Липецкий государственный технический университет

Факультет автоматизации и информатики

Кафедра автоматизированных систем управления

ИНДИВИДУАЛЬНОЕ ДОМАШНЕЕ ЗАДАНИЕ

по дисциплине «Прикладные интеллектуальные системы и экспертные системы»

Методы искусственного интеллекта. Генетические алгоритмы.

Студент Сухоруких А.О.

Группа М-ИАП-22

Руководитель Кургасов В.В.

доцент

Липецк 2022 г.

Задание кафедры

Разработать генетический алгоритм для поиска минимума функции Бута

$$f(x_1, x_2) = (x_1 + x_2 - 7)^2 + (2x + y - 5)^2$$

Глобальный минимум находится в точке $f(1, 3) = 0$

$$-10 <= x_i <= 10$$

Ход работы

1 Теоретическая часть

Генетический алгоритм (англ. genetic algorithm) — это эвристический алгоритм поиска, используемый для решения задач оптимизации и моделирования путем последовательного подбора, комбинирования и вариации искомых параметров с использованием механизмов, напоминающих биологическую эволюцию. Является разновидностью эволюционных вычислений (англ. evolutionary computation). Отличительной особенностью генетического алгоритма является акцент на использование оператора «скрещивания», который производит операцию рекомбинации решений-кандидатов, роль которой аналогична роли скрещивания в живой природе.

Постановка задачи

Для функции приспособления $W(x)$ в пространстве поиска X требуется найти $x\ *= argmaxW(x)$ или $x\ *= argminW(x)$.

Описание алгоритма

1. Случайным образом генерируется конечный набор пробных решений: $P^1 = \{p_1^1 ... p_n^1\},\ p_i^1 \in X$ (первое поколение, n - размер популяции).

2. Оценка приспособленности текущего поколения: $F^k = \{f_1^k ... f_n^k\},\ f_i^k = W(p_i^k)$

3. Выход, если выполняется критерий останова, иначе

4. Генерация нового поколения посредством операторов селекции S, скрещивания C и мутаций M: $P^{k+1} = M \cdot C \cdot S(P^k, F^k)$ и переход к пункту 2.

Процессе селекции отбирают только несколько лучших пробных решений, остальные далее не используются. Скрещивание за место пары решений создаёт другую, элементы которой перемешаны каким-то особым образом. Мутация случайным образом меняет какую-нибудь компоненту пробного решения на иную.

Иные обозначения

Несмотря на внушительный возраст, в генетических алгоритма до сих пор используют различную терминологию, проистекающей как из генетики, так и из кибернетики.

Встречаются такие обозначения:

- Функция приспособленности (Fitness) $W(x)$ - целевая функция;
- Особь - пробное решение $p_i^k \in X$;
- Популяция - все поколения, вносящие вклад в последующее.

Чаще всего поколение и популяция - синонимы;

- Ген - компонент вектора x пространства поиска X;
- Скрещивание - кроссинговер (crossingover).

Применение генетических алгоритмов

Генетические алгоритмы применяются для решения следующих задач:

Оптимизация функций

Оптимизация запросов в базах данных

Разнообразные задачи на графах (задача коммивояжёра, раскраска, нахождение паросочетаний)

Настройка и обучение искусственной нейронной сети

Задачи компоновки

Составление расписаний

Игровые стратегии

Теория приближений

Искусственная жизнь

Биоинформатика (свёртывание белков)

Кодирование пространства поиска

В ГА часто используют следующие типы кодирования компонент пространства поиска:

Бинарный, если признак сам по себе является бинарным;

Численный в двоичной системе. Расширенный вариант бинарного, где используется фиксированное число бит.

Кодирование кодом Грея. Избавляет от проблем предыдущего варианта, но добавляет накладные расходы на кодирование/декодирование;

С небинарными операторами скрещивания и мутаций:

Числа с плавающей запятой. Используются в том случае, когда масштаб изменения признака заранее не известен;

Номинальные типы и более абстрактные сущности;

Типы с автоподстройкой...

Начальная популяция

Начальная популяция генерируется обычно случайно. Единственный критерий - достаточное разнообразие особей, чтобы популяция не свалилась в ближайший экстремум.

Оценка приспособленности

Оценка приспособленности часто проводится в две стадии. Первая - собственно оценка: $F^k = \{f_1^k \dots f_n^k\}, \ f_i^k = W(p_i^k)$. Вторая - дополнительные преобразования. Например, ею может быть нормировка к виду $F^{k'} = \{f_1^{k'} \dots f_n^{k'}\}, \ f_i^{k'} = (f_i^k - f_0)/(f_1 - f_0)$, где f_1 и f_0, соответственно, лучший и худший показатели в текущей популяции.

Оператор отбора (селекции)

На этом этапе отбирается оптимальная популяция для дальнейшего размножения. Обычно берут определённое число лучших по приспособленности. Имеет смысл также отбрасывать "клонов", т.е. особей с одинаковым набором генов.

Оператор скрещивания

Чаще всего скрещивание производят над двумя лучшими особями. Результатом является также обычно две особи с компонентами, взятыми от их родителей. Цель этого оператора - распространение хороших генов по

популяции и стягивание плотности популяции к тем областям, где она и так велика в том предположении, что "нас много там, где хорошо".

