ΑΡΙΣΤΟΤΕΛΕΙΟ

ΣΧΟΛΗ

Τυχαίοι Περίπατοι στη Στατιστική Φυσική

Επιβλέπων:
Ονοματεπώνυμο

Συγγραφέας:
Ονοματεπώνυμο

Abstract

The scope of this thesis is random simulations and in particular random walk experiments. We start with some basic theory on random walks regarding mean square displacement, number of distinct sites that the random walker visits and the Rosenstock approximation. After that, we begin with a random walk on one and two dimensions calculating the mean square displacement experimentally. Then, we do a continuous random walk while calculating again the mean square displacement. The third experiment calculates the number of distinct sites the random walker has visited in one and two dimension. At last, we run an experiment in a grid with molecule traps and compare our results with the Rosenstock Approximation in order to check its validity.

Περίληψη

Ο στόχος της εργασίας είναι οι στοχαστικές προσομοιώσεις και συγκεκριμένα τα πειράματα τυχαίων περιπάτων. Αρχικά αναφέρουμε την βασική θεωρία γύρω από τους τυχαίους περιπάτους όσον αφορά την μέση τετραγωνική μετατόπιση, τον αριθμό των διαφορετικών θέσεων που επισκέφτηκε το σωματίδιο και την προσέγγιση Rosenstock. Στη συνέχεια, ξεκινάμε με ένα τυχαίο περίπατο σε μία και δύο διαστάσεις υπολογίζοντας την μέση τετραγωνική μετατόπιση. Παρακάτω, επαναλαμβάνουμε τον υπολογισμό αλλά αυτή τη φορά σε δισδιάστατο συνεχή χώρο. Το τρίτο πείραμα υπολογίζει τον αριθμό των διαφορετικών θέσεων που επισκέπτεται ο περιπατητής σε μία και δύο διαστάσεις. Καταλήγουμε υπολογίζοντας την πυκνότητα των χρόνων παγίδευσης σε ένα πλέγμα με μόρια-παγίδες και συγκρίνοντας τα αποτελέσματά μας με την θεωρητική προσέγγιση Rosenstock.

Περιεχόμενα

Κεφάλαιο 1

Εισαγωγή

Οι τυχαίοι περίπατοι είναι μοντέλα που χρησιμοποιούνται σε πάρα πολλούς τομείς της επιστήμης όπως η Οικολογία, η Ψυχολογία, η Επιστήμη Υπολογιστών, η Φυσική, η Χημεία, η Βιολογία και τα Οικονομικά. Συμπεράσματα που βγάζουμε από την μελέτη των τυχαίων περιπάτων, μπορούν να μας βοηθήσουν να βγάλουμε συμπεράσματα για τις τιμές μετοχών [3], να εκτιμήσουμε το μέγεθος του διαδικτύου [1], και να εξηγήσουμε τις ανθρώπινες αποφάσεις σε σχέση με το χρόνο τον οποίο θα παρθούν [6].

Εμείς θα ασχοληθούμε περισσότερο με τυχαία πειράματα προσαρμοσμένα στις ανάγκες της στατιστικής φυσικής, όπως για παράδειγμα την μεταφορά ηλεκτρονίων σε νανοκρυσταλλικά TiO_2 ηλεκτρόδια [5], την κίνηση Brown των ατόμων σε αέρια, και άλλα.

1.1 Ο κώδικας

Όλα τα πειράματα θα πραγματοποιηθούν στην γλώσσα Python. Η Python είναι μία αντικειμενοστραφής γλώσσα με πάρα πολλά πακέτα και βιβλιοθήκες για ότι ανάγκη μπορεί κανείς να φανταστεί. Αναφέρουμε εδώ τις βιβλιοθήκες που θα εισάγουμε στο πρόγραμμα, αν και οι περισσότερες χρησιμοποιούνται σχεδόν σε κάθε επιστημονική εφαρμογή της γλώσσας. Οποιοσδήποτε μπορεί να βρει στο διαδίκτυο το documentation και τον κώδικα τον οποίο χρησιμοποιούν αυτά τα πακέτα.

- Η βασικότερη βιβλιοθήκη της Python. Πακέτο αριθμητικών υπολογισμών γραμμικής άλγεβρας και προσομοιώσεων. Πολύ γρήγορο και αποτελεσματικό κατασκευασμένο σε C και C++.

```
import numpy as np
```

- Built-in πακέτο της Python εξοπλισμένο με γεννήτριες τυχαίων αριθμών για τυχαίο sampling από κατανομές.

```
import random
```

- Built-in πακέτο της Python για τον υπολογισμό του χρόνου εκτέλεσης προγραμμάτων.

```
import time
```

- Το καλύτερο πακέτο για το σχεδιασμό γραφημάτων οποιουδήποτε τύπου.

```
import matplotlib.pyplot as plt
```

- Συμπληρωματικό πακέτο για το σχεδιασμό γραφημάτων κατασκευασμένο με την matplotlib.

```
import seaborn as sns
```

- Το καλύτερο πακέτο για την εφαρμογή αλγορίθμων μηχανικής μάθησης σε δεδομένα.

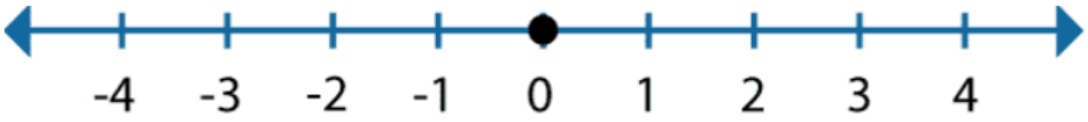
```
from sklearn.linear_model import LinearRegression
```

Κεφάλαιο 2

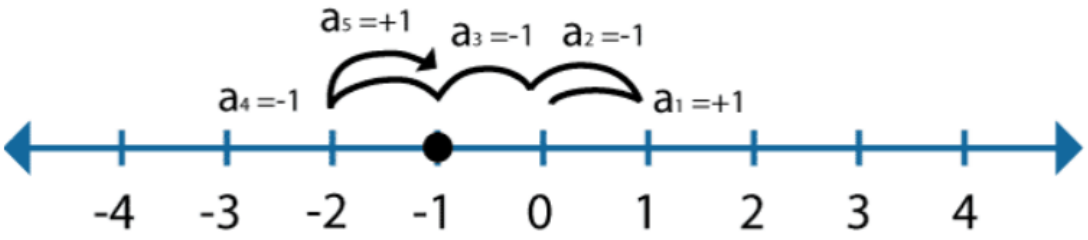
Θεωρία

2.1 Μέση τετραγωνική μετατόπιση

Μία από τις μεταβλητές τις οποίες θα υπολογίσουμε στα πειράματά μας είναι η μέση τετραγωνική μετατόπιση ενός τυχαίου περιπάτου. Ο απλούστερος τυχαίος περίπατος είναι μονοδιάστατος. Έστω μία μαύρη τελεία στην γραμμή των ακεραίων η οποία ξεκινάει από το κέντρο.



Η τελεία ξεκινά να κινείτε είτε αριστερά είτε δεξιά με ίση πιθανότητα $1/2$. Ας καλέσουμε το πρώτο βήμα a_1, το δεύτερο βήμα a_2 κ.ο.κ. Κάθε a έχει είτε την τιμή -1 (κίνηση αριστερά), είτε την τιμή $+1$ (κίνηση δεξιά). Για παράδειγμα:



Ας υπολογίσουμε την μέση τετραγωνική απόσταση ενός τυχαίου περιπάτου N βημάτων:

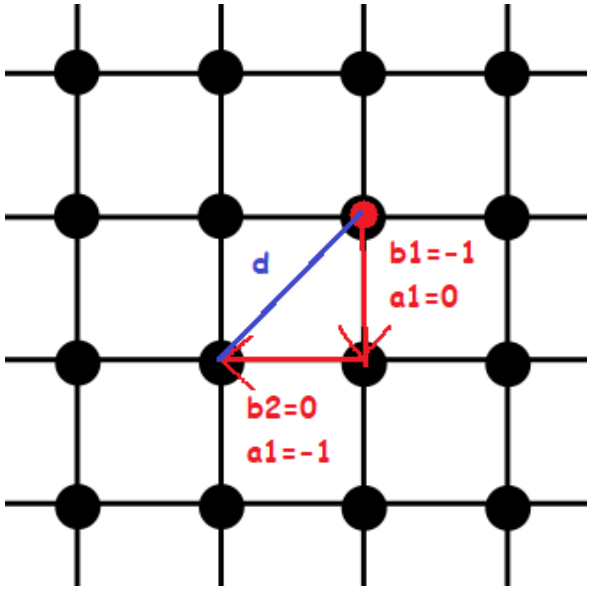
$$\left\langle d^2 \right\rangle = \left\langle (a_1 + a_2 + a_3 + \ldots + a_N)^2 \right\rangle = \langle (a_1 + a_2 + a_3 + \ldots + a_N)\,(a_1 + a_2 + a_3 + \ldots + a_N) \rangle$$
$$= \left(\left\langle a_1^2 \right\rangle + \left\langle a_2^2 \right\rangle + \left\langle a_3^2 \right\rangle + \ldots + \left\langle a_N^2 \right\rangle \right) + 2\left(\langle a_1 a_2 \rangle + \langle a_1 a_3 \rangle + \ldots \langle a_2 a_N \rangle + \ldots \right) \tag{2.1}$$

Είναι προφανές ότι τα a_i^2 με $i = 1, 2 \ldots N$, είναι πάντα ίσα με 1, μιας και η μόνες πιθανές τιμές των a_i είναι -1 και 1. Επομένως, ο πρώτος όρος της (2.1) δίνει $(1 + 1 \ldots . + 1) = N$. Ο δεύτερος όρος έχει το άθροισμα όλων των μέσων όρων των γινομένων διαφορετικών βημάτων, δηλαδή των $\langle a_i a_j \rangle$ με $i \neq j$. Οι πιθανές τιμές του των γινομένων είναι:

$$\begin{array}{ccc} a_i & a_j & a_i a_j \\ 1 & 1 & 1 \\ 1 & -1 & -1 \\ -1 & 1 & -1 \\ -1 & -1 & 1 \end{array} \tag{2.2}$$

Δηλαδή έχουμε -1 με πιθανότητα $1/2$ και 1 με πιθανότητα $1/2$. Προφανώς ο μέσος όρος αυτών των τιμών είναι 0. Συνεπώς, ο δεύτερος όρος δίνει $2(0 + 0 \ldots + 0)$. Άρα τελικά, η αναμενόμενη τετραγωνική μετατόπιση ενός τυχαίου περιπάτου σε μία διάσταση, είναι ίση με τον αριθμό N των βημάτων του τυχαίου περιπάτου, εφόσον step $= 1$.

Μπορεί κανείς εύκολα με την ίδια λογική, να αποδείξει το ίδιο αποτέλεσμα για ένα δισδιάστατο τυχαίο περίπατο. Έχουμε τώρα δύο κάθετους μεταξύ τους άξονες $\mathbb{Z}$, δηλαδή ένα πλέγμα, στο οποίο τώρα η μετατόπιση υπολογίζεται με το πυθαγώρειο θεώρημα (μπλε γραμμή).



