НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Фізико-математичний факультет

Кафедра математичного аналізу та теорії ймовірностей

«На правах рукопису»
УДК _____

До захисту допущено:
Завідувач кафедри
Клесов Олег Іванович
«___»_____20__ р.

Магістерська дисертація

на здобуття ступеня магістра

за освітньо-науковою програмою «Страхова та фінансова математика**»**

зі спеціальності 111 «Математика»

на тему: «Побудова та статистичний аналіз ГПВЧ на основі фракталів**»**

Виконав (-ла):
студент (-ка) ІІ курсу магістратури, групи ОМ-91мн
Манич Іван Олегович _____

Науковий керівник:
Доктор фізико-математичних наук
Клесов Олег Іванович _____

Рецензент:
Завідувач кафедри інформаційних систем і технологій Національного транспортного університету, доктор фіз.-мат. наук, професор Гавриленко Валерій Володимирович

_____

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент (-ка) _____

Київ – 2021 року

ЗМІСТ

ВСТУП

У даній роботі розглянута проблема необхідності випадкових чисел у сучасних галузях, що використовують прикладну математику. Дана робота є дослідженням нових методів для постійного генерування псевдовипадкових послідовностей довільної довжини. Методи, розглянуті у роботі передбачають можливість генерування дихотомічних псевдовипадкових за допомогою побудов фрактальних структур.

Одними з головних об’єктів дослідження є фрактали. В ході роботи будуть запропоновані їх нові прикладні значення, розглянуті різні методи взаємодії з ними, підходи до їх програмної реалізації.

Актуальність даної теми пов’язана з великим попитом на генератори псевдовипадкових чисел, які відповідають загальновизнаним стандартам, таким як NIST, тести Diehard, частотним перевіркам та багатьом іншим. Попит викликаний широким спектром застосування псевдовипадкових вибірок у найрізноманітніших сферах, таких як Методи Монте-Карло, криптографія, імітаційне моделювання, тощо.

РОЗДІЛ 1: МЕТОДИ ПОБУДОВИ ФРАКТАЛЬНИХ СТРУКТУР

1.1 Інстременти, що були використані для дослідженні

Побудова фракталів буде здійснбватись програними засобами. Для побудови самоподібних фігур у роботі буде використано наступні інструменти:

- IDE Pycharm (середовище розробки для мови програмування Python)
- Інтерпритатор Python v3.9
- Бібліотека numpy
- Бібліотека matplotlib
- Бібліотека math
- Бібліотека turtle
- Бібліотека svg_write

Використання саме цього набору інструментів обумовлено декількома важливими факторам, такими як:

- Легкочитаємий код
- Швидкість роботи
- Зручність візуалізації
- Наявність великої кількості бібліотек для математичних операцій
- Гнучкість у методах виводу інформації
- Зручність перевірки правильності обчислень
- Наявність навчальної інформації по використанню інструментів.

1.2 L-системи для побудови самоподібних фігур

Побудова самоподібних фігур – це процес, до якого існує немало підходів. Одним з найбільш зручних, наглядних та дієвих, на мій погляд, є так звані L-системи або системи Ліндемайєра, що визначається, як «система переписування» або «вигляд формальної граматики». Щоб пояснити, що це значить, розглянемо сам хід побудови

деякого фрактала, наприклад, трикутника Серпінського. Для того, щоб побудувати данну фігуру на n-й ітерації, необхідно виконати наступні дії:

1. Побудувати трикутник
2. З'єднати середини його сторін відрізками
3. Розглянути утворені цими сторонами трикутники, крім середнього.
4. Для кожного з них виконати кроки 1-4 n-1 разів.

Якби ми не скористались формальним описом рекурсії у кроці 4, нам необхідно було би прописати всі 3n правил. L-система так само пропонує закодувати перелік необхідних дій у вигляді так званої «аксіоми», які являють собою звичайні символи англійсьокого алфавіту та математичні символи. Після чого за допомогою «породжуючих правил» перетворити аксіому на повний перелік операцій, необхідних для побудови фрактала на певній ітерації.

Формально L-системи можна визначити як кортеж $G = (V, \omega, P)$, де

- *V (алфавіт)*- множина символів, які будуть замінені (змінні) та не будуть замінені (константи) – при використанні породжуючих правил.
- *ω (аксіома)* – початковий рядок, складений з символів, що належать *V*, що визначає початковий стан системи.
- *P* - множина «породжуючих правил», що визначають, який вигляд прийме аксіома для того, щоб стати правилом побудови фрактала до n-ї ітерації. Для прикладу, якщо правило вказує, що символ «A» аксіоми, має перейти у набір символів «AB», то аксіома «A» після 3 ітерацій претвориться на рядок правил «ABBB».

РОЗДІЛ 2: ВИКОРИСТАННЯ ФРАКТАЛІВ ДЛЯ ГПВЧ

2.1 Вибір фракталів для дослідження

Розглядання фракталів у якості засобів для генерування псевдовипадкових чисел не є абсолютно новою ідеєю, проте, зазвичай розглядають фрактали розмірності 2 та більше. Я вирішив підійти до цього питання з іншого боку та почати розглядати можливість використання найпростіших, з точки зору відтворення, фракталів та подібних їм структур у якості ГПВЧ. Я пропоную розглянути Трикутник Серпінського та його видозмінену версію.

Для початку визначимось, що може слугувати джерелом для псевдовипадкових чисел при формуванні зображення деякої ітерації фрактала? У даній роботі описана ідея розглядати у якості такого джерела координати точки на деякому кроці побудови зображення.

2.2. Зображення фракталів за допомогою L-системи

Підхід, запропонований у попередньому пункті роботи, відображав набір правил, які, зазвичай, викладаються на лекціях, коли починають вивчати фрактали. Він є достатньо легким для розуміння студентами, проте запрограмувати Трикутник Серпінського на порядок легше, якщо починати описувати його не з великого трикутника, а з найменших трикутників на n-й ітерації. По суті, для зображення такої фігури нам необхідний набір з трьох дій, які позначимо відповідними символами:

1. Малювання відрізку декої довжини «F»
2. Поворот напрямку «пера» на 120 градусів за годинниковою стрілкою «+»
3. Поворот напрямку «пера» на 120 градусів проти годинникової стрілки «-»

Зауважу, що малювання ліній на площині за допомогою бібліотеки turtle починається горизонтально зліва направо. Враховуючи даний набір правил, легко бачити, що для зображення звичайного рівностороннього трикутника можна

скористатись правилом «F+F+F». Шляхом аналізу та експериментів, було з’ясовано, що для зображення Трикутника Серпінського добре підходить паттерн «F+F-F-F-F+F». Якщо замість побудови сторін (простого зміщення «пера» для зображення відрізку), малювати цей паттерн, то ми отримаємо першу ітерацію зображення Трикутника Серпінського. Тобто, вводимо «породжуюче правило» L-системи: змінну «F» замінити на рядок «F+F-F-F-F+F». Проте, виникає проблема: якщо для першої ітерації цей варіант підходить, то для другої та подальших підстановок ми продовжимо маливати першу ітерацію, оскільки ми будемо продовжувати з тієї ж точки, з якої почали. Найбільш елегантним та правильним рішенням у даному випадку буде зміна аксіоми таким чином, щоб ми повертались не у початкову точку, а у іншу вершину зображеного трикутника. Дійсно, зміна аксіоми на «F+F+F+F» допомогла і тепер ми можемо спостерігати, як виглядають зображення декількох перших ітерацій нашої системи:

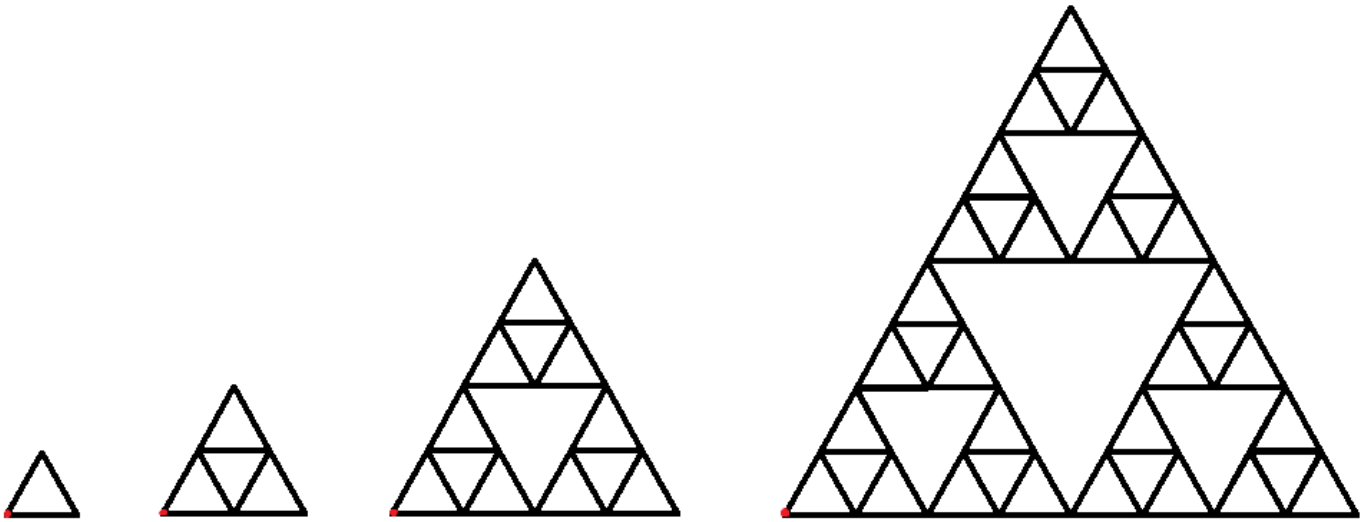


Рисунок 1 зображення перших 4 ітерації заданої L-системи

Таким чином можемо написати функцію, яка при виклику буде малювати даний фрактал. Аргументом даної функції вкажемо необхідну кількість ітерації для побудови. В загальному код буде виглядати наступним чином:

```
def Serpinskiy(countIteration):
    axiom = "F+F+F+F" #Початкова аксіома
    axmTemp = "" #Змінна для перетворення аксіоми
    #Набір породжуючих правил
    translate = {"+":"+","-":"-","F":"F+F-F-F+F"}

    #Цикл створення набору правил для побудови
    for k in range(countIteration):
        for ch in axiom:
            axmTemp += translate[ch]
        axiom = axmTemp
        axmTemp = ""

    #Цикл, у якому будується Трикутник Серпінського
    for ch in axiom:
        if ch == "+":
            turtle.left(120) #Поворот додатний на 120 градусів
        elif ch == "-":
            turtle.right(120) #Поворот від'ємний
        else:
            turtle.forward(50)

    turtle.update()
    turtle.mainloop()
```

2.3 Визначення формул для задання координат пера

2.3.1 Визначення загальних формул при сталих кутах

Ще однією перевагою даного методу кодування послідовності дій, необхідних для зображення самоподібних фігур, є те, що при цьому неважко визначити формулу, яка задає координати точки зображення на певному кроці. Покажемо, як це визначається.

