dont les lignes sont données en exemple :

Labo	Geom	Mat	Densite (g/cm3)	R (mm)	Kn (N/m)	Gamma ($N.m^{-1}s$)	Kt (N/m)	Mu	Note	Votant	...
MSC	poly	verre	2.1	-	3.10^7	20	2.10^6	0.5	2	2	...
LPMDI	poly	acier	6	-	5.10^6	1	4.10^5	0.7	3	6	...
LPGP	sphere	metal	4.7	4	10^6	10	3.10^4	0.2	3	3	...
LMGC	sphere	verre	1.56	1	2.10^7	5	10^7	0.3	4	5	...
PMMH	quartz	verre	1.46	2	7.10^8	0.5	2.10^8	0.4	2	1	...
...	...	...	...	...	...	...	...	...	...	...	...

Après une phase d'initialisation qui permet d'affecter les différentes variables caractéristiques avec les valeurs issues de la base de données, on exécute le code python suivant :

```
class grain : # Objet grain i
   def __init__ (self ,x,y):
      self .pos = (x,y) # Position (Xi ,Yi)
      self .vit = (0 ,0) # Vitesse intermediaire (Vix ,Viy)
      self . force =(0 ,0) # Force d'interaction (Fix , Fiy)
      #...
```

*Q11. Analyser et expliquer **en une phrase** le rôle de cette partie de code*

Puis on exécute :

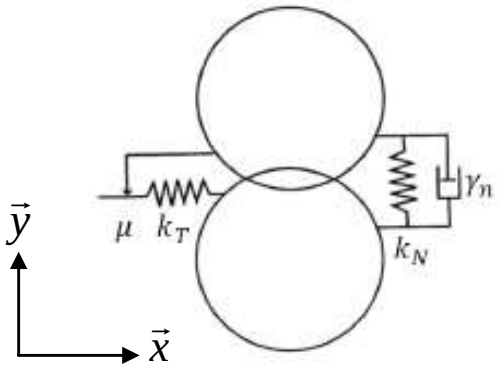
```
R=1 # Rayon des grains
n=5 # Nombre de niveaux
tas =[] # Initialisation de la liste
for i in range (n) :
   for j in range (n-i):
      tas += [ grain (0+i*R+j*2*R,R+i* sqrt (3)* R)]
```

Q12. Justifier et esquisser la forme du "tas" de grain obtenu.

Dans la suite du problème, nous nous intéressons à la résolution du Principe Fondamental de la Dynamique en résultante selon $\vec{x}$ et $\vec{y}$ pour chaque grain i :

- On utilise un modèle de ressort (raideur k_N) associé à une dissipation visqueuse (coefficient γ_n) permettant de reproduire une collision inélastique.

- Pour les forces tangentielles, un ressort (raideur k_T)) couplé à un patin (limite du glissement μ) permet de modéliser la force de friction



Les équations qui régissent le mouvement d'un grain i sont donc :

$$m_i \frac{d^2x_i(t)}{dt^2} = -K_T x_i(t) - \mu . m_i g \qquad et \qquad m_i \frac{d^2y_i(t)}{dt^2} = -K_N y_i(t) - \gamma_n \frac{dy_i(t)}{dt} - m_i g$$

#intiailisation des variables
Kt,Kn=1e-6,1e-6
m=10e-6
g=9.81
nu=0.5
gamma=5e-5

Q13. ***Ecrire*** *ces équations en utilisant la forme matricielle* $\dot{X} = F(X)$ *et* $\dot{Y} = G(Y)$

$$avec\ X = \begin{pmatrix} x \\ \dot{x} \end{pmatrix} \rightarrow \frac{dX}{dt} = \dot{X} = \begin{pmatrix} \dot{x} \\ \ddot{x} \end{pmatrix}\ et\ Y = \begin{pmatrix} y \\ \dot{y} \end{pmatrix} \rightarrow \frac{dY}{dt} = \dot{Y} = \begin{pmatrix} \dot{y} \\ \ddot{y} \end{pmatrix}$$

Q14. Ecrire les fonctions F et G sur Python

TESTER *ces fonctions avec X1=np.array([0,0]) et Y1= np.array([1e-5,0]) et en affichant F(X1) et G(Y1)*

On suppose qu'à l'instant initial, $x(0) = x_0 = 0$; $y_0 = 0.01mm$ et $V(0) = v_0 = 0$ De manière générale, pour simplifier les notations, on désigne par x_i l'approximation de la valeur de $x(t_i)$. On se propose de résoudre cette équation en prenant un pas de temps $h = (a - b)/n$

On rappelle que la résolution d'Euler explicite pour la grandeur *x* vectorielle sur un intervalle *n*h* s'écrit :

$$\forall k \in [0, n-1]\ \ x_{k+1} = x_k + h * F(t_k, x_k)$$

Q15. Ecrire la fonction ***euler(F, G, a, b , n)*** *qui prend donc en argument F, G, a, b , n et qui calcul les matrices X et Y sur les temps allant de a à b avec un pas h.*

Elle renvoie le vecteur temps T, les matrices X et Y à 2 colonnes et n lignes.

TESTER *votre fonction en affectant euler(F,G, 0, 20, 2000) aux variables T, X et Y et en affichant séparément T, X et Y*

Q16. Tracer avec subplot, dans deux figures cote à cote, les déplacements x(t) et y(t) en fonction du temps.

AMELIORER LA PRCEISION DE VOTRE CALCUL …

Q17. Puis tracer dans une nouvelle figure y(t) en fonction de x(t) d'un coté et le portrait de phase de y de l'autre