

# MonteCarlo python program to study liquid/gas phase equilibrium

## Introducció

En aquest treball he buscat implementar una simplificació del codi que s'exposa en el capítol 8 del llibre Understanding Molecular Simulation, de Daan Frenkel i Berend Smith.

Per realitzar aquesta simplificació em basaré en les idees del "Gibbs Ensemble", que proposa una simulació en la que les condicions a les diverses fases que contingui seran iguals, mateixa pressió, temperatura, potencials químics... D'aquesta manera aquesta simplificació permetrà veure de forma ràpida i eficaç el comportament d'un equilibri líquid gas.

Però com serà el sistema que simularé? Aquest es basa en simular dos caixes separades, cada una contenint N o M partícules de gas o líquid, mitjançant MonteCarlo la meua simulació prendrà una partícula de una de les caixes, aleatòriament, i la canviarà a l'altra, en una posició aleatòria, després calcularà l'energia que tindria aquest sistema amb el canvi de partícula, i si s'està reduint la energia total del sistema acceptarà el canvi, de manera similar als canvis d'Spin que vam introduir a la simulació d'Ising.

Les simplificacions que es faran seran sobretot en la forma de calcular l'energia, aquí es on es poden ultimar els detalls de la simulació, fent servir mètodes més precisos que consumeixin més temps de simulació o mètodes més simples que permetin aproximacions prou vàlides, en el meu cas he treballat amb mètodes que ja hem vist en altres moments en aquest curs, bàsicament fent càlculs de potencial energètic segons Lennard-Jones, model que considera les interaccions tant atractives com repulsives entre parelles electròniques a partir del seu radi.

## Codi (metodologia)

El codi programat consta de 3 seccions clares, la primera d'elles, que veiem en la figura 1, es tracta de la definició de constants i paràmetres varis que utilitzarem en diferents punts del codi i la simulació:

```
# Constants Lennard-Jones
EPSILON = 1.0
SIGMA = 1.0 |

# Parametres simulacio MC
PASOS_MC = 1000 # Numero de passos de la simulacio
PARTICULES_GAS = 80 # Numero inicial de particules de gas
PARTICULES_LIQUID = 10 # Numero inicial de particules de líquid
```

Figura 1: Definició de constants i parametres

Com es veu a la figura 1 tenim dos parts diferents, primerament, la definició de les constants epsilon i sigma de la formula de Lennard-Jones (unitats reduïdes adimensionals, en altres simulacions i sistemes podríem considerar variar aquests valors). Per altra banda també definim diversos paràmetres del càlcul Montecarlo, com podrien ser el numero de passos

aleatoris que fem i la situació inicial de cada caixa, amb el numero de partícules que tenim en cada una.

La segona secció del codi (que veiem reduïda en la figura 2) és on preparem tots els passos a realitzar, com són els càlculs energètics o els loops Montecarlo.

```
def ini_liquid(particules):  
  
    for step2 in range(PARTICLES_LIQUID):  
        x=random.uniform(0, 10)  
        y=random.uniform(0, 10)  
        z=random.uniform(0, 10)  
        particules_liquid.append((x, y, z))  
  
def energia_liquid(particules):  
    energia_liquid = 0.0  
    for triplet in particules_liquid:  
        for triplet2 in particules_liquid:  
            if triplet != triplet2:  
                temp=[  
                    triplet2[0] - triplet[0],  
                    triplet2[1] - triplet[1],  
                    triplet2[2] - triplet[2]]  
                r_2 = temp[0]**2 + temp[1]**2 + temp[2]**2  
                r_6 = r_2**3  
                r_12 = r_6**2  
                energia_liquid += 4 * EPSILON * ((SIGMA**12 / r_12) - (SIGMA**6 / r_6))  
    return energia_liquid  
  
def montecarlo(particules_gas, particules_liquid, energia_total):  
    # Fer Monte Carlo  
    for pas in range(PASOS_MC):  
        # Selecciono aleatoriament si canvio un gas o un liquid de caps, si al generar numero aleatori (0 o 1) de  
        if random.random() < 0.5: # Gas a liquid  
            if len(particules_gas) > 0:  
                particules_gas.pop(random.randint(0, len(particules_gas) - 1))  
                particules_liquid.append((random.uniform(0, 10), random.uniform(0, 10), random.uniform(0, 10)))  
        else: # Liquid a gas  
            if len(particules_liquid) > 0:  
                particules_liquid.pop(random.randint(0, len(particules_liquid) - 1))  
                particules_gas.append((random.uniform(0, 10), random.uniform(0, 10), random.uniform(0, 10)))  
  
        energia_temp=energia_gas(particules_gas)+energia_liquid(particules_liquid)  
        if energia_temp < energia_total:  
            energia_total=energia_temp  
  
    # Imprimir el estado de equilibrio  
    print("Partícules de gas:", len(particules_gas))  
    print("Partícules de líquid:", len(particules_liquid))  
    return energia_total
```