Визначимо, як змінюються координати пера по осі абсцисс. Перо може зсунутись прямо за початковим напрямком (додатний напрямок осі абсцисс), а далі рухатись під кутом від цього напрямку. Позначимо довжину відрізка, яку малює перо через літеру d. Це значить, що рухаючись під кутом α від осі абсцисс, абсцисса пера буде зсуватись на $d\cos\ \alpha$ при додатньому повороті та на $-d\cos\alpha$ при від'ємному.

Порядок виконення поворотів у додатному та від'ємному напрямках є заданим L-системою, тому для ясності поглянемо на цей порядок на різних ітераціях:

Ітерація 1: $+\ -\ -\ +\ +\ +\ -\ -\ +\ +\ +\ -\ -\ +\ +\ +\ -\ -\ +$

Ітерація 2: $+\ -\ -\ +\ +\ +\ -\ -\ +\ -\ +\ -\ -\ +\ -\ +\ -\ -\ +\ +\ +\ -\ -\ +\ +\ +\ -\ -\ +\ +$
$+\ -\ -\ +\ -\ +\ -\ -\ +\ -\ +\ -\ -\ +\ +\ +\ -\ -\ +\ +\ +\ -\ -\ +\ +\ +\ -\ -\ +\ -\ +\ -\ -\ +\ -\ +\ -\ -$
$+\ +\ +\ -\ -\ +\ +\ +\ -\ -\ +\ +\ +\ -\ -\ +\ -\ +\ -\ -\ +\ -\ +\ -\ -\ +\ +\ +\ -\ -\ +$

...

При більш детальному погляді можна побачити, що завжди діє наступне правило: знаки, позиції котрих діляться націло на 5 у степені заданої ітерації – завжди є плюсами. Знаки, позиції яких при діленні на $5^1, 5^2, 5^3\ \ldots 5^i$ дають в остачі 1,4,6,9 – теж є знаками «+», інші – «−». Отже, можемо задати послідовність коефіцієнтів

$$c = \{c_j : j \geq 1\}$$

Введемо наступні позначення для $m_k -$ остача від ділення $\ j\ $ на 5^k

$$c_j = \begin{cases} 1, \min\limits_k m_k \in \{0; 1; 4; 6; 9\} \\ -1, \min\limits_k m_k \in \{2; 3; 7; 8\} \end{cases}$$

В такому випадку формула для координати абсцисси пера на i-му кроці буде мати наступний вигляд

$$x_i = d \sum_{j=0}^{i} c_j \cos \alpha j$$

Очевидно, що аналогічними міркуваннями можна прийти до задання ординати пера на i-му кроці:

$$y_i = d \sum_{j=0}^{i} c_j \sin \alpha j$$

Знехтуємо тим, що генерація псеводивадкових чисел може здійснюватись тільки до деякої ітерації та розглянемо теоретичний варіант, коли $n \to \infty$. При цьому легко бачити, що значення x_i та y_i будуть повторюватись, проте не з сталим періодом. Це пояснюється тим, що тригонометричні функції – періодичні і $\cos \alpha_j$ та $\sin \alpha_j$ будуть приймати тільки 3 різних значення при $\alpha = 120°$, але через деяку кількість кроків відбувається зсув для повторення заданого паттерну. Для наглядності наведу приклад зміни значеннь x_i та y_i для декількох кількостей ітерацій. Дані графіки побудовані за допомогою бібліотеки matplotlib.