- В одноточечном варианте, результатом скрещивания родителей $p_i^k = \{a_1, a_2, \ldots a_n\},\ p_j^k = \{b_1, b_2, \ldots b_n\} \in P^k$ в k -ой популяции станут два элемента популяции $k+1$, такие что $p_i^{k+1} = \{a_1, \ldots, a_c, b_{c+1}, \ldots, b_n\},\ p_j^{k+1} = \{b_1, \ldots, b_c, a_{c+1}, \ldots a_n\} \in P^{k+1}$, где точка c выбирается случайно. В двухточечном варианте, соответственно, точек пересечения будет две, и они также выбираются случайно. Легко расширить эту конструкцию и до n точек. Нужно заметить, что в случае нечётного n, происходит n+1-точечный кроссинговер с n+1-ой точкой между последней и первой компонентами.

- Скрещиванием с маской является результат в виде двух потомков с компонентами, принадлежность которых определяется по битовой маске. Т.е. результатом скрещивания родителей $p_i^k = \{a_1, a_2, \ldots, a_n\},\ p_j^k = \{b_1, b_2, \ldots, b_n\} \in P^k$ в k -ой популяции станут два элемента популяции $k+1$, такие что $p_i^{k+1} = \{c_1, \ldots, c_n\},\ p_j^{k+1} = \{d_1, \ldots, d_n\} \in P^{k+1}$, где $c_i = a_i$ при $m_i = 0$ и $c_i = b_i$ при $m_i \neq 0$, и противоположные условия для второго отпрыска. Маска выбирается случайно. Для простоты, ею может быть третья особь.

- В непрерывном пространстве можно ввести такую аналогию для скрещивания: $P^{k+1}(x) = \int_X dy P^k(x) P^k(y) \exp\left[-\frac{(x-y)^T M_c (x-y)}{\rho(x,y)}\right]$, где $P^k(x)$ - плотность генофонда к-ой популяции, $\rho(x,y)$ - расстояние между двумя особями с генами x и y, M_c - матрица силы скрещивания.

Оператор мутаций

Оператор мутаций просто меняет произвольное число элементов в особи на другие произвольные. Фактически он является неким диссипативным элементом, с одной стороны вытягивающим из локальных экстремумов, с другой - приносящим новую информацию в популяцию.

- Инвертирует бит в случае бинарного признака.
- Изменяет на некоторую величину числовой признак. Причём, скорее на ближайший.
- Заменит на другой номинальный признак.
- В непрерывном пространстве можно ввести следующую аналогию:

$$P^{k+1}(x) = \int_X dy P^k(y) \exp\left[-(x-y)^T M_m (x-y)\right]$$, где $P^k(x)$ - плотность генофонда к-ой популяции, M_c - матрица силы мутаций.

Критерии останова

- нахождение глобального, либо субоптимального решения;
- выходом на «плато»;
- исчерпание числа поколений, отпущенных на эволюцию;
- исчерпание времени, отпущенного на эволюцию;
- исчерпание заданного числа обращений к целевой функции.

Эвристики

Генетические алгоритмы богаты возможностями встраивания различных эвристик. До сих пор не существует (и не будет!) точных критериев оптимального размера популяции, способов мутаций и скрещивания, выбора начальной популяции и т.п.

Плоидность

Каждая особь состоит не из одного, а нескольких пробных решений. Каждый кратный элемент пробного решения имеет активный(доминантный) или неактивный(рецессивный) статус, и, тем самым, проявляет или не проявляет (или же проявляет с определённой интенсивностью) себя при

вычислении целевой функции. Кратность пробных решений в особи называется плоидностью (2 - диплоидный набор, 3 - триплоидный, n - n-плоидный набор).

Преимущества:

Вытягивает популяцию из локального экстремума, т.к:

Поддерживает генетическое разнообразие, не позволяет вырождаться;

Позволяет естественным путём воссоздать аналог инцеста.

Недостатки:

Усложнение алгоритма;

Требует большего числа итераций до схождения в экстремум.

Преимущества ГА

Большое число свободных параметров, позволяющим эффективно встраивать эвристики;

Эффективное распараллеливание;

Работает заведомо не хуже абсолютно случайного поиска;

Связь с биологией, дающая некоторую надежду на исключительную эффективность ГА в природе.

Недостатки ГА

Большое количество свободных параметров, которое превращает "работу с ГА" в "игру с ГА";

Недоказанность сходимости;

В простых целевых функциях (гладкие, один экстремум и т.п.) генетика всегда проигрывает по скорости простым алгоритмам поиска.

Пропорциональная селекция

Это наиболее часто используемый вид селекции в ГА

Пусть f_i - пригодность индивида i, $\overline{f} = \frac{1}{N} * \sum_i f_i$ - средняя пригодность популяции. Тогда в пропорциональной селекции индивид i выбирается для репродукции с вероятностью $p_i = \frac{f_i}{\sum_j f_j} = \frac{f_i}{\overline{f} * N}$.

Другая возможность - выбирать случайное вещественное число $r \in \left[0, \sum_{j=1}^{N} f_j\right]$ и отобрать того индивида, для которого $\sum_{j=1}^{i-1} f_j \leq r < \sum_{j=1}^{i} f_j$, где $\sum_{j=1}^{0} f_j = 0$. *Проблема:* такой алгоритм работает относительно медленно.

