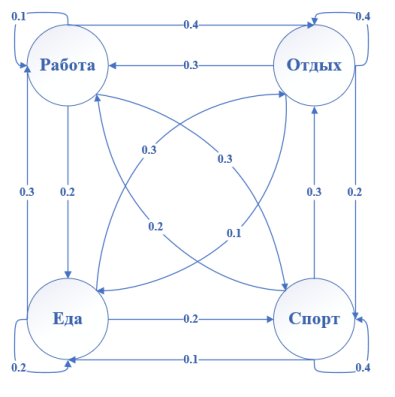
**Марковские цепи**

**1. Придумать эргодическую марковскую цепь, содержащую не менее 4-х состояний**

Для моделирования была выбрана марковская цепь с четырьмя состояниями: "Работа", "Отдых", "Спорт", "Еда". Переходы между этими состояниями заданы матрицей переходных вероятностей.

**2. Диаграмма переходов и матрица переходных вероятностей** Диаграмма переходов (граф) изображена на следующем рисунке:



Матрица переходных вероятностей имеет следующий вид:

�� = (

0.1 0.4 0.3 0.4 0.2 0.3 0.3 0.3

0.3 0.2 0.2 0.1 0.4 0.1 0.2 0.2

)

**3. Реализация марковой цепи и построение графиков изменения компонентов финальных векторов.**

Моделирование Марковской цепи пошагово с различными начальными векторами вероятностей состояний

Были проведены симуляции для разных начальных состояний:

1. Начальное состояние: "Отдых"

2. Начальное состояние: "Работа"

3. Начальное состояние: "Спорт"

4. Начальное состояние: "Еда"

Из диаграммы состояний видно, что все состояния имеют ненулевые вероятности перехода в другие состояния. Это предполагает, что марковская цепь может быть неприводимой, так как существует вероятность перейти из любого состояния в любое другое. Также в диаграмме не видны явные циклы с периодом больше единицы, что делает возможным апериодичность цепи. Однако, для точного утверждения о неприводимости и апериодичности требуется дополнительный анализ, которым мы еще займемся. Если цепь неприводима и апериодична, и если каждое состояние возвратно с конечным средним временем возврата, то представленная марковская цепь является эргодической.

**Код реализации:**

import numpy as np

import matplotlib.pyplot as plt

*# Пространство состояний*

states = ["Работа", "Отдых", "Спорт", "Еда"]

*# Матрица вероятностей (матрица переходов) из main.py*

trans\_matrix = np.array([

[0.1, 0.4, 0.3, 0.2], *# Работа -> Работа, Отдых, Спорт, Еда* [0.3, 0.4, 0.2, 0.1], *# Отдых -> Работа, Отдых, Спорт, Еда* [0.2, 0.3, 0.4, 0.1], *# Спорт -> Работа, Отдых, Спорт, Еда* [0.3, 0.3, 0.2, 0.2] *# Еда -> Работа, Отдых, Спорт, Еда* ])

*# Начальные вектора вероятностей для каждого состояния*

init\_prob\_vectors = [

np.array([1.0, 0.0, 0.0, 0.0]), *# Начало в "Работа"*

np.array([0.0, 1.0, 0.0, 0.0]), *# Начало в "Отдых"*

np.array([0.0, 0.0, 1.0, 0.0]), *# Начало в "Спорт"*

np.array([0.0, 0.0, 0.0, 1.0]), *# Начало в "Еда"*

np.array([0.25, 0.25, 0.25, 0.25]) *# Равномерное начальное распределение* ]

def markov\_step(state\_prob, trans\_matrix):

return state\_prob @ trans\_matrix

def simulate\_chain(init\_prob, trans\_matrix, steps, epsilon=1e-6): current\_prob = init\_prob

history = [current\_prob]

for \_ in range(steps):

next\_prob = markov\_step(current\_prob, trans\_matrix) history.append(next\_prob)

if np.linalg.norm(next\_prob - current\_prob) < epsilon: break

current\_prob = next\_prob

return history

*# Моделирование марковской цепи*

steps = 50

chains = [simulate\_chain(vec, trans\_matrix, steps) for vec in init\_prob\_vectors]

*# Вывод результатов моделирования*

for i, chain in enumerate(chains):

init\_state = states[i] if i < len(states) else 'Равномерное распределение'

print(f"Начальное состояние: {init\_state}")

print(f"Возможные состояния:\n{' -> '.join(states)}")