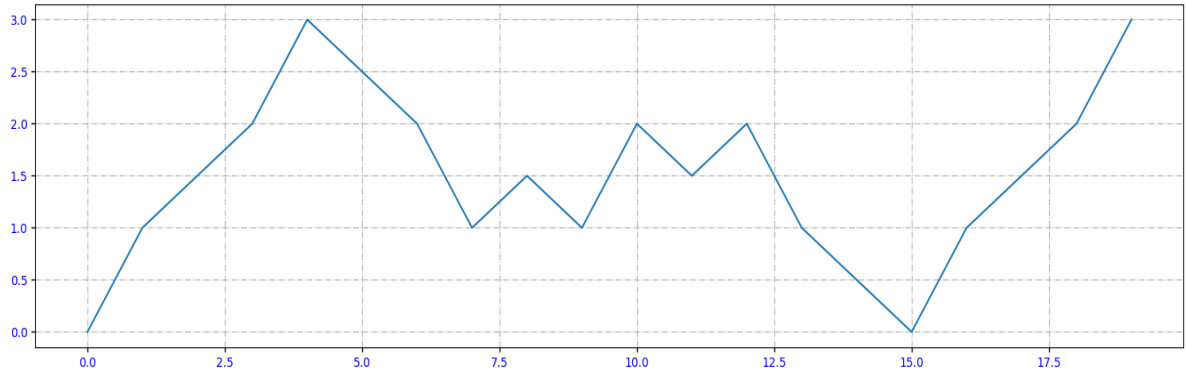


Рисунок 2 зміна абсцисси пера для малювання 1-ї ітерації Трикутника Серпінського

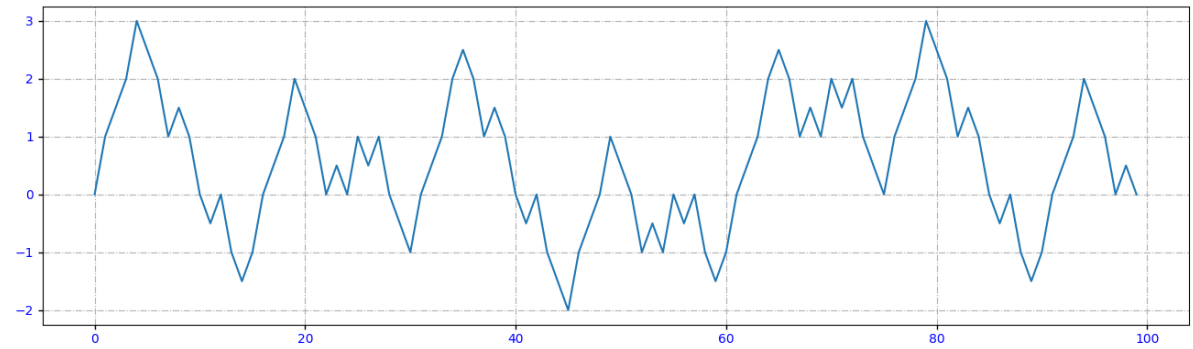


Рисунок 3 зміна абсцисси пера для малювання 2-ї ітерації Трикутника Серпінського

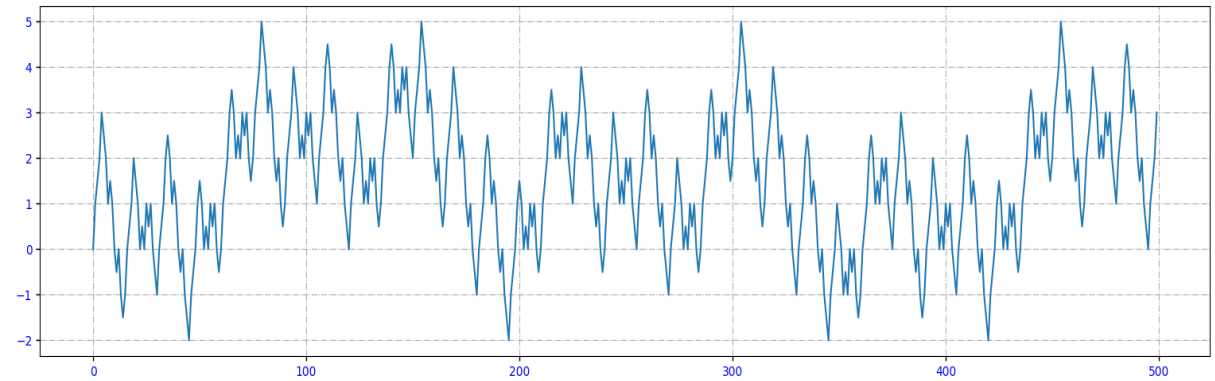


Рисунок 4 зміна абсцисси пера для малювання 3-ї ітерації Трикутника Серпінського

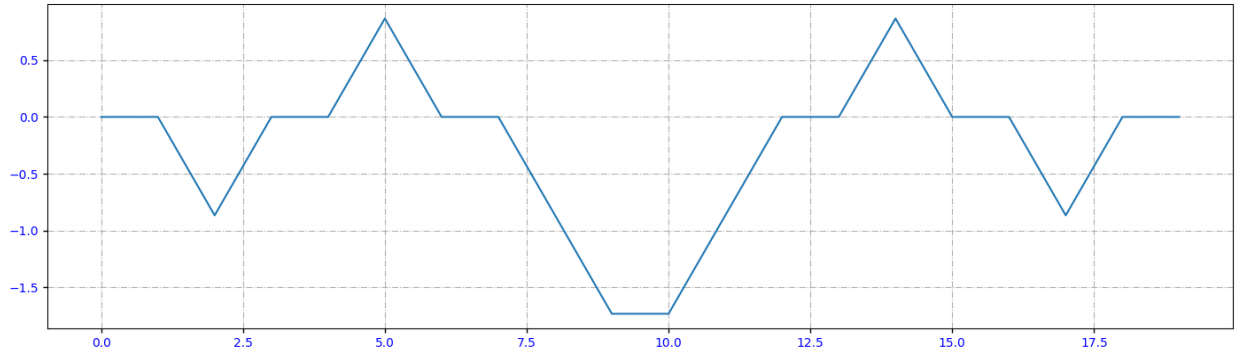


Рисунок 5 зміна ординати пера для малювання 1-ї ітерації Трикутника Серпінського

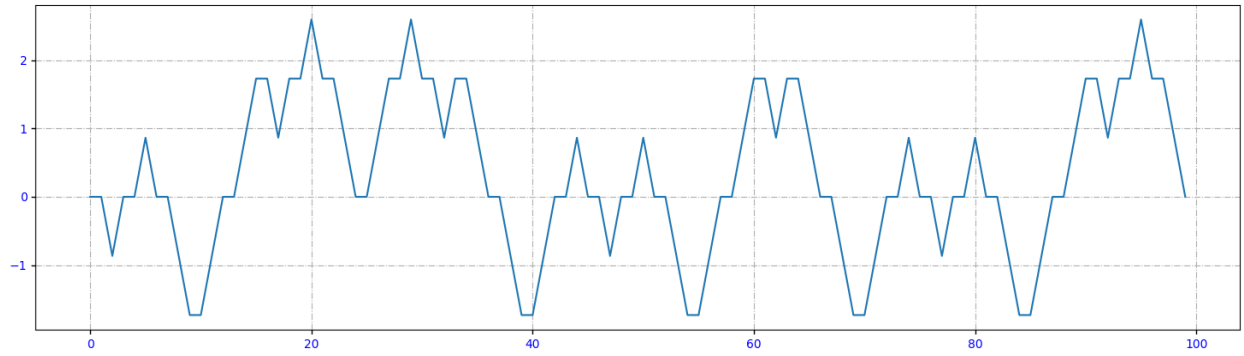


Рисунок 6 зміна ординати пера для малювання 2-ї ітерації Трикутника Серпінського

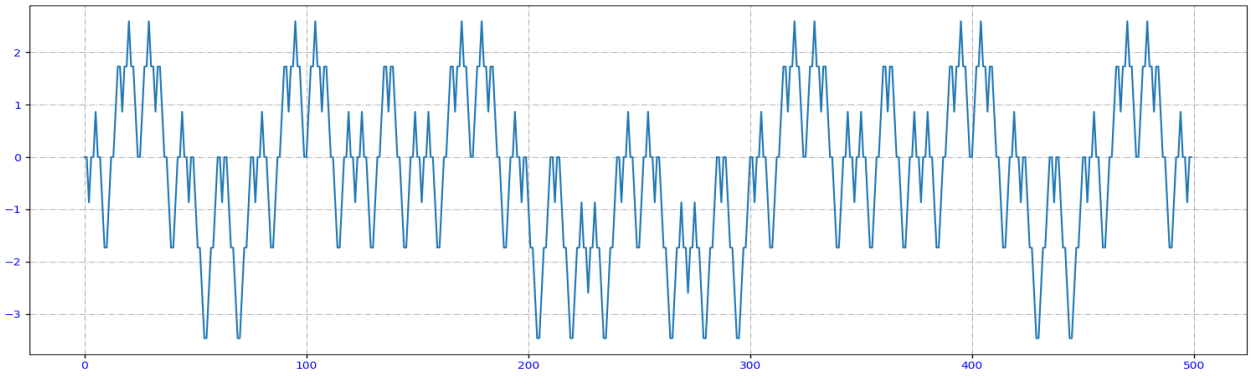


Рисунок 7 зміна ординати пера для малювання 3-ї ітерації Трикутника Серпінського

2.3.2 Рішення проблеми періодичності

Для уникнення даної проблеми є декілька рішень. Одне з яких: змінити кут нахилу у додатному (від'ємному) напрямку на деяке достатньо невелике ірраціональне число, наприклад, $ln2$. Як наслідок, отримане зображення з кожним збільшенням кількості ітерацій буде все більше і більше відхилятись від зображення трикутника Серпінського, проте, нагадаємо, що основна задача полягає не у зображенні фрактала, а у використанні його як джерела для псевдовипадкових чисел. Дана модифікація дозволяє позбутись ознак періодичності, тому врахуємо її для розрахунку кута повороту пера α_j.

$$\alpha_j = \begin{cases} \dfrac{120 + ln2}{180}\pi, & \min\limits_k m_k \in \{0; 1; 4; 6; 9\} \\ -\dfrac{3}{2}\pi, & \min\limits_k m_k \in \{2; 3; 7; 8\} \end{cases}$$

2.4 Побудова ГПВЧ на основі зазначеної L-системи

2.4.1 Визначення ГПВЧ та зерна

Для детермінованих ГПВЧ характерна наступна структура, запропонована П. Лекуером $(S, \mu, f, \mathcal{U}, g)$. S – кінцевий набір станів, μ – ймовірнісний розподіл у просторі станів S, $f: S \to S$ – функція переходу, $\mathcal{U}$ – простір вихідних значень, $g: S \to \mathcal{U}$ – функція вихідних значень. Стани, зазвичай задаються через рекурентну формулу: $s_i = f(s_{i-1})$. Вихідним значенням генератора псевдовипадкових чисел для $i \geq 1$ називається число $u_i = g(s_i) \in \mathcal{U}$. Для ГПВЧ необхідне також значення s_0, через який задано початковий стан. Таке число називається зерном (англ. seed) та дозволяє відтворити задану послідовність псевдовипадкових чисел. Зауважимо, що можливість відтворити задану одного разу послідовність псевдовипадкових чисел є невід'ємною властивістю саме для генератора псевдовипадкових чисел. Генератори, які дають на виході послідовності, що неможливо повторити, називаються істинними генераторами

випадкових чисел. Зазвичай, вони можуть базуватись на ГПВЧ, якому у якості зерна буде передаватись деяке значення, отримане з фізичних процесів (джерел ентропії).

Питання про те, яке зерно використовувати у ході роботи з Трикутником Серпінського зводиться до наступних пропозицій:

- Змінювати довжину відрізків
- Змінювати градусні міри кутів

Моя пропозиція полягає у постійній зміні кута на величину між 0 та 1 градусом. Звісно, така умова є опціональною і в якості зерна можна брати довільне число. Таким чином, у якості зерна можна приймати довільне дійсне число.

2.4.2 Зміна формул з урахуванням відхилення та зерна

Ці корективи треба внести у формулу для відображення того, як будуть змінюватись координати пера. Нехай зміна буде відбуватись тільки для кута нахилу у додатньому напрямку.

Введемо змінну s для зерна. $s \in \mathbb{R}_+$. Також врахуємо постійне відхилення на ln2 при повороті у додатньому напрямку і α_j задана у радіанах наступним чином:

$$\alpha_j = \begin{cases} \dfrac{120 + ln2 + s}{180}\pi, \min\limits_k m_k \in \{0; 1; 4; 6; 9\} \\ -\dfrac{3}{2}\pi, \quad \min\limits_k m_k \in \{2; 3; 7; 8\} \end{cases}$$

$$x_i = d\sum_{j=0}^{i}\left(c_j \cos\left(\sum_{k=0}^{j}\alpha_j\right)\right)$$

$$y_i = d\sum_{j=0}^{i}\left(c_j \sin\left(\sum_{k=0}^{j}\alpha_j\right)\right)$$

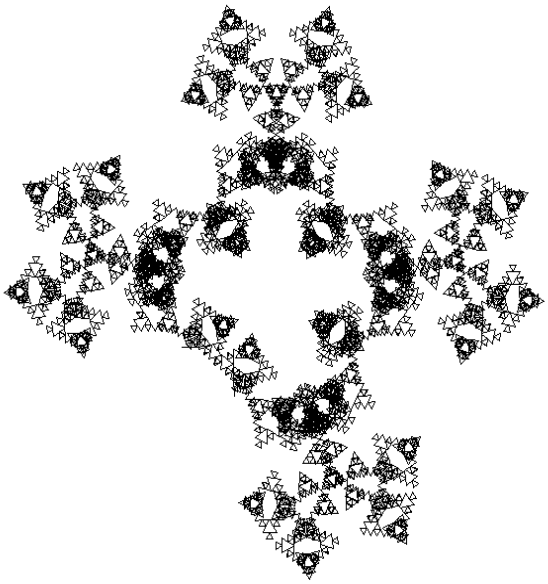
Внесемо дані корективи у програму: додамо ще один аргумент функції Serpinskiy – seed. Та перепишемо цекл наступним чином:

```
seed = seed/1000
deflection = math.log(2, math.e)
for ch in axiom:
    if ch == "+":
        turtle.left(120 + deflection + seed)
    elif ch == "-":
        turtle.right(120)
    else:
        turtle.forward(7)
```