В равновесном ГА популяциям не разрешается увеличиваться или сокращаться, поэтому для репродукции отбирают N индивидов. Отсюда следует, что *ожидаемое число* копий каждого индивида в промежуточной популяции равно $N_i = p_i * N = \frac{f_i}{\overline{f}}$.

N_i обычно является вещественным числом. Реальное число копий (целое) может варьироваться около N_i.

Индивиды с пригодностью выше средней имеют более одной копии в промежуточной популяции, а индивиды ниже средней пригодности могут не иметь ни одной (в среднем).

Проблемы с пропорциональной селекцией:

Преждевременная сходимость:

Индивид с $f_i >> \overline{f}$, но $f_i << f_{max}$ был получен на ранних поколениях.

Так как $N_i >> 1$, гены такого индивида довольно быстро распространятся на всю популяцию.

В таком случае рекомбинация не может более производить новых индивидов (только мутация может) и $\overline{f} << f_{max}$ навсегда.

Стагнация:

Ближе к концу работы алгоритма все индивиды могут получить относительно высокую и примерно равную пригодность, т.е. $\forall i f_i \approx f_{max}$.

Тогда $N_1 \approx ... \approx N_N \approx 1$, что приводит к очень маленькому селективному давлению, т.е. наилучшее решение предпочитается лишь немного больше, чем наихудшее.

Замечание: обе проблемы могут быть преодолены применением методики *масштабирования пригодности* (будет объяснено позже).

Ранговая селекция

Индивиды сортируются (ранжируются) на основе их пригодности таким образом, чтобы $f_i \geq f_j$ для $i > j.$

Затем каждому индивиду назначается вероятность p_i быть отобранным, взятая из заданного распределения с ограничением $\sum_i p_i = 1$

Типичные распределения:

1. Линейное: $p_i = a * i + b(a < 0)$.
2. Отрицательное экспоненциальное: $p_i = a * exp(b * i + c)$. Это эквивалентно назначению первому индивиду вероятности p, второму - p_2, третьему - p_3, и т.д.

Преимущества:

1. Нет преждевременной сходимости, т.к. нет индивидов с $N_i >> 1$.
2. Нет стагнации, так как и к концу работы алгоритма $N1 \neq N2 \neq ...$.
3. Нет необходимости в *явном вычислении пригодности*, т.к. для упорядочения индивидов достаточно иметь возможность их попарного сравнения.

Недостатки: значительные накладные расходы на переупорядочивание и трудность теоретического анализа сходимости.

Замечание: ранговая селекция имеет мало общего с биологической.

Турнирная селекция

Для отбора индивида создается группа из $N(N \geq 2)$ индивидов, выбранных случайным образом

Индивид с наибольшей пригодностью в группе отбирается, остальные - отбрасываются (турнир)

Турнирную селекцию можно рассматривать как ранговую селекцию при наличии шума.

Преимущества:

1. Нет преждевременной сходимости
2. Нет стагнации
3. Не требуется глобальное переупорядочивание
4. Не требуется явное вычисление функции пригодности

Замечание: турнирная селекция имеет глубокие аналоги в биологии.

Элитарная селекция

Как минимум одна копия лучшего индивида популяции всегда проходит в следующее поколение

Преимущество:

гарантия сходимости, т.е. если глобальный максимум будет обнаружен, то ГА сойдется к этому максимуму.

Недостаток: большой риск захвата локальным минимумом. Альтернатива: сохранять лучшую найденную структуру в специальной ячейке памяти и в конце эксперимента использовать ее в качестве решения вместо лучшего в последнем поколении.

