ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»**

Департамент прикладной математики

Дисциплина: «Введение в специальность»

Направление подготовки: 0.1.03.04 «Прикладная математика»

Факультет: Московский институт электроники и математики им. А.Н. Тихонова

Учебный 2023/2024 год, 2 семестр

Проектная работа на тему:

«ОПТИМИЗАЦИЯ СТРАТЕГИИ ПРИНЯТИЯ РЕШЕНИЙ В УСЛОВИЯХ НЕОПРЕДЕЛЕННОСТИ С ИСПОЛЬЗОВАНИЕМ ГЕНЕТИЧЕСКОГО АЛГОРИТМА»

Выполнил:
студент группы БПМ232
Барбарич Е. И.

Руководитель:
преп.
Прытков Р. С.

Москва 2024

Содержание

Аннотация

В данной проектной работе рассматривается задача оптимизации стратегии принятия решений в условиях неопределенности. Для решения используется генетический алгоритм, основанный на принципах эволюции и естественного отбора, реализованный на языке Python 3.11. Также в работе реализован прототип игры для моделирования условий неопределенности. Проведен анализ влияния параметров алгоритма и случайности на его точность, получены оптимальные стратегии для игровой модели. Результаты сравниваются с теоретическим решением для оценки эффективности предложенного метода. Работа демонстрирует потенциал генетических алгоритмов в решении оптимизационных задач.

Введение

В современном мире принятие решений в условиях неопределенности окружает нас повсюду, от личных решений до стратегических решений в бизнесе и политике. Неопределенность может возникать из-за различных факторов, таких как изменчивость рынка, нестабильность политической обстановки, климатические изменения и другие. В подобных условиях традиционные методы принятия решений, основанные на определенных правилах и моделях, могут быть неэффективными.

Оптимизация стратегии принятия решений в условиях неопределенности требует разработки новых подходов и инструментов, среди которых выделим понятие генетического алгоритма.

Генетические алгоритмы являются эффективным инструментом, когда целевая функция не может быть задана в явном виде. Они основаны на принципах естественного отбора и генетики и позволяют находить оптимальные решения путем эволюционного поиска в пространстве возможных стратегий.

Целью данной проектной работы является оптимизация стратегии в условиях неопределенности, с использованием генетического алгоритма.

Для проведения вычисления, реализации игровой модели, построения алгоритма и последующей его эволюции взят язык программирования Python 3.11.

Актуальность генетического алгоритма заключается в решении NP-трудных задач [1], способности к самообучению (из этого следует адаптация к изменению условий), а также в его применимости к широкому кругу оптимизационных задач.

Теоретическая справка

Генетический алгоритм

Генетический алгоритм [2] - метод оптимизации, в основе которого лежит принцип эволюционной теории Дарвина. Перечислим основные положения этого метода :

1. Создание начальной популяции стратегий-индивидуумов.
2. Оценка приспособленности каждого индивидуума.
3. Отбор: Выбор лучших решений для скрещивания и создания нового поколения.
4. Кроссинговер (скрещивание) и мутация: Применение функций кроссинговера (скрещивания) и мутации для создания новых стратегий.
5. Повторение процесса отбора, скрещивания и мутации.

Для имитации естественного отбора, вводится следующее понятие: Фитнес функция (функция приспособленности) - числовой показатель пригодности индивидуума (стратегии), в нашем случае он достаточно прост: мы имитируем игру некоторого числа раздач, используя нашу стратегию, а затем оставшийся банк делим на исходный.

Правила игры

Для раскрытия потенциала генетического алгоритма рассмотрим следующую карточную игру:

В этой игре задействованы два человека, игрок и дилер. Они получают по две карты, игрок видит обе свои карты и одну карту дилера. Игрок решает вытянуть ли новую карту, чтобы приблизиться к 21, или остаться с нынешним кол-вом карт. Если игрок превышает 21, то он проиграл, игрок набравший максимальное ≤ 21 становится победителем. Стоимость карт: карты с номиналами от 2 до 10 имеют стоимость, равную их номиналу. Валет, дама и король считаются за 10 очков, а туз может быть как 1, так и 11 очков в зависимости от ситуации. Действия дилера регламентированы: при общей стоимости карт < 17 дилер берет карту, иначе остается с нынешним кол-вом.

Также определим несколько вспомогательный понятий: рука - карты, которые находятся у игрока. Виды рук: жесткая - рука без туза, мягкая - с тузом, пара - с двумя одинаковыми картами.

