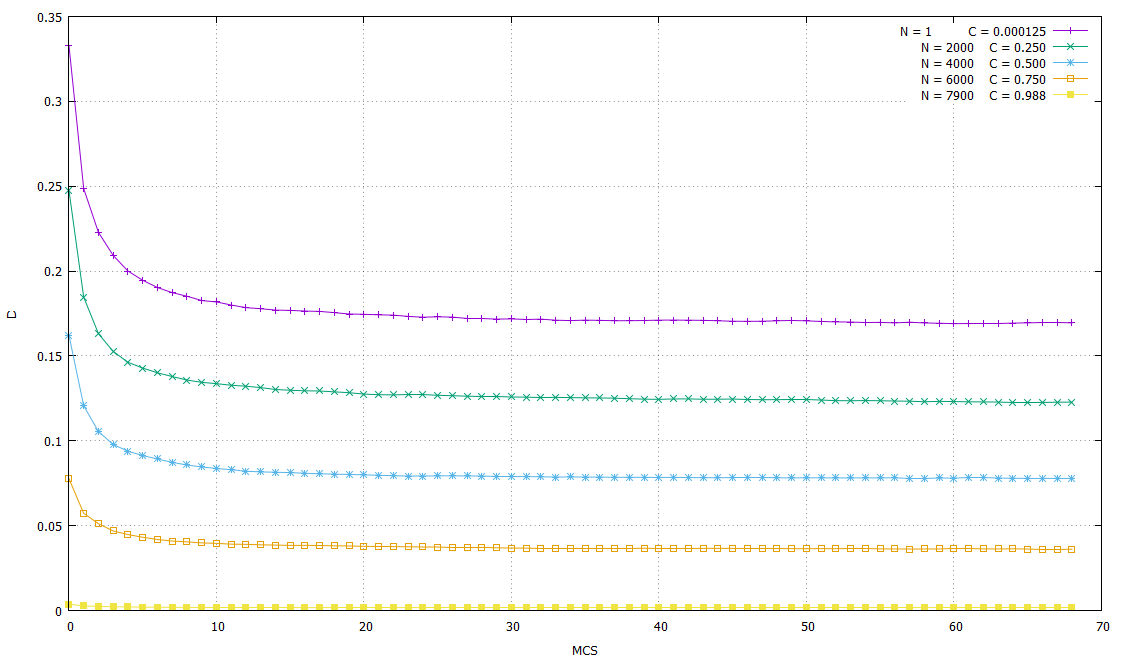
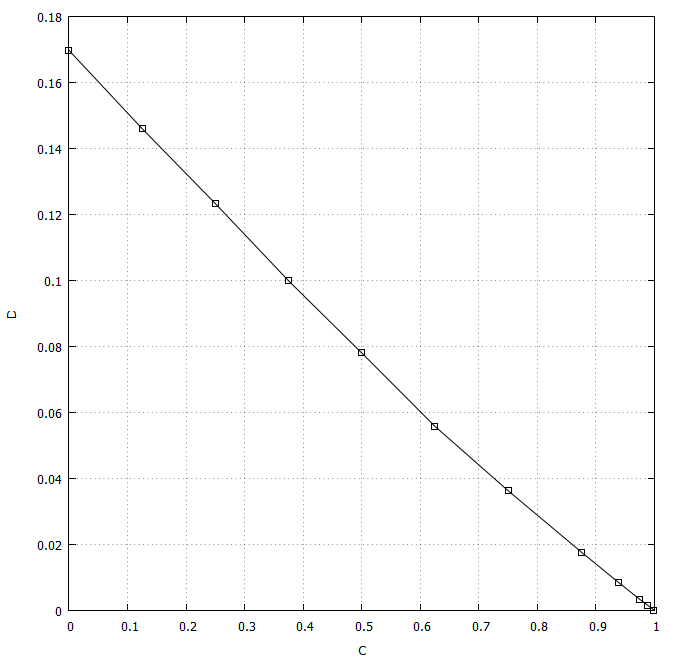
Symulacje Monte Carlo: Dyfuzja 3D