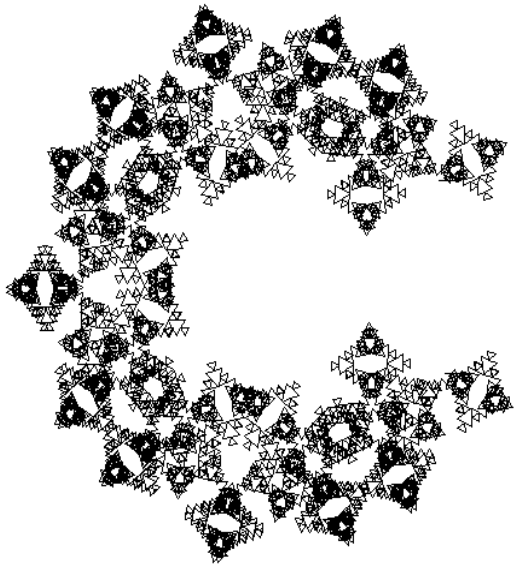
Тепер допишемо у кінець циклу запис значень координат у змінну. Шляхом дослідження даного методу було виявлено наступну послідновність дій, яка є досатньо ефективною для генерації псевдовипадкових чисел: При побудові самоподібної фігури обираємо 10-20 значень координати x або y. Переведемо їх модулі у двійкову систему числення та запишемо у рядок. Отримаємо деяку дихотомічну послідовність, яку надалі можна використовувати для генерації наступних дихотомічних послідовностей у якості зерна, просто дописавши попереду «0.» та перевівши число назад у десяткову систему. Для запису 20 значень координат пера у змінну та поверненню їх при виклику функції, об'явимо 2 нових змінних для координат та 1 для рахування ітерацій циклу зображення у початку функції Serpinskiy:

```
x = y = []
iteration = 0
   if (iteration % (len(axiom)//20) == 0):
       x.append(turtle.pos()[0])
       y.append(turtle.pos()[1])
   iteration += 1
```

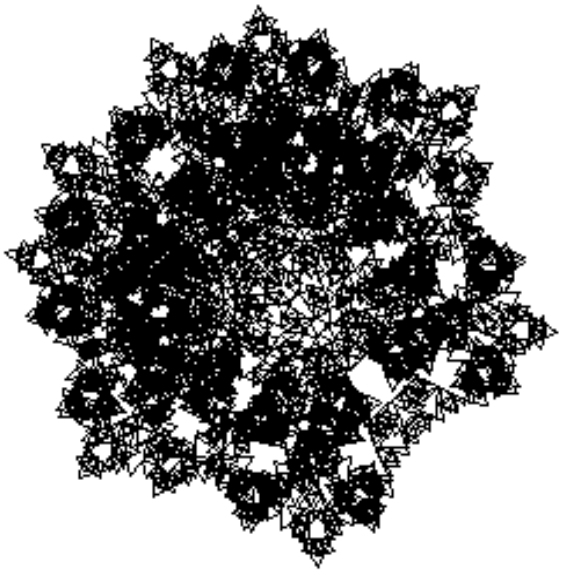
Наведемо приклади того, що виходить при різних значеннях seed. Для наглядності візьмемо 5 ітерацій



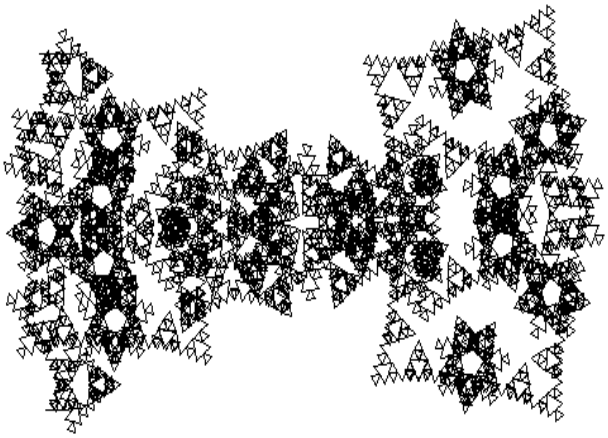
а)



б)



в)



г)

Рисунок 8 зміна вигляду заданого фракталу при значенні зерна

а)101 б)203 в)300 г)-203

2.5 Перевірка отриманих дихотомічних послідовностей псевдовипадкових чисел

2.5.1 Базова перевірка кількості серій

Введемо наступні гіпотези:

Нульова гіпотеза H_0 – дихотомічна послідовність є випадковою.

Альтернативна гіпотеза H_1 – дихотомічна послідовність не є випадковою.

Задамо рівень значущості гіпотези $\alpha = 0.05$.

Для перевірки гіпотез необхідно проверсти ряд розрахунків для визначення таких характеристик дихотомічної послідовності, як:

- Загальна кількість елементів
- кількість одиниць
- кількість нулів
- кількість 1-серій
- кількість 0-серій
- загальна кількість серій
- Ймовірності вибірок з різними кількостями серій
- Значення меж області відхилення нульової гіпотези $\{k_1, k_2\}$

Отже, виходячи з того, що нульову гіпотезу відхиляємо при кількості серій R, що ймовірність $P(k_1 < R < k_2) \geq \alpha$, напишемо функцію-перевірку, що буде проводити всі необхідні розрахунки та запустимо цю перевірку для 100 різних значень зерна.

2.5.2 Кодування перевірки гіпотези випадковості

Для перевірки отриманих даних напишемо функцію, в якій реалізуємо алгоритм розрахунку ймовірності $P(k_1 < R < k_2)$. В кінці алгоритму порівнюємо значення R з k_1 та k_2 й повертаємо результат у вигляді булевого значення (true або false). Таким

чином буде зручно обробити результати після прохождення великої кількості перевірок. Ці значення запишемо у масив.

```
def checkPRN(prn):
    dht = toDikhotomic(prn)
    print("Медіана = ", statistics.median(prn))
    print("Дихотомічна послідовність: ", dht)
    k1 = k2 = n0 = n1 = R = R0 = R1 = 0

    for i in dht:
        if i == 0: n0+=1
        else: n1+=1

    if dht[0] == 0: R0 = 1
    else: R1 = 1

    for i in range(len(dht)-1):
        if dht[i] != dht[i+1]:
            if dht[i+1] == 0: R0 += 1
            else: R1 += 1

    R = R0 + R1
    print("кількість нулів: ", n0)
    print("кількість нулів: ", n1)
    print("Кількість 0-серій: ", R0)
    print("Кількість 1-серій: ", R1)
    print("Кількість серій загалом: ", R)
```

```
#Оберемо рівень значущості alpha = 0.05, P(R<=k1) = P(R>=k2) = 0.025
    alpha = 0.05
    P = []
    for i in range(len(dht)):
        P.append(countProbability(n0,n1,i+1))
    sumP = 0
    sumN = 0
    for i in range(len(P)):
        sumP += P[i]
        if sumP > alpha/2:
            k1 = i
            break

    for i in range(len(P)):
        sumN += P[len(P) - 1 - i]
        if sumN > alpha / 2:
            k2 = len(P)-1-i
            break
    print(" k1 = ", k1)
    print(" k2 = ", k2)

    if R > k1 and R < k2:
        print(" k1 < R < k2 – гіпотезу H0 не відхиляємо")
        return True
    else:
        print(" R >= k2 або R <= k1 - гіпотезу H0 відхиляємо ")
        return False
```

2.5.3 Критерій Колмогорова-Смірнова

Критерій узгодженості Колмогорова-Смірнова, взагалі кажучи, використовується для порівняння двох емпіричних розподілів. В нашому випадку для оцінки рівномірності розподілу згенерованих псевдовипадкових чисел будемо генерувати рівномірно розподілу вибірку необхідної довжини одразу під час порівняння. Для перевірки даним критерієм напишемо функцію, що буде розраховувати та повертати значення λ.

В залежності від отриманих значень λ можна буде зробити висновок, чи можемо ми відхилити гіпотезу про рівномірний розподіл згенерованої послідовності псевдовипадкових чисел з відповідним рівнем значущості.

α	0,1	0,05	0,01
λ	1,22	1,36	1,63

Таблиця 1 Критичні значення λ критерію Колмогорова-Смірнова для відповідних рівней значущості α.

```
def Kolmogorov(x):
    z = []#< - c()
    x = sorted(x)#< - sort(x)

    for k in range(len(x)):#1:n)
        a1 = x[k] - k/ len(x)
        a2 = (k+1)/ len(x) - x[k]
        z.append(max(a1, a2))
```

```
m = max(z)

Lambda = m * len(x) ** 0.5


print('D (statistics) = ', m)

print('Lambda = ', Lambda)


return Lambda
```

2.5.4 Критерій ω^2

Критерій омега-квадрат, також відомий як критерій фон Мізеса або критерій Крамера-Мізеса-Смірнова є одним з найбільш популярних для перевірки гіпотези про те, що дейска дискретна випадкова величина підпадає під деякий закон розподілу. Перевірка може проводитись для довільного виду розподілу.

Критерій вважається достатньо надійним для вибірок, кількість елементів яких не менша за 15. Основною ідеєю методу є оцінка, що базується на сумі квадратів різниць між вибіркою, що перевіряється та теоретичною вибіркою обраного розподілу. Статистика критерію ω^2 рахується за формулою

$$\omega^2 = \left(\frac{1}{12n} + \sum_{i=1}^{n}\left(F(x_i) - \frac{2i-1}{2n}\right)^2 - \frac{0{,}4}{n} + \frac{0{,}6}{n^2}\right)\left(1 + \frac{1}{n}\right),$$

Де $F(x_i)$ – значення теоретичної функції розподілу.

Розраховане значення порівнюють з даними у наступній таблиці. Якщо отримане значення статистики менше за табличне – то гіпотеза про розподіл вибірки не відхиляється при відповідному значенні α.

α	0,1	0,05	0,01
ω^2	0,347	0,461	0,743

Таблиця 2 Критичні значення ω^2 критерію фон Мізеса для відповідних рівней значущості α.