Το σωματίδιο (κόκκινο) έχει $1/4$ πιθανότητα να κινηθεί σε μία από τις 4 κατευθύνσεις, άρα έχει $1/4$ να κάνει $+1$, $1/4$ να κάνει -1, και $1/2$ να μην κινηθεί. Επομένως, σε αυτή την περίπτωση αν ονομάσουμε x και y τις αποστάσεις που διένυσε ανά άξονα, τότε:

$$\left\langle d^2 \right\rangle = \left\langle x^2 + y^2 \right\rangle = \left\langle x^2 \right\rangle + \left\langle y^2 \right\rangle \tag{2.3}$$

Είναι προφανές ότι $\left\langle x^2 \right\rangle = \left\langle y^2 \right\rangle$ αφού πρόκειται για ανεξάρτητους άξονες στους οποίους στατιστικά συναντάμε ακριβώς την ίδια συμπεριφορά. Άρα:

$$\left\langle y^2 \right\rangle = \left(\left\langle b_1^2 \right\rangle + \left\langle b_2^2 \right\rangle + \left\langle b_3^2 \right\rangle + \ldots + \left\langle b_N^2 \right\rangle\right) + 2\left(\langle b_1 b_2 \rangle + \langle b_1 b_3 \rangle + \ldots \langle b_2 b_N \rangle + \ldots\right) \tag{2.4}$$

Για τον μέσο όρου του τετραγώνου γνωρίζουμε από τα παραπάνω ότι έχουμε $1/2$ πιθανότητα να δώσει 1 ($1/4$ για το -1 και $1/4$ για το $+1$) και $1/2$ πιθανότητα να δώσει 0 (δηλαδή να μη κινηθεί στον άξονα ψ αλλά στον ξ). Άρα:

$$\langle b_i^2 \rangle = \frac{1}{2} * 1 + \frac{1}{2} * 0 = \frac{1}{2} \tag{2.5}$$

Για τα γινόμενα διαφορετικών όρων φτιάχνουμε ξανά τον πίνακα:

b_i	b_j	$b_i b_j$
1	1	1
1	−1	−1
1	0	0
1	0	0
−1	1	−1
−1	−1	1
−1	0	0
−1	0	0
0	1	0
0	−1	0
0	0	0
0	0	0
0	1	0
0	−1	0
0	0	0
0	0	0

(2.6)

Συμπεριλάβαμε δύο φορές το 0 στο b_i και στο b_j, για να διακρίνουμε καλύτερα ότι είναι δύο φορές πιο πιθανή τιμή από το 1 και το -1 σε κάθε περίπτωση. Δηλαδή, κάθε γραμμή/ενδεχόμενο του πίνακα, έχει την ίδια πιθανότητα να λάβει χώρα. Από τον πίνακα παρατηρούμε ότι έχουμε 12/16=3/4 πιθανότητα το γινόμενο να είναι 0, 2/16=1/8 να είναι 1 και 2/16=1/8 να είναι -1. Άρα:

$$\langle b_i b_j \rangle = \frac{1}{8} * 1 + \frac{1}{8} * (-1) + \frac{3}{4} * 0 = 0 \tag{2.7}$$

Εύκολα φαίνεται από την (2.4) ότι:

$$\left\langle y^2 \right\rangle = \left(\frac{1}{2} + \frac{1}{2} + \ldots \frac{1}{2} \right) + 2\,(0 + 0 + \ldots + 0) = \frac{1}{2} * N \tag{2.8}$$

Συνεπώς, γνωρίζοντας ότι το ίδιο ισχύει και για το $\langle x^2 \rangle$, από τη (2.3) μαθαίνουμε ότι και στην δισδιάστατη περίπτωση η αναμενόμενη τετραγωνική μετατόπιση είναι ίση με $N/2 + N/2 = N$.

2.2 Αριθμός διαφορετικών θέσεων που επισκέφτηκε το σωματίδιο

Η δεύτερη σημαντική παράμετρος που θα μελετήσουμε κυρίως σε δισδιάστατα πλέγματα, είναι ο αριθμός των διαφορετικών τοποθεσιών που έχει επισκεφτεί το σωματίδιο κατά τη διάρκεια του τυχαίου περιπάτου. Ο υπολογισμός της S_n όπως συνήθως συμβολίζουμε αυτή την μεταβλητή, είναι μακροσκελής και δεν είναι στους στόχους της συγκεκριμένης εργασίας. Θα αναφέρουμε λοιπόν τα απαραίτητα για την μελέτη μας, και θα παραπέμψουμε στις δημοσιεύσεις για την αναλυτικότερη προσέγγιση του θέματος.

Μία σημαντική παρατήρηση είναι ότι ο αριθμός των διαφορετικών θέσεων σε ένα τυχαίο περίπατο εξαρτάται προφανώς από τον αριθμό των βημάτων που έχουν εκτελεστεί. Επομένως ο αριθμός S_n είναι ουσιαστικά μία συνάρτηση των βημάτων n. Ο αναλυτικός υπολογισμός των συναρτήσεων για μία, δύο και τρεις διαστάσεις βρίσκεται στην αναφορά [4], από την οποία εμείς θα κρατήσουμε τις περιπτώσεις:

$$1\mathrm{D} \quad S_n \sim (8n/\pi)^{\frac{1}{2}} \tag{2.9}$$

και

$$2\mathrm{D} \quad S_n \sim \pi n / \log n \tag{2.10}$$

Παρατηρείστε ότι πρόκειται για αναλογίες και όχι για ισότητες. Αναμένουμε λοιπόν συναρτήσεις που να έχουνε τις ίδιες ιδιότητες και την ίδια μορφή, αλλά να μην συμπίπτουν σε απόλυτους αριθμούς.

2.3 Προσέγγιση Rosenstock

Στα πειράματά μας, θα ελέγξουμε την εγκυρότητα της θεωρητικής προσέγγισης Rosenstock. Η προσέγγιση Rosenstock αφορά τυχαίους περιπάτους σε πλέγματα με μόρια-παγίδες.

Έστω λοιπόν ότι έχουμε ένα πλέγμα και τοποθετούμε μόρια-παγίδες σε τυχαίες θέσεις του με βάση κάποια συγκέντρωση c. Για ένα πλέγμα δύο διαστάσεων x γραμμών και y στηλών:

$$c = \frac{number\ of\ traps}{number\ of\ grid\ positions} = \frac{N}{xy} \tag{2.11}$$

Στη συνέχεια αφήνουμε ένα σωματίδιο να εκτελέσει ένα τυχαίο περίπατο από μία κενή θέση του πλέγματος, μέχρις ότου να πέσει σε ένα μόριο-παγίδα. Τα βήματα που εκτέλεσε κατά τον περίπατο μέχρι τη στιγμή που παγιδεύτηκε, ονομάστηκε χρόνος παγίδευσης και συμβολίζεται είτε με n είτε με t.

Η Θεωρητική προσέγγιση Rosenstock αφορά την κατανομή πυκνότητας αυτών των χρόνων παγίδευσης για δεδομένη συγκέντρωση. Η πρώτη εμφάνιση της προσέγγισης οφείλεται στη δημοσίευση του 1970 [7] ενώ μία απλοποιημένη παρουσίαση της λογικής πίσω από την φόρμουλα μπορεί να βρεθεί στην [2]. Ο κατανομή λοιπόν έχει τη μορφή:

$$\Phi(n) = (1 - c)^{S_n} \tag{2.12}$$

Στην δισδιάστατη περίπτωση που μας ενδιαφέρει περισσότερο, ακολουθούμε την μορφή της (2.10). Για να την χρησιμοποιήσουμε, βάζουμε μία σταθερά κανονικοποίησης Γ μπροστά στην συνάρτηση με αποτέλεσμα να έχω:

$$\Phi(n) = (1 - c)^{\Gamma\pi n/\log n} = (1 - c)^{\Gamma}(1 - c)^{\pi n/\log n} = \Delta(1 - c)^{\pi n/\log n} \tag{2.13}$$

Επομένως, ορίσαμε την νέα σταθερά Δ κανονικοποίησης της κατανομής των χρόνων παγίδευσης, και αποδείξαμε ότι για τις δύο διαστάσεις:

$$\Phi(n) \sim (1 - c)^{\pi n/\log n} \tag{2.14}$$

Συνεπώς, όταν θα έχουμε την κατανομή και τις τιμές της θα υπολογίζουμε την σταθερά κανονικοποίησης λύνοντας ως προς Δ την

$$\sum_n \Delta(1-c)^{\pi n/\log n} = 1 \tag{2.15}$$

Κεφάλαιο 3

Πειράματα

3.1 Διακριτός Τυχαίος Περίπατος - Μέση τετραγωνική μετατόπιση

3.1.1 Σε μία διάσταση

Η πρώτη και απλούστερη προσομοίωση που θα κάνουμε, αφορά ένα τυχαίο περίπατο σωματιδίου σε ένα πλέγμα μίας διάστασης. Ο σκοπός είναι να υπολογίσουμε την μέση τετραγωνική μετατόπιση 100000 προσομοιώσεων που ονομάζουμε runs, όπως φαίνεται και στον κώδικα. Το κάθε run θα κάνει t=1000 τυχαία βήματα είτε αριστερά είτε δεξιά μεγέθους step=1.

Για να μοντελοποιήσουμε έναν τέτοιο τυχαίο περίπατο, φανταζόμαστε την γραμμή των ακεραίων αριθμών και επιλέγουμε αυθαίρετα ένα σημείο εκκίνησης. Για απλότητα, επιλέγουμε ως αρχική θέση την αρχή του άξονα δηλαδή position=0. Στη συνέχεια ένας βρόγχος επανάληψης t=1000 βημάτων ανανεώνει την θέση του σωματιδίου προσθέτοντας ή αφαιρώντας step=1 στη μεταβλητή position. Η τυχαία επιλογή του βήματος αριστερά ή δεξιά γίνεται με την εντολή random.choice που επιλέγει με ίση πιθανότητα ένα μέλος της λίστας [-step,+step]. Έτσι το βήμα αριστερά γίνεται αν στην θέση που βρίσκεται το σωματίδιο προσθέσουμε -1, ενώ το βήμα δεξιά αν προσθέσουμε +1. Επομένως, ο αλγόριθμος ενός πειράματος έχει ως εξής:

```
//=============================================================//
set starting position

For t=1000 steps:
    choose randomly(uniformly) if the particle goes left or right
    update particle position
//=============================================================//
```

Η ομοιόμορφη κατανομή εξασφαλίζει ότι σε κάθε βήμα το σωματίδιο έχει 1/2 πιθανότητα να μεταβεί στην θέση αριστερά του, και 1/2 πιθανότητα να μεταβεί στη θέση δεξιά του. Ο κώδικας που υλοποιεί τον παραπάνω αλγόριθμο:

```
position = 0
for j in range(t):
    move = random.choice([-step,+step])
    position = position + move
```