Figura 2: Càlculs del codi

En aquesta figura podem veure dos seccions clares, el càlcul energètic de líquid (en les línies anteriors a la imatge trobem una secció paral·lela per el gas) i el loop Montecarlo.

En el càlcul del líquid estem definint dos seqüències diferents que utilitzarem en el futur, primerament ini\_liquid crearà una situació inicial de N (on N val PARTICLES\_LIQUID) partícules posicionades en posicions aleatòries de 0 a 10 en les tres dimensions, el comandament “append” ens afegeix una la partícula que estiguem creant a cada pas a una llista comuna que utilitzarem en “energia\_liquid”, la següent seqüència que definim. Aquesta conté dos “for”s, que ens permetran per cada partícula (triplet de coordenades) calcular la distancia que la separa amb cada una de totes les altres partícules que tinguem (guardo cada coordenada en el nom “temp” i a la línia r\_2 calculo i sumo la distancia en cada eix), per després elevar a 6 i 12 aquest radi, tenint així totes les incògnites que necessito per calcular l’energia per Lennard-Jones (en cada pas ho va sumant, arribant així a una energia total de la caixa liquida). Aquest pas és dels que més consumeixen en potencia de la simulació, i molts cops ha estat el factor

limitant de la simulació que estigués fent, ja que estem passant per aquests “for”s N-1 vegades per N per poder fer el càlcul sencer, per exemple si tenim 10 partícules per cada una farem 9 passos de càlcul, significat un total de 90 passos amb un numero de partícules bastant petit.

Finalment només quedarà programar el loop aleatori, en aquest, en cada pas que fem generarà un numero aleatori (binari, 0 o 1) i en cas de ser 0 traurà una partícula aleatòria de la caixa de gasos per afegir-la a una posició aleatòria de la de gasos, i si surt un 1 al revés. Després, però, caldrà decidir si el canvi es vàlid o no, i això ho farem definint la variable energia\_temp, l'energia del sistema en aquesta nova situació, aquesta serà comparada amb energia\_total (que veurem definida més endavant, però com diu el seu nom es la energia total del sistema vàlid que tinguem, l'inicial en el primer pas) i en cas de donar un valor d'energia més baix, aquesta energia\_temp substituirà a la energia\_total, passant a ser aquest el sistema vàlid que tinguem, el de menys energia.

Finalment, tenim la tercera part del codi representada a la figura 3:

```
if __name__ == '__main__':
    ini_gas(particules_gas)
    ini_liquid(particules_liquid)
    energia_gas_total = energia_gas(particules_gas)
    energia_liquid_total = energia_liquid(particules_liquid)
    energia_total = energia_gas_total+energia_liquid_total
    print("L'energia total inicial serà de : " + str(energia_total)) #str es per fer
    energia_montecarlo=montecarlo(particules_gas, particules_liquid, energia_total)
    print("L'energia de l'estat d'equilibri serà de:" +str(energia_montecarlo))
```

*Figura 3: Part final de crida del codi*

En aquesta part final el que farem es fer que la simulació es realitzi, ja que fins ara només estàvem definint el que faríem en cada moment, però fins que no cridem (call, llamada) els segments que hem definit no estarem fent la simulació realment.

D'aquesta forma doncs veiem com definim les situacions inicials, calculem la energia total que utilitzarem al cridar el loop Montecarlo com a la suma de l'energia total de gas i de liquid, demanem que s'imprimeixi l'energia inicial (important, cal fer que la variable energia\_total sigui una string igual que el text ja que sinó es crea un error al imprimir dos classes diferents) definim l'energia final després de passar Montecarlo i l'Imprimim.

