МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

КАФЕДРА КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ
И ПРОГРАММНОЙ ИНЖЕНЕРИИ (КАФЕДРА №43)

ПРЕПОДАВАТЕЛЬ

профессор, д-р техн. наук, профессор		Ю.А. Скобцов
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ОТЧЁТ О ЛАБОРАТОРНОЙ РАБОТЕ № 4

ГЕНЕТИЧЕСКОЕ ПРОГРАММИРОВАНИЕ

по дисциплине: ЭВОЛЮЦИОННЫЕ МЕТОДЫ ПРОЕКТИРОВАНИЯ ПРОГРАММНО-ИНФОРМАЦИОННЫХ СИСТЕМ

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. _____ _____ _____
подпись, дата инициалы, фамилия

Санкт-Петербург
2024

1. Цель работы:

Решение задачи аппроксимации функции с помощью генетического программирования. Графическое отображение найденных решений.

2. Индивидуальное задание по варианту

Вариант 10

Ackley’s Path function

1. Разработать эволюционный алгоритм, реализующий ГП для нахождения заданной по варианту функции (таб. 4.1).

- Структура для представления программы – древовидное представление.
- Терминальное множество: переменные x1, x2, x3, …, xn, и константы в соответствии с заданием по варианту.
- Функциональное множество: +, -, *, /, abs(), sin(), cos(), exp(), возведение в степень,
- Фитнесс-функция – мера близости между реальными значениями выхода и требуемыми.

2. Представить графически найденное решение на каждой итерации.

3. Сравнить найденное решение с представленным в условии задачи.

Терминальное множество: x_1, x_2, x_3, x_4.

Константы: 10, n, 2, pi, -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5.

3. Краткие теоретические сведения

В генетическом программировании (ГП) в качестве особи выступает программа, представленная в определенном формате, которая решает некоторую задачу.

Программы составляются из переменных, констант и функций, которые связаны некоторыми синтаксическими правилами. Поэтому определяется терминальное множество, содержащее константы и переменные, и функциональное множество, которое состоит, прежде всего, из операторов и необходимых элементарных функций.

Древовидное представление

В качестве примера рассмотрим арифметическую формулу, которую удобно представлять деревом. Рассмотрим арифметическую формулу

$$\frac{d}{e} - a * (b + c)$$

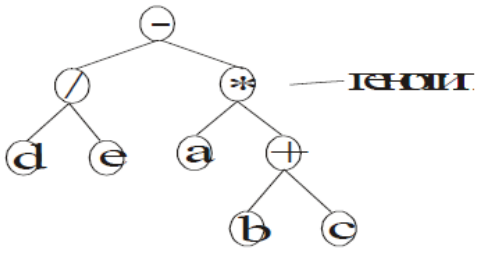


Рис.4.1 Древовидное представление формулы d/e-a*(b+c).

При этом листья дерева соответствуют терминалам, а внутренние узлы – функциям.

4. Листинг программы

Цель – минимизация суммы квадратов разности значений заданной функции и значений, построенных с помощью особи.

Метод eaSimpleWithElitism() аналогичен оригинальному методу eaSimple(), но теперь объект halloffame используется для реализации механизма элитизма. Индивидуумы, хранящиеся в объекте halloffame, просто копируются в следующее поколение, не подвергаясь воздействию операторов отбора, скрещивания и мутации. Для этого нужно внести следующие модификации: вместо того чтобы отбирать индивидуумов в количестве, равном размеру популяции, мы отбираем их меньше на столько, сколько индивидуумов находится в зале славы:

```
offspring = toolbox.select(population, len(population) - hof_size)
```

после применения генетических операторов индивидуумы добавляются из зала славы в популяцию:

```
offspring.extend(halloffame.items)
```

Генетические операторы:

В качестве оператора отбора (с псевдонимом select) используется турнирный отбор (размер турнира 2):

```
toolbox.register("select", tools.selTournament, tournsize=2)
```

В качестве оператора скрещивания (с псевдонимом mate) - специализированный для генетического программирования оператор cxOnePoint(), предоставляемый библиотекой deap. Поскольку

эволюционирующие программы представлены в виде деревьев, этот оператор принимает два родительских дерева и переставляет местами их участки, создавая два дерева-потомка:

```
toolbox.register("mate", gp.cxOnePoint)
```