2 Блок-схема алгоритма

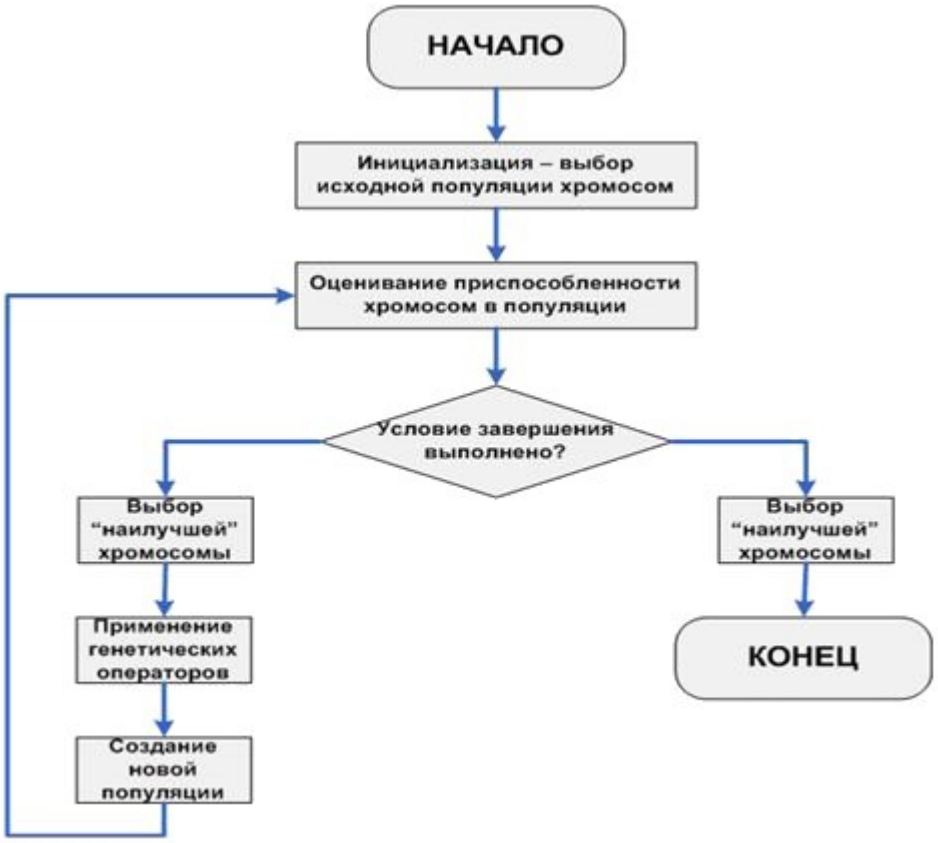


Рисунок 1 - Блок-схема генетического алгоритма

3 Код программы

Файл с настройками для генетического алгоритма и инициализация поиска минимума функции. Кол-во популяций 50, кол-во запуска алгоритма 50.

Методы селекции:

элитарный,

турнирный,

рулетка.

Методы кросенговера:

одноточечный,

двуточечный,

двойнойкросинговер(cxSimulatedBinaryBounded)

За основу был взять пакет deap. Ген - вектор значений индивидуума, обучаемый алгоритмом. Популяция - коллекция генов. Функция приспособленности - или целевая функция. Функция, которую мы

оптимизируем. Индивидуумы, для которых функция дает наилучшую оценку будут отобраны генетическим алгоритмом.

Класс creator. Метафабрика. позволяющая расширять существующие классы. Используется обычно для создания классов Fitness и Individual.

Fitness

В этом классе инкапсулированы значения приспособленности. Приспособленность в deap можно определять по нескольким компонентам (целями). у каждой из которых есть вес. Комбинация весов определяет поведение или стратегию приспособления в конкретной задаче

base.Fitness - абстрактный класс. содержащий кортеж weights. Чтобы определить стратегию, кортежу наджо присвоить значения.

С помощью этого класса определяются индивидуумы, образующие популяцию. В данном случае все гены индивидумов будут иметь тип лист, а класс каждого индивидума будет содержать экземляр FitnessMin.

Toolbox

Это контейнер для функций и операторов и позволяет создавать новые операторы путем назначения псевдонимов. Первым аргументом реигстрируем имя функциииии. Вторым передаем ей выполняемую функцию. Остальные рагументы - необязательны (аргументы выполняемой функциииии)