Алгоритм даного тесту також закодуємо для перевірки згенерованих чисел.

```
def omega_sq(x):
    x = sorted(x)
    Fx = 0
    omega_arr = []
    for i in range(len(x)):
        Fx += x[i]
        omega_arr.append((Fx - (2 * (i + 1) - 1) / (2 * len(x))) **
2)

    nw2 = 1/(12*len(x)) + Sum(omega_arr)

    w = - 0.4/len(x) + 0.6/len(x)**2

    omega_square = (1 / (12 * len(x))+Sum(omega_arr) + w) * (1 +
1/len(x)))

    w2 = nw2/len(x)
    print('nw^2 statistics: ', nw2)
    print('w^2 = ', w2)

    return w2
```

2.5.5 Критерій Пірсона χ^2

Критерій χ^2 базується на групуванні даних та аналізі частот на певних інтервалах. Цю перевірку ми проведемо у декілька етапів. Для стандартної перевірки випадковості за критерієм узгодженості Пірсона розіб'ємо проміжок [0;1] на інтервали та визначимо, скільки чисел попадає у кожен з цих інтервалів. Дані значення будуть складати масив частот, який будемо порівнювати з масивом прогнозованих частот. Очевидно, що для рівномірно розподіленої вибірки з n елементів, кількість елементів у кожному з k інтервалів буде $\frac{n}{k}$. Визначимось з тим, на скільки інтервалів оптимально буде робити розбиття області [0;1]. Для цього є наступна формула

$$k = 11 + [\log_2 n], \text{де } [\log_2 n] - \text{ціла частина } \log_2 n$$

Висунувши гіпотезу H_0 про випадковість отриманної вибірки чисел, визначимо кількість чисел у кожному з інтервалів, які позначимо $A_1, \dots, A_k$. Позначимо через $\nu_j, j = 1, \dots, k$ число елементів вибірки, у відповідному проміжку. Позначимо також теоретичну ймовірність потрапляння елементу у інтервал через p_j. Для рівномірного розподілу і однакових розмірів інтервалів $p_j = \frac{1}{k}, \forall j = 1, .., k$. Тепер позначимо характеристику критерію Пірсона

$$\rho(X) = \sum_{j=1}^{k} \frac{\left(\nu_j - np_j\right)^2}{np_j}$$

Визначивши дану характеристику можна легко знайти p-значення за таблицею, врахувавши, що число ступенів свободи буде рівне k-1. Оскільки для тестування згенерованих вибірок число ступенів свободи буде достатньо велике, зручніше буде описати розрахунок p-значення в залежності від кількості елементів. Зручніше буде використати функцію stats.chisquare() з бібліотеки scipy, яка одразу розраховує p-значення.

Критерій Пірсона χ^2

Число ступенів свободи f	Рівень значимості					
	0,01	0,025	0,05	0,95	0,975	0,99
1	6,6	5,0	3,8	0,0039	0,00098	0,00016
2	9,2	7,4	6,0	0,103	0,051	0,020
3	11,3	9,4	7,8	0,352	0,216	0,115
4	13,3	11,1	9,5	0,711	0,484	0,297
5	15,1	12,8	11,1	1,15	0,831	0,554
6	16,8	14,4	12,6	1,64	1,24	0,872
7	18,5	16,0	14,1	2,17	1,69	1,24
8	20,1	17,5	15,5	2,73	2,18	1,65
9	21,7	19,0	16,9	3,33	2,70	2,09
10	23,2	20,5	18,3	3,94	3,25	2,56
11	24,7	21,9	19,7	4,57	3,82	3,05
12	26,2	23,3	21,0	5,23	4,40	3,57
13	27,7	24,7	22,4	5,89	5,01	4,11
14	29,1	26,1	23,7	6,57	5,63	4,66
15	30,6	27,5	25,0	7,26	6,26	5,23
16	32,0	28,8	26,3	7,96	6,91	5,81
17	33,4	30,2	27,6	8,67	7,56	6,41
18	34,8	31,5	28,9	9,39	8,23	7,01
19	36,2	32,9	30,1	10,1	8,91	7,63
20	37,6	34,2	31,4	10,9	9,59	8,26
21	38,9	35,5	32,7	11,6	10,3	8,90
22	40,3	36,8	33,9	12,3	11,0	9,54
23	41,6	38,1	35,2	13,1	11,7	10,2
24	43,0	39,4	36,4	13,8	12,4	10,9
25	44,3	40,6	37,7	14,6	13,1	11,5
26	45,6	41,9	38,9	15,4	13,8	12,2
27	47,0	43,2	40,1	16,2	14,6	12,9
28	48,3	44,5	41,3	16,9	15,3	13,6
29	49,6	45,7	42,6	17,7	16,0	14,3
30	50,9	47,0	43,8	18,5	16,8	15,0

Таблиця 3 Критичні значення статистики критерію Пірсона χ^2 для відповідних рівней значущості α.

Окрім звичайного розбиття проміжку [0;1] на k проміжків проведемо ще декілька незвичайних перевірок. Проаналізуємо не вибірку псевдовипадкових чисел, а вибірку впорядкованих пар, які будемо сприймати як координати точок на площині. Чим ближчий розподіл отриманої послідовності до рівномірного – тим більше буде заповнюватись площа $[0;1] \times [0;1]$ при збільшенні кількості генерованих чисел. Для спостереження даного явища виведемо результати для згенерованих 1000, 10000 та 100000 чисел.

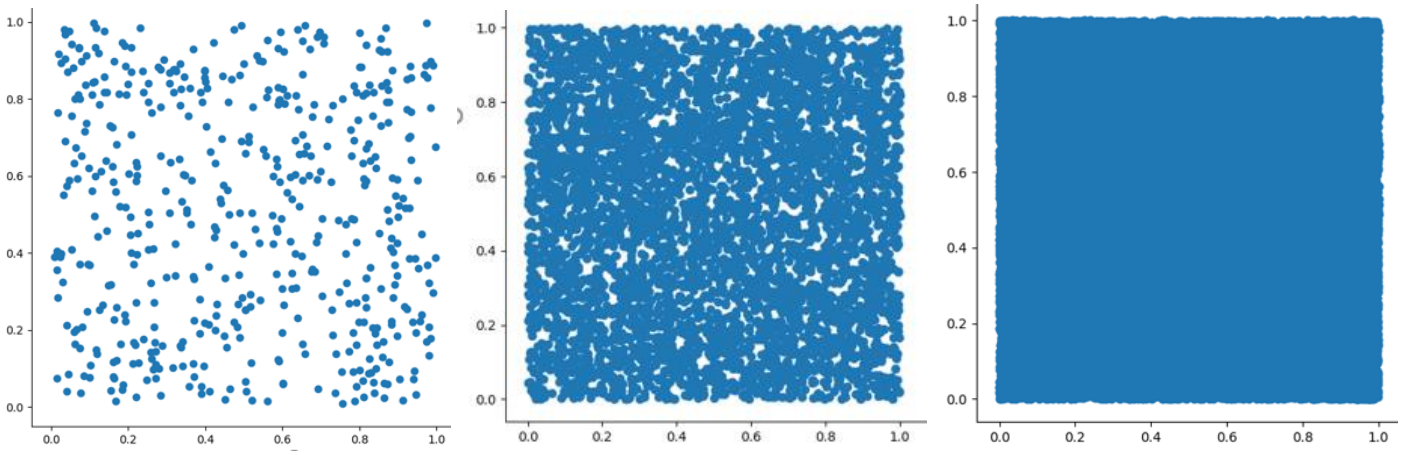


Рисунок 9 поступове заповнення квадрату $[0;1] \times [0;1]$ при збільшенні кількості згенерованих псевдовипадкових чисел.

Розіб'ємо квадрат $[0;1] \times [0;1]$ на рівні 100 квадратів, провівши 8 вертикальних прямих та 8 горизонтальних. У кожному отриманому квадраті порахуємо кількість точок, які в нього попадають. Таким чином визначимо частоти попадання впорядкованих пар у кожну область розбиття. На основі даних частот теж порахуємо значення критерію узгодженості Пірсона. Відповідно, кількість елементів тестованої множини буде вже не n, а $\left[\frac{n}{2}\right]$, а кількість ступенів свободи буде дорівнювати 99.

Аналогічну перевірку зробимо і для трьохвимірного випадку: з елементів множини згенерованих чисел візьмемо впорядковані трійки, які будемо сприймати як координати точок у просторі. Зрозуміло, що аналогічно до двовимірного випадку, вони будуть заповнювати область у кубі $[0;1] \times [0;1] \times [0;1]$ із збільшенням кількості

генерованих чисел у випадку, якщо їх розподіл близький до рівномірного. Виведемо результат побудови для 1000, 10000 та 100000 чисел.

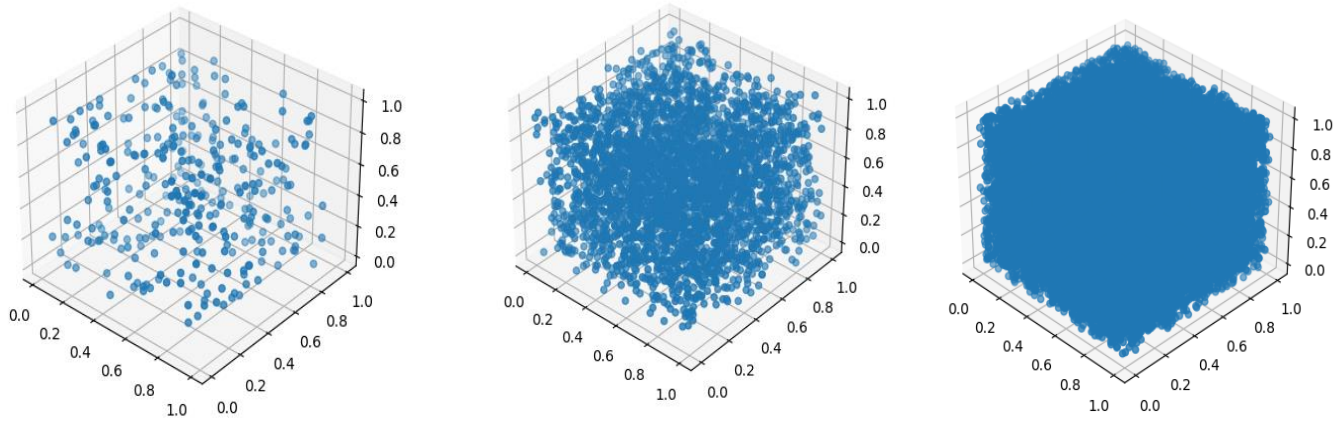


Рисунок 10 поступове заповнення кубу $[0;1] \times [0;1] \times [0;1]$ при збільшенні кількості згенерованих псевдовипадкових чисел.

