1. Теоретическая часть

Пусть двое играют в некоторую антагонистическую игру. Проводится некоторое достаточно большое число партий. Накопленный выигрыш — это суммарный выигрыш во всех партиях. Средний выигрыш — это накопленный выигрыш, разделенный на число сыгранных партий. Средний выигрыш стремится к цене игры, когда число партий велико. Однако при малом числе партий тенденция может быть не так заметна, поскольку разброс может быть большим. Для исследования вопроса о скорости сходимости среднего выигрыша к цене игры следует использовать метод статистических испытаний. Разыгрывается достаточно длинная серия игр, в каждой из которых оба игрока применяют свои оптимальные смешанные стратегии. (Эти стратегии могут быть найдены как аналитически, так и методом Брауна - Робинсон, описанном в предыдущей лабораторной работе). В каждой партии стратегия выбирается случайным образом в соответствии с оптимальными вероятностями смешанной стратегии.

1. Программный код

"""

Программа, моделирующая последовательность

антагонистических игр с заданной матрицей платежей

и вычисляющая средний и суммарный выигрыш

Автор: Афанасьев И.Е.

Дата написания: 20.09.2020

"""

iterations = 5650 # Количество ходов

start\_money = 1000 # Начальное количество денег у каждого игрока

matrix1 = [ [-2, 3], [3, -4] ] # Матрица платежей

# Инвертируем матрицу (Платежи для второго игрока)

q = len (matrix1)

matrix2 = []

for i in range(q):

matrix2.append([])

for i in matrix1:

for pos, j in enumerate(i):

matrix2[pos % q].append(- j)

# Класс игрок с матрицей платежей, текущим количеством

# денег, коэффициентами для принятия решения на основе

# стратегии противника и последним ходом

class gamer:

def \_\_init\_\_(self, matrice, money):

self.m = matrice

self.money = money

self.koef = [0] \* len(matrice[0])

self.last = 0

def strikeback(self, enemy\_strat):

self.koef[enemy\_strat] += 1

self.money += self.m[self.last][enemy\_strat]

return self.money

# Вычисление оптимального хода на основе

# предыдущих ходов противника

def math(self):

M = []

for i in self.m:

process = 0

for j in range(len(self.koef)):

process += self.koef[j] \* i[j]

M.append(process)

self.last = M.index(max(M))

return self.last

# Импортируем графическую библиотеку

import matplotlib.pyplot as plt

# Создаём два объекта класса "игрок"

player1 = gamer(matrix1, start\_money)

player2 = gamer(matrix2, start\_money)

# Инициализируем фигуру, на которой будем рисовать

fig = plt.figure()

plt.ion()

# Вычисляем суммарный и средний выигрыш

sum\_gain = 0

average\_gain = []

x = [] # ось x

# Игра

for i in range(iterations):

str1 = player1.math() # Вычисление текущего хода

str2 = player2.math() # Вычисление текущего хода

player1.strikeback(str2)

player2.strikeback(str1)

sum\_gain += matrix1[str1][str2]

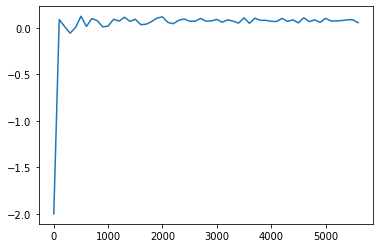
average\_gain += [sum\_gain / (i + 1)]

x += [i]

plt.plot(x[::100], average\_gain[::100])

plt.show()

1. Результаты работы программы



Аналогично предыдущей лабораторной работе:

S(x, y) = −2xy + 3x(1 − y) + 3(1 − x)y − 4(1 − x)(1 − y) = = −12xy + 7x + 7y − 4.

x = 7/12

y = 7/12

S(7/12, 7/12) = **1/12** – соответствует практическому результату