Код файла main.py

```
from deap import base, algorithms
from deap import creator
from deap import tools
import graycode

import algelitism

from prettytable import PrettyTable
```

```
import random
import matplotlib.pyplot as plt
import numpy as np
import time
import math

LOW, UP = -10, 10
ETA = 20
LENGTH_CHROM = 2  # длина хромосомы, подлежащей оптимизации

# константы генетического алгоритма
POPULATION_SIZE = 25   # количество индивидуумов в популяции
P_CROSSOVER = 0.2    # вероятность скрещивания
P_MUTATION_MAX = 0.8      # вероятность мутации индивидуума
P_MUTATION_MIN = 0.2      # вероятность мутации индивидуума
MAX_GENERATIONS = 50  # максимальное количество поколений
HALL_OF_FAME_SIZE = 25

def func(individial):
return
((individial[0]+2*individial[1]-7)**2+(2*individial[0]+individial[1]-5)**2),
#return (math.sin((individial[0]**2 +
individial[1]**2)**(0.5))**2)/((individial[0]**2 + individial[1]**2)**0.5),

def funcBin(individial):
point_1 = int(individial[0])
point_2 = int(individial[1])
return((point_1+2*point_2-7)**2+(2*point_1+point_2-5)**2),
```

```
#return (math.sin((point_1**2 + point_2**2)**(0.5))**2)/((point_1**2 + point_2**2))+100,


def metrics(minvalues):
count = 0
avg = 0
for i in minvalues:
        if i < 0.0001:
        count += 1
return [count/len(minvalues), len(minvalues) - count]



def show(ax, xgrid, ygrid, f):

ax.clear()
ax.contour(xgrid, ygrid, f)
ax.scatter(*zip(*population), color='black', s=3, zorder=1)

plt.draw()
plt.gcf().canvas.flush_events()

time.sleep(0.01)

def showBin(ax, xgrid, ygrid, f):

ax.clear()
ax.contour(xgrid, ygrid, f)
```

```
populationTmp = population
print(len(populationTmp))

for i in range(len(populationTmp)):
        populationTmp[i][0] = int(populationTmp[i][0])
        populationTmp[i][1] = int(populationTmp[i][1])
ax.scatter(*zip(*populationTmp), color='black', s=3, zorder=1)

plt.draw()
plt.gcf().canvas.flush_events()

time.sleep(0.01)

def randomPoint(a, b):
return [random.uniform(a, b), random.uniform(a, b)]

def randomPoint_bin(a, b):
return [random.randint(a, b), random.randint(a, b)]

if __name__ == '__main__':

hof = tools.HallOfFame(HALL_OF_FAME_SIZE)

RANDOM_SEED = 1245
random.seed(RANDOM_SEED)

creator.create("FitnessMin", base.Fitness, weights=(-1.0,))

creator.create("Individual", list, fitness=creator.FitnessMin)
```

```
best_num = []
best_bin = []
best_grey = []

table = PrettyTable()

table.field_names = ["Способ отбора родителей/Метод рекомбинации", "Тип рекомбинации", "Значение вероятности", "Тип числа", "Результат точка 1", "Результат точка 2", "Функция","Надежность", "Ср. число итераций"]

for i in range(3): ## Тип числа Вещ/Бин/Грея
    for j in range(3): ## Селекция
    for k in range(3): ## Кросс
        for m in range(2): ## Мутация
        toolbox = base.Toolbox()

        if i == 0:
        toolbox.register("randomPoint", randomPoint, LOW, UP)
        toolbox.register("individualCreator", tools.initIterate, creator.Individual, toolbox.randomPoint)
        toolbox.register("populationCreator", tools.initRepeat, list, toolbox.individualCreator)
        if i == 1:
        toolbox.register("randomPoint", randomPoint_bin, LOW, UP)
        toolbox.register("individualCreator", tools.initIterate, creator.Individual, toolbox.randomPoint)
        toolbox.register("populationCreator", tools.initRepeat, list, toolbox.individualCreator)
        if i == 2:
```

```
                toolbox.register("randomPoint", randomPoint_bin, LOW, UP)
                toolbox.register("individualCreator", tools.initIterate,
creator.Individual, toolbox.randomPoint)
                toolbox.register("populationCreator", tools.initRepeat, list,
toolbox.individualCreator)
                population =
toolbox.populationCreator(n=POPULATION_SIZE)

                if i == 0:
                toolbox.register("evaluate", func)
                else:
                toolbox.register("evaluate", funcBin)

                if j == 0:
                toolbox.register("select", tools.selBest)
                if j == 1:
                toolbox.register("select", tools.selRoulette)
                if j == 2:
                toolbox.register("select", tools.selTournament, tournsize=3)

                if k == 0:
                toolbox.register("mate", tools.cxOnePoint)
                if k == 1:
                toolbox.register("mate", tools.cxTwoPoint)
                if k == 2:
                toolbox.register("mate", tools.cxSimulatedBinaryBounded,
low=LOW, up=UP, eta=ETA)

                if m == 0:
```

```
                toolbox.register("mutate", tools.mutPolynomialBounded,
low=LOW, up=UP, eta=ETA, indpb=1.0/LENGTH_CHROM)
                if m == 1:
                toolbox.register("mutate", tools.mutPolynomialBounded,
low=LOW, up=UP, eta=ETA, indpb=1.0/LENGTH_CHROM)


                stats = tools.Statistics(lambda ind: ind.fitness.values)
                stats.register("min", np.min)
                stats.register("avg", np.mean)

                x = np.arange(-10, 10, 0.1)
                y = np.arange(-10, 10, 0.1)
                xgrid, ygrid = np.meshgrid(x, y)

                f_himmelbalu =
