Concours Blanc — Informatique — corrigé

Simulation de la cinétique d'un gaz parfait (centrale 2018)

1 Initialisation

1.1 Placement en dimension 1

Q 1. La ligne 11 permet de stocker dans la variable p un nombre aléatoire entre 0 et L.

Q 2. Le paramètre c représente la position du centre de la particule.

Q 3. La ligne 4 permet de rejeter les particules qui dépasseraient à droite ou à gauche de l'espace alloué.

Q 4. Les lignes 5 et 6 déterminent si la particule que l'on souhaite ajouter se superpose à l'une des autres particules.

Q 5. La fonction possible prend en paramètre la position d'une particule et renvoie `True` si on peut la rajouter à la liste des particules déjà ajoutées.

Q 6. On peut directement choisir un nombre aléatoire entre R et $L-R$ en faisant

```
p = R + (L-2*R)*random.random()
```

Q 7. Dans ces conditions il sera impossible de trouver une position possible pour la dernière particule à ajouter, la fonction `placement1D` sera dans une boucle infinie et ne terminera jamais.

Q 8. Si le nombre de particules à placer est faible, il est très peu probable que l'on choisisse une position impossible pour une nouvelle particule à ajouter donc la fonction `possible` renverra toujours `True`. La boucle `while` s'exécutera N fois et à chaque fois, la boucle `for` de la fonction `possible` s'exécutera 0, puis 1, puis 2, ..., puis $N-1$ fois. La complexité sera alors

$$C(N) = 1+2+3+\ldots+N-1 = \frac{N(N-1)}{2} = O(N^2)$$

Q 9. Pour remédier de manière simple (mais non optimale) à la situation de la question 7, on décide de recommencer à zéro le placement des particules dès qu'une particule est rejetée par la fonction `possible`. Réécrire les lignes 9 à 13 de la fonction `placement1D` pour mettre en œuvre cette décision.
Les lignes 9 à 13 deviennent :

```
res=[]
while len(res) < N:
  p = R + (L-2*R)*random.random()
  if possible(p): res.append(p)
  else: res=[]
return res
```

1.2 Optimisation du placement en dimension 1

Q 10. On écrit la fonction suivante

```
def placement1Drapide(N:int, R:float, L:float) -> [float]:
  res = []
  l = L-N*2*R
  for i in range(N):
    res.append(l*random.random())
  res = sorted(res) # On trie les particules pour rendre la suite plus efficace
  for i in range(N):
    p[i] += R   # Décale la particule courante de R
    for j in range(i, N):
      p[j] += 2*R  # Décale toutes les suivantes de 2*R
  return res
```

Q 11. La fonction `placement1Drapide` a une complexité quadratique $O(N^2)$ du fait de la double boucle. Ceci n'est pas mieux que la fonction précédente mais même lorsque le nombre de particules est grand, cette fonction s'exécute sans problème (contrairement à la fonction précédente)

1.3 Dimension quelconque

Q 12. On donne la fonction suivante :

```
def placement(D:int, N:int, R:float, L:float) -> [[float]]:
  def dist(c,p):  # Renvoie la distance entre les points p et c
    S=0
    for i in range(len(c)):
      S+= (c[i]-p[i])**2
    return math.sqrt(S)

  def possible(c):
    for p in res:
      if dist(c,p)<2*R: return False
    return True

  res=[]
  while len(res)<N:
    p=[]
    for i in range(D):
      p = p + [R+(L-2*R)*random.random()]
    if possible(p): res.append(p)
    else: res=[]
  return res
```

2 Mouvement des particules

2.1 Analyse physique

Q 13. Entre deux événements une particule a un mouvement rectiligne uniforme.

Q 14. Lorsque $m_1 = m_2$ ces formules deviennent $\vec{v}_1' = \vec{v}_2$ et $\vec{v}_2' = \vec{v}_1$. Les vitesses des particules sont échangées.

Q 15. Lorsque $m_1 \ll m_2$ on obtient $\vec{v}_1' = -\vec{v}_1 + 2\vec{v}_2$ et $\vec{v}_2' = \vec{v}_2$
Dans ce problème cela correspond au choc d'une particule avec une paroi, dans ce cas $v_2 = 0$

2.2 Evolution des particules

Q 16. Fonction `vol` :

```
def vol(p:[[float], [float]], t:float) -> None:
  for i in range(len(p[0])):    # pour chaque coordonnée d'espace
    p[0][i] += t*p[1][i]        # x(t+dt) = x(t) + vx*dt
```

Q 17. Fonction `rebond`

```
def rebond(p:[[float], [float]], d:int) -> None:
  p[1][d] *= -1    # Inverse la composante de la vitesse perpendiculaire à la paroi
```

Q 18. Fonction `choc`

```
def choc(p1:[[float],[float]], p2:[[float], [float]]) -> None:
  p1[1][0],p2[1][0] = p2[1][0],p1[1][0]   # On échange les vitesses des deux particules
```

