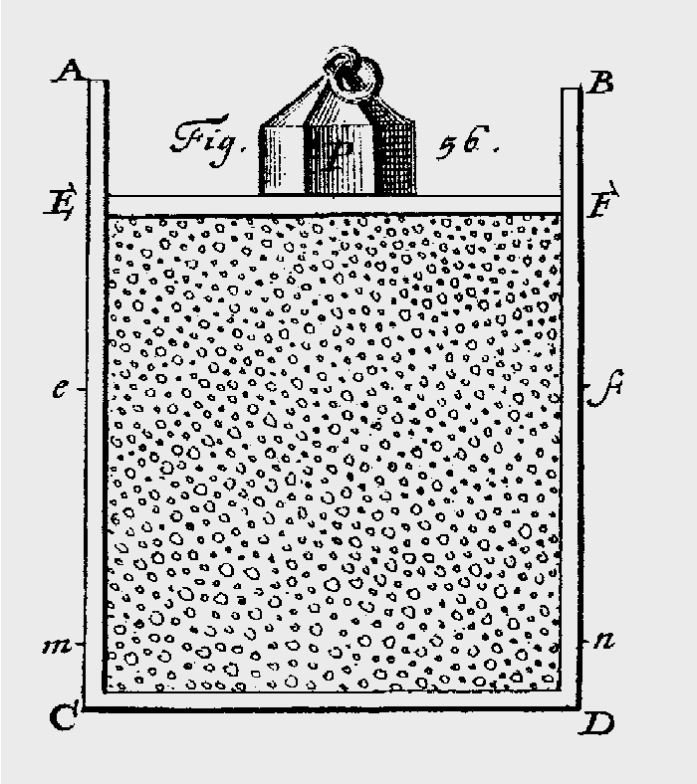
Symulacje komputerowe w fizyce



Ćwiczenia III - GAZ

Zadanie

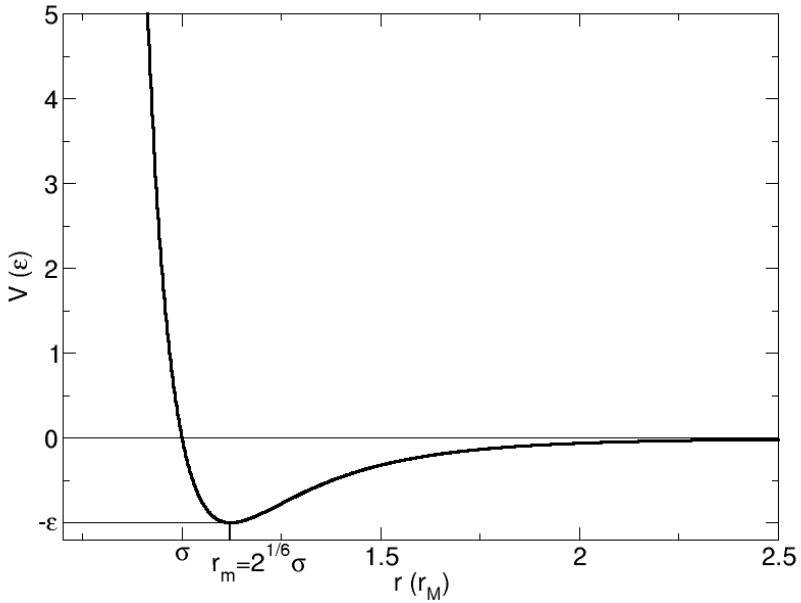
- napisz symulację dynamiki molekularnej dwuwymiarowego gazu szlachetnego (cząstek odziałujących siłami Lennarda-Jonesa) w periodycznych warunkach brzegowych

potencjał Lennarda-Jonesa:

$$V(r) = 4\varepsilon\left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right]$$

•prosty potencjał przybliżający oddziaływanie między obojętnymi elektrycznie atomami

•człon r^{-12} opisuje odpychanie chmur elektronowych na bliskich odległościach podczas gdy człon r^{-6} przyciąganie na dużych odległościach (siły van der Waalsa)