(1-np.sin((xgrid**2+ygrid**2)**0.5)**2)/(1+0.001*(xgrid**2+ygrid**2)) + 100


                algelitism.eaSimpleElitism
                algorithms.eaSimple
                if i == 0:
                try: population_1, logbook_1 =
algelitism.eaSimpleElitism(population, toolbox,
                                cxpb=P_CROSSOVER,
                                mutpb=P_MUTATION_MAX
if m==0 else P_MUTATION_MIN,
                                ngen=MAX_GENERATIONS,
                                halloffame=hof,
```

```
                                                stats=stats,
                                                #callback=(show, (ax, xgrid,
ygrid, f_himmelbalu)),
                                                verbose=True)
                        except:
                                continue
                        else:
                        try: population_1, logbook_1 =
algelitism.eaSimpleElitism(population, toolbox,
                                                cxpb=P_CROSSOVER,
                                                mutpb=P_MUTATION_MAX if
m==0 else P_MUTATION_MIN,
                                                ngen=MAX_GENERATIONS,
                                                halloffame=hof,
                                                stats=stats,
                                                #callback=(showBin, (ax, xgrid,
ygrid, f_himmelbalu)),
                                                verbose=True)
                        except:
                                continue

                        maxFitnessValues, meanFitnessValues, genValues =
logbook_1.select("min", "avg", "gen")

                        best = hof.items[0]
                        result_str = []
                        if j == 0:
                        result_str.append('Случайная')
                        if j == 1:
```

```
result_str.append('Рулетка')
if j == 2:
result_str.append('Турнир')
if k == 0:
result_str.append('Одноточечная')
if k == 1:
result_str.append('Двуточечная')
if k == 2:
result_str.append('cxSimulatedBinaryBounded ')
if m == 0:
result_str.append('Большая вероятность')
if m == 1:
result_str.append('Малая вероятность')
if i == 0:
result_str.append('Число')
result_str.append(best[0])
result_str.append(best[1])
result_str.append(func(best))
if i == 1:
result_str.append('Бин')
result_str.append(format(int(best[0]), '04b'))
result_str.append(format(int(best[1]), '04b'))
# result_str.append(bin(int(best[0]))[2:])
# result_str.append(bin(int(best[1]))[2:])
result_str.append(funcBin([best[0], best[1]]))
if i == 2:
result_str.append('Код грея')

result_str.append('{:03b}'.format(graycode.tc_to_gray_code(int(best[0]))))
```

```
result_str.append('{:03b}'.format(graycode.tc_to_gray_code(int(best[1]))))
                    result_str.append(funcBin([best[0], best[1]]))
                    coef = metrics(maxFitnessValues)
                    result_str.append(coef[0])
                    result_str.append(coef[1])

                    table.add_row(result_str)
        print(table)
        with open('res.txt', 'w') as file:
            file.write(str(table))
        from deap import tools
        from deap.algorithms import varAnd


        def eaSimpleElitism(population, toolbox, cxpb, mutpb, ngen, stats=None,
            halloffame=None, verbose=__debug__, callback=None):
        """Перелеланный алгоритм eaSimple с элементом элитизма
        """

        logbook = tools.Logbook()
        logbook.header = ['gen', 'nevals'] + (stats.fields if stats else [])

        # Evaluate the individuals with an invalid fitness
        invalid_ind = [ind for ind in population if not ind.fitness.valid]
        fitnesses = toolbox.map(toolbox.evaluate, invalid_ind)
        for ind, fit in zip(invalid_ind, fitnesses):
            ind.fitness.values = fit
```

```
record = stats.compile(population) if stats else {}
logbook.record(gen=0, nevals=len(invalid_ind), **record)
if verbose:
        print(logbook.stream)

# Begin the generational process
for gen in range(1, ngen + 1):
        # Select the next generation individuals
        offspring = toolbox.select(population,  int(len(population)/2))

        # Vary the pool of individuals
        offspring = varAnd(offspring, toolbox, cxpb, mutpb)

        # Evaluate the individuals with an invalid fitness
        invalid_ind = [ind for ind in offspring if not ind.fitness.valid]
        fitnesses = toolbox.map(toolbox.evaluate, invalid_ind)
        for ind, fit in zip(invalid_ind, fitnesses):
        ind.fitness.values = fit

        offspring.extend(halloffame.items)

        # Update the hall of fame with the generated individuals
        if halloffame is not None:
        halloffame.update(offspring)

        # Replace the current population by the offspring
        population[:] = offspring

        # Append the current generation statistics to the logbook
```

```
        record = stats.compile(population) if stats else {}
        logbook.record(gen=gen, nevals=len(invalid_ind), **record)
        if verbose:
        print(logbook.stream)

        if callback:
        callback[0](*callback[1])

    return population, logbook
```