Έχοντας λοιπόν την τελική θέση είναι εύκολο να υπολογίσουμε την τετραγωνική μετατόπιση του πειράματος, απλά υψώνοντας την μεταβλητή position στο τετράγωνο. Σκοπός όμως είναι να τρέξουμε το πείραμα 100000 φορές και να υπολογίσουμε την μέση τετραγωνική απόσταση όλων των runs. Άρα στον τελικό κώδικα αρχικοποιούμε μία μεταβλητή sd_sum=0 (ακρωνύμιο του square distance sum), και σε κάθε run προσθέτουμε την τελική τετραγωνική μετατόπιση. Έτσι, υπολογίζουμε το άθροισμα όλων των τετραγωνικών μετατοπίσεων για τα 100000 πειράματα. Τέλος, διαιρούμε αυτό τον αριθμό με τον αριθμό των πειραμάτων που τρέξαμε προκειμένου να βρούμε την μέση τετραγωνική μετατόπιση meam_sd.

```
start_time = time.time()

sd_sum = 0
runs = 100000
t=1000
step = 1
for i in range(runs):
    position = 0
    for j in range(t):
        move = random.choice([-step,+step])
        position = position + move
    sd_sum+=position**2
mean_sd = sd_sum/runs
```

```
print(f"The Mean Square Displacement is {mean_sd}.")
print(f"Execution Time: {(time.time() - start_time)} seconds.")
```

και το output:

```
The Mean Square Displacement is 995.37292.
Execution Time: 55.54194784164429 seconds.
```

3.1.2 Σε δύο διαστάσεις

Την ίδια λογική θα χρησιμοποιήσουμε στο πείραμα δύο διαστάσεων. Αυτή την φορά αντί να έχουμε μία γραμμή ακεραίων φανταζόμαστε δύο, κάθετες μεταξύ τους, με σημείο τομής το (0,0) όπως σε ένα καρτεσιανό σύστημα αξόνων. Η διαφορά είναι ότι ο χώρος μας αποτελείται από τα διακριτά ζεύγη των ακεραίων $(x, y) \in \mathbb{Z} \times \mathbb{Z}$. Έχουμε δηλαδή ένα πλέγμα δύο διαστάσεων πάνω στο οποίο το σωματίδιό μας θα εκτελέσει τον τυχαίο περίπατο. Η αρχική μας θέση πλέον έχει δύο συνιστώσες, την τετμημένη και την τεταγμένη.

Αρχικοποιούμε για το ένα run δύο μεταβλητές position_x=0 και position_y=0. Στη συνέχεια για την μία εκτέλεση του πειράματος t=1000 βημάτων, επιλέγουμε σε κάθε επανάληψη τυχαία, αν το σωματίδιο θα κινηθεί αριστερά, δεξιά, πάνω ή κάτω. Ανανεώνουμε την τετμημένη και την τεταγμένη αναλόγως, και έχουμε πλέον ως νέα θέση του σωματιδίου στο δισδιάστατο πλέγμα τις νέες τιμές (position_x, position_y). Συνεπώς, ο αλγόριθμος ενός πειράματος έχει ως εξής:

```
//=============================================================//
set starting position
For t=1000 steps:
    choose randomly(uniformly) if the particle goes
                            left, right, up or down
    update particle position
//=============================================================//
```

Προφανώς η ομοιόμορφη τυχαία επιλογή μετάβασης δίνει $1/4$ πιθανότητα για την κάθε κατεύθυνση που μπορεί να κινηθεί το σωματίδιο. Η μετάβαση αριστερά σημαίνει πρόσθεση -1 στην τετμημένη και 0 στην τεταγμένη, η μετάβαση επάνω πρόσθεση $+1$ στην τεταγμένη και 0 στην τετμημένη κ.ο.κ. Ο κώδικας που υλοποιεί το ένα πείραμα είναι:

```
position_x = 0
position_y = 0
for j in range(t):
    move = random.choice([[-step,0],[step,0],[0,-step],[0,step]])
    position_x+= move[0]
    position_y+= move[1]
```

Έχοντας την τελική θέση του σωματιδίου μπορούμε να υπολογίσουμε την τετραγωνική μετατόπιση με χρήση του πυθαγόρειου θεωρήματος ως $d^2 = x^2 + y^2$.

Τρέχουμε λοιπόν το πρόγραμμα 100000 φορές και κάθε φορά προσθέτουμε την τετραγωνική μετατόπιση στην μεταβλητή sd_sum:

```
sd_sum+=position_x**2+position_y**2
```

Φυσικά στο τέλος διαιρούμε με τον αριθμό των πειραμάτων που εκτελέσαμε για να βρούμε την μέση τετραγωνική μετατόπιση των 100000 runs. Ο συνολικός κώδικας είναι ως εξής

```
start_time = time.time()

sd_sum = 0
runs = 100000
t=1000
step = 1
for i in range(runs):
    position_x = 0
    position_y = 0
    for j in range(t):
        move = random.choice([[-step,0],[step,0],[0,-step],[0,step]])
        position_x+= move[0]
        position_y+= move[1]
    sd_sum+=position_x**2+position_y**2
mean_sd = sd_sum/runs

print(f"The Mean Square Displacement is {mean_sd}.")
print(f"Execution Time: {(time.time() - start_time)} seconds.")
```

με output:

```
The Mean Square Displacement is 1009.55048.
Execution Time: 79.88995146751404 seconds.
```

3.2 Συνεχής τυχαίος περίπατος σε δύο διαστάσεις-Μέση Τετραγωνική Απόσταση

Στην συνέχεια των πειραμάτων με σωματίδιο σε τυχαίο περίπατο θα προσομοιώσουμε ένα συνεχή χώρο και όχι ένα πλέγμα με ακέραιες τιμές. Ο τρόπος να προσεγγίζουμε συνεχής χώρους υπολογιστικά είναι να δεχτούμε μικρές υποδιαιρέσεις των παραμέτρων. Ένα δισδιάστατο χώρο μπορούμε να τον παραμετροποιήσουμε με χρήση πολικών συντεταγμένων. Ο τρόπος με τον οποίο θα ανανεώνεται η θέση του σωματιδίου στον συνεχή χώρο είναι με την τυχαία επιλογή γωνίας. Χωρίζουμε το επίπεδο σε 360 ίσες διαμερίσεις ενός κυκλικού δίσκου. Επομένως, για ένα πείραμα, θα έχουμε μία επανάληψη χιλίων βημάτων και θα επιλέγουμε τυχαία μία γωνία. Συνεπώς ο αλγόριθμος θα έχει τη μορφή:

```
//=============================================================//
set starting position
For t=1000 steps:
    choose randomly(uniformly) an angle
    update particle position based on that angle
//=============================================================//
```

Για την τυχαία επιλογή της γωνίας εκμεταλλευόμαστε την εντολή random.randint(0,359) σε μοίρες και μετατρέπουμε σε rad με χρήση του τύπου:

$$\frac{angle}{180} = \frac{rad}{\pi} \tag{3.1}$$

άρα:

```
angle = (random.randint(0,359)/180)*np.pi
```

Η ανανέωση της θέσης γίνεται προβάλλοντας το ευθύγραμμο τμήμα μήκους step=1 στον άξονα x πολλαπλασιάζοντας με το συνημίτονο της γωνίας, και στον άξονα y πολλαπλασιάζοντας με το ημίτονο της γωνίας. Συνεπώς για ένα πείραμα κώδικας έχει τη μορφή:

```
position_x = 0
position_y = 0
for j in range(t):
    angle = (random.randint(0,359)/180)*np.pi
    position_x+=round(step*np.cos(angle),2)
    position_y+=round(step*np.sin(angle),2)
```

Η συνάρτηση round στρογγυλοποιεί τους υπολογισμούς των τριγωνομετρικών στο δεύτερο δεκαδικό. Εφόσον θέλουμε να υπολογίσουμε ξανά την μέση τετραγωνική μετατόπιση, τρέχουμε τον κώδικα για runs=100000 πειράματα και προσθέτουμε την τετραγωνική απόσταση από την αρχή των αξόνων (την αρχική μας θέση) σύμφωνα με το πυθαγόρειο θεώρημα. Στο τέλος διαιρούμε το άθροισμα με τον αριθμό των πειραμάτων για να βρούμε τη μέση τιμή. Ο συνολικός κώδικας έχει ως εξής:

```
start_time = time.time()

sd_sum = 0
runs = 100000
t=1000
step = 1
for i in range(runs):
    position_x = 0
    position_y = 0
    for j in range(t):
        angle = (random.randint(0,359)/180)*np.pi
        position_x+=round(step*np.cos(angle),2)
        position_y+=round(step*np.sin(angle),2)
    sd_sum+=position_x**2+position_y**2
mean_sd = sd_sum/runs

print(f"The Mean Square Displacement is {mean_sd}.")
print(f"Execution Time: {(time.time() - start_time)} seconds.")
```

με output:

```
The Mean Square Displacement is 999.8212898829913.
Execution Time: 963.7810792922974 seconds.
```

Τώρα θα υπολογίσουμε ξανά την μέση τετραγωνική μετατόπιση αλλά αυτή τη φορά θα πάρουμε τους μέσους όρους των 100000 πειραμάτων ανά 100 τυχαία βήματα. Δηλαδή θα βρούμε την μέση τετραγωνική μετατόπιση από 100000 πειράματα με 100 βήματα, την μέση τετραγωνική μετατόπιση από 100000 πειράματα με 200 βήματα κ.ο.κ. Η βασική δομή του κώδικα είναι όμοια με το προηγούμενο πείραμα σε συνεχές χώρο αλλά με κάποιες τροποποιήσεις.

Αρχικοποιούμε μία λίστα 10 θέσεων ονόματι averages με την τιμή μηδέν σε κάθε στοιχείο της. Σε αυτή τη λίστα θα προσθέτουμε τις τιμές της τετραγωνικής μετατόπισης για τα 100,200,300....1000 βήματα και θα διαιρούμε με τον αριθμό των πειραμάτων προκειμένου να έχουμε τις μέσες τιμές ανά αριθμό βημάτων.

Το σχήμα του αλγορίθμου ενός πειράματος έχει ως εξής:

```
//=================================================================//
set starting position
create an empty list named values

For 1000 steps:
    choose randomly(uniformly) an angle
    update particle position based on that angle
    if steps are 100,200.....1000
              then fill the list values
              with the square displacement
//=================================================================//
```

Ο κώδικας που υλοποιεί το ένα πείραμα είναι:

```
position = [0,0]
values = []
for j in range(1,t+1):
    angle = (random.randint(0,359)/180)*np.pi
    position[0]+=round(np.cos(angle),2)
    position[1]+=round(np.sin(angle),2)
    if j%100==0:
        values.append(position[0]**2+position[1]**2)
```

Λεπτομερέστερα, το block κώδικα:

```
if j%100==0:
    values.append(position[0]**2+position[1]**2)
```

ελέγχει πότε ο αριθμός των βημάτων είναι πολλαπλάσιο του 100 (κοιτώντας να μηδενίζεται το υπόλοιπο της διαίρεσης με το 100) και γεμίζει την λίστα values με τις 10 τετραγωνικές μετατοπίσεις όπως φαίνεται και στον ψευδοκώδικα. Ο λόγος που επιλέχθηκε το range(1,t+1) και όχι το range(0,t) είναι καθαρά για την ευκρίνεια του κώδικα και τη χρήση του mod100 αντί του mod99.