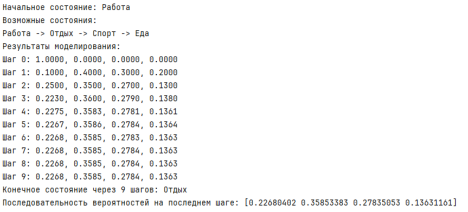
print("Результаты моделирования:")

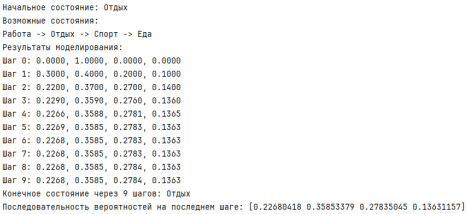
for step, vec in enumerate(chain):

print(f"Шаг {step}: {', '.join([f'{p:.4f}' for p in vec])}")

final\_state = states[np.argmax(chain[-1])]

print(f"Конечное состояние через {len(chain) - 1} шагов: {final\_state}") print(f"Последовательность вероятностей на последнем шаге: {chain[-1]}") print("-" \* 50)





def plot\_chains(chains, states, init\_states):

for i, chain in enumerate(chains):

chain = np.array(chain)

plt.figure(figsize=(12, 6))

for j, state in enumerate(states):

plt.plot(range(len(chain)), chain[:, j], label=f'{state}') plt.title(f'Изменение компонентов вероятности (Начальное состояние: {init\_states[i]})')

plt.xlabel('Шаги')

plt.ylabel('Вероятность')

plt.legend()

plt.grid(True)

plt.show()

def plot\_rmse(chains, init\_states):

for i, chain in enumerate(chains):

rmse = []

for j in range(1, len(chain)):

rmse.append(np.sqrt(np.mean((chain[j] - chain[j-1])\*\*2))) plt.figure(figsize=(12, 6))

plt.plot(range(1, len(chain)), rmse, label=f'Начальное состояние: {init\_states[i]}')

plt.title(f'Изменение среднеквадратического отклонения (Начальное состояние: {init\_states[i]})')

plt.xlabel('Шаги')

plt.ylabel('RMSE')

plt.legend()

plt.grid(True)

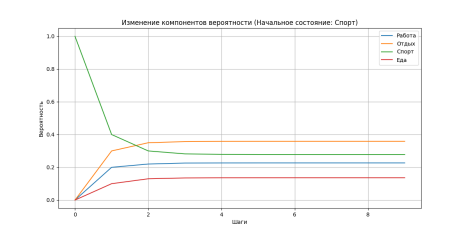
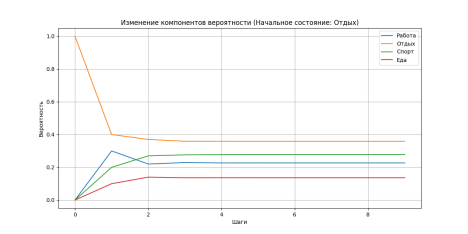
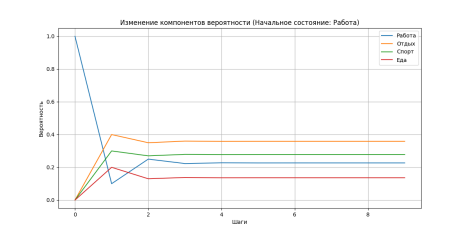
plt.show()

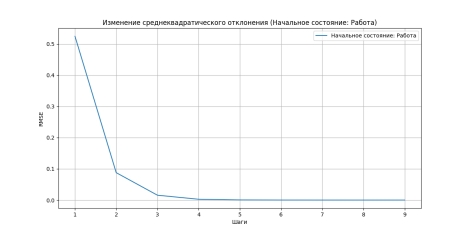
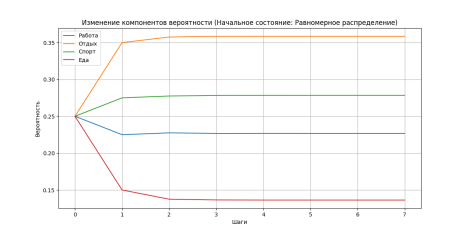
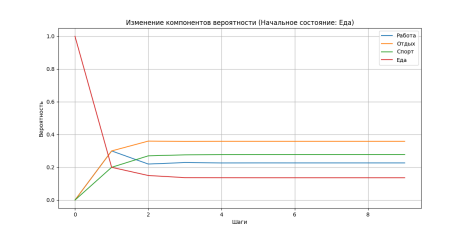
*# Пример использования функций для визуализации:*