Оператор мутации определяется в два этапа. Сначала определим вспомогательный оператор, пользующийся специальной функцией генетического программирования genGrow() из библиотеки deap. Этот оператор создает поддерево в рамках ограничений, заданных двумя константами. Затем определим сам оператор мутации, который заменяет случайно выбранное поддерево случайным же деревом, сгенерированным вспомогательным оператором:

```
toolbox.register("expr_mut", gp.genGrow, min_=MUT_MIN_TREE_HEIGHT,
max_=MUT_MAX_TREE_HEIGHT)
toolbox.register("mutate", gp.mutUniform, expr=toolbox.expr_mut, pset=primitiveSet)
```

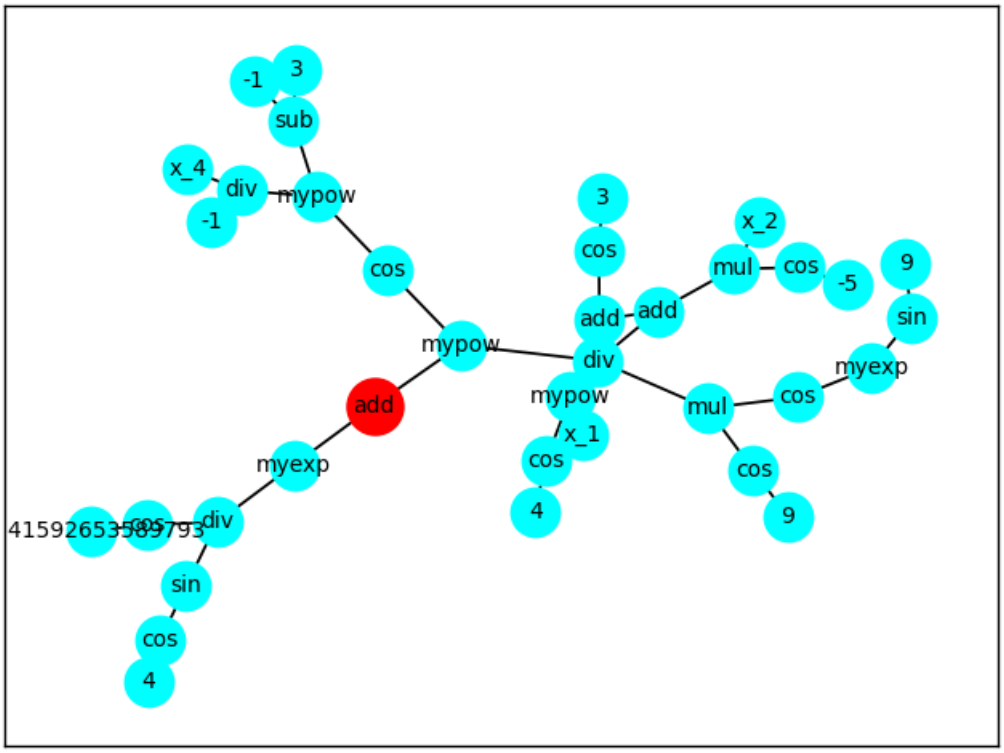
5. Результаты выполнения программы

При заданных значениях констант:

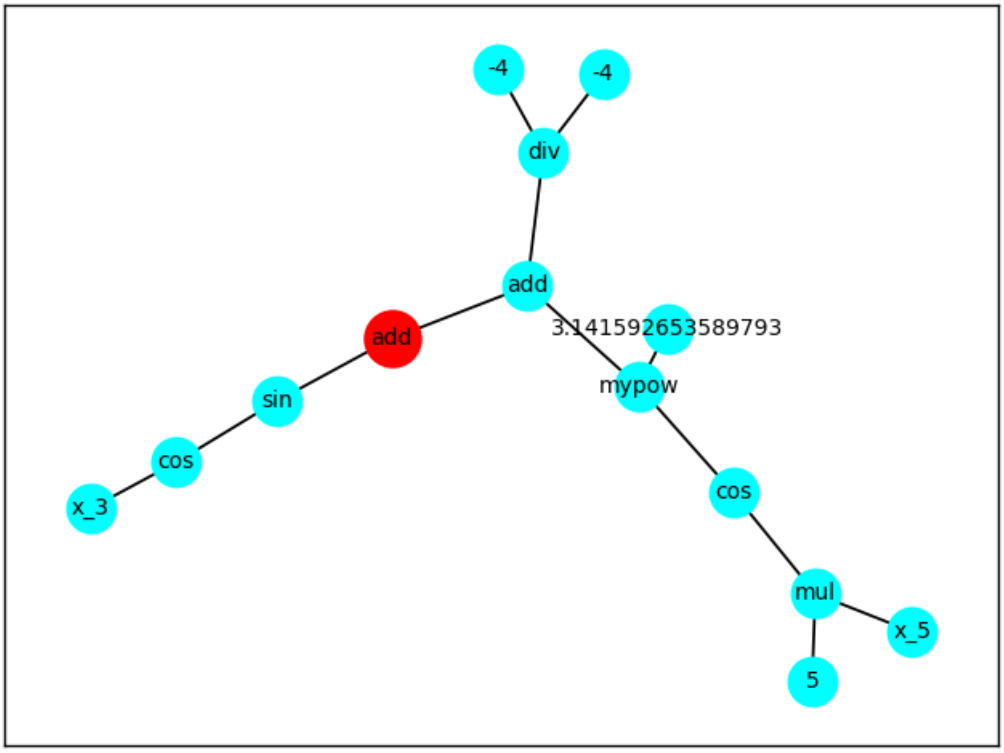
```
NUM_INPUTS = 9
NUM_OUTPUTS = 20


# Константы генетического алгоритма:
POPULATION_SIZE = 40
P_CROSSOVER = 0.5
P_MUTATION = 0.01
MAX_GENERATIONS = 30
HALL OF FAME SIZE = 10

# Константы, специфичные для генетического программирования:
MIN_TREE_HEIGHT = 3
MAX_TREE_HEIGHT = 5
LIMIT_TREE_HEIGHT = 17
MUT_MIN_TREE_HEIGHT = 0
MUT_MAX_TREE_HEIGHT = 2

# интервал
LEFT_BORDER_X1 = -1
RIGHT_BORDER_X1 = 1
LEFT_BORDER_X2 = -1
RIGHT_BORDER_X2 = 1
```

Результат:

Среднее поколение:



Лучшее решение:



Первое поколение

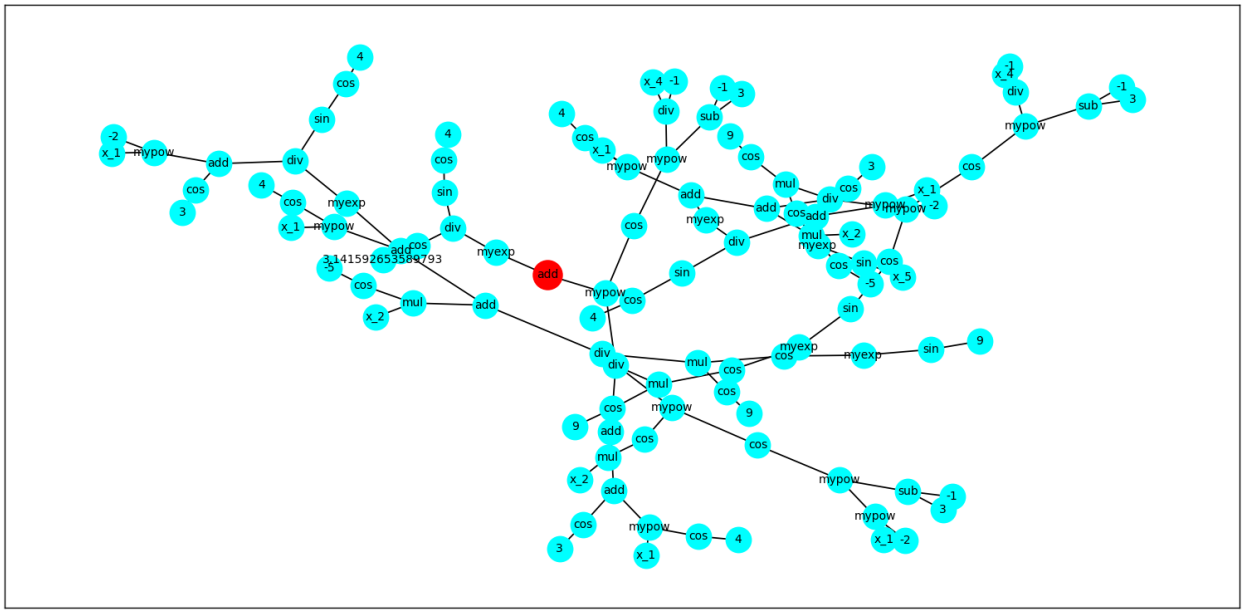


Таблица статистики (nevals – количество подсчетов фитнес-функции):