Реализация генетического алгоритма

Влияние случайности в игре очевидно, и для определения числа раздач, на основе которых будет рассчитываться фитнес-функция, необходимо проанализировать влияние случайности на точность этой функции. Интуитивно понятно, что при тестировании стратегии на большом количестве раздач влияние случайности будет усреднено, и мы получим более точное представление о пригодности стратегии. Рассмотрим графики 1

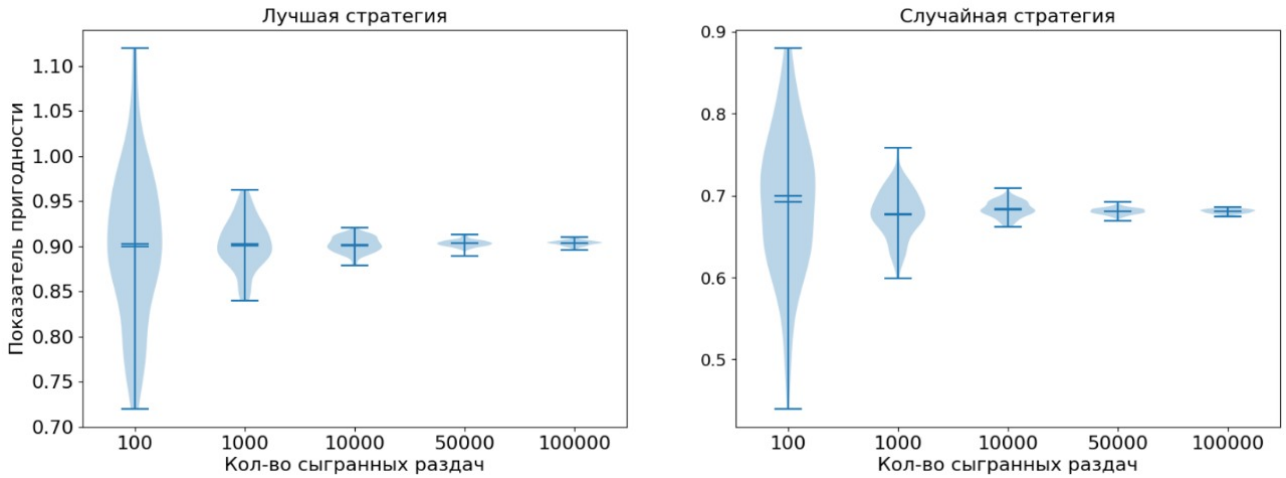

Рис. 1: Зависимость разброса показателя пригодности от числа раздач

Смотря на первые графики, мы видим, что оптимальная точность достигается уже для 50 000 сыгранных рук. Посмотрим на следующий график 2:

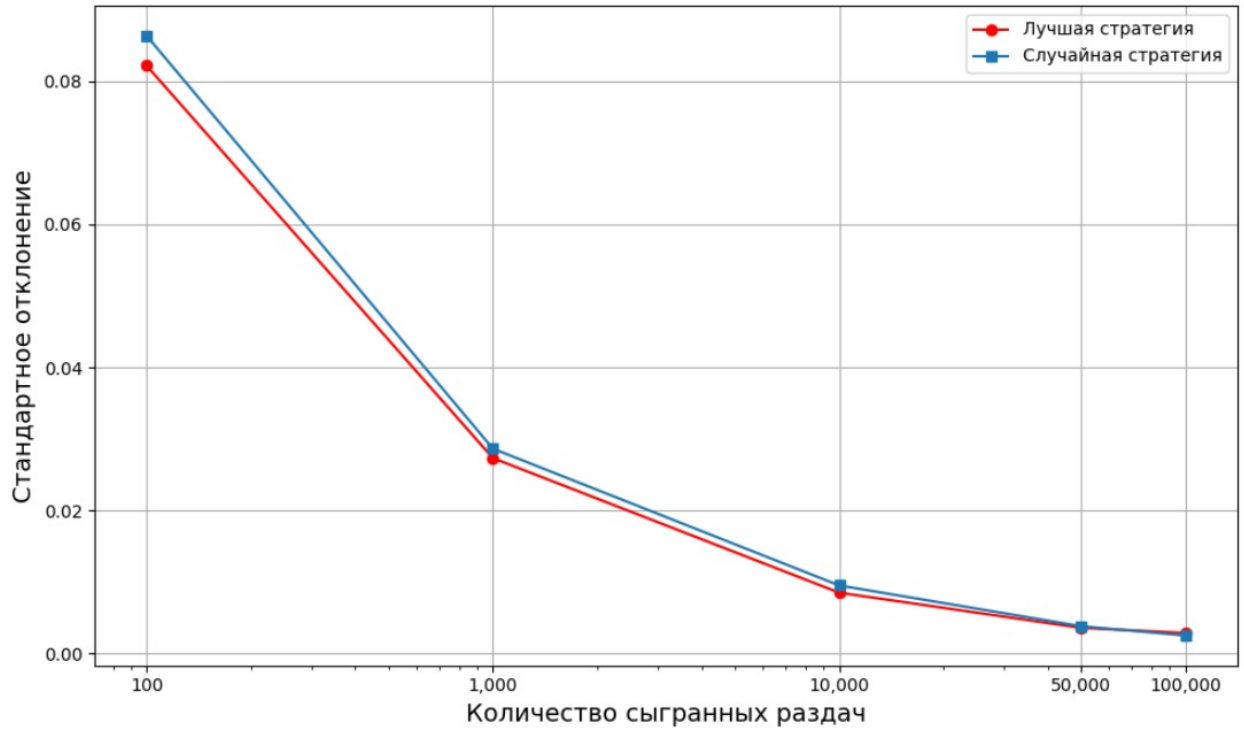

Рис. 2: Стандартное отклонение

Проанализировав графики выше, понимаем, для того чтобы уменьшить стандартное отклонение хотя бы в 2 раза, нам необходимо увеличить кол-во раздач более чем в два раза, поэтому для нашей задачи 50 000 игр дадут разумную точность вычислений и оптимальное вычислительное время.

Стратегия-индивидуум представляет собой таблицу 1 для жестких рук (16x10), где названия столбцов - открытая карта дилера, а названия строк - суммарное значение руки у игрока. На пересечении строк и столбцов стоят два действия "Hit" - взять карту и "Stay" - остаться с тем же набором карт, в таблице обозначены "h" и "s" соответственно.

	2	3	4	5	6	7	8	9	10	11
5	h	h	s	h	s	h	s	h	s	h
6	h	h	s	h	h	s	h	s	s	s
7	s	s	h	s	s	s	h	s	h	s
8	s	h	h	h	h	s	h	s	s	s
9	s	h	s	h	h	h	s	h	s	s
10	h	h	h	h	h	h	h	s	s	s
11	s	h	h	h	s	h	h	s	s	s
12	h	h	h	s	s	s	h	s	s	s
13	h	h	h	s	h	s	s	h	s	s
14	h	s	s	s	s	h	h	h	h	s
15	s	s	h	s	s	s	s	s	h	h
16	h	s	s	h	s	s	s	h	s	s
17	s	s	s	h	s	h	s	s	h	s
18	h	s	s	s	s	s	h	s	h	s
19	s	s	s	h	s	h	s	h	s	s
20	s	s	s	s	s	s	s	s	s	s

Таблица 1: Стратегия

Выбор параметров и настроек генетического алгоритма широк, можно выбрать размер популяции, число поколений, фитнес функцию, способ отбора индивидуумов, кроссинговера, мутации и ее вероятности и прочие, но мы остановимся на следующих:

1. Размер популяции - 450 индивидуумов. (вычислено с помощью прямых тестов)

2. Число поколений - 300.

3. Вероятность мутации - 0.05.

4. Отбор для скрещивания (он же отбор лучших особей) - турнирный по два (отбираются 200 лучших, исходя из результата турнира).

5. Кроссинговер - скрещиваются два индивидуума по строчкам, получаем еще 200 индивидуумов.

6. Мутация - случайные 50 индивидуумов из 400 подвергаются мутации.

С помощью таких настроек генетического алгоритма мы гарантируем, что лучшие индивидуумы буду поддерживаться, а худшие не будут задействованы, причем мы не теряем генетическое разнообразие благодаря турнирному отбору и мутации, а также сохраняем баланс между производительностью, временем и точностью.

Анализ точности стратегий-индивидуумов

Для анализа точности стратегии мы будем использовать уже готовое аналитическое решение Эдварда Торпа, представленное в книге "Обыграй дилера"[3]. Т.к. он нашел оптимальную стратегию для игры в BlackJack, то в его стратегии для жесткой руки будет действие удвоение ставки (в таблице представлена как буква D от англ. Double), но наши правила немного отличаются, поэтому вместо удвоения ставки мы говорим просто о взятии дополнительной карты. В рамках нашей задачи это не сыграет существенной роли, ведь в оригинальный правилах при удвоении берется еще одна карта.

Спустя 30 часов непрерывного выполнения кода мы получаем 300 потенциальных стратегий. Для того чтобы найти самое оптимальное решение, воспользуемся графиком 3, показывающий зависимость максимальной и средней приспособленности от поколения:

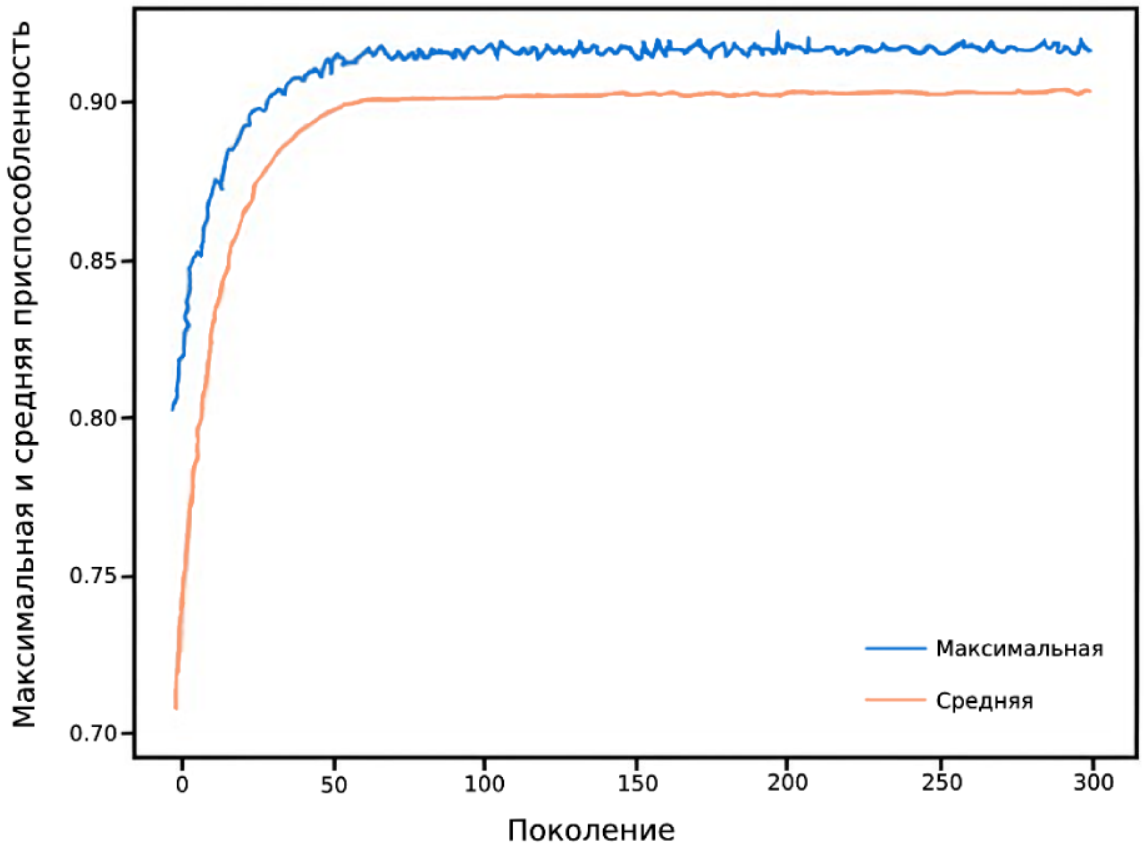

Рис. 3: Зависимость макс. и ср. приспособленности от поколения

Одна из самых интересных вещей в данной работе - это наблюдать за тем, как генетический алгоритм с каждым поколением улучшает стратегию.

Проанализировав график и перебрав некоторые скачки максимальной приспособленности, обратим внимание на лучших индивидуумов в поколении 202 4a и 298 4b:

	2	3	4	5	6	7	8	9	10	11
5	s	h	h	h	h	h	h	h	h	h
6	h	h	h	h	h	h	h	h	h	h
7	h	h	h	h	h	h	h	h	h	h
8	h	h	h	h	h	h	h	h	h	h
9	h	h	h	h	h	h	h	h	h	h
10	h	h	h	h	h	h	h	h	h	h
11	h	h	h	h	h	h	h	h	h	h
12	s	h	s	s	s	h	h	h	h	h
13	s	s	s	s	s	h	h	h	h	h
14	s	s	s	s	s	h	h	h	h	h
15	s	s	s	s	s	h	h	h	h	h
16	s	s	s	s	s	h	h	h	h	h
17	s	s	s	s	s	s	s	s	s	s
18	s	s	s	s	s	s	s	s	s	s
19	s	s	s	s	s	s	s	s	s	s
20	s	s	s	s	s	s	s	s	s	s

(a) Поколение 202

	2	3	4	5	6	7	8	9	10	11
5	h	h	h	h	h	h	h	h	h	h
6	h	h	h	h	h	h	h	h	h	h
7	h	h	h	h	h	h	h	h	h	h
8	h	h	h	h	h	h	h	h	h	h
9	h	h	h	h	h	h	h	h	h	h
10	h	h	h	h	h	h	h	h	h	h
11	h	h	h	h	h	h	h	h	h	h
12	h	h	s	s	s	h	h	h	h	h
13	s	s	s	s	s	h	h	h	h	h
14	s	s	s	s	s	h	h	h	h	h
15	s	s	s	s	s	h	h	h	h	h
16	s	s	s	s	s	h	h	h	h	h
17	s	s	s	s	s	s	s	s	s	s
18	s	s	s	s	s	s	s	s	s	s
19	s	s	s	s	s	s	s	s	s	s
20	s	s	s	s	s	s	s	s	s	s

(b) Поколение 298

Возвращаясь к графику мы видим, что стратегия из 202 поколения (совпадение с истинным решением = 0.9875) имеет более высокую приспособленность, чем стратегия из 298 (совпадение = 1.0), хотя последняя является решением нашей задачи, ведь полностью совпадает с теоретическим решением Эдварда Торпа. Почему же так? Все дело в случайности, хоть в начале работы мы поняли как уменьшить ее влияние, но все же мы не искоренили ее полностью. Возвращаясь к графикам зависимости разброса пригодности 1 мы можем предположить, что 100 000 раздач дадут нам такую точность, которая бы полностью исключила промахи, но в первую очередь мы искали компромисс между точностью и вычислительным временем, поэтому результаты нас более чем устраивают.