3 Inventaire des événements

3.1 Prochains événements dans un espace à une dimension

Q 19. Fonction `tr` :

```
def tr(p:[float],R:float,L:float) -> None or (float,int):
  if p[1]>0 :   # La vitesse est positive, la particule rencontrera la paroi de droite
    return ((L-p[0]-R)/p[1], 0)
  elif p[1]<0 :   # La vitesse est négative, rebond sur la paroi de gauche
    return (-(p[0]-R)/p[1], 0)
  else:   # La vitesse est nulle, pas de rebond
    return None
```

Q 20. Fonction `tc`

```
def tc(p1:[float], p2:[float], R:float) -> None or float:
  if p2[0]>p1[0] :  # Si la particule p2 est à droite
    vr = p2[1] -p1[1]     # Vitesse relative des particules
    d = p2[0] -p1[0]-2*R  # Distance à parcourir
  else:             # p1 est à droite
    vr = p1[1] -p2[1]     # Vitesse relative
    d = p1[0] - p2[0]-2*R # distance
  if vr<=0:         # Vitesse relative nulle, pas de collision
    return None
  else :
    return d/vr     # temps de collision
```

3.2 Catalogue d'événements

Q 21. Fonction `ajoutEv`

```
def ajoutEv(catalogue:[[bool, float, int, int or None, int or None]],
            e:[bool, float, int, int or None, int or None]) -> None:
    i=0
    while i<len(catalogue) and catalogue[i][1]>e[1]
      i+=1
    catalogue.insert(e,i)
```

Q 22. Fonction `ajout1p`

```
def ajout1p(catalogue:[[bool, float, int, int or None, int or None]], i:int,
            R:float, L:float, particules:[[[float], [float]]]) -> None:
    r = tr(particules[i],R,L)
    if r != None:
      ajoutEv(catalogue, [True, r[0], i, None, r[1]])
    for j in range(len(particules)):
      if j != i:
        c = tc(particules[i], particules[j], R)
        if c != None:
          ajoutEv(catalogue, [True, c, i, j, None])
```

Q 23. Fonction `initCat`

```
def initCat(particules, R, L) :
  catalogue = []
  for i in range(len(particules)):
    ajout1p(catalogue, i, R, L, particules)
  return catalogue
```

Q 24. Toutes les collisions entre particules vont être comptées 2 fois avec cette méthode.

Q 25. On pourrait utiliser un algorithme de recherche par dichotomie pour détermier la position dans la liste où insérer l'élément.

4 Simulation

Q 26. S'il existe au moins une particule qui n'est pas à l'arrêt, cette particule entrera en collision avec une paroi dans le futur. Si toutes les particules sont à l'arrêt, il n'y aura plus de collisions entre les particules où avec les parois.

Q 27. Fonction `etape`

```
def etape(particules,e):
  # fait évoluer les particules jusqu'à l'événement
  for i in range(len(particules)):
    vol(particules[i], e[1])
  if e[4] != None: # C'est un rebond
    rebond(particules[e[2]], e[4])
  else:  # C'est un choc
```

```
    choc(particules[e[2]], particules[e[3]])
```

Q 28. fonction `majCat`

```
def majCat(catalogue, particules, e, R, L) :
  for i in range(len(catalogue)):
    catalogue[i][1] -= e[1]  # Décompte le temps écoulé
    # Marque non valides les événements où interviennent les particules de e
    if catalogue[i][2] == e[2] or catalogue[i][2]==e[3]:
      catalogue[i][0] = False;
    if catalogue[i][3] != None:
      if catalogue[i][3] == e[2] or catalogue[i][3]==e[3]:
        catalogue[i][0] = False;
  # détermine les prochains événements des particules de l'événement e
  ajout1p(catalogue, e[2], R, L, particules)
  if e[3] != None:
    ajout1p(catalogue, e[3], R, L, particules)
```

Q 29. Fonction `simulation`

```
def simulation(bdd, D, N, R, L, T) :
  particules = situationInitiale(D, N, R, L)
  catalogue = initCat(particules, R, L)
  t = 0   # Temps de la simulation
  n = 0   # Nombre d'événements
  while  t<T
    e = catalogue.pop()
    if e[0]:    # Vérifie si l'événement est valide
      t += e[1]
      n += 1
      etape(particules, e)
      majCat(catalogue, e)
      enregistrer(bdd, t, e, particules)
  return n
```

Q 30. Les doublons sont automatiquement marqués invalide lorsque le premier a été traité, car ils impliquent les mêmes particules.

5 Explotation des résultats

Q 31.

```
SELECT SI_NUM FROM SIMULATION WHERE SI_DUR>10;
```

Q 32.

```
SELECT SI_DIM, COUNT(*) FROM SIMULATION GROUP BY SI_DIM;
```

Q 33.

```
SELECT SI_NUM, COUNT(*), AVG(RE_VIT)
FROM REBOND GROUP BY SI_NUM;
```

Q 34.

```
SELECT RE_DIR SUM(2*PA_M*RE_VP)
FROM REBOND JOIN PARTICULE ON REBOND.PA_NUM = PARTICULE.PA_NUM
WHERE SI_NUM = n
GROUP BY RE_DIR;
```