gen	nevals	min	avg
0	40	43.9625	1479
1	14	39.4132	62.1199
2	12	39.4132	58.9384
3	18	39.4132	54.1955
4	18	35.6539	4104.42
5	18	35.6539	50.9008
6	19	35.6539	63.1218
7	18	25.9465	6.76068e+06
8	12	25.9465	54.4872
9	8	22.6711	43.0262
10	12	22.6711	1.35213e+07
11	16	20.3477	40.6674
12	8	20.3477	44.4631
13	12	18.8725	397.624
14	18	18.8725	41.6662
15	16	18.8725	1.1118e+07
16	12	18.8725	25.4522
17	20	18.8725	38.4174
18	10	18.8515	25.0147
19	16	18.5009	35.6127
20	14	18.1927	31.5758
21	12	18.1927	37.5239
22	12	18.1927	35.6971
23	24	18.1927	37.9127
24	18	18.1927	26.8186
25	14	16.4794	24.2805
26	20	16.4794	26.5537
27	14	16.4794	23.1966
28	14	16.4794	20.522
29	16	16.4794	22.4912
30	16	16.4794	1.1118e+07

Таблица значений функций и ошибки:

Выход, ошибка в метрике абсолютных значений, ошибка квадратичная

```
0.475  2.572  6.614
0.713  2.863  8.195
1.772  1.667  2.779
2.564  0.634  0.401
2.612  0.750  0.562
3.109  0.499  0.249
3.204  0.380  0.144
3.653  0.549  0.302
3.733  0.295  0.087
3.985  0.396  0.157
```

Вывод:

```
-- Лучший индивидуум 1 поколения =  add(sin(cos(x_3)), add(div(-4, -4), mypow(cos(mul(x_5, 5)), 3.141592653589793)))
-- длина=14, высота=5
-- Лучшая приспособленность =  39.41315503399046
-- Лучшая ошибка =  39.363155033990466

-- Лучший индивидуум 15.0 поколения =
add(mypow(div(add(mul(cos(-5), x_2), add(mypow(cos(4), x_1), cos(3))), mul(cos(9), cos(myexp(sin(9))))),
cos(mypow(div(x_4, -1), sub(-1, 3)))), myexp(div(cos(3.141592653589793), sin(cos(4)))))
-- длина=37, высота=7
-- Лучшая приспособленность =  18.87249564933184
-- Лучшая ошибка =  18.80249564933184

-- Лучший индивидуум =  add(mypow(div(add(mul(cos(mypow(div(add(mul(cos(-5), x_2), add(mypow(cos(4), x_1),
myexp(div(add(mypow(-2, x_1), cos(3)), sin(cos(4))))))), mul(cos(9), cos(myexp(sin(9))))),
cos(mypow(mypow(-2, x_1), sub(-1, 3))))), x_2), add(mypow(cos(4), x_1), cos(3))), mul(cos(9),
 cos(myexp(sin(cos(mypow(div(add(mul(cos(-5), x_2), add(mypow(cos(4), x_1), myexp(div(add(mypow(-2, x_1), cos(3)),
  sin(cos(4))))))), mul(cos(9), cos(myexp(sin(x_5))))), cos(mypow(div(x_4, -1), sub(-1, 3))))))))))), cos(mypow(div(x_4, -1),
  sub(-1, 3)))), myexp(div(cos(3.141592653589793), sin(cos(4)))))
-- длина=112, высота=16
-- Лучшая приспособленность =  16.479358343038715
-- Лучшая ошибка (mnk) =  16.319358343038715
-- Лучшая ошибка (abs) =  24.67477242576978
```

Код программы

elitism.py

```python
from deap import tools
from deap import algorithms


def eaSimpleWithElitism(population, toolbox, cxpb, mutpb, ngen, stats=None,
                        halloffame=None, verbose=__debug__):
    """This algorithm is similar to DEAP eaSimple() algorithm, with the
modification that
    halloffame is used to implement an elitism mechanism. The individuals
contained in the
    halloffame are directly injected into the next generation and are not
subject to the
    genetic operators of selection, crossover and mutation.
    """
```

```
    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

    if halloffame is None:
        raise ValueError("halloffame parameter must not be empty!")

    halloffame.update(population)
    hof_size = len(halloffame.items) if halloffame.items else 0

    record = stats.compile(population) if stats else {}
    logbook.record(gen=0, nevals=len(invalid_ind), **record)
    if verbose:
        print(logbook.stream)
    sols = list()
    # Begin the generational process
    for gen in range(1, ngen + 1):

        # Select the next generation individuals
        offspring = toolbox.select(population, len(population) - hof_size)

        # Vary the pool of individuals
        offspring = algorithms.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

        # add the best back to population:
        offspring.extend(halloffame.items)

        # Update the hall of fame with the generated individuals
        halloffame.update(offspring)

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

        if gen == 1:
            sols.append(halloffame.items[0])

        if gen == ngen / 2:
            sols.append(halloffame.items[0])

        # 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)

    return population, logbook, sols
```