Заключение

В данной проектной работе с помощью языка программирования Python 3.11 была проведена оптимизация стратегии принятия решения в условиях неопределенности с использованием генетического алгоритма.

В результате данной оптимизации мы получили наилучшую стратегию, которая полностью идентична стратегии, полученной теоретически. Данный результат показывает, что генетический алгоритм - ценный инструмент в современном программировании.

Литература

[1] Томас Кормен, Чарльз Лейзерсон, Рональд Ривест, and Клиффорд Штайн. *Алгоритмы. Построение и анализ:[пер. с англ.]*. Издательский дом Вильямс, 2009.

[2] John H Holland. Genetic algorithms. *Scientific american*, 267(1):66–73, 1992.

[3] Edward O Thorp. *Beat the dealer: A winning strategy for the game of twenty-one*. Vintage, 2016.

Приложения

Реализация игры

[19]:

```
from itertools import product
import random

RANKS = {'2', '3', '4', '5', '6', '7', '8', '9', '10', 'jack', 'queen', 'king', 'ace'}
SUITS = {"\u2665", "\u2660", "\u2666", "\u2663"}


class GameForGA:

    def __init__(self):
        self.deck = self.create_deck()
        random.shuffle(self.deck)

        self.player_hand = []
        self.dealer_hand = []

    @staticmethod
    def create_deck():
        deck = []
        for suit, rank in product(SUITS, RANKS):
            deck.append([rank, suit])
        return deck

    def get_card(self):
        return self.deck.pop()

    def start_game(self):
        self.player_hand = [self.get_card(), self.get_card()]
        self.dealer_hand = [self.get_card(), self.get_card()]

    @staticmethod
    def hand_value(hand):
        value = 0
        aces = 0
        for card in hand:
            if card[0] == 'ace':
                value += 11
                aces += 1
            elif card[0].isdigit():
                value += int(card[0])
            else:
                value += 10
        while value > 21 and aces > 0:
            value -= 10
            aces -= 1
        return value

    @staticmethod
    def print_hand(hand):
        res = ""
        for card in hand:
            res += card[0] + ' ' + card[1] + ' '
        return res

    def player_action(self, action):
        if action == "h":
            self.player_hand.append(self.get_card())
        return self.game_status()

    def dealer_action(self):
        while self.hand_value(self.dealer_hand) < 17:
            self.dealer_hand.append(self.get_card())
            # print("Dealer hits card and has:", self.print_hand(self.dealer_hand), self.hand_value(self.dealer_hand))

    def game_status(self):
        player_value = self.hand_value(self.player_hand)
        if player_value > 21:
            # return "player BUST"
            return 'l'
        elif player_value == 21:
            # return "player BLACKJACK"
            return 'w'
        else:
            return "continue"

    def game_result(self):
        player_value = self.hand_value(self.player_hand)
        dealer_value = self.hand_value(self.dealer_hand)
```

```
        if player_value > 21:
            # return f"player have {player_value} \nLOSS"
            return 'l'
        elif dealer_value > 21 or player_value > dealer_value:
            # return f"player have {player_value} dealer have {dealer_value} \nWIN"
            return 'w'
        elif player_value == dealer_value:
            # return "equal"
            return 'e'
        else:
            # return f"player have {player_value} dealer have {dealer_value} \nLOSS"
            return 'l'
```

