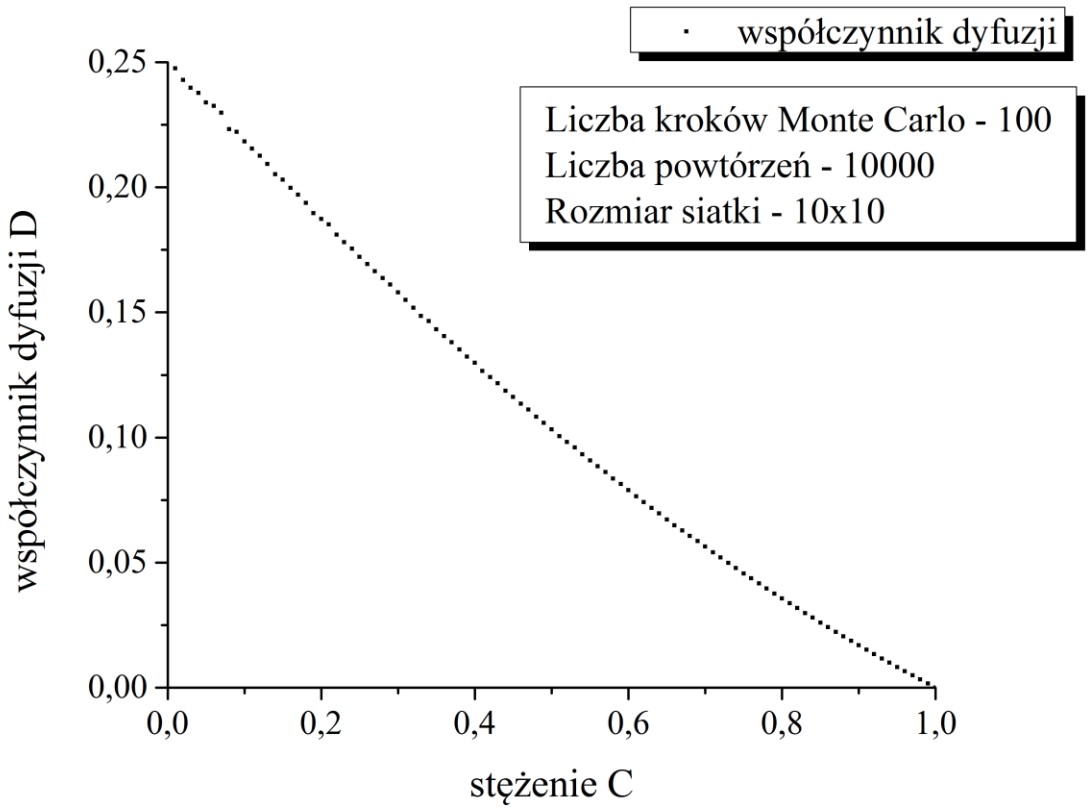
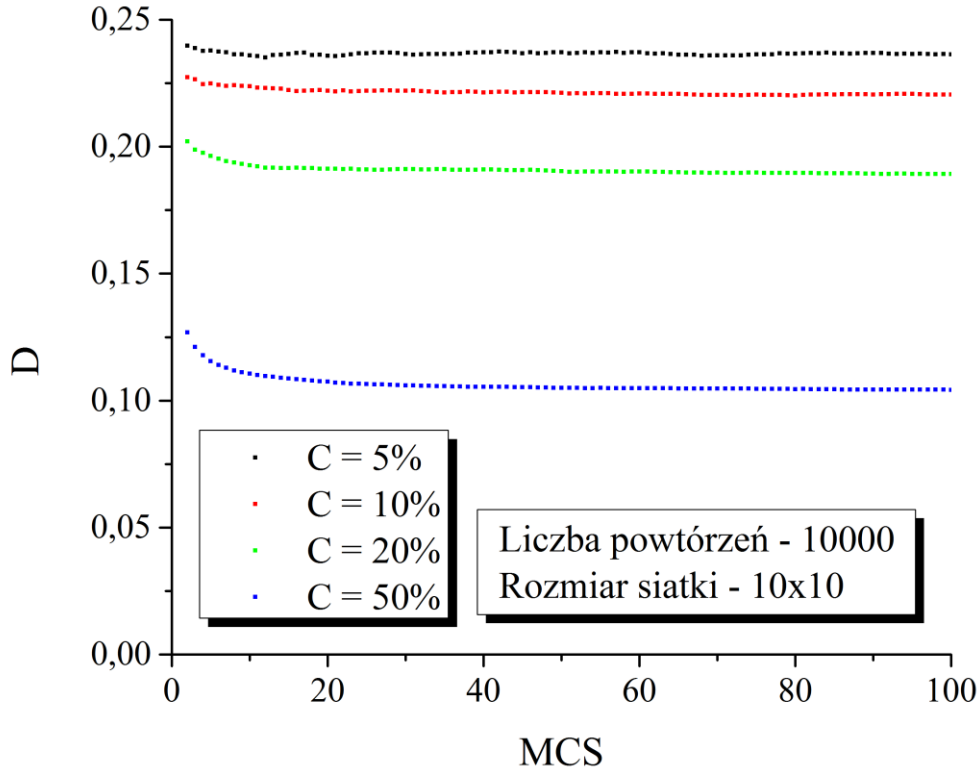
Mateusz Krzykowski 204211





```
# Libraries import
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats


class atom:
    # Class defining atoms used in simulation

    # Initialize object
    def __init__(self, r):
        self.r = np.array(r)
        self.deltar = np.array([0, 0])

    # Calculate diffusion coefficient D
    def D(self):
        return np.sum(np.power(self.deltar, 2))


L = 10  # Length of the 2D map
L2 = L ** 2  # Number of sites in the map
d = 2  # Dimentionality
nrepeat = 10000  # Number of repeats

random = np.random.randint # Assign random
function

tmax = 100

LEFT = np.array([-1, 0])
RIGHT = np.array([1, 0])
UP = np.array([0, -1])
DOWN = np.array([0, 1])

randomstep = [LEFT, RIGHT, UP, DOWN]


def output(A, filename):
    # Output [x, y] vector for output.txt file
    f = open(filename, 'w')
    for i in A:
        f.write(str(i[0]) + " " + str(i[1]) + "\n")
    f.close()


def output2(A, filename):
    # Output [x, y] vector for output.txt file
    f = open(filename, 'w')
    for i in A:
        f.write(str(i) + "\n")
    f.close()


DofC = []

# Main loop
for N in range(1, L2):
    is_graph = N in [5, 10, 20, 50]
    c = N / L2
    DofC.append([c, 0.0])
    print("c = " + str(int(c * 100)) + "%")

    graph = np.zeros((nrepeat, tmax))

    for a in range(nrepeat):
        points = set()  # Initialize set of points

        # Generate N points
        for i in range(N):
            while True:
                A = (random(L), random(L))
                if not A in points:
                    points.add(A)
                    break

        items = [atom(point) for point in points] #
Generate array of atoms
        result = []  # Initialize results array

        # Loop over MCS
        for t in range(1, tmax):
            for item in items:  # Loop over atoms
                deltar = randomstep[random(4)]
                r = (item.r + deltar) % L

                if not tuple(r) in points:
                    points.remove(tuple(item.r))
                    points.add(tuple(r))
                    item.r = r
                    item.deltar += deltar

                if is_graph:
                    graph[a, t] += item.D() / t

        # Append calculatedD to DofC list
        DofC[-1][1] += np.sum(np.array([i.D()
                                        for i in items])) / tmax / N / 4.0
/ nrepeat

    if is_graph:
        graph = np.average(graph, axis=0) / N / 2.0 / d
        output2(graph, 'output_N = ' + str(N) + '.txt')

DofC.append([1, 0])
output(DofC, 'output.txt') # Print D of c
```