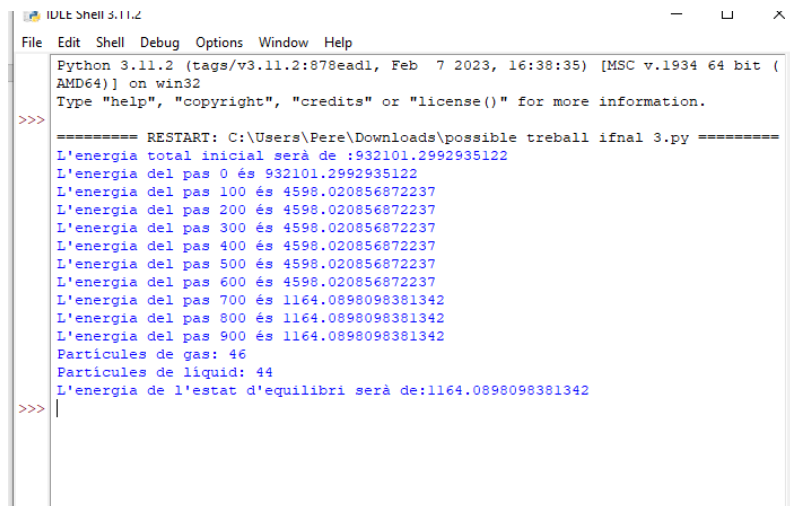
A més a més, mentre realitzava aquest informe, m'ha semblat que seria curiós i significatiu afegir alguna manera de poder veure com va evolucionant aquest sistema, així que al final del tot de la variable definida com a montecarlo he afegit ràpidament les següents línies (figura 4) que ens permetran veurà la evolució cada x passos (en el meu pas, cada 100 passos ja que he demanat que m'ho imprimís sempre que es trobes en un pas divisible entre 100).

```
if pas%100 == 0:
    print("L'energia del pas " + str(pas)+" és |"+str (energia_total))
```

*Figura 4: Línies afegides per visualitzar l'evolució dinàmicament*

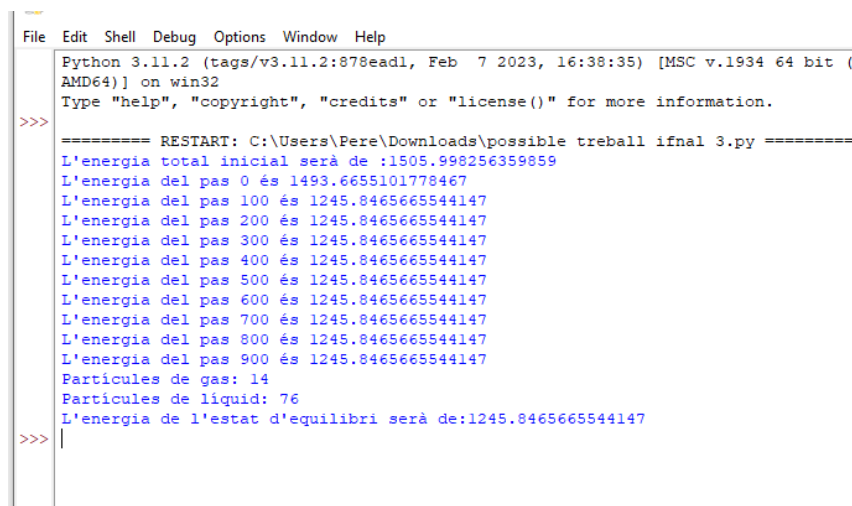
## Resultats

En aquest apartat afegiré les imatges de tres simulacions diferents, amb diferents situacions inicials per visualitzar si la simulació es realitza bé:



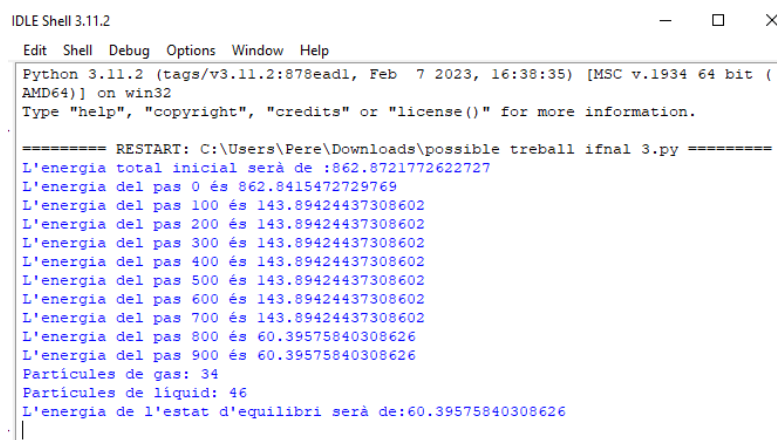
```
Python 3.11.2 (tags/v3.11.2:878ead1, Feb 7 2023, 16:38:35) [MSC v.1934 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:\Users\Pere\Downloads\possible treball ifnal 3.py =====
L'energia total inicial serà de :932101.2992935122
L'energia del pas 0 és 932101.2992935122
L'energia del pas 100 és 4598.020856872237
L'energia del pas 200 és 4598.020856872237
L'energia del pas 300 és 4598.020856872237
L'energia del pas 400 és 4598.020856872237
L'energia del pas 500 és 4598.020856872237
L'energia del pas 600 és 4598.020856872237
L'energia del pas 700 és 1164.0898098381342
L'energia del pas 800 és 1164.0898098381342
L'energia del pas 900 és 1164.0898098381342
Particules de gas: 46
Particules de líquid: 44
L'energia de l'estat d'equilibri serà de:1164.0898098381342
>>> |
```

Figura 5: Situació inicial 80 gas 10 líquid



```
Python 3.11.2 (tags/v3.11.2:878ead1, Feb 7 2023, 16:38:35) [MSC v.1934 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:\Users\Pere\Downloads\possible treball ifnal 3.py =====
L'energia total inicial serà de :1505.998256359859
L'energia del pas 0 és 1493.6655101778467
L'energia del pas 100 és 1245.8465665544147
L'energia del pas 200 és 1245.8465665544147
L'energia del pas 300 és 1245.8465665544147
L'energia del pas 400 és 1245.8465665544147
L'energia del pas 500 és 1245.8465665544147
L'energia del pas 600 és 1245.8465665544147
L'energia del pas 700 és 1245.8465665544147
L'energia del pas 800 és 1245.8465665544147
L'energia del pas 900 és 1245.8465665544147
Particules de gas: 14
Particules de líquid: 76
L'energia de l'estat d'equilibri serà de:1245.8465665544147
>>> |
```

Figura 6: Situació inicial 10 gas 80 líquid



```
Python 3.11.2 (tags/v3.11.2:878ead1, Feb 7 2023, 16:38:35) [MSC v.1934 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:\Users\Pere\Downloads\possible treball ifnal 3.py =====
L'energia total inicial serà de :862.8721772622727
L'energia del pas 0 és 862.8415472729769
L'energia del pas 100 és 143.89424437308602
L'energia del pas 200 és 143.89424437308602
L'energia del pas 300 és 143.89424437308602
L'energia del pas 400 és 143.89424437308602
L'energia del pas 500 és 143.89424437308602
L'energia del pas 600 és 143.89424437308602
L'energia del pas 700 és 143.89424437308602
L'energia del pas 800 és 60.39575840308626
L'energia del pas 900 és 60.39575840308626
Particules de gas: 34
Particules de líquid: 46
L'energia de l'estat d'equilibri serà de:60.39575840308626
>>> |
```

Figura 7: Situació inicial 40 gas 40 líquid

Aquí podem veure que el codi corre correctament, oferint els resultats de energia que l'hi demanem cada 100 passos i la situació final d'energia.

### Conclusions

Evidentment podria ser casualitat al ser un mètode aleatori, però com que ha passat varis cops al fer diverses simulacions crec que es important remarcar una curiositat dels resultats, i es que en casos amb més gas o situacions semblants s'arriba més o menys a un equilibri, amb aproximadament 50% de molècules de cada tipus, en canvi en els casos on hi ha més líquid inicialment la situació final manté una gran quantitat de líquid. Una altra vegada ressalto que podria ser casualitat, ja que no veig un motiu teòric pel que pugui passar tenint en compte que els càlculs realitats per una espècie i l'altre són paral·lels, però trobo important remarcar aquesta singularitat.

A part d'això com ja he dit, el programa funciona correctament, i ha sigut un programa complicat d'elaborar al combinar varis loops com seria en la part de [triplets] que ha sigut bastant difícil d'enllestir.

### Referencies

([*Computational Science (San Diego, Calif.)*] Daan Frenkel\_ Berend Smit - *Understanding Molecular Simulation \_from Algorithms to Applications (2002, Academic Press )* - Libgen.Lc, n.d.)