Розіб'ємо куб $[0;1] \times [0;1] \times [0;1]$ на 1000 однакових кубів за допомогою площин, паралельних координатним площинам. Порахуємо частоти попадання точок у кожен з цих кубів. На основі цих данних також порахуємо характеристику критерію Пірсона, врхувавши, що кількість елементів множини рівна $\left[\frac{n}{3}\right]$, а кількість ступенів свободи буде рівна 999.

Для обчислення характеристики критерію Пірсона та p-значення скористаємось функцією stats.chisquare з вільної бібліотеки scipy для python.

2.5.6 Спектральний тест

Спектральний тест підходить для тестування дихотомічної послідовності. Послідовність розглядається як дискретний сигнал, у якого необхідно виявити частотні піки за доромогою спектрального розкладу. З наявності та кількості таких піків можна судити про періодичність даної дихотомічної послідовності. Перевірка данного тесту з рівнем значущості 0.05 буде значити, що кількість піків частотності має не перевищувати 5%.

Для данного тесту необхідно пригадати дискретне перетворення Фур'є, адже його необхідно буде використати задля представлення послідовності у вигляді суми періодичних складових.

Для початку переведемо дихотомічну послідовність у вигляд послідовності з елементів -1 та 1. Для цього замінимо кожен 0 на -1. За допомогою дискретного перетворення Фурьє отримаємо значення комплексних амплітуд, де дійсна частина відповідає за значення амплітуди, а уявне – за її фазу.

$$s_i = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i}{N} kn}$$

В нашому випадку необхідно звертати увагу саме на модуль значення амплітуди. При обчисленні цих значень бачимо, що вони симетричні відносно середини вибірки. Тому для подальшого аналізу ми візьмемо підпослідовність від 1го елемента до n/2.

Наступним кроком обчислюємо ще одну важливу характеристику. Введемо позначення:

$$T = \sqrt{\ln\left(\frac{1}{0.05}\right) \cdot n} \approx 5.47$$

Де 0.05 – рівень значущості для нашої перевірки. Якщо послідовність випадкова – то 95% «піків» не мають перевищувати значення T. Позначимо через N_0 частку піків, які не мають перевищувати 95%.

$$N_0 = \frac{0.95n}{2} = 4.75$$

Позначимо через d задану наступним чином оцінку різниці очікуваного значення частки піків та реального значення.

$$d = \frac{N_1 - N_0}{\sqrt{n \cdot 0.95 \cdot \frac{0.05}{4}}}$$

Та обчислюємо p-значення

$$P_{value} = erfc\left(\frac{|d|}{\sqrt{2}}\right)$$

Якщо дане значення буде більше за 0.01, то гіпотеза про випадковість не відхиляється і перевірка на випадковість вважається пройденою.

Реалізуємо даний алгоритм у нашій програмі:

```
def spectral(selection):
    n = len(selection)
    plus_minus_one = []

    #Переводимо дихотомічну послідовність у -1,1
    for i in selection:
        if i == 0:
            plus_minus_one.append(-1)
        elif i == 1:
            plus_minus_one.append(1)

    #Знаходимо дискретне перетворення Фур'є
    s = np.fft.fft(plus_minus_one)
```

```
#Розраховуємо коефіцієнти
M = np.abs(s[0:n // 2])
T = np.sqrt(np.log(1 / 0.05) * n)
count_n0 = 0.95 * (n / 2)
count_n1 = len(np.where(M < T)[0])

#Знаходимо d
d = (count_n1 - count_n0) / np.sqrt(n * 0.95 * 0.05 / 4)

#Знаходимо p-значення
p_val = math.erfc(abs(d) / np.sqrt(2))
return p_val
```

2.5.8 Результати первевірок послідовностей згенерованих за допомогою запропонованого ГПВЧ

Запустимо цикл, у якому згенеруємо, зведемо до дихотомічних та протестуємо 100 числових послідовностей методом серій.

```
countTrue = 0
countFalse =0

for i in range(100):
    prn = plotFormula(5,i)
    binArr = toBin(prn)
    result = checkPRN(prn)
    if result: countTrue += 1
    else : countFalse += 1

print("Перевірок пройдено: ", countTrue)
print("Перевірок провалено: ", countFalse)
```

Отримаємо результат:

Перевірок пройдено: 100

Перевірок провалено: 0

Отже, згенеровані запропонованим ГПВЧ дихотомічні послідовності є достатньо схожими на випадкові, щоб пройти дану перевірку. Це дає нам підставу вважати, що запропонований ГПВЧ є одним з рішень розглянутої у роботі проблеми.

Проведемо також інші тести, результати яких занесемо у масиви для подальшого аналізу. Для оцінки ефективності тестованого генератора псевдовипадкових чисел поглянемо на те, як змінювались значення кожної з перевірок, а також максимальні, мінімальні та середні значення у кожній з перевірок.

Для тестування згенерованих вибірок були вибрані наступні тести:

- Тест на рівномірність розподілу випадкових чисел за критерієм Колмогорова-Смірнова
- Тест на рівномірність розподілу випадкових чисел за критерієм фон Мізеса
- Тест на рівномірність розподілу випадкових чисел за критерієм χ^2 Пірсона
- Спектральний тест дихотомічної послідовності

У кожному з даних тестів однією з головних характеристик p-значення.

Означення: p-значенням називається ймовірність того, що згенерована послідовність псевдовипадкових чисел не гірша за гіпотетичну істинну випадкову послідосність.

Якщо p-значення = 1 – то запропонована послідовність в рамках тесту вважається «ідеально випадковою», якщо ж p-значення = 0, то послідовність «абсолютно передбачувана». Для обраного рівню значущості α p-значення має бути не нижчим, ніж даний показник. В такому випадку гіпотеза про випадковість не відхиляється. В інакшому випадку гіпотеза відхиляється.

Для перевірки було проведено 100 генерацій з різними згенерованими вибірками по 10 000 чисел кожна.

В результаті перевірки отримали наступні результати:

Середнє p-значеня критерію χ^2: 0.28224817914959066

Середнє значення тесту ω^2: 0.19694233011372855

Середнє значення параметра λ критерію Колмогорова: 0.009712890624999993

Середнє p-значеня спектрального тесту : 0.46053770850662656

Мінімальне p-значеня критерію χ^2: 0.0005340629923535132

Мінімальне значення тесту ω^2: 0.02891029713948573

Мінімальне значення параметра λ критерію Колмогорова: 0.004648437499999991

Мінімальне p-значеня спектрального тесту: 0.0010193326220783953

Максимальне p-значеня критерію χ^2: 0.9265960076090426

Максимальне значення тесту ω^2: 0.9416377843221007

Максимальне значення параметра λ критерію Колмогорова: 0.019267187500000005

Максимальне p-значеня спектрального тесту: 0.9972390169448668

Врахуємо необхідні значення для проходження тестів з рівнем значущості 0.05. З отриманих результатів можна зробити висновок, що отримані послідовності в середньому відповідають заданій планці якості у вигляді випадковості із рівнем значущості 0.05. Проте, максимальні та мінімальні значення дають нам зрозуміти, що не всі послідовності проходять усі тести достатньо добре. Для наглядності побудуємо графік для статистики кожного тесту окремо щоб побачити, скільки раз їх значення не відповідають необхідним у критерії. Для зручності значення статистик критеріїв позначимо синім кольором, а критичні значення – червоним.

Для критерію Пірсона χ^2 p-значення не відповідають необхідним у 24 випадках з 100. Це можна спостерігати на рисунку 11. На даний момент тільки для цього критерію є стільки вибірок, що не відповідають його вимогам та змушують відхилити гіпотезу про рівномірний розподіл, як мінімум, з обраним рівнем значущості.

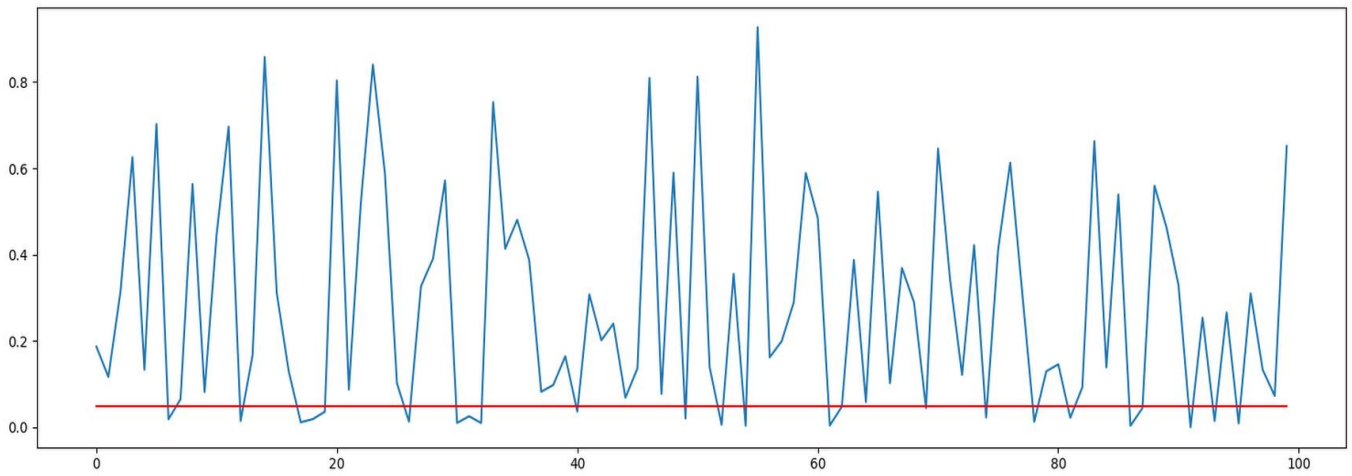


Рисунок 11 Графік зміни p-значення критерію χ^2

Для критерію фон Мізеса значення ω^2 перевищують критичний рівень для рівня значущості 0,05 у 7 випадках з 100. Це можна спостерігати на рисунку 12.