main.py

```
import random
import operator
import numpy as np
from deap import base
from deap import creator
from deap import tools
from deap import gp
from mpmath import *
import decimal
import itertools
import matplotlib.pyplot as plt
import networkx as nx
import elitism
import math

# количество входов программы контроля
NUM_INPUTS = 9
NUM_OUTPUTS = 20

# Константы генетического алгоритма:
POPULATION_SIZE = 40
P_CROSSOVER = 0.5
P_MUTATION = 0.01
MAX_GENERATIONS = 30
HALL_OF_FAME_SIZE = 10

# Константы, специфичные для генетического программирования:
MIN_TREE_HEIGHT = 3
MAX_TREE_HEIGHT = 5
LIMIT_TREE_HEIGHT = 17
MUT_MIN_TREE_HEIGHT = 0
MUT_MAX_TREE_HEIGHT = 2

# интервал
LEFT_BORDER_X1 = -1
RIGHT_BORDER_X1 = 1
LEFT_BORDER_X2 = -1
RIGHT_BORDER_X2 = 1

# константы по варианту
aa = 1
bb = 5.1 / (4 * math.pi ** 2)
cc = 5 / math.pi
dd = 6
ee = 10
ff = 1 / (8 * math.pi)

# скорость:
RANDOM_SEED = 42
toolbox = base.Toolbox()


def ackleys path function(x, y):
    return -20 * np.exp(-0.2 * np.sqrt(0.5 * (x**2 + y**2))) - np.exp(0.5 *
(np.cos(2 * math.pi * x) + np.cos(2 * math.pi * y))) + np.e + 20


# формирование таблицы
In = []
for i in range(NUM_OUTPUTS):
    ra = []
```

```
    x = [random.uniform(LEFT_BORDER_X1, RIGHT_BORDER_X1) for _ in
range(NUM_INPUTS)]
    ra.extend(x)
    In.append(ra)

# Update the output data using Ackley's Path Function
Out = [ackleys_path_function(x[0], x[1]) for x in In]

# вычисляем значения функции
# f1a(x) = sum(i * x(i)^2)
for x in In:
    fitness = sum(i * xi**2 for i, xi in enumerate(x))
    Out.append(fitness)
    Out.sort()

def div(a, b):
    if b == 0:
        return 0
    elif a == 0:
        return 0
    else:
        return a / b

def mypow(a, b):
    if isinstance(b, int):
        return pow(a, b)
    elif a == 0:
        return 0
    else:
        return a

def myexp(a):
    if isinstance(a, int):
        mp.dps = 300
        return exp(a)
    else:
        return a

primitiveSet = gp.PrimitiveSet("main", NUM_INPUTS, "x_")
primitiveSet.addPrimitive(operator.abs, 1)
primitiveSet.addPrimitive(operator.mul, 2)
primitiveSet.addPrimitive(operator.add, 2)
primitiveSet.addPrimitive(operator.sub, 2)
primitiveSet.addPrimitive(div, 2)
primitiveSet.addPrimitive(math.sin, 1)
primitiveSet.addPrimitive(math.cos, 1)
primitiveSet.addPrimitive(myexp, 1)
primitiveSet.addPrimitive(mypow, 2)
primitiveSet.addTerminal(2)
primitiveSet.addTerminal(10)
primitiveSet.addTerminal(math.pi)
primitiveSet.addTerminal(NUM_INPUTS)
primitiveSet.addTerminal(-1)
primitiveSet.addTerminal(-2)
primitiveSet.addTerminal(-3)
primitiveSet.addTerminal(-4)
primitiveSet.addTerminal(-5)
primitiveSet.addTerminal(1)
primitiveSet.addTerminal(2)
primitiveSet.addTerminal(3)
primitiveSet.addTerminal(4)
primitiveSet.addTerminal(5)

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