Szczegóły zadania

- weź N cząstek (na początek niech N=16), ustaw je na regularnej sieci
- nadaj im prędkości odpowiadające początkowej temperaturze T
- prowadź symulację przez pewien czas t (korzystając np. z algorytmu żabki), i – co jakiś czas - zapisuj konfiguracje
- zrób krótką animację
- (z gwiazdką) znajdź ewolucję temperatury i ciśnienia w Twoim układzie

Klasa „cząstka”

- Klasa to wygodna konstrukcja pozwalająca trzymać wszystkie atrybuty cząstki w jednym miejscu

```
class czastka:
    """ Klasa opisujaca pojedyncza czastke
gazu """
    def __init__(self,promien,pos,vel):
        self.promien = promien
        self.r=pos      # położenie
        self.v=vel      # prędkosc
```

pierwszy argumentem każdej metody (funkcji składowej w klasie) jest zawsze odniesieniem do przetwarzanego właśnie egzemplarza klasy i zwyczajowo jest nazywany *self*

automatycznie wywoływane przy tworzeniu nowego egzemplarza klasy

nową cząstkę można utworzyć za pomocą:

```
g=czastka(promien,polozenie,predkosc)
```

(uwaga – w argumenty nie włączamy *self*)

własności cząstki mogą byc uzyskane poprzez :

```
g.promien, g.r, g.v etc.
```

Ogólna struktura programu

inicjalizacja
t=0
while $t<t_{max:}$
    oblicz siły
    aktualizuj prędkości i położenia
    t=t+dt
    (opcjonalnie) obliczaj wielkości makroskopowe (temperatura, ciśnienie, etc)

Przykładowe wartości parametrów:

particleNumber=16
boxsize=8.0
eps=1.0
sigma=1.0
promien=0.5
deltat=0.0001
temp=2.5

Inicjalizacja

```
particles = []
for i in range(nx):
        for j in range(ny):
            polozenie = array([i*delta, j*delta])
            predkosc=array([(random.random()-1./2),(random.random() -1./2)])
            particles.append(czastka(promien,polozenie,predkosc) )
```

utwórz siatkę $n_x x n_y$ z cząstkami (żeby nie nachodziły na siebie)

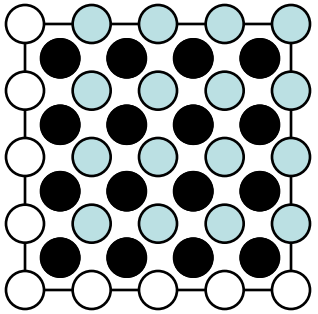
przypadkowe predkosci poczatkowe (numpy)

utwórz cząstki i dopisz je do listy cząstek

```
sumv=0.0
sumv2=0.0
for p in particles:
    sumv=sumv+p.v
sumv=sumv/particleNumber  # prędkość środka masy
for p in particles:
    p.v=(p.v-sumv) # teraz środek masy spoczywa
for p in particles:
    sumv2=sumv2+dot(p.v,p.v)/2.0
sumv2=sumv2/particleNumber  # średnia energia kinetyczna
fs=sqrt(temp/sumv2)        # czynnik skalujący,  temp - żądana temperatura
for p in particles:
    p.v=p.v*fs     # skalujemy
```

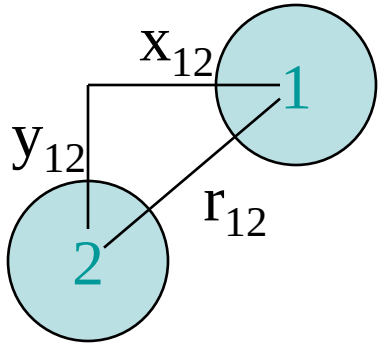
$$\mathcal{T} = \frac{2}{2Nk_B} K$$

dwa wymiary



Siły

$$\mathbf{F}_{2\to 1} = -\frac{f(r_{12})}{r_{12}}\left[x_{12}\mathbf{e}_x + y_{12}\mathbf{e}_y\right]$$