[20]:
```
def playGA(strategy):
    g = GameForGA()
    g.start_game()
    type_of_hand = 0
    help_type = 1
    # print("Dealer shows:", g.print_hand(g.dealer_hand[:1]))
    g.print_hand(g.dealer_hand[:1])
    status = "continue"
    while status == "continue":
        # print(g.print_hand(g.player_hand), 'your value:', g.hand_value(g.player_hand))
        g.print_hand(g.player_hand)
        g.hand_value(g.player_hand)

        if help_type == 1:
            if (g.player_hand[0][0] == 'ace' or g.player_hand[1][0] == 'ace'):
                type_of_hand = 1
            if (g.player_hand[0][0] == g.player_hand[1][0]):
                type_of_hand = 2
        help_type = 0
        if type_of_hand != 0:
            break
        if g.hand_value(g.player_hand) == 21 or g.hand_value(g.dealer_hand) == 21:
            # if g.hand_value(g.player_hand) != g.hand_value(g.dealer_hand):
            # print('BLACKJACK')
            status = 'BLACKJACK'

        if status == 'BLACKJACK':
            break
        # action = input("Enter an action (h/s): ") это для игры в ручную
        # action = strategy[type_of_hand][g.hand_value(g.dealer_hand[:1])][g.hand_value(g.player_hand)] для 3 раздач

        action = strategy[g.hand_value(g.player_hand) - 5][
            g.hand_value(g.dealer_hand[:1]) - 2]  # только для жесткой раздачи
        status = g.player_action(action)

        if action == "s":
            break
```

```
    if status == "continue":
        # print("Dealer has:", g.print_hand(g.dealer_hand), g.hand_value(g.dealer_hand))
        g.print_hand(g.dealer_hand)
        g.hand_value(g.dealer_hand)
        g.dealer_action()
    if type_of_hand == 0:
        return g.game_result()
    else:
        return 'e'  # сделали так, чтобы рассматривать только жесткие раздачи (не меняем остаток банка)
#основная часть закомментированного кода - реализация пользовательского интерфейса игры
```

Реализация генетического алгоритма

[21]:
```
import pandas as pd
import random
#from game import playGA
import matplotlib.pyplot as plt
import copy
import multiprocessing as mp

from loguru import logger


def creator_df():
    df_hard = pd.DataFrame(index=range(5, 21), columns=range(2, 12))
    # df_soft = pd.DataFrame(index=range(13, 21), columns=range(2, 12))
    # df_double = pd.DataFrame(index=range(2, 21, 2), columns=range(2, 12))
    for i in range(5, 21):
        for j in range(2, 12):
            df_hard.loc[i, j] = random.choice(['h', 's'])
    # for i in range(13, 21):
    #     for j in range(2, 12):
    #         df_soft.loc[i, j] = random.choice(['h', 's'])
    # for i in range(2, 21, 2):
    #     for j in range(2, 12):
    #         df_double.loc[i, j] = random.choice(['h', 's'])
    df_hard = df_hard.values.tolist()
    return df_hard  # , df_soft, df_double
#закомментированный код в creator_df - создание случайной стратегии для мягких и парных рук


class FitnessMax():
    def __init__(self):
        self.values = 0


class Individual(list):
    def __init__(self, *args):
        super().__init__(*args)
        self.fitness = FitnessMax()
```

```
def blackjack_fitness(individual):
    cash = 50000
    dep = 1
    for _ in range(50000):
        result = playGA(individual)
        if result == 'l':
            cash -= dep
        elif result == 'w':
            cash += dep
        else:
            cash += 0
    return cash / 50000


def individual_creator():
    return Individual(creator_df())


def population_creator(n):
    return [individual_creator() for i in range(n)]


def tournament(population):
    offspring = []
    for i in range(0, 450, 2):
        offspring.append(max([population[i], population[i + 1]], key=lambda ind: ind.fitness.values))
    sorted_offspring = sorted(offspring, key=lambda ind: ind.fitness.values, reverse=True)
    return sorted_offspring[:200]


def crossing(parent_1, parent_2):
    child_1 = copy.deepcopy(parent_1)
    child_2 = copy.deepcopy(parent_2)
    for i in range(len(child_1)):
        s = random.randint(len(child_1[0]) // 2 + 1, len(child_1[0]) - 1)
        p = random.randint(1, s)
        child_1[i][p:s], child_2[i][p:s] = child_2[i][p:s], child_1[i][p:s]

    return child_1, child_2
```

```python
def mutation(parent):
    mutant = copy.deepcopy(parent)
    for i in range(len(mutant)):
        for j in range(len(mutant[0])):
            if random.random() < 0.05:
                mutant[i][j] = 'h' if mutant[i][j] == 's' else 's'
    return mutant
```

[ ]:
```python
logger.info("Start. Create population")
population = population_creator(450)
logger.info("Population created")
generationCounter = 0

logger.info("Start BlackJackFitness")
fitnessValues = list(map(blackjack_fitness, population))
logger.info("End BlackJackFitness")

for individual, fitnessValue in zip(population, fitnessValues):
    individual.fitness.values = fitnessValue

maxFitnessValue = []
meanFitnessValue = []

fitnessValues = [individual.fitness.values for individual in population]

while generationCounter < 300:
    generationCounter += 1
    offspring = tournament(population)
    # print(len(offspring))

    for i in range(0, len(offspring), 2):
        cross_1, cross_2 = crossing(offspring[i], offspring[i + 1])
        offspring.append(cross_1)
        offspring.append(cross_2)

    # print(len(offspring))
    for i in range(0, len(offspring), 8):
        mutantik = mutation(offspring[i])
        offspring.append(mutantik)
    # print(len(offspring))
```

```python
    freshFitnessValues = list(map(blackjack_fitness, offspring))
    for individual, fitnessValue in zip(offspring, freshFitnessValues):
        individual.fitness.values = fitnessValue

    population[:] = offspring
    # print(len(population))
    fitnessValues = [ind.fitness.values for ind in population]

    maxFitness = max(fitnessValues)
    meanFitness = sum(fitnessValues) / len(population)
    maxFitnessValue.append(maxFitness)
    meanFitnessValue.append(meanFitness)
    print(f"Поколение {generationCounter}: Макс. приспособленность = {maxFitness}, Средняя = {meanFitness}")

    best_i = fitnessValues.index(max(fitnessValues))
    ans = [*population[best_i]]
    df_ans = pd.DataFrame(ans, index=range(5, 21), columns=range(2, 12))
    print('Лучший инидивид \n', df_ans, '\n')

plt.plot(maxFitnessValue, label='Максимальная')
plt.plot(meanFitnessValue, label='Средняя')
plt.xlabel('Поколение')
plt.ylabel('Максимальная и средняя приспособленность')
plt.legend()
plt.show()
```

