```
import numpy as np
import random
import time
import matplotlib.pyplot as plt
import matplotlib.animation as animation
import seaborn as sns
%matplotlib notebook
```

Завдання 1: Написати програму розв’язування систем 3 лінійних рівнянь з 3 невідомими, та вказати розв’язок X системи AX= B.
(Системи таких рівнянь розв'язуються методом Гауса(чи іншими), але я використала готову функцію, що не винаходити велосипед.)

```
# Коефіцієнти рівнянь
A = np.array([[2, 1, -1],
              [1, 3, 2],
              [3, 2, 4]])

b = np.array([1, 9, 7])

# Розв'язання системи рівнянь
x = np.linalg.solve(A, b)

x
```

```
array([-0.44,  2.64,  0.76])
```

Завдання 2: Написати програму, котра приймає на вхід матрицю зі значеннями 1 або 0 (живий або мертвий стани) та ітеративно замінює значення в матриці за наступними правилами:
● якщо в живої клітини два чи три живих сусіди, то вона лишається жити;
● якщо в живої клітини один чи немає живих сусідів, то вона помирає від «самотності»;
● якщо в живої клітини чотири та більше живих сусідів, то вона помирає від «перенаселення»;
● якщо в мертвої клітини рівно три живих сусіди, то вона оживає. Кожна клітинка має вісім сусідів.

```
def count_neighbors(matrix, i, j):
    n, m = matrix.shape
    count = 0

    for x in range(i - 1, i + 2):
        for y in range(j - 1, j + 2):
            if 0 <= x < n and 0 <= y < m and (x != i or y != j):
                count += matrix[x, y]

```

```
    return count

def iterate(matrix):
    n, m = matrix.shape
    new_matrix = np.copy(matrix)

    for i in range(n):
        for j in range(m):
            neighbors = count_neighbors(matrix, i, j)

            if matrix[i, j] == 1:
                if neighbors < 2 or neighbors > 3:
                    new_matrix[i, j] = 0
            else:
                if neighbors == 3:
                    new_matrix[i, j] = 1

    return new_matrix
```

```
initial_matrix = np.array([[1, 0, 0, 0, 0, 0, 0],
                           [0, 0, 1, 0, 0, 1, 1],
                           [1, 0, 0, 1, 0, 0, 1],
                           [0, 1, 1, 0, 1, 1, 0],
                           [1, 1, 1, 1, 0, 0, 1],
                           [1, 1, 1, 1, 1, 1, 1],
                           [1, 1, 0, 1, 1, 0, 1]])

# Кількість ітерацій
iterations = 7

# Виконуємо ітерації
for i in range(iterations):
    initial_matrix = iterate(initial_matrix)

# Виводимо результат
initial_matrix
```

```
array([[0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0]])
```

Завдання 2.1*: Модифікувати програму так, щоб вона випадково генерувала початковий стан матриці з заданим розміром і мала можливість безкінечно симулювати ітерації.

```
def random_matrix(n, m):
    return np.random.randint(2, size=(n, m))

def count_neighbors(matrix, i, j):
    n, m = matrix.shape
```

```
    count = 0

    for x in range(i - 1, i + 2):
        for y in range(j - 1, j + 2):
            if 0 <= x < n and 0 <= y < m and (x != i or y != j):
                count += matrix[x, y]

    return count

def iterate(matrix):
    n, m = matrix.shape
    new_matrix = np.copy(matrix)

    for i in range(n):
        for j in range(m):
            neighbors = count_neighbors(matrix, i, j)

            if matrix[i, j] == 1:
                if neighbors < 2 or neighbors > 3:
                    new_matrix[i, j] = 0
            else:
                if neighbors == 3:
                    new_matrix[i, j] = 1

    return new_matrix
```

```
# Розмір матриці
n = 50
m = 50

initial_matrix = random_matrix(n, m)

# Безкінечна симуляція ітерацій

# while True:
#     print(initial_matrix)
#     initial_matrix = iterate(initial_matrix)
#     time.sleep(1)
```