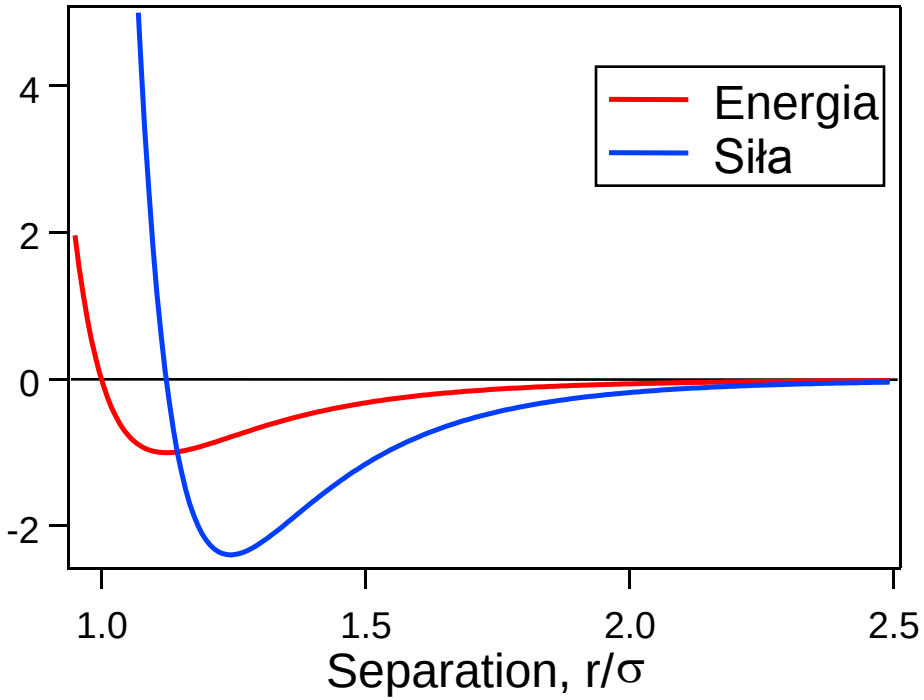
potencjał Lennarda-Jonesa

$$r_{12} = \left[(x_2 - x_1)^2 + (y_2 - y_1)^2\right]^{1/2}$$

$$u(r) = 4\varepsilon\left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right]$$

$$f(r) = -\frac{du}{dr} = \frac{48\varepsilon}{\sigma}\left[\left(\frac{\sigma}{r}\right)^{13} - \frac{1}{2}\left(\frac{\sigma}{r}\right)^{7}\right]$$

$$\mathbf{F}_{2\to 1} = -\frac{48\varepsilon}{\sigma^2}\left[\left(\frac{\sigma}{r_{12}}\right)^{14} - \frac{1}{2}\left(\frac{\sigma}{r_{12}}\right)^{8}\right]\left[x_{12}\mathbf{e}_x + y_{12}\mathbf{e}_y\right]$$