Έτσι, για κάθε ένα πείραμα πρέπει να προσθέσουμε στην αρχικοποιημένη με μηδενικά λίστα averages τις τιμές της λίστας values διαιρεμένες με τον αριθμό των πειραμάτων προκειμένου να έχουμε στο τέλος τις μέσες τετραγωνικές μετατοπίσεις 100000 πειραμάτων ανά 100 βήματα του σωματιδίου.

Ο τελικός κώδικας θα είναι:

```
start_time = time.time()

runs = 10000
t=1000
averages = [0]*10
for i in range(runs):
    position = [0,0]
    values = []
    for j in range(1,t+1):
        angle = (random.randint(0,359)/180)*np.pi
        position[0]+=round(np.cos(angle),2)
        position[1]+=round(np.sin(angle),2)
        if j%100==0:
            values.append(position[0]**2+position[1]**2)
    for j in range(10):
        averages[j]+=values[j]/runs

print(f"The averages are {averages}.")
print(f"Execution Time: {(time.time() - start_time)} seconds.")
```

με έξοδο:

```
The averages are [99.67193910999949,
                  201.10285323000062,
                  298.58902693000033,
                  399.82761215000033,
                  494.9311487600011,
                  597.2043227299978,
                  693.3227494999987,
                  791.3730152799965,
                  901.8665181699997,
                  1003.6801849000012].
Execution Time: 94.43011784553528 seconds.
```

Χρησιμοποιούμε τώρα την μέθοδο ελαχίστων τετραγώνων μέσω του πακέτου `sklearn.linear_model` και του αντικειμένου LinearRegression(). Αρχικά κατασκευάζουμε τα np.array αντικείμενα έτσι ώστε να μπορούν να εισαχθούν στις εντολές που θα χρησιμοποιήσουμε. Ως τετμημένες θα χρησιμοποιήσουμε τους αριθμούς των βημάτων που πραγματοποιήθηκαν στην μεταβλητή t. Η εντολή np.arange(100,1001,100).reshape(-1,1) ουσιαστικά κατασκευάζει ένα στηλοδιάνυσμα με τα πολλαπλάσια του 100 έως και το 1000. Ως τεταγμένες φυσικά τις μέσες τετραγωνικές μετατοπίσεις ανά 100 βήματα.

Ο κώδικας:

```
t = np.arange(100,1001,100).reshape(-1,1)
av = np.array(averages)
```

Εν συνεχεία, κατασκευάζουμε το μοντέλο γραμμικής παλινδρόμησης μέσω των μεθόδων που αναφέραμε, και στη συνέχεια εκτελούμε το fitting, δηλαδή το την εύρεση της ευθείας ελαχίστων τετραγώνων. Στη μεταβλητή m τοποθετούμε την κλίση της ευθείας και στην μεταβλητή c την σταθερά, έτσι ώστε η ευθεία μας να γράφετε ως $y = mt + c$. Ο κώδικας:

```
reg = LinearRegression()
model = reg.fit(t,av)
c = model.intercept_
m = model.coef_[0]
print(f"y = {m}x+{c}")
```

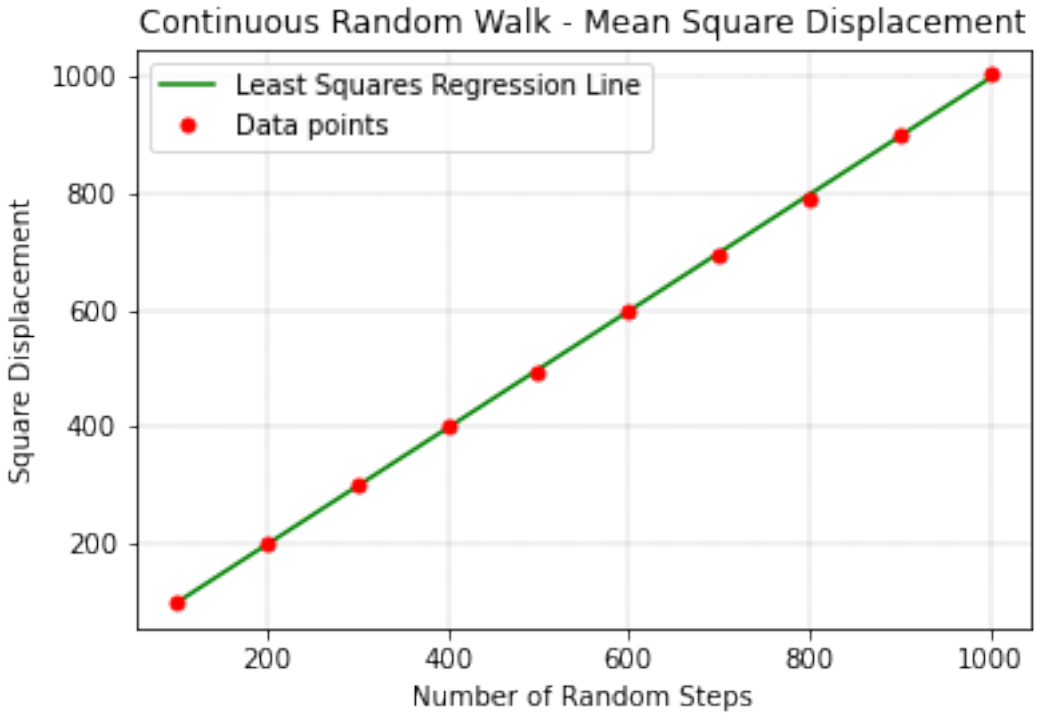
με έξοδο

```
y = 0.999278690573332x+-1.4463427393329766
```

Πλέον μένει να παραστήσουμε γραφικά την ευθεία και τα δεδομένα που χρησιμοποιήθηκαν προκειμένου να καταλήξουμε στα κατάλληλα συμπεράσματα. Για να το κάνουμε αυτό θα χρησιμοποιήσουμε το πακέτο matplotlib.pyplot με το ακρωνύμιο plt όπως συνήθως χρησιμοποιείτε. Χρησιμοποιούμε την εξίσωση της ευθείας και την αναπαριστούμε γραφικά με γραμμή, ενώ τα αρχικά δεδομένα με τελείες. Ονομάζουμε τους άξονες και τα χρώματα σύμφωνα με το πρόβλημά μας. Ο τελικός κώδικας:

```
y = m*t+c
plt.plot(t,y,'-',color ='green',label='Least Squares Regression Line')
plt.plot(t,av,'.',color = 'red',label='Data points',ms=10.0)
plt.legend()
plt.grid(color='0.25', linestyle='--', linewidth=0.2)
plt.ylabel('Square Displacement')
plt.xlabel('Number of Random Steps')
plt.title('Continuous Random Walk - Mean Square Displacement')
plt.show()
```

και το γράφημα:



Σχήμα 3.1: Μέση τετραγωνική απόσταση αν 100 βηματισμούς του σωματιδίου για συνεχή χώρο δύο διατάσεων.

3.3 Αριθμός πλεγματικών θέσεων που το σωματίδιο επισκέφτηκε τουλάχιστον μία φορά

3.3.1 Σε μία διάσταση

Η λογική που θα εκτελεστεί το πείραμα είναι προφανώς παρόμοια με τα προηγούμενα πειράματα μιας και πρόκειται για μονοδιάστατο τυχαίο περίπατο. Πάμε πρώτα να μελετήσουμε πως θα εκτελεστεί το ένα πείραμα σε αυτή την περίπτωση και έπειτα στον τελικό κώδικα θα τρέξει για runs=10000 φορές.

Ψάχνουμε να υπολογίσουμε τον αριθμό των διαφορετικών θέσεων που έχει λάβει το σωματίδιο κατά το ταξίδι του στην γραμμή των ακεραίων αριθμών $\mathbb{Z}$. Θα εκμεταλλευτούμε το γεγονός ότι στη μία διάσταση ο μόνος τρόπος να μεταβεί το σωματίδιο στη θέση $n > 0$ είναι πρώτα να έχει περάσει από τις θέσεις $n-1, n-2 ... 1$, μιας και η εκκίνηση πάντα γίνεται από το 0. Ομοίως, για να έχει βρεθεί στην θέση $-n < 0$ θα πρέπει να έχει περάσει από τις $-n+1, -n+2 ...$ κ.ο.κ. Επομένως, αν γνωρίζουμε τις δύο μεγαλύτερες αποστάσεις (μία προς τα θετικά μία προς τα αρνητικά) από το σημείο εκκίνησης, είμαστε

σίγουροι ότι το σωματίδιο έχει περάσει από όλες τις ενδιάμεσες τοποθεσίες. Έτσι, μπορούμε να βρούμε την maximum και την minimum θέση, και να υπολογίσουμε τον αριθμό των διαφορετικών θέσεων με την φόρμουλα $maxpos - minpos + 1$. Από την μέγιστη θέση αφαιρούμε την αρνητική ελάχιστη θέση, και προσθέτουμε 1 (για να συμπεριληφθεί και η αρχική θέση 0). Για παράδειγμα, αν η μέγιστη θέση είναι 5 και η ελάχιστη -2, τότε ο αριθμός των διαφορετικών θέσεων θα είναι 5-(-2)+1=8 όπου αναφέρεται στις θέσεις -2,-1,0,1,2,3,4,5. Όπως και πριν, πρέπει να κρατήσουμε τους αριθμούς των 10000 πειραμάτων ανά 100 βήματα επομένως πάλι θα πρέπει να αρχικοποιήσουμε μία 10-λίστα `s_t` με μηδενικά. Σε αυτή θα προσθέτουμε κάθε 100 βήματα (από 100 μέχρι 1000) τον αριθμό S των διαφορετικών θέσεων. Ο αλγόριθμος ενός πειράματος:

```
//=============================================================//
set starting position
set min and max variables to zero
for 1000 steps:
    update position randomly(left or right)
    if position bigger than max set position as max
    if position smaller than min set position as min
    if step 100 or 200 or.....1000:
          add to s_t list
//=============================================================//
```

ο κώδικας που τον υλοποιεί:

```
position = 0
min_pos = 0
max_pos = 0
for t in range(1,1001):
    move = random.choice([-step,+step])
    position+= move
    if position<min_pos:
        min_pos = position
    elif position>max_pos:
        max_pos = position
    if t%100==0:
        s_t[t//100-1]+= max_pos-min_pos+1
```

Παρατηρείστε ότι η εντολή `s_t[t//100-1]+= max_pos-min_pos+1` προσθέτει στην θέση 0 της λίστας, τον αριθμό των διαφορετικών θέσεων που επισκέφτηκε το σωματίδιο στα 100 βήματα, στην θέση 1 της λίστας των 200 βημάτων κ.ο.κ.