Parametry:

* rozmiar pudełka
* liczba kroków
* liczba niezależnych symulacji zależna od liczby atomów ()

Obliczone współczynniki dyfuzji:

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
| 1 | 20000 | 0.000125 | 0.169726 |
| 1000 | 20 | 0.125 | 0.146013 |
| 2000 | 10 | 0.250 | 0.123302 |
| 3000 | 6 | 0.375 | 0.099928 |
| 4000 | 5 | 0.500 | 0.078053 |
| 5000 | 4 | 0.625 | 0.055812 |
| 6000 | 3 | 0.750 | 0.036308 |
| 7000 | 2 | 0.875 | 0.017721 |
| 7500 | 2 | 0.938 | 0.008584 |
| 7800 | 2 | 0.975 | 0.003358 |
| 7900 | 2 | 0.988 | 0.001656 |
| 7990 | 2 | 0.999 | 0.000156 |

Kod programu

(Napisany w języku Python, z użyciem biblioteki NumPy)

import numpy as np

rnd = np.random

# configuration -----------------------------------------------------

box\_size = 20

n\_atoms = 1000

mc\_steps = 70

independent\_simulations = 20000//n\_atoms

assert n\_atoms < box\_size \*\* 3

# initialization -----------------------------------------------------

print("#    L = {}\n"

      "#    N = {}\n"

      "#    C = {}\n"

      "#    {} MCS, {} independent simulations"

      .format(box\_size, n\_atoms, n\_atoms\*box\_size\*\*(-3), mc\_steps, independent\_simulations))

# possible moves of an atom in a step

moves = [[1, 0, 0], [-1, 0, 0], [0, 1, 0],

         [0, -1, 0], [0, 0, 1], [0, 0, -1]]

results\_avg = np.zeros(mc\_steps)    # R2[MCS] averaged over atoms and ind. simul.

for sim\_no in range(independent\_simulations):

    results = np.zeros\_like(results\_avg)    # R2[MCS] averaged over atoms

    # occupied places

    occupied = np.zeros((box\_size, box\_size, box\_size), dtype=bool)

    # positions array

    atoms = np.zeros((n\_atoms, 3), dtype=int)

    # [x1,y1,z1]

    # [x2,x2,z2]

    #    ...

    # [xn,yn,zn]

    # inserting atoms in random positions

    for i in range(n\_atoms):

        r = rnd.randint(box\_size, size=3)  # r = [x,y,z] (random position)

        while occupied[r[0], r[1], r[2]]:

            r = rnd.randint(box\_size, size=3)

        atoms[i] = r

        occupied[r[0], r[1], r[2]] = True

    start\_pos = atoms.copy()  # save a copy of starting positions

# simulation ---------------------------------------------------------

    for i in range(mc\_steps):

        # Monte Carlo step -----------------------------------------------

        for r in atoms:

            dr = moves[rnd.choice(6)]   # random direction (vector from moves[])

            nb = (r + dr) % box\_size    # coordinates of the neighbor (in PBC)

            rb = r % box\_size           # coordinates of the atom (in PBC)

            if not occupied[nb[0], nb[1], nb[2]]:

                occupied[nb[0], nb[1], nb[2]] = True

                occupied[rb[0], rb[1], rb[2]] = False

                r += dr

        # Calculating distance travelled

        DeltaR2 = np.linalg.norm(atoms - start\_pos, axis=1) \*\* 2

        results[i] = np.average(DeltaR2)    # average over atoms

    results\_avg += results/independent\_simulations  # average over independent simulations

# Calculating and printing the diffusion coefficient ---------------------

D = [results\_avg[t] / (6\*t) for t in range(1,mc\_steps)]

for item in D:

    print(item)