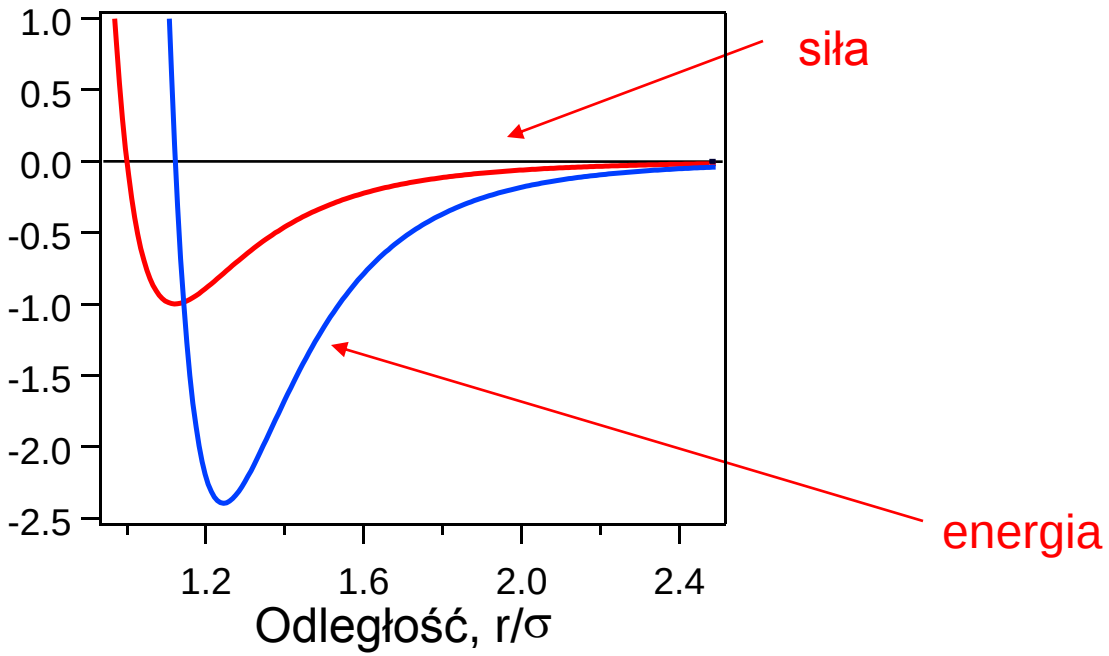


Obcięcie potencjału LJ

zwykle obcinany w $r_c = 2.5\sigma$

$$u_s(r) = \begin{cases} u(r) - u(r_c) & r \le r_c \\ 0 & r > r_c \end{cases}$$

obcinamy i przesuwamy
(aby potencjał był ciągły)



Jednostki

Wygodnie skalowac wszystko przez parametry modelu

- długość σ
- energię ε
- masę m

r* - bezwymiarowa odległość r/σ

T* - bezwymiarowa temperatura kT/ε

t* - bezwymiarowy czas $t/\left(\sigma\sqrt{m/\varepsilon}\right)$

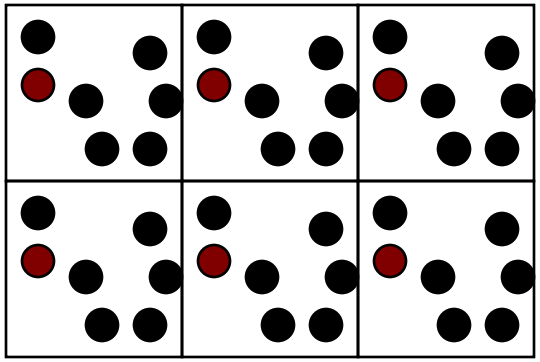
ρ* - bezwymiarowa gęstość $\rho\sigma^2$

potencjał Lennard-Jones po przeskalowaniu (bezwymiarowy)

$$u^*(r^*) = 4\left[\left(\frac{1}{r^*}\right)^{12} - \left(\frac{1}{r^*}\right)^{6}\right]$$

Periodyczne warunki brzegowe

- Modelowanie ścianek jest niepraktyczne
  - Silne artefakty związane ze skończonymi rozmiarami układu
  - Sztuczny wpływ ściany na własności układu
- Zamiast modelowac ścianki, lepiej jest otoczyć nasz układ jego kopiami
  - “Periodyczne warunki brzegowe” (PBC)



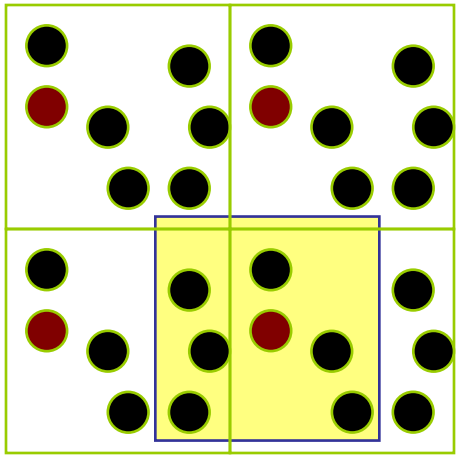
PBC - implementacja

```
if newX > b:
    newX -= b
if newX < 0:
    newX += b

if newY >b:
    newY -= b
if newY < 0:
    newY += b
```