Ο συνολικός κώδικας:

```
start_time = time.time()

runs = 10000
step = 1
s_t = [0]*10
for i in range(runs):
    position = 0
    min_pos = 0
    max_pos = 0
    for t in range(1,1001):
        move = random.choice([-step,+step])
        position+= move
        if position<min_pos:
            min_pos = position
        elif position>max_pos:
            max_pos = position
        if t%100==0:
            s_t[t//100-1]+= max_pos-min_pos+1
mean_s = np.array([(x/runs) for x in s_t])

print(f"Average Number of Distinct Sites per 100 steps:{mean_s}")
print(f"Execution Time: {(time.time() - start_time)} seconds.")
```

όπου απευθείας δημιουργούμε την numpy array με τις μέσες τιμές των διαφορετικών θέσεων ανά 100 βηματισμούς για όλα τα 10000 runs. Το output:

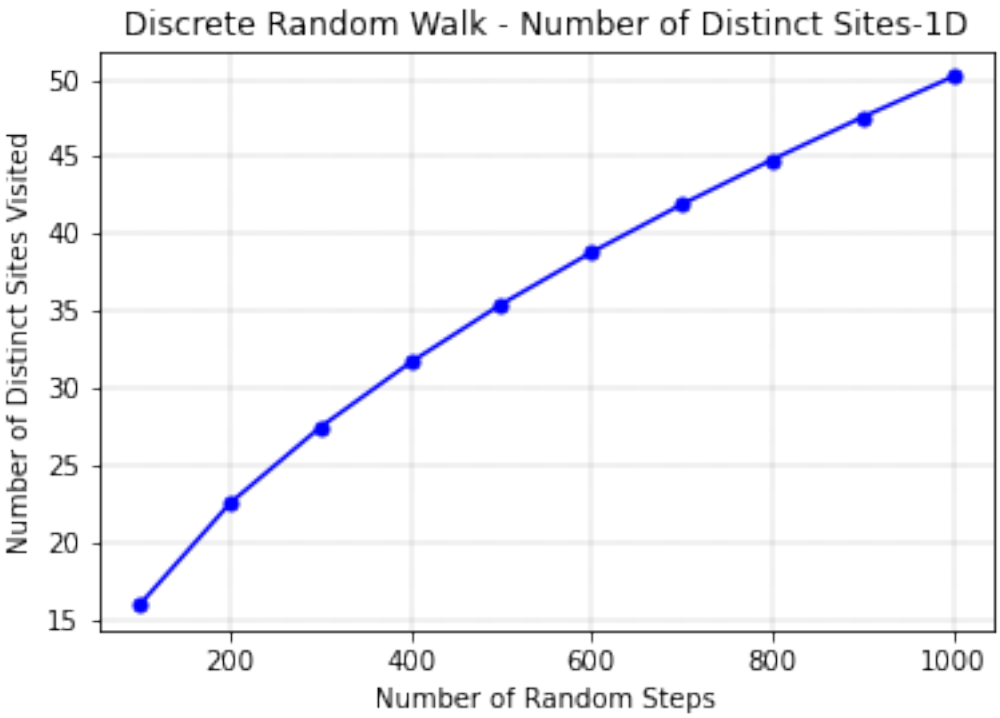
```
Average Number of Distinct Sites per 100 steps:[15.9749
                                                22.5574
                                                27.4858
                                                31.6908
                                                35.4432
                                                38.8125
                                                41.9199
                                                44.8198
                                                47.5799
                                                50.197 ]
Execution Time: 7.148834943771362 seconds.
```

Κατασκευάζουμε τώρα το γράφημα με τετμημένη των αριθμό των βημάτων όπου δημιουργούμε με την εντολή np.arange και τεταγμένη τους μέσους αριθμούς mean_s.

Ο κώδικας:

```
t = np.arange(100,1001,100)
plt.plot(t, mean_s, '.-',color = 'red', ms = 10)
plt.grid(color='0.25', linestyle='--', linewidth=0.2)
plt.ylabel('Number of Distinct Sites Visited')
plt.xlabel('Number of Random Steps')
plt.title('Discrete Random Walk - Number of Distinct Sites-1D')
plt.show()
```

με output:



Σχήμα 3.2: Μέσος αριθμός διαφορετικών τοποθεσιών σε μία διάσταση, που το σωματίδιο επισκέφτηκε τουλάχιστον μία φορά.

3.3.2 Σε δύο διαστάσεις

Στην περίπτωση των δύο διαστάσεων η λογική παραμένει η ίδια όμως χρειάζονται τροποποιήσεις. Η πρώτη τροποποίηση αφορά την επιλογή της επόμενης κίνησης όπως έγινε και σε προηγούμενο πείραμα. Επιλέγουμε τυχαία και ομοιόμορφα μία από τις κινήσεις πάνω, κάτω, αριστερά, δεξιά και ανανεώνουμε την θέση του σωματιδίου. Η δεύτερη τροποποίηση έχει να κάνει με την γνώση μας για το αν η θέση που θα μεταβαίνει το σωματίδιο είναι μία από τις θέσεις που έχει επισκεφθεί προηγουμένως. Το τρικ που χρησιμοποιήσαμε

στην περίπτωση της μίας διάστασης δεν δουλεύει στην περίπτωση των δύο διαστάσεων. Το γεγονός ότι το σωματίδιο βρίσκεται στην θέση (x, y) μας εξασφαλίζει ότι έχει βρεθεί σε μία εκ των $(x-1, y), (x+1, y), (x, y-1), (x, y+1)$ χωρίς όμως να γνωρίζουμε ποια ή ποιες. Ως αποτέλεσμα αυτού, πρέπει σε κάθε βήμα να ελέγχουμε αν η ανανεωμένη θέση είναι θέση που έχει επισκεφτεί το σωματίδιο προηγουμένως. Γι΄ αυτό και σε κάθε πείραμα θα πρέπει να αποθηκεύουμε όλες τις νέες θέσεις που επισκέπτεται το σωματίδιο σε μία λίστα ονόματι `all_positions`. Στο τέλος το μήκος αυτής της λίστας θα είναι και ο αριθμός των διαφορετικών θέσεων που έχει επισκεφθεί το σωματίδιο. Φυσικά διατηρούμε τη στρατηγική των ανά 100 μέσων όρων για τα 10000 runs με χρήση της λίστας `s_t`. Ο αλγόριθμός του ενός πειράματος έχει ως εξής:

```
//=============================================================//
set starting position
set list of different positions visited
for 1000 steps:
    update position randomly(left, right, up, down)
    if position not visited before:
        add new position to the list of different positions
    if steps 100 or 200 ....:
        add length of the different pos list to the s_t list
//=============================================================//
```

Ο κώδικας που υλοποιεί το ένα πείραμα:

```
position_x = 0
position_y = 0
all_positions = [[0,0]]
for j in range(1,t+1):
    move = random.choice([[-step,0],[step,0],[0,-step],[0,step]])
    position_x+= move[0]
    position_y+= move[1]
    position = [position_x,position_y]
    if position not in all_positions:
        all_positions.append(position)
    if j%100==0:
        s_t[j//100-1]+= len(all_positions)
```

Προφανώς όπως και πριν, η `s_t` αρχικοποιείται με μηδενικά στο τελικό πρόγραμμα ώστε να λειτουργήσει για όλα τα runs. Ας σημειώσουμε εδώ ότι η λίστα `all_positions` έχει ως μέλη της, λίστες μήκους 2, που αναπαριστούν την τετμημένη και την τεταγμένη των διαφορετικών θέσεων.

Ο συνολικός κώδικας:

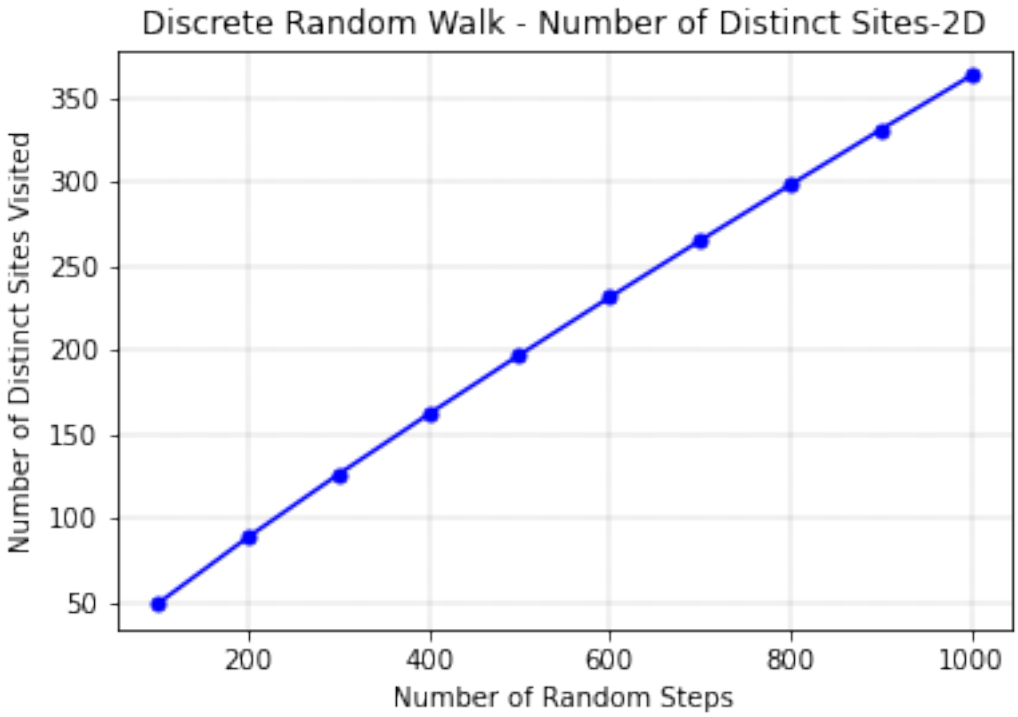
```
start_time = time.time()

runs = 10000
step = 1
s_t = [0]*10
t = 1000
for i in range(runs):
    position_x = 0
    position_y = 0
    all_positions = [[0,0]]
    for j in range(1,t+1):
        move = random.choice([[-step,0],[step,0],[0,-step],[0,step]])
        position_x+= move[0]
        position_y+= move[1]
        position = [position_x,position_y]
        if position not in all_positions:
            all_positions.append(position)
        if j%100==0:
            s_t[j//100-1]+= len(all_positions)

mean_s = np.array([(x/runs) for x in s_t])

print(f"Average Number of Distinct Sites per 100 steps:{mean_s}")
print(f"Execution Time: {(time.time() - start_time)} seconds.")
```

με έξοδο

```
Average Number of Distinct Sites per 100 steps:[49.4849
                                                89.0422
                                                126.3679
                                                162.2419
                                                197.2506
                                                231.6082
                                                265.2785
                                                298.4961
                                                331.2296
                                                363.4049]
Execution Time: 40.61035513877869 seconds.
```

Κατασκευάζουμε όμοια με πριν το γράφημα:

```
t = np.arange(100,1001,100)
plt.plot(t, mean_s, '.-',color = 'blue', ms = 10)
plt.grid(color='0.25', linestyle='--', linewidth=0.2)
plt.ylabel('Number of Distinct Sites Visited')
plt.xlabel('Number of Random Steps')
plt.title('Discrete Random Walk - Number of Distinct Sites-2D')
plt.savefig('../pdf/figures/DRW_NDS_2D.png')
plt.show()
```



Σχήμα 3.3: Μέσος αριθμός διαφορετικών τοποθεσιών σε δύο διαστάσεις, που το σωματίδιο επισκέφτηκε τουλάχιστον μία φορά.