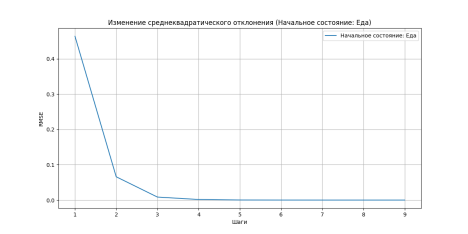
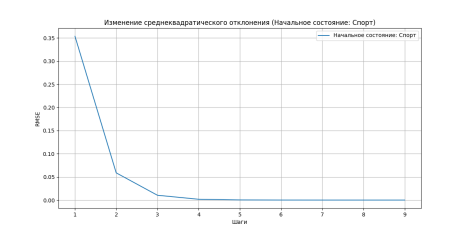
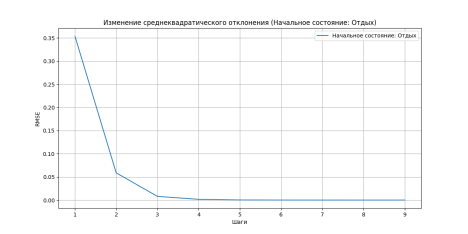
init\_states = ["Работа", "Отдых", "Спорт", "Еда", "Равномерное распределение"]

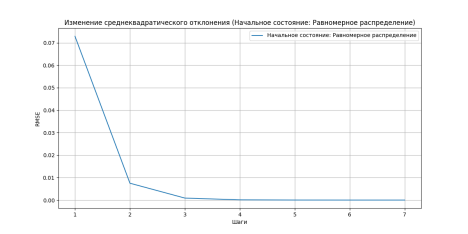
plot\_chains(chains, states, init\_states)

plot\_rmse(chains, init\_states)







Поиск стационарного распределения аналитически:

*#Аналитическое решение*

def find\_stationary\_distribution(transition\_matrix):

n = len(transition\_matrix)

*# (P^T - I) \* pi = 0*

A = np.transpose(transition\_matrix) - np.eye(n)

A = np.vstack([A, np.ones(n)])

b = np.zeros(n + 1)

b[-1] = 1

*# lsm*

pi, residuals, rank, s = np.linalg.lstsq(A, b, rcond=None) return pi

stationary\_distribution = find\_stationary\_distribution(trans\_matrix) print("Стационарное распределение:", stationary\_distribution) sum = 0

for i in stationary\_distribution:

sum += i

print("sum = ", sum)



Сравнение вектора из пункта 3 и вектор, рассчитанный аналитически:

*#Сравнение финальные распределения цепей со стационарным*

def compare\_distributions(chains, stationary\_dist, init\_states): *"""Сравнивает финальные распределения цепей со стационарным распределением."""*

for i, chain in enumerate(chains):

final\_dist = chain[-1]

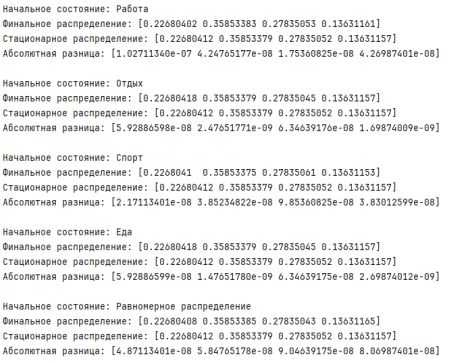
print(f"Начальное состояние: {init\_states[i]}")

print(f"Финальное распределение: {final\_dist}")

print(f"Стационарное распределение: {stationary\_dist}") print(f"Абсолютная разница: {np.abs(final\_dist -

stationary\_dist)}\n")

*# Пример использования:*

compare\_distributions(chains, stationary\_distribution, init\_states) 

**Выводы**

Полученные результаты моделирования подтверждают аналитически найденное стационарное распределение. Сравнение данных показывает, что финальные векторы вероятностей, полученные в результате моделирования, хорошо согласуются с аналитическим решением.