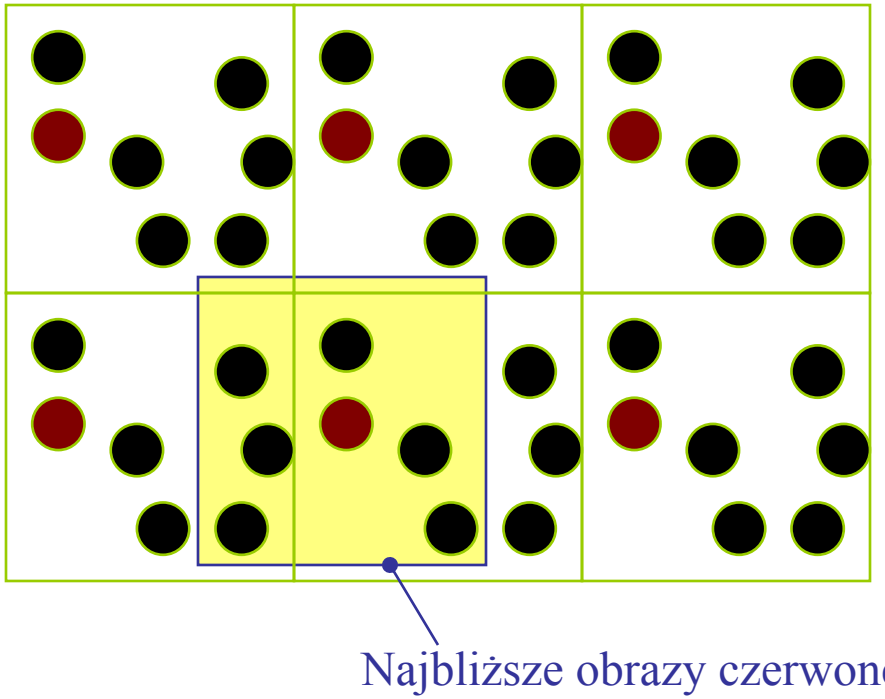
rozmiar pudełka

newX, newY – nowe (zaktualizowane) współrzędne x i y cząstki



Najbliższy obraz

Rozważaj tylko najbliższy obraz danej cząstki przy wyznaczaniu sił

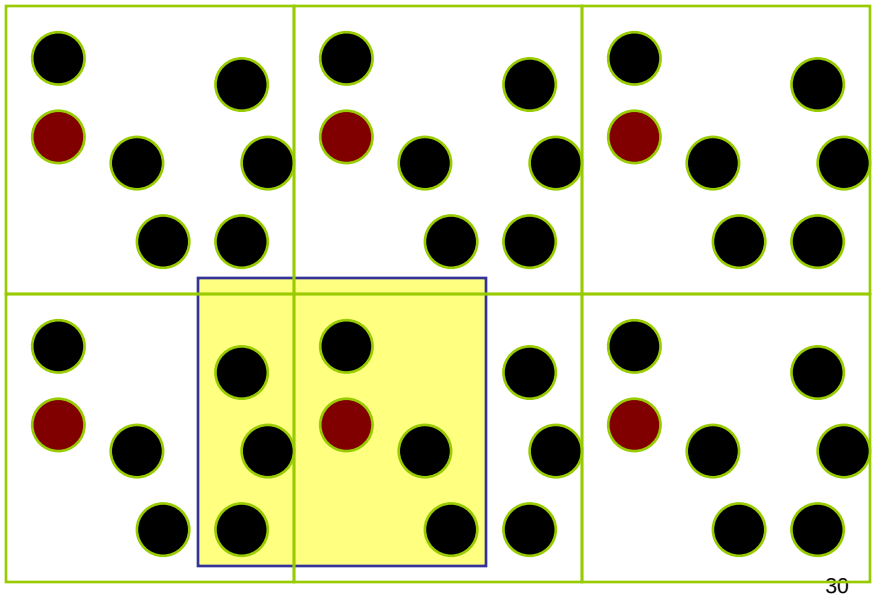


(można stosowac wtedy kiedy obcięcie sił jest mniejsze niż połowa bokiu pudełka, dla sił długozasięgowych jest trudniej!)