3.4 Χρόνος Παγίδευσης Διακριτού τυχαίου περιπάτου με παγίδες

Ο σκοπός αυτού του πειράματος είναι ο υπολογισμός του χρόνου παγίδευσης ενός σωματιδίου που εκτελεί τυχαίο περίπατο, σε ένα πλέγμα με μόρια-παγίδες σε κάποιες θέσεις του. Η μεθοδολογία που θα ακολουθήσουμε σε αυτό το πείραμα διαφέρει κατά πολύ από αυτές των προηγούμενων πειραμάτων κυρίως λόγω της δομής του κώδικα. Για

την καλύτερη δόμηση και την επίτευξη γενικότητας στον κώδικα, θα χρησιμοποιηθεί αντικειμενοστραφής προγραμματισμός. Θα δημιουργήσουμε μία κλάση ονόματι grid_2D όπου θα έχει όλες τις ιδιότητες που χρειάζονται για την εκτέλεση του τυχαίου περιπάτου με παγίδες. Ενώ εξηγούμε την δομή του κώδικα και τις λεπτομέρειες των εντολών, θα αναδύεται και η μοντελοποίηση του προβλήματος από υπολογιστικής σκοπιάς.

Ξεκινάμε λοιπόν με την magic method __init__ που εκτελείτε με το που δημιουργείτε το αντικείμενο (κάποιο instance) αυτής της κλάσης.

```
def __init__( self, x , y):
    self.x = x
    self.y = y
    self.grid = np.zeros((x,y),dtype=int)
```

Δημιουργούμε ένα αντικείμενο της κλάσης grid_2D δίνοντας τις διαστάσεις ενός πίνακα (np.array) τον οποίο γεμίζουμε με μηδενικά. Αυτό είναι το δισδιάστατο πλέγμα μας x γραμμών και y στηλών μοντελοποιημένο σε ένα πίνακα $x \times y$.

Η επόμενη μέθοδος που θα εντάξουμε στην κλάση μας είναι για την προσθήκη παγίδωνμορίων συγκεκριμένης συγκέντρωσης:

```
N = int((self.x*self.y)*c)
molecules = 0
while molecules<N:
    x = random.randint(0,self.x-1)
    y = random.randint(0,self.y-1)
    if self.grid[x,y]==0:
        self.grid[x,y]=1
        molecules+=1
```

Για να γεμίσουμε με κατάλληλο τρόπο το πλέγμα βρίσκουμε πρώτα τον αριθμό των παγίδων που πρέπει να τοποθετήσουμε με βάση τη συγκέντρωση (2.11). Συνεπώς, $N = xyc$ θα είναι ο αριθμός των παγίδων που πρέπει να τοποθετήσουμε στο grid προκειμένου να έχει συγκέντρωση c. Στην συνέχεια, ξεκινάμε να τοποθετούμε τον αριθμό 1(παγίδες) σε θέσεις του πλέγματος που έχουν 0 μέχρι να τοποθετηθούν και τα N παγίδες-μόρια. Στο τέλος έχουμε στην κατοχή μας έναν πίνακα με N άσσους-παγίδες και $xy - N$ ελεύθερες θέσεις για τον τυχαίο περίπατο.

Η επόμενη μέθοδος της κλάσης μας επιλέγει μία κενή θέση του πλέγματος τυχαία για την αρχική θέση του σωματιδίου και την εκκίνηση του τυχαίου περιπάτου.

Ο κώδικας:

```
def add_particle(self):
    while True:
        x = random.randint(0,self.x-1)
        y = random.randint(0,self.y-1)
        if self.grid[x,y]==0:
            self.particle_position = [x,y]
            break
```

Έχουμε δηλαδή μία αυθαίρετη επανάληψη όπου επιλέγει τυχαία θέσεις του πλέγματος μέχρι να βρει κάποια κενή. Όταν βρει κενή θέση σταματάει η επανάληψη και αποθηκεύει την τοποθεσία του πλέγματος ως αρχική θέση του σωματιδίου.

Η επόμενη μέθοδος της κλάσης μας θα είναι η εκτέλεση του τυχαίου περιπάτου. Για να το κάνουμε αυτό αρχικά ας δημιουργήσουμε με ξεχωριστή συνάρτηση η οποία θα επιστρέφει την νέα θέση του σωματιδίου. Ο λόγος για να δημιουργηθεί μία τέτοια συνάρτηση ανεξάρτητα από το υπόλοιπο πρόβλημα, είναι η συνοριακές συνθήκες που θέλουμε να θέσουμε στο πλέγμα. Στην προκειμένη περίπτωση, όταν το σωματίδιο βρίσκεται σε μία θέση στην άκρη του πλέγματος και κινηθεί προς την κατεύθυνση που δεν υπάρχει άλλη θέση στο πλέγμα, τότε πρέπει να βρεθεί στην απέναντι πλευρά. Δηλαδή, η απέναντι πλευρές του πλέγματος λειτουργούν ως επιφάνεια κυλίνδρου. Παράδειγμα, αν έχω ένα πλέγμα 3x3, το σωματίδιο βρίσκεται στην θέση [0,0] και η τυχαία επιλογή βήματος δώσει up, η νέα θέση θα είναι η [2,0]. Αν η τυχαία επιλογή δώσει left τότε η νέα θέση πρέπει να είναι η [0,2]. Για να πετύχουμε την παραπάνω συμπεριφορά και τις κατάλληλες θεωρητικές εκμεταλλευόμαστε την συνάρτηση mod ως εξής:

```
def get_next_position(pos,rows,columns):
    move = random.choice(['left','right','up','down'])
    if move=='left':
        return [pos[0],(pos[1]-1)%columns]
    elif move=='right':
        return [pos[0],(pos[1]+1)%columns]
    elif move=='up':
        return [(pos[0]-1)%rows, pos[1]]
    elif move=='down':
        return [(pos[0]+1)%rows, pos[1]]
```

Η συνάρτηση δέχεται ως είσοδο την προηγούμενη θέση του σωματιδίου και τις διαστάσεις του πλέγματος, προκειμένου να γνωρίζει πότε βρίσκεται σε ακριανή θέση, καθώς και πως να ανανεώσει και να επιστρέψει τη νέα θέση.

Ο τυχαίος περίπατος :

```
def random_walk(self):
    position = self.particle_position
    steps=0
    while True:
        next_position = get_next_position(position,self.x,self.y)
        if self.grid[next_position[0],next_position[1]]==1:
            steps+=1
            break
        else:
            position = next_position
            steps+=1
    return steps
```

Έχουμε λοιπόν το σωματίδιο σε μία τυχαία επιλεγμένη θέση και ξεκινάει τον τυχαίο περίπατο μέχρι να συναντήσει μία παγίδα. Μόλις βρεθεί στην παγίδα, επιστρέφει τον αριθμό των βημάτων που έκανε το σωματίδιο. Αυτό ήταν και το ζητούμενο, δηλαδή ο χρόνος παγίδευσης.

Για να εκτελέσουμε λοιπόν ένα πείραμα τυχαίου περιπάτου θα πρέπει να δημιουργήσουμε το grid, να προσθέσουμε τις παγίδες, να επιλέξουμε αρχική θέση για το σωματίδιο, και να εκτελέσουμε τον τυχαίο περίπατο. Ο αλγόριθμος μοναδικού πειράματος έχεις ως εξής:

```
//=============================================================//
create grid
add molecule_traps
add particle
do the random walk - get trap_times
//=============================================================//
```

3.4.1 Πλέγμα 500x500 - Συγκέντρωση παγίδων c=0.01

Πάμε τώρα να χρησιμοποιήσουμε τις μεθόδους αυτές για να εκτελέσουμε το πείραμα. Ας δημιουργήσουμε το πλέγμα 500x500 ως instance της κλάσης μας, και ας τοποθετήσουμε και τις παγίδες με συγκέντρωση 0.01:

```
c = 0.01
grid1 = grid_2D(500,500)
grid1.add_molecule_traps(c)
```

Στη συνεχεία θα πρέπει να εκτελέσουμε σε αυτό το πλέγμα 100000 πειράματα και να αποθηκεύσουμε του χρόνους διαφυγής.

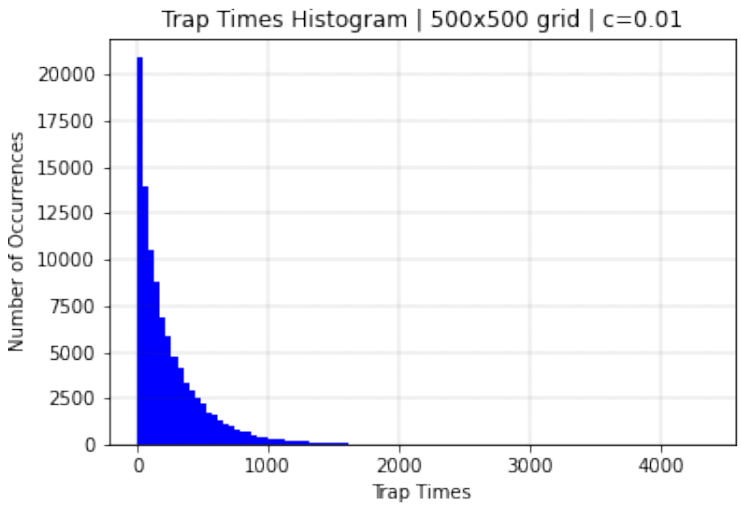
```
start_time = time.time()

trap_times1 = []
for i in range(100000):
    grid1.add_particle()
    trap_times1.append(grid1.random_walk())

data1 = trap_times1.copy()
print(f"Execution time: {(time.time() - start_time)} seconds")
```

με Execution time: 25.733628273010254 seconds.

Έχοντας τώρα ένα μεγάλο δείγμα των χρόνων παγίδευσης μπορούμε να σχεδιάσουμε την κατανομή τους. Δύο τρόποι μπορούν να περιγράψουν την αριθμητική κατανομή των χρόνων παγίδευσης, το ιστόγραμμα και το γράφημα της κατανομής. Για το ιστόγραμμα χρησιμοποιούμε το πακέτο matplotlib.pyplot με το ακρωνύμιο plt:

```
plt.hist(data1, bins = 100, color = 'blue')
plt.grid(color='0.25', linestyle='--', linewidth=0.2)
plt.ylabel('Number of Occurrences')
plt.xlabel('Trap times')
plt.title('Trap Times Histogram | 500x500 grid | c=0.01')
plt.show()
```

και το γράφημα:

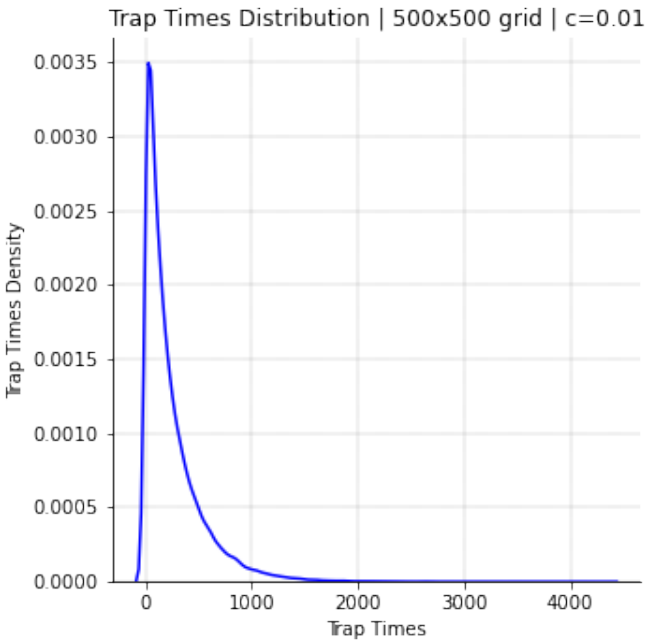


Σχήμα 3.4: Ιστόγραμμα αριθμητικών περιστατικών ανά χρόνο διαφυγής για πλέγμα 500x500 με πυκνότητα παγίδων c=0.01