```
creator.create("Individual", gp.PrimitiveTree, fitness=creator.FitnessMin)
toolbox.register("expr", gp.genFull, pset=primitiveSet, min_=MIN_TREE_HEIGHT,
max_=MAX_TREE_HEIGHT)
toolbox.register("individualCreator", tools.initIterate, creator.Individual,
toolbox.expr)
toolbox.register("populationCreator", tools.initRepeat, list,
toolbox.individualCreator)
toolbox.register("compile", gp.compile, pset=primitiveSet)

def Error_mnk(individual):
    func = toolbox.compile(expr=individual)
    return math.fsum(abs((func(*pIn) - pOut)) for pIn, pOut in zip(In, Out))

def Error_abs(individual):
    func = toolbox.compile(expr=individual)
    return math.fsum((func(*pIn) - pOut) ** 2 for pIn, pOut in zip(In, Out))

def Error2(individual):
    func = toolbox.compile(expr=individual)
    return [(func(*pIn) - pOut) ** 2 for pIn, pOut in zip(In, Out)]

def Error3(individual):
    func = toolbox.compile(expr=individual)
    return [abs((func(*pIn) - pOut)) for pIn, pOut in zip(In, Out)]

def getCost(individual):
    return Error_mnk(individual) + individual.height / 100,

toolbox.register("evaluate", getCost)
toolbox.register("select", tools.selTournament, tournsize=2)
toolbox.register("mate", gp.cxOnePoint)

# genGrow() из библиотеки deap. Этот оператор создает
toolbox.register("expr_mut", gp.genGrow, min_=MUT_MIN_TREE_HEIGHT,
max_=MUT_MAX_TREE_HEIGHT)
toolbox.register("mutate", gp.mutUniform, expr=toolbox.expr_mut,
pset=primitiveSet)

toolbox.decorate("mate", gp.staticLimit(key=operator.attrgetter("height"),
max_value=LIMIT_TREE_HEIGHT))
toolbox.decorate("mutate", gp.staticLimit(key=operator.attrgetter("height"),
max_value=LIMIT_TREE_HEIGHT))

def plotsol(best):
    nodes, edges, labels = gp.graph(best)
    g = nx.Graph()
    g.add_nodes_from(nodes)
    g.add_edges_from(edges)
    pos = nx.spring_layout(g)
    nx.draw_networkx_nodes(g, pos, node_color='cyan')
    nx.draw_networkx_nodes(g, pos, nodelist=[0], node_color='red',
node_size=400)
    nx.draw_networkx_edges(g, pos)
    nx.draw_networkx_labels(g, pos, **{"labels": labels, "font_size": 8})
    plt.show()

if __name__ == "__main__":
    population = toolbox.populationCreator(n=POPULATION_SIZE)
    stats = tools.Statistics(lambda ind: ind.fitness.values)
    stats.register("min", np.min)
    stats.register("avg", np.mean)
    hof = tools.HallOfFame(HALL_OF_FAME_SIZE)
    population, logbook, sols = elitism.eaSimpleWithElitism(population,
toolbox, cxpb=P_CROSSOVER, mutpb=P_MUTATION,
```

```
ngen=MAX_GENERATIONS, stats=stats, halloffame=hof, verbose=True)
    print("\n\n")
    best = hof.items[0]
    best1 = sols[0]
    best2 = sols[1]
    for i in range(round(0.5 * NUM_OUTPUTS)):
        print("%6.3f %6.3f %6.3f" % (Out[i], Error3(best)[i],
Error2(best)[i]))
    print("-- Лучший индивидуум 1 поколения = ", best1)
    print("-- длина={}, высота={}".format(len(best1), best1.height))
    print("-- Лучшая приспособленность = ", best1.fitness.values[0])
    print("-- Лучшая ошибка = ", Error_mnk(best1), '\n')
    print("-- Лучший индивидуум {} поколения = ".format(MAX_GENERATIONS / 2),
best2)
    print("-- длина={}, высота={}".format(len(best2), best2.height))
    print("-- Лучшая приспособленность = ", best2.fitness.values[0])
    print("-- Лучшая ошибка = ", Error_mnk(best2), '\n')
    print("-- Лучший индивидуум = ", best)
    print("-- длина={}, высота={}".format(len(best), best.height))
    print("-- Лучшая приспособленность = ", best.fitness.values[0])
    print("-- Лучшая ошибка (mnk) = ", Error_mnk(best))
    print("-- Лучшая ошибка (abs) = ", Error_abs(best))
    plotsol(best)
    plotsol(best1)
    plotsol(best2)
```

Вывод

В ходе выполнения лабораторной работы была решена задача аппроксимации функции с помощью генетического программирования, графически отображены найденные решения.