Графики и вспомогательные функции

```
[ ]: import pandas as pd
     def creatorDF():
         df_hard = pd.DataFrame(index=range(5, 21), columns=range(2, 12))
         # df_soft = pd.DataFrame(index=range(13, 21), columns=range(2, 12))
         # df_double = pd.DataFrame(index=range(2, 21, 2), columns=range(2, 12))
         for i in range(5, 21):
             for j in range(2, 12):
                 df_hard.loc[i, j] = random.choice(['h', 's'])
         # for i in range(13, 21):
         #     for j in range(2, 12):
         #         df_soft.loc[i, j] = random.choice(['h', 's'])
         # for i in range(2, 21, 2):
         #     for j in range(2, 12):
         #         df_double.loc[i, j] = random.choice(['h', 's'])
         df_hard = df_hard.values.tolist()
         return df_hard #, df_soft, df_double
```

```
[ ]: def blackjack_fitness(individual, value):
         cash = value
         dep = 1
         for _ in range(value):
             result = playGA(individual)
             if result == 'l':
                 cash -= dep
             elif result == 'w':
                 cash += dep
             else:
                 cash += 0
         return cash/value
```

```
[ ]: df1 = creatorDF()
     df = [['h' ,'h', 'h' ,'h', 'h' ,'h', 'h' , 'h', 'h', 'h'],
           ['h' ,'h', 'h' ,'h', 'h' ,'h', 'h' , 'h', 'h', 'h'],
           ['h' ,'h', 'h' ,'h', 'h' ,'h', 'h' , 'h', 'h', 'h'],
           ['h' ,'h', 'h' ,'h', 'h' ,'h', 'h' , 'h', 'h', 'h'],
           ['h' ,'h', 'h' ,'h', 'h' ,'h', 'h' , 'h', 'h', 'h'],
           ['h' ,'h', 'h' ,'h', 'h' ,'h', 'h' , 'h', 'h', 'h'],
           ['h' ,'h', 'h' ,'h', 'h' ,'h', 'h' , 'h', 'h', 'h'],
           ['h' ,'h', 's' ,'s', 's' ,'h', 'h' , 'h', 'h', 'h'],
           ['s' ,'s', 's' ,'s', 's' ,'h', 'h' , 'h', 'h', 'h'],
           ['s' ,'s', 's' ,'s', 's' ,'h', 'h' , 'h', 'h', 'h'],
           ['s' ,'s', 's' ,'s', 's' ,'h', 'h' , 'h', 'h', 'h'],
           ['s' ,'s', 's' ,'s', 's' ,'h', 'h' , 'h', 'h', 'h'],
           ['s' ,'s', 's' ,'s', 's' ,'s', 's' , 's', 's', 's'],
           ['s' ,'s', 's' ,'s', 's' ,'s', 's' , 's', 's', 's'],
           ['s' ,'s', 's' ,'s', 's' ,'s', 's' , 's', 's', 's'],
           ['s' ,'s', 's' ,'s', 's' ,'s', 's' , 's', 's', 's']]
```

```
[ ]: import matplotlib.pyplot as plt


     arr100 = [blackjack_fitness(df, 100) for i in range(100)]
     arr1000 = [blackjack_fitness(df, 1000) for _ in range(100)]
     arr10000 = [blackjack_fitness(df, 10000) for _ in range(100)]
     arr50000 = [blackjack_fitness(df, 50000) for _ in range(100)]
     arr100000 = [blackjack_fitness(df, 100000) for _ in range(100)]

     arr1_100 = [blackjack_fitness(df1, 100) for i in range(100)]
     arr1_1000 = [blackjack_fitness(df1, 1000) for _ in range(100)]
     arr1_10000 = [blackjack_fitness(df1, 10000) for _ in range(100)]
     arr1_50000 = [blackjack_fitness(df1, 50000) for _ in range(100)]
     arr1_100000 = [blackjack_fitness(df1, 100000) for _ in range(100)]

     fig, (ax1, ax2) = plt.subplots(nrows=1, ncols=2, figsize=(16, 6))
```

```
ax1.violinplot([arr100, arr1000, arr10000, arr50000, arr100000], showmeans=True, showmedians=True)
ax1.set_xticks([1, 2, 3, 4, 5])
ax1.set_xticklabels(['100', '1000', '10000', '50000', '100000'])
ax1.set_ylabel('Показатель пригодности')
ax1.set_xlabel('Кол-во сыгранных раздач')
ax1.set_title('Лучшая стратегия')

ax2.violinplot([arr1_100, arr1_1000, arr1_10000, arr1_50000, arr1_100000], showmeans=True, showmedians=True)
ax2.set_xticks([1, 2, 3, 4, 5])
ax2.set_xticklabels(['100', '1000', '10000', '50000', '100000'])
#ax2.set_ylabel('Показатель пригодности')
ax2.set_xlabel('Кол-во сыгранных раздач')
ax2.set_title('Случайная стратегия')

plt.show()
```

```
[ ]: import numpy as np
std = [np.std(arr100), np.std(arr1000),np.std(arr10000),np.std(arr50000), np.std(arr100000)]
std1 = [np.std(arr1_100), np.std(arr1_1000),np.std(arr1_10000),np.std(arr1_50000), np.std(arr1_100000)]
val = [100, 1000, 10000, 50000, 100000]

plt.figure(figsize=(10, 6))
plt.plot(val, std, marker='o', label= 'Лучшая стратегия', color = 'r')
plt.plot(val, std1, marker='s', label= 'Случайная стратегия')
plt.grid(True)
plt.legend()

plt.xscale('log', base=10)


plt.xticks(val, [f'{v:,}' for v in val], rotation=0)
plt.xlabel('Количество сыгранных раздач', fontsize=14)
plt.ylabel('Стандартное отклонение', fontsize=14)
plt.tight_layout()
plt.show()
```