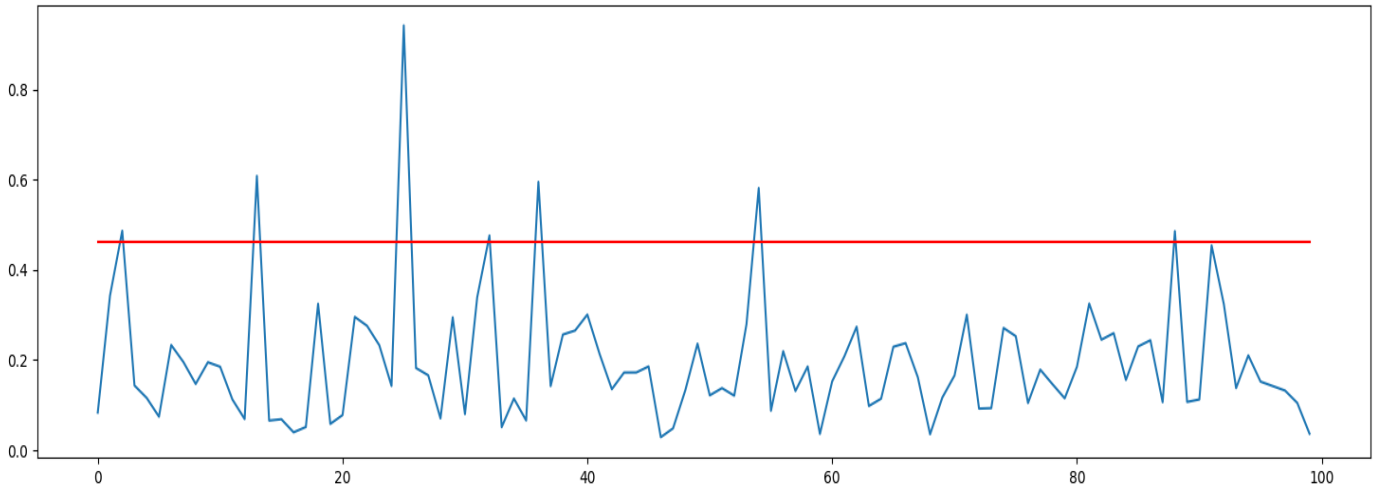


Рисунок 12 Графік зміни значення ω^2 критерію фон Мізеса

Критерій Колмогорова-Смірнова пройшла кожна із згенерованих вибірок. Це можна бачити на рисунку 13. Проте для спектрального тесту є 10 значень, які не дозволяють вважати відповідні дихотомічні послідовності випадковими. Візуалізація наведена на рисунку 14.

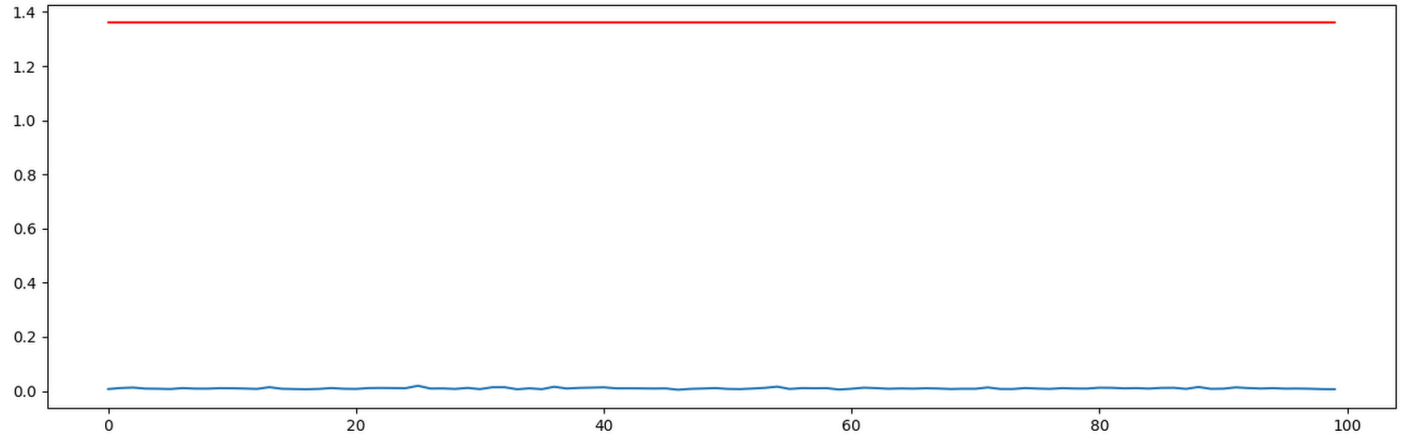


Рисунок 13 Графік зміни значення λ критерію Колмогорова-Смірнова

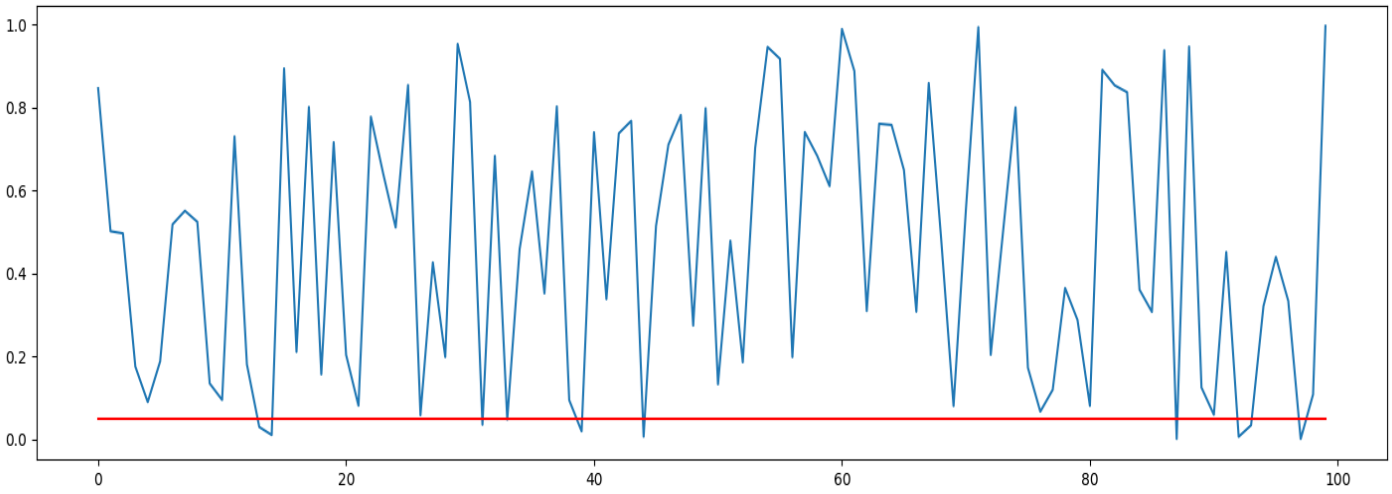


Рисунок 14 Графік зміни p-значення спектрального тесту для згенерованих 100 вибірок

Дані результати свідчать про те, що не всі значення зерна дають достатньо хорошу послідовність псевдовипадкових чисел. Проте, переважна більшість значень зерна дає змогу генерувати послідовності, що проходять усі тести. Строго кажучи, якщо взяти 100 підпослідовністей деяких істинно випадкових чисел, то немає ніяких гарантій, що кожна з них пройде усі тести. Саме тому в рамках цього дослідження ми звертаємо увагу саме на середнє значення статистик критеріїв та відсоток вибірок, які пройшли тести успішно.

2.6 Порівняння надійності запропонованого методу з існуючими ГПВЧ у

2.6.1 Порівняння швидкодії генерації псевдовипадкових чисел

Одним з важливих параметрів у сучасних генераторах псевдовипадкових чисел є продуктивність генерації. Порівняємо швидкість генерації у запропонованого генератора псевдовипадкових чисел. Для цього проведемо тести на різних мовах програмування. На кожній з них створимо по 100 000 псевдовипадкових чисел, розподілених у проміжку від 0 до 1. Для перевідрки оберемо стандартні методи генерації псевдовипадкових чисел у мовах програмування C++, C# та Python. Код наведемо нижче.

Зауважимо, що для генерації 100 000 псевдовипадкових чисел описаним методом потрібно згенерувати фрактал, щонайменше на сьомій ітерації.

Розглянемо результати:

- Час генерації на C++ становить 0.0070516 секунд
- Час генерації на C# становить 0.3290037 секунд
- Час генерації на Python становить 0.316581 секунд
- Час генерації запропонованим методом реалізованим на Python із 7 ітераціями становить 2.16409 секунд

З цього можна зробити висновок, що запропонований метод побудови генератора псевдовипадкових чисел є менш продуктивним. Зауважимо, що python є мовою програмування, основною перевагою якої є швидкість та зручність написання коду, що йде не на користь продуктивності, тому очікувано, що час генерації на мові C++ та C# кращий.