Najbliższy obraz - implementacja

```
r_vect=pj.r-pi.r     # wektor pomiędzy cząstkami i oraz j
      if r_vect[0] > b2:     # b2 – połowa pudełka b2=b/2
         r_vect[0] =r_vect[0] –b  # przesuwamy współrzędną x wektora r_vect
      elif r_vect[0] <-b2:
         r_vect[0] =r_vect[0]+b   # b – bok pudełka

      if r_vect[1] > b2:              # to samo dla y
         r_vect[1] =r_vect[1] -b
      elif r_vect[1] <-b2:
         r_vect[1] =r_vect[1] +b
```



Metoda całkowania

Żabka:

$$\mathbf{v}(t+\delta t/2) = \mathbf{v}(t-\delta t/2) + \left(\frac{\mathbf{F}(t)}{m}\right)\Delta t$$

$$\mathbf{r}(t+\delta t) = \mathbf{r}(t) + \mathbf{v}(t+\delta t/2)\Delta t$$

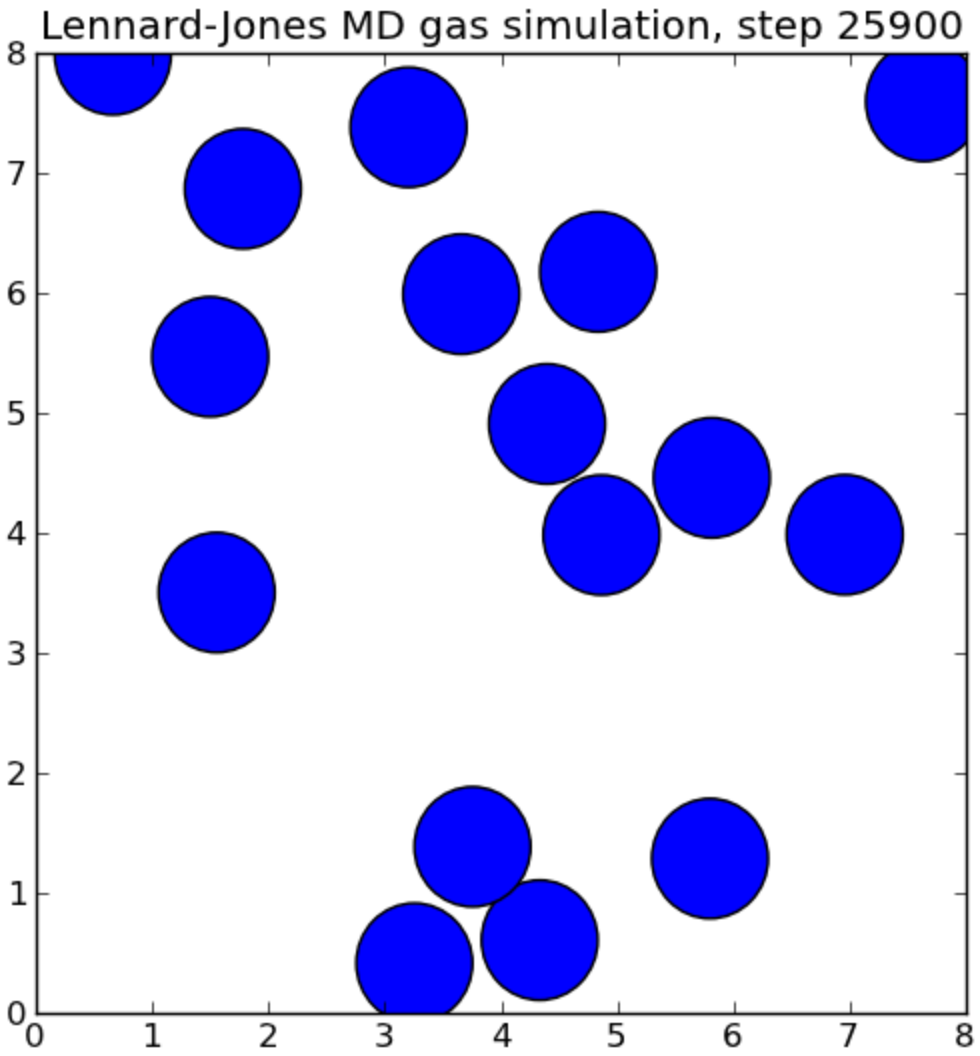
- załóż, że cząstki mają równe masy
- pracuj z wektorami (e.g. **r0**=array([2.,0.])
- iloczyn skalarny **a**.**b**=dot(**a**,**b**)
- zdefiniuj funkcje sila(**r**), potencjal (**r**) etc.