Για την κατανομή πυκνότητας των χρόνων διαφυγής θα χρησιμοποιήσουμε το πακέτο seaborn με το ακρωνύμιο sns:

```
sns.displot(data1, kind="kde", color = "blue")
plt.grid(color='0.25', linestyle='--', linewidth=0.2)
plt.ylabel('Trap Times Density')
plt.xlabel('Trap Times')
plt.title('Trap Times Density Distribution | 500x500 grid | c=0.01')
plt.tight_layout()
plt.show()
```

και το γράφημα:



Σχήμα 3.5: Κατανομή πυκνότητας χρόνων διαφυγής για πλέγμα 500x500 με πυκνότητα παγίδων c=0.01

3.4.2 Πλέγμα 500x500 - Συγκέντρωση παγίδων c=0.001

Όμοια με προηγουμένως, με την μόνη τροποποίηση την αλλαγή της συγκέντρωσης. Δημιουργούμε το πλέγμα:

```
c = 0.01
grid1 = grid_2D(500,500)
grid1.add_molecule_traps(c)
```

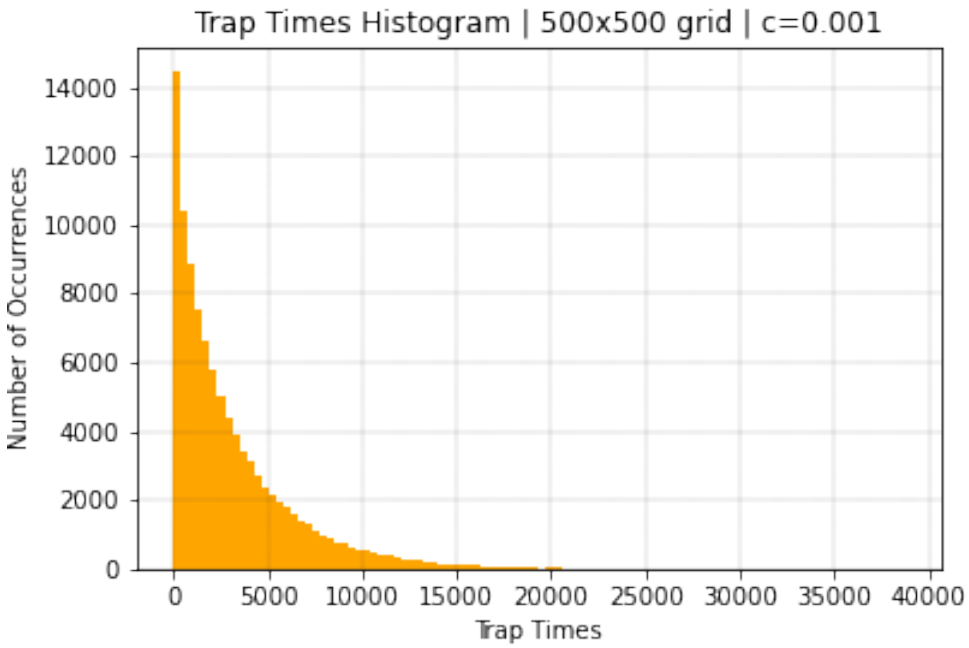
Στη συνεχεία θα πρέπει να εκτελέσουμε σε αυτό το πλέγμα 100000 πειράματα και να αποθηκεύσουμε του χρόνους διαφυγής.

```
start_time = time.time()

trap_times2 = []
for i in range(100000):
    grid2.add_particle()
    trap_times2.append(grid2.random_walk())

data2 = trap_times2.copy()
print(f"Execution time: {(time.time() - start_time)} seconds")
```

με Execution time: 337.0471861362457 seconds. Το ιστόγραμμα:

```
plt.hist(data2, bins = 100, color = 'orange')
plt.grid(color='0.25', linestyle='--', linewidth=0.2)
plt.ylabel('Number of Occurrences')
plt.xlabel('Trap Times')
plt.title('Trap Times Histogram | 500x500 grid | c=0.001')
plt.show()
```

και το γράφημα:

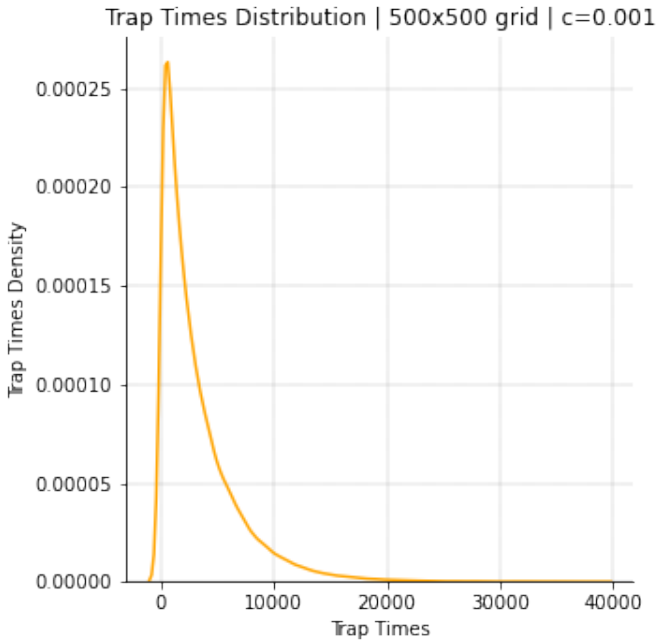


Σχήμα 3.6: Ιστόγραμμα αριθμητικών περιστατικών ανά χρόνο διαφυγής για πλέγμα 500x500 με πυκνότητα παγίδων c=0.001

Για την κατανομή πυκνότητας των χρόνων διαφυγής:

```
sns.displot(data2, kind="kde", color = "orange")
plt.grid(color='0.25', linestyle='--', linewidth=0.2)
plt.locator_params(axis='x', nbins=5)
plt.ylabel('Trap Times Density')
plt.xlabel('Trap Times')
plt.title('Trap Times Density Distribution | 500x500 grid | c=0.001')
plt.tight_layout()
plt.show()
```

και το γράφημα:



Σχήμα 3.7: Κατανομή πυκνότητας χρόνων διαφυγής για πλέγμα 500x500 με πυκνότητα παγίδων c=0.001

3.4.3 Σύγκριση των συγκεντρώσεων 0.01 και 0.001

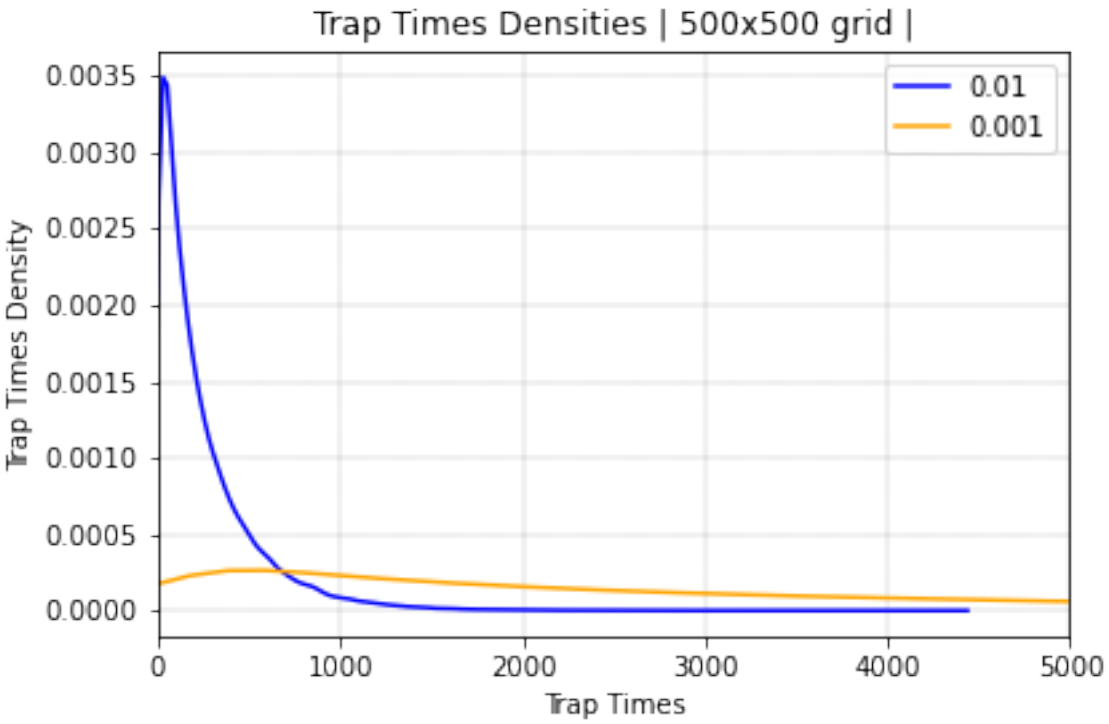
Τοποθετούμε λοιπόν τα δύο διαγράμματα πυκνότητας στο ίδιο διάγραμμα ώστε να παρατηρήσουμε καλύτερα την μεγάλη διαφορά που έχουν. Για να το κάνουμε αυτό εξάγουμε τα δεδομένα της γραμμής των εντολών displot που δημιούργησαν τα διαγράμματα πυκνότητας:

```
(line1_x, line1_y) = sns.distplot(data1).get_lines()[0].get_data()
(line2_x, line2_y) = sns.distplot(data2).get_lines()[0].get_data()
```

και τα χρησιμοποιούμε για να σχεδιάσουμε μαζί τις γραμμές:

```
plt.plot(line1_x, line1_y, color='blue', label = '0.01')
plt.plot(line2_x, line2_y, color='orange', label = '0.001')
plt.xlim(0,5000)
plt.grid(color='0.25', linestyle='--', linewidth=0.2)
plt.ylabel('Trap Times Density')
plt.xlabel('Trap Times')
plt.title('Trap Times Densities | 500x500 grid |')
plt.tight_layout()
plt.legend()
plt.show()
```

και το γράφημα:



Σχήμα 3.8: Κατανομή πυκνότητας χρόνων διαφυγής για πλέγμα 500x500 με πυκνότητα παγίδων c=0.01 και c=0.001.