Перевірка часу генерації C++:

```
int main()
{
    int N = 100000;
    float arr [100000];
    auto start = std::chrono::system_clock::now();
    srand((unsigned)time(NULL));
    for (int i = 0; i < N; i++) {
        arr[i] = (float)rand() / RAND_MAX;
        //cout << arr[i] << "\n";
    }
    auto end = std::chrono::system_clock::now();
    std::chrono::duration<double> elapsed_seconds = end -
start;
    std::time_t end_time =
std::chrono::system_clock::to_time_t(end);
    std::cout << "finished computation at " << &end_time
        << "elapsed time: " << elapsed_seconds.count() <<
"s\n";
    system("pause");
}
```

Перевірка часу генерації C#:

```
static void Main(string[] args)
        {
            Stopwatch stopwatch = new Stopwatch();



            int N = 100000;
            double[] arr = new double[N];


            stopwatch.Start();


            for (int i = 0; i < N; i++)
            {                 Random rand = new
Random(DateTime.Now.Millisecond+i);
                arr[i] = rand.Next(0, 10000000)/10000000f;
                //Console.WriteLine("arr[" + i + "] = " + arr[i]);
            }


            stopwatch.Stop();


            Console.WriteLine("Elapsed Time is {0} ms",
stopwatch.ElapsedMilliseconds);
            Console.ReadLine();


        }
```

Перевірка часу генерації Python:

```
    #імпортуємо стандартну бібліотеку для генерації
псевдовипадкових чисел
    import random

    arr = []

   start = datetime.now()

   print(start)

   for i in range(100000):
       arr.append(random.uniform(0, 1))

   end = datetime.now()

   test = end - start

   print(test, " seconds ")
```

2.6.2 Порівняння якості вибірок, що генерують ГПВЧ

Як було встановлено у попередньому пункті, запропонований метод генерації псевдовипадкових чисел є менш швидким на фоні стандартних методів генерування у сучасних мовах програмування. Проте, швидкодія є не такою важливою характеристикою, як відповідність критеріям. Тому порівняємо згенеровані мовами програмування послідовності нашими критеріями та порівняємо отримані результати з тими, що були отримані для послідовностей, згенерованих за допомогою фракталів. Для тестування будемо брати не повну вибірку у 100 000 чисел, а підпослідовності по 10 000 для коректного порівняння з отриманими раніше результатами. Також згенеруємо ще 100 000 чисел моїм методом та розглянемо їх підпослідовності для того, щоб всі ГПВЧ порівнювались в однакових умовах.

З технічної точки зору, реалізовано це було через запис згенерованих послідовностей у файл для подальшого читання даних файлу у Python. Тому для всіх згенерованих послідовностей проводились одні й ті самі тести.

Результати наступні:

1. ГПВЧ C++ згенерував послідовності, які в середньому пройшли всі тести. Тільки 1 з 10 послідовностей не пройшла тест на рівномірність розподілу за критерієм фон Мізеса, 1 з 10 не пройшла тест за критерієм Колмогорова-Смірнова та 1 з 10 не пройшла спектральний тест.
   Варто зауважити, що додаткову перевірку критерієм Пірсона на площині та в просторі згенеровані послідовності пройшли.
2. ГПВЧ C# 1 з 10 разів провалив тест за критерієм фон Мізеса, 1 з 10 разів провалив тест за критерієм Колмогорова-Смірнова та 1 з 10 разів провалив спектральний тест.
   Проте додаткову перевірку критерієм Пірсона на площині та в просторі згенеровані послідовності пройти не змогли.

3. ГПВЧ Python згенерував 10 послідовностей, 1 з яких провалила тест за критерієм Пірсона та 1 з 10 вибірок провалила спектральний тест.
   Додаткову перевірку критерієм Пірсона на площині та в просторі згенеровані послідовності пройшли.
4. ГПВЧ на основі фракталів згенерував 10 послідовностей, 1 з яких не пройшла тест фон Мізеса і одна з яких не пройшла тест Колмогорова-Смірнова.
   Додаткову перевірку критерієм Пірсона на площині та у просторі послідовності пройшли успішно.

Отже, якщо порівнювати дані генератори псевдовипадкових чисел із запропонованим у дисертації, то очевидно, що ГПВЧ C# не складає конкуренцію, а ГПВЧ C++ має відсоток завалених тестів, що зрівнюється з таким у запропонованого метода генерації на основі фракталів.

РОЗДІЛ 3: ПОБУДОВА ГЕНЕРАТОРА ВИПАДКОВИХ ЧИСЕЛ НА ОСНОВІ ЗАПРОПОНОВАНИХ ГЕНЕРАТОРІВ ПСЕВДОВИАВДКОВИХ ЧИСЕЛ

3.1 Ідея побудови ГВЧ на основі запроаонованого ГПВЧ

Генератор випадкових чисел (ГВЧ) відрізняється від генератора псевдовипадкових чисел (ГПВЧ) тим, що результат генератора випадкових чисел неможливо відтворити, а відповідно, зерно не є необхідним для роботи такого генератора.

Є 2 простих способи побудувати генератор випадкових чисел на основі запропонованого:

1. Замість зерна використовувати значення деякого випадкового шуму
2. Використовувати значення випадкового шуму для зміни кута відхилення на кожній ітерації побудови зображення фрактала.

Можна запропонувати безліч інших варіантів, але будемо розглядати саме ці, оскільки вони найбільш очевидні та ті, що більш-менш легко можна реалізувати без масштабних змін коду програми. Є багато джерел випадкового шуму, якими можна скористатись. До них відносяться будь-які електричні компоненти, природні явища, джерела світла обчислювальні прилади. Для електричних компонентів характерний шум, що породжується випадковими скачками струму. Такий тип шуму називається імпульсним і з’являється через низькочастотні модуляції струму через захват і еміссію носіїв заряду. Такий шум з деяких компонентів компьютера можна отримати програмними засобами.

3.2 Реалізація даної ідеї для запропонованого ГПВЧ

Для наших цілей найпростішим та найефективнішим методом буде оцінка часу виконання процесором деяких операцій у мілісекундах. Реалізуємо це у програмі наступним чином:

1. Введемо змінну для значення випадкового шуму.
2. Фіксуємо час перед деяким набором операцій
3. Виконуємо операції
4. Фіксуємо час після завершення виконнання операцій
5. Фіксуємо різницю часу у мікросекундах

У випадку, коли ми хочемо використовувати шум у якості зерна для генерації псевдовипадкових чисел, у якості операцій з кроку 3 можна використовувати будь-що, що можна реалізувати на мові програмування. Доцільно обрати деякі достатньо швидкі розрахунки, щоб вони виконувались менше секунди. Інакше доцільно буде використовувати різницю часу у секундах.

У випадку, коли планується використовувати значення псевдовипадкового шуму для зміни кути відхилення на кожній ітерації побудови зображення фрактала, набором операцій з кроку 3 буде слугувати проходження циклу виконання правил згенерованої L-системи. Виглядати це буде наступним чином:

```
calculationgNoise = 0
for ch in axiom:
      start = datetime.now()
      if ch == "+":
          turtle.left(120 + deflection + calculationgNoise)
      elif ch == "-":
```

```
        turtle.right(120)
    else:
        turtle.forward(d)
        if (iteration % (len(axiom)//20) == 0):
            x.append(turtle.pos()[0])
            y.append(turtle.pos()[1])
    iteration += 1
    end = datetime.now()
    calculationgNoise = (end - start).microseconds
```

ВИСНОВКИ

В ході проведеної роботи було виконано дослідження проблеми генерування псевдовипадкових чисел. Детально представлені основні поняття і об’єкти вивчення у даній галузі. Розглянуті необхідні умови для створення ГПВЧ, в рамках яких можна знайти рішення. Запропоновано один з методів підходу до створення ГПВЧ на основі побудови фракталів, розмірність яких не перевищує 2. Розглянуті сучасні інструменти зображення фракталів. Детально описані методи створення послідовності правил для побудови зображення фракталів з необхідною точністю та показана реалізація цих методів у на мові програмування Python.

Проведені дослідження запропонованого методу на ефективність, запропонований спосіб позбавлення основного недоліку – періодичності даних, що генерує метод. Описані шляхи застосування даного методу, запропоновані варіанти зведення отриманих результатів до дихотомічних послідовностей.

Проведена перевірка згенерованих запропонованим ГПВЧ дихотомічних послідовностей на випадковість із рівнем значущості 0.05 за допомогою ряду перевірок, порядок яких було описано та реалізовано програмними засобами. Отримані результати задовольнили припущення про випадковість генерованої вибірки.

Отже поставлена задача була вирішена.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

Книги	Один автор	1. Grzegorz Rozenberg, Arto Salomaa. The mathematical theory of L systems. — New York: Academic Press, 1980 2. Aristid Lindenmayer. Mathematical models for cellular interaction in development
	Декілька авторів	1. N.G. Bardis, A.P. Markovskyi, N. Doukas, N. V. Karadimas. True Random Number Generation Based on Environmental Noise Measurements for Military Applications Proceedings of the 8th WSEAS International Conference on SIGNAL PROCESSING, ROBOTICS and AUTOMATION. — 2009. — С. 68—73. — ISBN 978-960-474-054-3. — ISSN 1790-5117. 2. Von Neumann, John. Various techniques used in connection with random digits National Bureau of Standards Applied Mathematics Series. — 1951. — № 12. — С. 36—38.
Наукові статті		1. Sherif H. AbdElHaleem, Ahmed G. Radwan and Salwa K. Abd-El-Hafiz Design of Pseudo Random Keystream Generator Using Fractals 2. 1Thamizhchelvy, K. and 2G. Geetha An Efficient Image Generation Algorithm Using Fractals and Chaos Theory
Електронні ресурси		1. wikipedia.org 2. https://www.youtube.com/channel/UCoxcjq-8xIDTYp3uz647V5A 3. https://www.youtube.com/channel/UCHnyfMqiRRG1u-2MsSQLbXA 4. https://www.youtube.com/channel/UCP1JsJgeNs86oqLGnjfGo9Q 5. https://www.youtube.com/channel/UCzdmz_lLWT_dPqOvFjXAMVg