Rysowanie klatek

```
import matplotlib.pyplot as plt
from matplotlib.patches import Circle

if (en%100==0):     #  co 100-na klatka
    plt.clf()              #  wyczyść obrazek
    F = plt.gcf()       #   zdefiniuj nowy
    for i in range(particleNumber):    # pętla po cząstkach
        p = particles[i]
        a = plt.gca()                          # 'get current axes' (to add smth to them)
        cir = Circle((p.r[0],p.r[1]), radius=p.radius)  # kółko tam gdzie jest cząstka
        a.add_patch(cir)                # dodaj to kółko do rysunku
        plt.plot()                          # narysuj
    plt.xlim((0,boxsize))               # obszar do narysowania
    plt.ylim((0,boxsize))
    F.set_size_inches((6,6))        # rozmiar rysunku
    nStr=str(en)     #nagraj na dysk – numer pliku z 5 cyframi, na początku zera, np 00324.png
    nStr=nStr.rjust(5,'0')
    plt.title(„Symulacja gazu Lennarda-Jonesa, krok "+nStr)
    plt.savefig('img'+nStr+'.png')
```

Przykładowa klatka

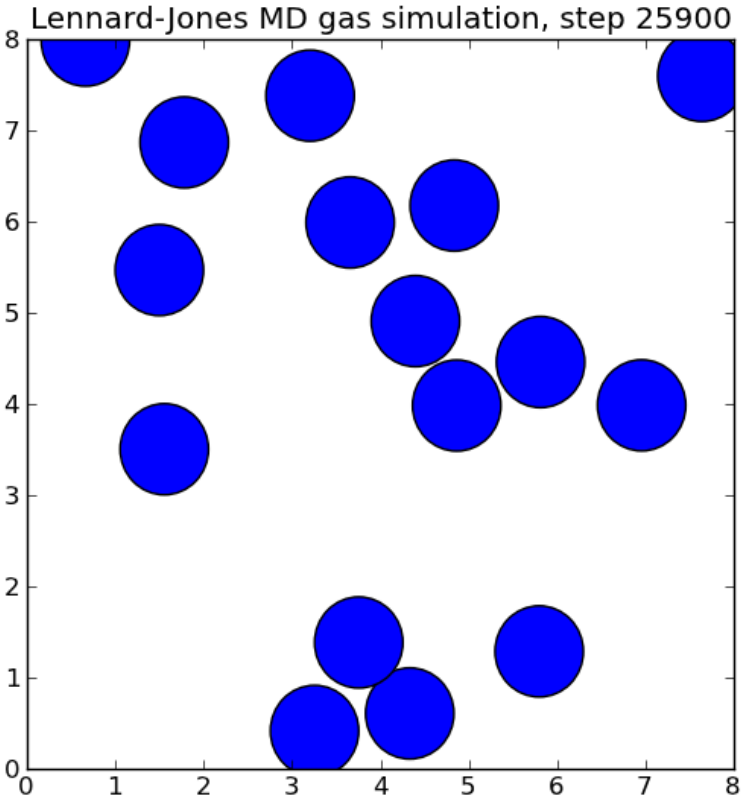


Robienie filmu

convert -delay 20 *.png anim.gif

opóźnienie między klatkami

nazwa animowanego gifu



Istotna uwaga:

- Operacja obliczania sił jest rzędu $O(N^2)$
- Bardzo kosztowne dla dużych układów
- Istnieją efektywne metody O(N) aby to przyspieszyć (lista sąsiadów, lista komórek)

Punktowanie zadania:

- Implementacja „gazu doskonałego” (periodyczne warunki brzegowe, brak sił, rysowanie poszczególnych klatek) – 50%
- Dodanie sił i numerycznego znajdowania trajektorii (żabka) – 50%
- (z gwiazdką) – wyznaczanie chwilowej temperatury i ciśnienia w układzie – extra 20%