4 Функции для оптимизации

Эффективность генетических алгоритмов зависит от выбора способов кроссинговера, мутации, выбора родительской пары, формирования новой популяции, а также от таких параметров как размер популяции, длина хромосомы. Оценить ГА можно с помощью разнообразных тестовых функций.

4.1 Функция для оптимизации

Для оптимизации возьмем функцию Бута. График функции представлен на рисунке 1.

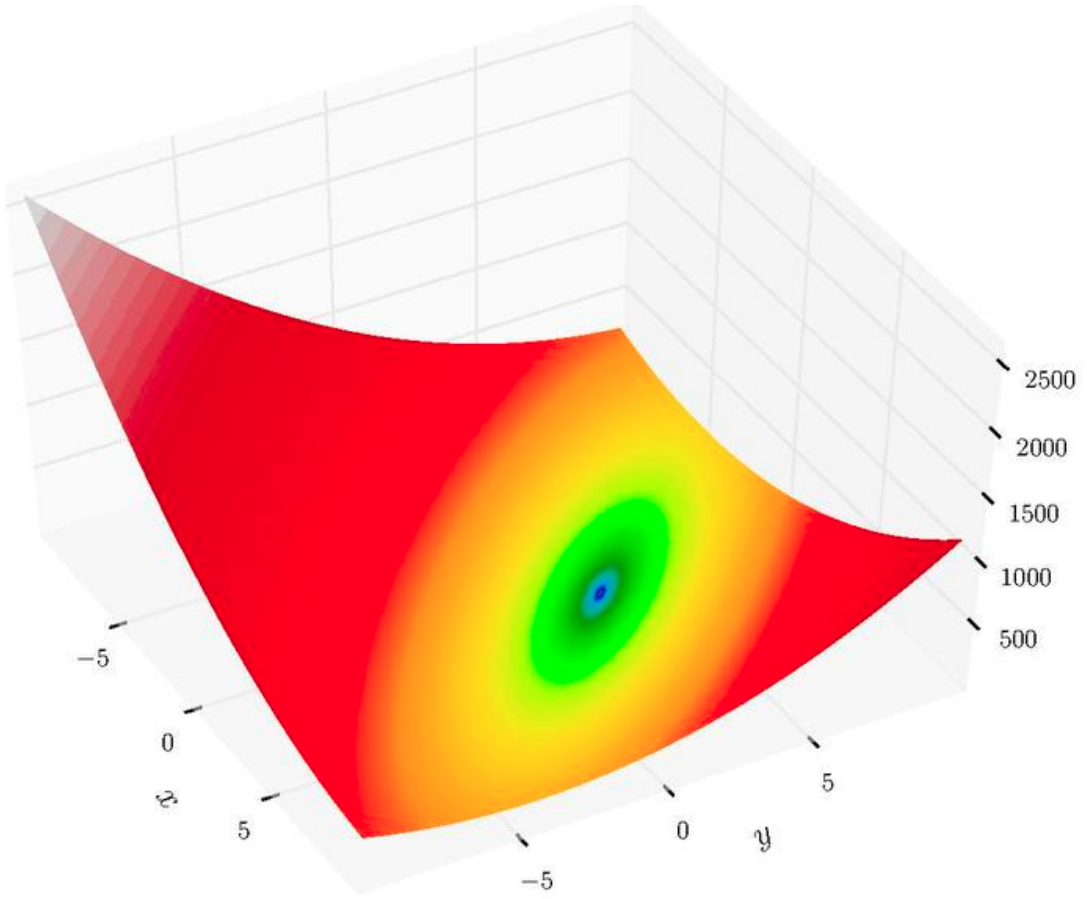


Рисунок 1 - График исходной функции

Данная функция имеет глобальный минимум в точке 1,3. Методы селекции:

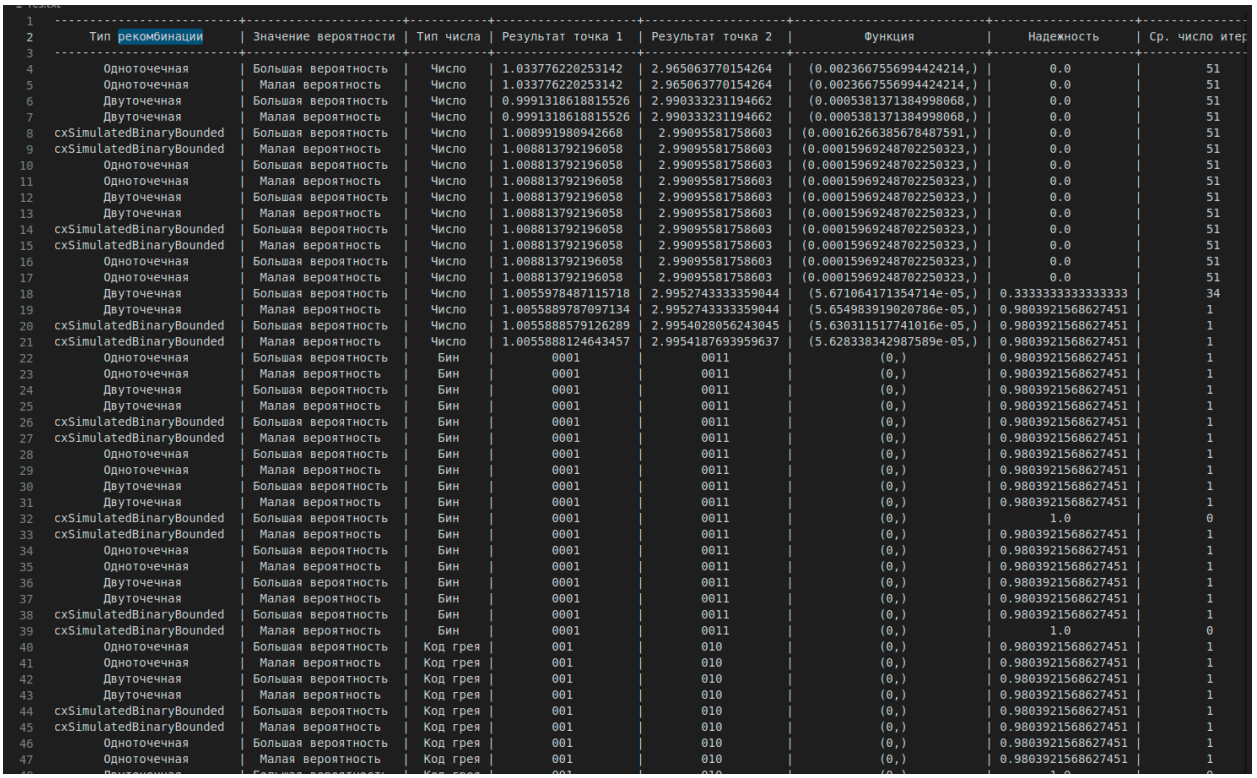
элитарный,

турнирный,

рулетка.

Минимальная вероятность мутации: 0,2. Максимальная вероятность мутации: 0,8. Методы рекомбинации: одноточечный, двухточечный и двоичный кроссовер.

В результате был проведен поиск минимума функции, с различными методами селекции. Всего тестовых вариантов генетического алгоритма 53 с различными методами селекции, вероятности мутации и кросенговера. Все варианты генетического алгоритма показали точный результат.