$$\int_1^x \frac{e^{\sqrt{t}}}{t^2}\mathrm{d}t = \left\langle \begin{array}{cc} \sqrt{t}=z & \mathrm{d}t = 2z\mathrm{d}z \\ t=z^2 & t=x, z=\sqrt{x} \end{array} \right\rangle = \int_1^{\sqrt{x}} \frac{e^z 2z}{z^4}\mathrm{d}z = 2\int_1^{\sqrt{x}} \frac{e^z}{z^3}\mathrm{d}z =$$

$$= \left\langle \begin{array}{cc} u = \frac{1}{z^3} & \mathrm{d}u = -\frac{3\mathrm{d}z}{z^4} \\ \mathrm{d}v = e^z\mathrm{d}z & v = e^z \end{array} \right\rangle = \frac{2e^z}{z^3}\Bigg|_1^{\sqrt{x}} + 6\int_1^{\sqrt{x}} \frac{e^z}{z^4}\mathrm{d}z =$$

$$= \left\langle \begin{array}{cc} u = \frac{1}{z^4} & \mathrm{d}u = -\frac{4\mathrm{d}z}{z^5} \\ \mathrm{d}v = e^z\mathrm{d}z & v = e^z \end{array} \right\rangle = \frac{2e^z}{z^3}\Bigg|_1^{\sqrt{x}} + \frac{6e^z}{z^4}\Bigg|_1^{\sqrt{x}} + 24\int_1^{\sqrt{x}} \frac{e^z}{z^5}\mathrm{d}z$$

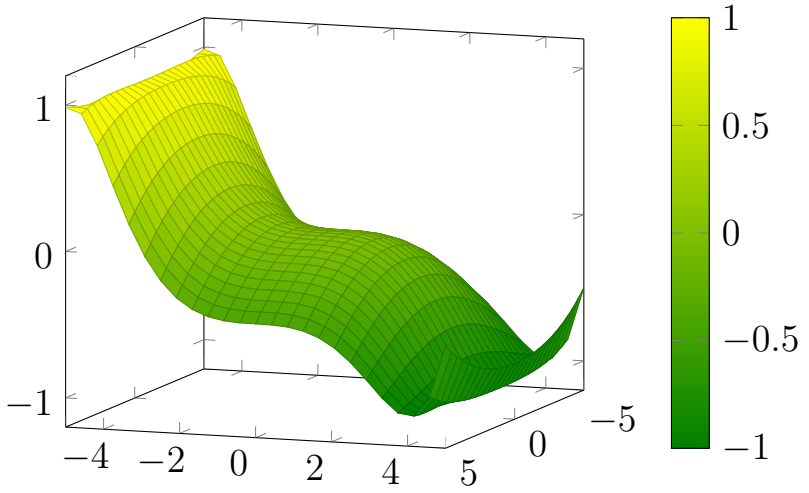

Рис. 5: График $-sin(x^2 + y^3)$