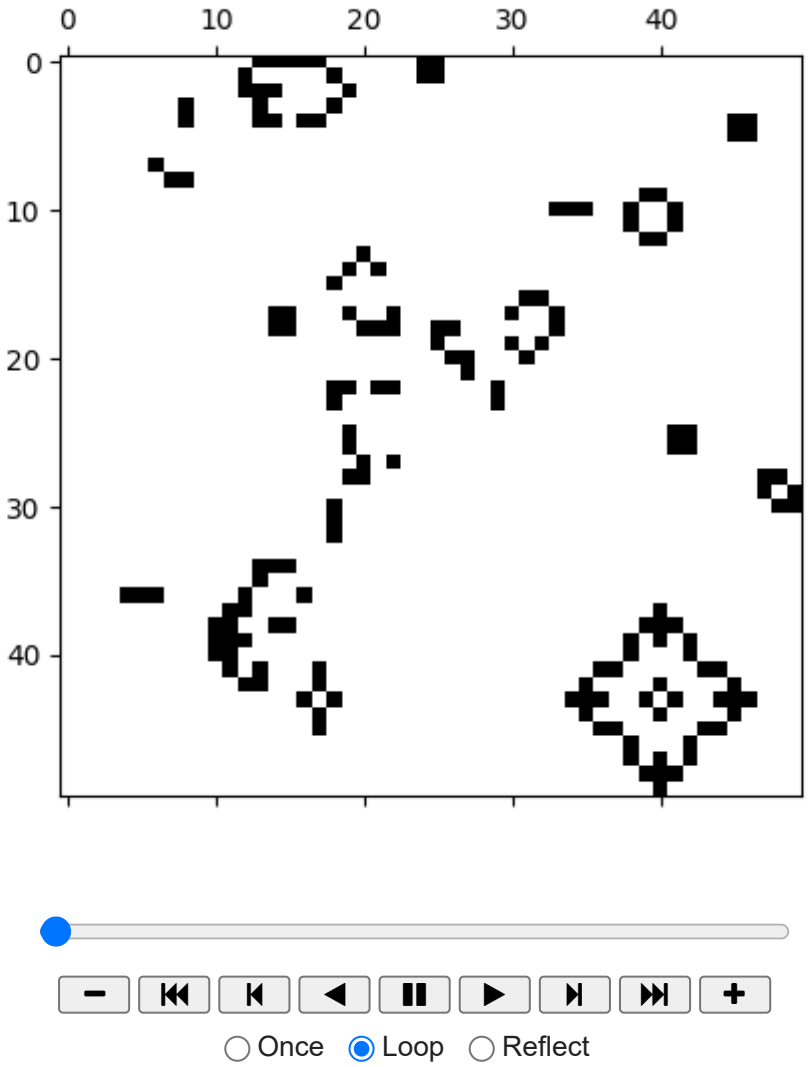
Завдання 2.2**: Візуалізувати симуляцію ітерацій (matplotlib / seaborn / plotly / etc).

```
# Створення фігури та підготовка для анімації
fig, ax = plt.subplots()
mat = ax.matshow(initial_matrix, cmap="binary")

def update(frame):
    global initial_matrix
    initial_matrix = iterate(initial_matrix)
    mat.set_data(initial_matrix)
    return mat,

# Створення анімації
ani = animation.FuncAnimation(fig, update, frames=200, interval=200)
```

```
13
14  from IPython.display import HTML
15
16  ani.save('animation.mp4', writer='ffmpeg', fps=20);
17  HTML(ani.to_jshtml())
```



Завдання 3: Ймовірність випадання сторони ‘H’ для кожної з 5 монет (назвемо їх m1, m2, m3, m4, m5) зі зміщеним центром ваги рівна відповідно [0.1, 0.2, 0.4, 0.8, 0.9]. З монет навмання вибрали одну і почали випробування. Визначити ймовірність випадання ‘H в наступному випробуванні після кожного з 8 фактично проведених випробувань: [H H H T H T H H] (тут ‘T’ протилежна сторона монети). Наприклад, до першого випробування ймовірність випадання ‘H’ рівна ~0.48 (за формулою повної ймовірності, з урахуванням рівноможливості вибрати кожну монету з наявних). Після випадання ‘H’ в першому випробуванні, ймовірності гіпотез, що вибрана монета є m1/m2/m3 зменшились, а відповідно ймовірності гіпотез, що вибрана монета m4/m5 збільшились а, отже і змінилась ймовірність випадання ‘H’ в наступному (другому) випробуванні і стала рівною ~0.69. Аналогічно після випадання ‘H’ в другому випробування треба переоцінити ймовірність випадання ‘H’ в третьому, і т.д.

Алгоритм розв'язання:

1)Визначаємо початкові ймовірності вибору кожної монети: p1 = 0.2 (для m1) p2 = 0.2 (для m2) p3 = 0.2 (для m3) p4 = 0.2 (для m4) p5 = 0.2 (для m5)

2)Обчислюємо початкову ймовірність випадання 'H': P('H') = p10.1 + p20.2 + p30.4 + p40.8 + p5*0.9 = 0.48*

*3)Після кожного випадання 'H' або 'T' оновлюємо ймовірності вибору монет згідно з формулою Байеса: P(mi|H) = P(H|mi)*P(mi) / P(H)

4)Обчислюємо оновлену P('H') за формулою повної ймовірності з використанням оновлених ймовірностей вибору кожної монети.

5)Повторюємо кроки 3-4 для кожного наступного випробування.

```
P_m_i = [0.2, 0.2, 0.2, 0.2, 0.2] # початкова ймовірність для кожної монети бути вибр
P_H_m_i = [0.1, 0.2, 0.4, 0.8, 0.9] # ймовірність випадання H для кожної монети

results = ['H', 'H', 'H', 'T', 'H', 'T', 'H', 'H']
result_probabilities = []

def bayesian_update(prior, likelihood, evidence):
  posteriors = [prior[i] * likelihood[i] for i in range(len(prior))]
  total = sum(posteriors)
  return [post / total for post in posteriors], total

# Починаємо з початкових імовірностей
posterior, probability = bayesian_update(P_m_i, P_H_m_i, results[0])
result_probabilities.append(round(probability,2))

# Проходимось по результатам випробувань і оновлюємо ймовірності
for i in range(1, len(results)):
  if results[i] == 'H':
    posterior, probability = bayesian_update(posterior, P_H_m_i, results[i])
  result_probabilities.append(round(probability, 2))

result_probabilities
```

```
[0.48, 0.69, 0.79, 0.79, 0.83, 0.83, 0.85, 0.86]
```

✓ 0 с завершено о 14:28