Тип рекомбинации	Значение вероятности	Тип числа	Результат точка 1	Результат точка 2	Функция	Надежность	Ср. число итер
Одноточечная	Большая вероятность	Число	1.033776220253142	2.965063770154264	(0.0023667556994424214,)	0.0	51
Одноточечная	Малая вероятность	Число	1.033776220253142	2.965063770154264	(0.0023667556994424214,)	0.0	51
Двуточечная	Большая вероятность	Число	0.9991318618815526	2.990333231194662	(0.0005381371384998068,)	0.0	51
Двуточечная	Малая вероятность	Число	0.9991318618815526	2.990333231194662	(0.0005381371384998068,)	0.0	51
cxSimulatedBinaryBounded	Большая вероятность	Число	1.008991980942668	2.99095581758603	(0.0001626638567487591,)	0.0	51
cxSimulatedBinaryBounded	Малая вероятность	Число	1.008813792196058	2.99095581758603	(0.0001596924870225032 3,)	0.0	51
Одноточечная	Большая вероятность	Число	1.008813792196058	2.99095581758603	(0.0001596924870225032 3,)	0.0	51
Одноточечная	Малая вероятность	Число	1.008813792196058	2.99095581758603	(0.0001596924870225032 3,)	0.0	51
Двуточечная	Большая вероятность	Число	1.008813792196058	2.99095581758603	(0.0001596924870225032 3,)	0.0	51
Двуточечная	Малая вероятность	Число	1.008813792196058	2.99095581758603	(0.0001596924870225032 3,)	0.0	51
cxSimulatedBinaryBounded	Большая вероятность	Число	1.008813792196058	2.99095581758603	(0.0001596924870225032 3,)	0.0	51
cxSimulatedBinaryBounded	Малая вероятность	Число	1.008813792196058	2.99095581758603	(0.0001596924870225032 3,)	0.0	51
Одноточечная	Большая вероятность	Число	1.008813792196058	2.99095581758603	(0.0001596924870225032 3,)	0.0	51
Одноточечная	Малая вероятность	Число	1.008813792196058	2.99095581758603	(0.0001596924870225032 3,)	0.0	51
Двуточечная	Большая вероятность	Число	1.0055978487115718	2.9952743333359044	(5.671064171354714e-05,)	0.3333333333333333	34
Двуточечная	Малая вероятность	Число	1.0055889787097134	2.9952743333359044	(5.654983919020786e-05,)	0.9803921568627451	1
cxSimulatedBinaryBounded	Большая вероятность	Число	1.0055888579126289	2.9954028056243045	(5.630311517741016e-05,)	0.9803921568627451	1
cxSimulatedBinaryBounded	Малая вероятность	Число	1.0055888124643457	2.9954187693959637	(5.628338342987589e-05,)	0.9803921568627451	1
Одноточечная	Большая вероятность	Бин	0001	0011	(0,)	0.9803921568627451	1
Одноточечная	Малая вероятность	Бин	0001	0011	(0,)	0.9803921568627451	1
Двуточечная	Большая вероятность	Бин	0001	0011	(0,)	0.9803921568627451	1
Двуточечная	Малая вероятность	Бин	0001	0011	(0,)	0.9803921568627451	1
cxSimulatedBinaryBounded	Большая вероятность	Бин	0001	0011	(0,)	0.9803921568627451	1
cxSimulatedBinaryBounded	Малая вероятность	Бин	0001	0011	(0,)	0.9803921568627451	1
Одноточечная	Большая вероятность	Бин	0001	0011	(0,)	0.9803921568627451	1
Одноточечная	Малая вероятность	Бин	0001	0011	(0,)	0.9803921568627451	1
Двуточечная	Большая вероятность	Бин	0001	0011	(0,)	0.9803921568627451	1
Двуточечная	Малая вероятность	Бин	0001	0011	(0,)	0.9803921568627451	1
cxSimulatedBinaryBounded	Большая вероятность	Бин	0001	0011	(0,)	1.0	0
cxSimulatedBinaryBounded	Малая вероятность	Бин	0001	0011	(0,)	0.9803921568627451	1
Одноточечная	Большая вероятность	Бин	0001	0011	(0,)	0.9803921568627451	1
Одноточечная	Малая вероятность	Бин	0001	0011	(0,)	0.9803921568627451	1
Двуточечная	Большая вероятность	Бин	0001	0011	(0,)	0.9803921568627451	1
Двуточечная	Малая вероятность	Бин	0001	0011	(0,)	0.9803921568627451	1
cxSimulatedBinaryBounded	Большая вероятность	Бин	0001	0011	(0,)	0.9803921568627451	1
cxSimulatedBinaryBounded	Малая вероятность	Бин	0001	0011	(0,)	1.0	0
Одноточечная	Большая вероятность	Код грея	001	010	(0,)	0.9803921568627451	1
Одноточечная	Малая вероятность	Код грея	001	010	(0,)	0.9803921568627451	1
Двуточечная	Большая вероятность	Код грея	001	010	(0,)	0.9803921568627451	1
Двуточечная	Малая вероятность	Код грея	001	010	(0,)	0.9803921568627451	1
cxSimulatedBinaryBounded	Большая вероятность	Код грея	001	010	(0,)	0.9803921568627451	1
cxSimulatedBinaryBounded	Малая вероятность	Код грея	001	010	(0,)	0.9803921568627451	1
Одноточечная	Большая вероятность	Код грея	001	010	(0,)	0.9803921568627451	1
Одноточечная	Малая вероятность	Код грея	001	010	(0,)	0.9803921568627451	1

Рисунок 1 - Результаты выполнения программы

Вывод

В ходе выполнения данного индивидуального домашнего задания мною были получены навыки разработки генетических алгоритмов. Была рассмотрена база построения генетических алгоритмов, различные методы селекций, типы рекомбинаций, кроссинговеров, и мутаций. Наилучшие данные были получены при представлении данных в бинарном виде или коде Грея.