Το διάγραμμα αυτό σχεδιάστηκε έμμεσα καθώς εξάγαμε τιμές μιας αριθμητικής προσέγγισης της κατανομής των χρόνων παγίδευσης. Γι' αυτό και μερικές φορές σε αυτά τα διαγράμματα παρατηρούμε μικρές παρεκκλίσεις από αυτό που θα περιμέναμε. Για παράδειγμα βλέπουμε τα displot στην αρχή του άξονα να μοιάζουν σαν να παίρνουν αρνητικές τιμές. Προφανώς είναι αδύνατο να υπάρξει αρνητικός χρόνος διαφυγής. Αυτό συμβαίνει γιατί η βιβλιοθήκη matplotlib κάνει fitting τα δεδομένα σε συγκεκριμένες κατανο-

μές φιξάροντας της παραμέτρους ώστε να αναπαράγουν τα δεδομένα μας. Αποτέλεσμα αυτού, μερικές φορές να υπάρχει μικρές αποκλίσεις από τα πραγματικά νούμερα. Πρόκειται δηλαδή για καθαρά υπολογιστικό λάθος και όχι για κάτι με φυσική φαινομενολογία.

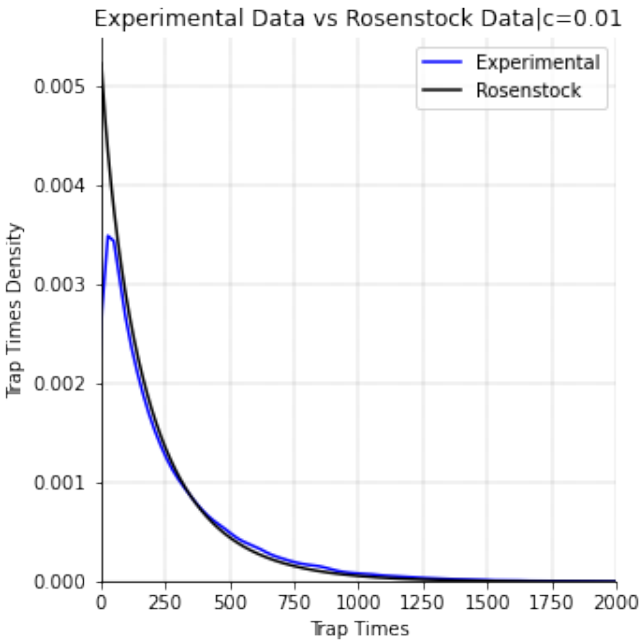
3.4.4 Πειραματική επιβεβαίωση της προσέγγισης Rosenstock

Τελευταίο βήμα στη μελέτη των χρόνων διαφυγής αποτελεί η σύγκριση με την θεωρητική προσέγγιση Rosenstock για να ελέγξουμε πειραματικά την εγκυρότητά της. Σχεδιάζουμε λοιπόν σε κοινό διάγραμμα για τις δύο διαφορετικές συγκεντρώσεις την πειραματική τιμή που υπολογίσαμε παραπάνω, και την καμπύλη που προκύπτει από την Rosenstock για δύο διαστάσεις. Για συγκέντρωση c=0.01 ο κώδικας:

```
n = np.arange(1,5000,1)
s_n = [(1-0.01)**((np.pi*n)/np.log(n)) for n in n]
norm_const1 = np.sum(s_n)
s_n = [x/norm_const1 for x in s_n]

sns.displot(data1, kind="kde", color = "blue", label = 'Experimental')
plt.plot(n,s_n,color='black',label = 'Rosenstock')
plt.xlim(0,2000)
plt.grid(color='0.25', linestyle='--', linewidth=0.2)
plt.ylabel('Trap Times Density')
plt.xlabel('Trap Times')
plt.title('Experimental Data vs Rosenstock Data|c=0.01')
plt.legend()
plt.tight_layout()
plt.show()
```

όπου στο πρώτο κομμάτι του κώδικα κατασκευάζουμε τις τετμημένες (s_n) και τις τεταγμένες (n) της θεωρητικής προσέγγισης, κανονικοποιώντας την κατανομή s_n για να είναι συμβατή με την κανονικοποίηση του displot και την φόρμουλα (2.15).

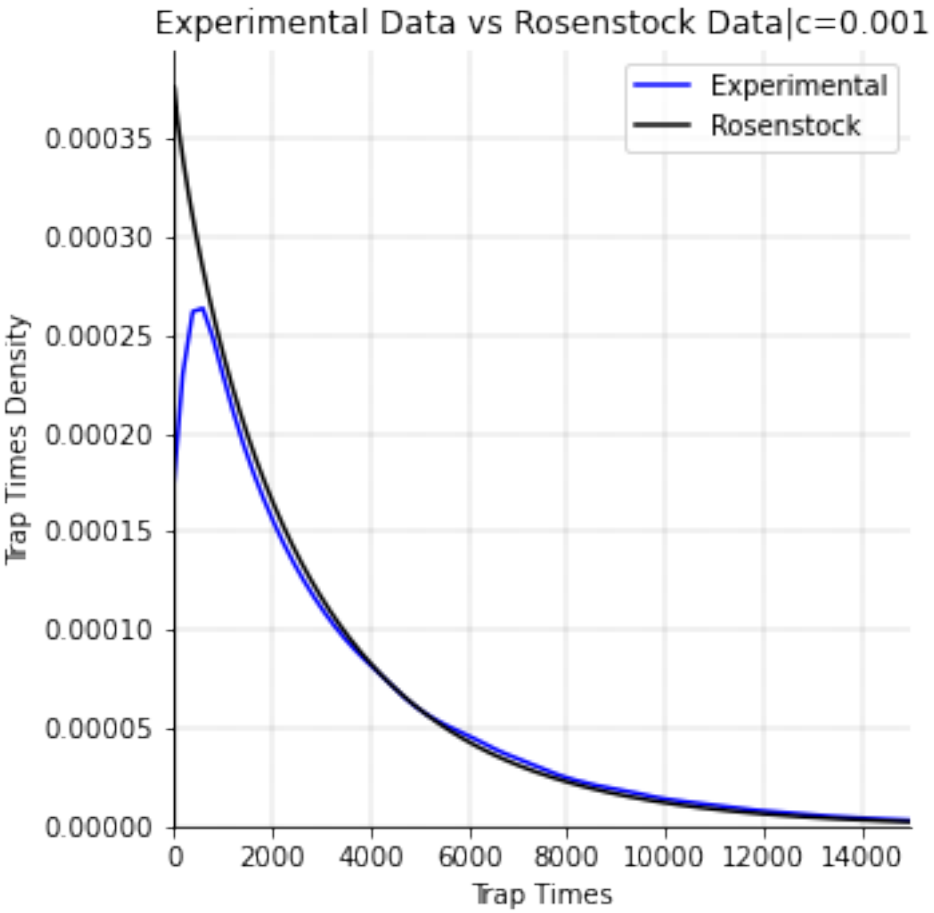
To output:



Ομοίως για συγκέντρωση c=0.001:

```
n = np.arange(1,30000,1)
s_n = [(1-0.001)**((np.pi*n)/np.log(n)) for n in n]
norm_const1 = np.sum(s_n)
s_n = [x/norm_const1 for x in s_n]

sns.displot(data2, kind="kde", color = "blue", label = 'Experimental')
plt.plot(n,s_n,color='black',label = 'Rosenstock')
plt.xlim(0,15000)
plt.grid(color='0.25', linestyle='--', linewidth=0.2)
plt.ylabel('Trap Times Density')
plt.xlabel('Trap Times')
plt.title('Experimental Data vs Rosenstock Data|c=0.01')
plt.legend()
plt.tight_layout()
plt.show()
```

To output:


Experimental Data vs Rosenstock Data|c=0.001
Experimental
Rosenstock
Trap Times Density
0.00035
0.00030
0.00025
0.00020
0.00015
0.00010
0.00005
0.00000
0
2000
4000
6000
8000
10000
12000
14000
Trap Times

Κεφάλαιο 4

Συμπεράσματα

- Από το 3.1.1 βρίσκουμε το αναμενόμενο από την θεωρία μας. Ο μέσος όρος των τετραγωνικών μετατοπίσεων των 100000 μονοδιάστατων περιπάτων, είναι περίπου ίσος με τον αριθμό των βημάτων του κάθε περιπάτου, εφόσον έχουμε βήμα ίσο με 1.
- Από το 3.1.2 βρίσκουμε το αναμενόμενο από την θεωρία μας. Ο μέσος όρος των τετραγωνικών μετατοπίσεων των 100000 δισδιάστατων περιπάτων, είναι περίπου ίσος με τον αριθμό των βημάτων του κάθε περιπάτου, εφόσον έχουμε βήμα ίσο με 1.
- Από το 3.2 βρίσκουμε ότι ανεξάρτητα από τον τρόπο που διαμερίζουμε τον χώρο, ο μέσος όρος των τετραγωνικών μετατοπίσεων παραμένει περίπου ίσος του αριθμού των βημάτων. Επίσης, από τον σχεδιασμό της ευθείας ελαχίστων τετραγώνων, παρατηρούμε ότι τα οι μέσες τετραγωνικές μετατοπίσεις είναι συνεπής με τη θεωρία μας, είδη από τα 100 βήματα. Η σχέση των βημάτων με τις τετραγωνικές μετατοπίσεις φαίνεται να είναι γραμμικές, μιας και τα error της ευθείας ελαχίστων τετραγώνων φαίνονται εξαιρετικά μικρά.
- Από το 3.3.1 παρατηρούμε ότι η καμπύλη S_n συναρτήσει του n που σχεδιάσαμε (έστω και ανά 100 βήματα) είναι σε σύμβαση με τη θεωρητική της μορφή (2.9).
- Από το 3.3.2 παρατηρούμε ότι η καμπύλη S_n συναρτήσει του n που σχεδιάσαμε (έστω και ανά 100 βήματα) είναι σε σύμβαση με τη θεωρητική της μορφή (2.10).
- Από το 3.4.4 παρατηρούμε ότι από τα 1000 περίπου βήματα και μετά, η θεωρητική προσέγγιση Rosenstock αναπαράγει πολύ καλά τα πειραματικά μας αποτελέσματα. Εφόσον ο τύπος της κατανομής προκύπτει από πιθανοκρατικές υποθέσεις και υπολογισμό μέσων όρων είναι λογικό. Σύμφωνα με το νόμο τον μεγάλων αριθμών, όσο περισσότερα βήματα κάνω τόσο καλύτερα θα προσεγγίζονται οι στατιστικές μου προβλέψεις.

Bibliography

[1] Z. Bar-Yossef and M. Gurevich. Random sampling from a search engine's index. *Journal of the ACM (JACM)*, 55(5):1–74, 2008.

[2] L. K. Gallos, P. Argyrakis, and K. W. Kehr. Trapping and survival probability in two dimensions. *Physical Review E*, 63(2):021104, 2001.

[3] D. A. Kodde and H. Schreuder. Forecasting corporate revenue and profit: Time-series models versus management and analysts. *Journal of Business Finance & Accounting*, 11(3):381–395, 1984.

[4] E. W. Montroll and G. H. Weiss. Random walks on lattices. ii. *Journal of Mathematical Physics*, 6(2):167–181, 1965.

[5] J. Nelson. Continuous-time random-walk model of electron transport in nanocrystalline tio 2 electrodes. *Physical Review B*, 59(23):15374, 1999.

[6] R. M. Nosofsky and T. J. Palmieri. " an exemplar-based random walk model of speeded classification": Correction to nosofsky and palmeri (1997). 2008.

[7] H. B. Rosenstock. Random walks on lattices with traps. *Journal of Mathematical Physics*, 11(2):487–490, 1970.