Міністерство освіти і науки України

Національний технічний університет України

«Київський політехнічний інститут ім. Ігоря Сікорського»

Кафедра автоматизації проектування енергетичних процесів і систем

Лабораторна робота №3

з дисципліни «Методології розробки інтелектуальних комп'ютерних програм»

«**Оптимізація функції із застосуванням генетичних алгоритмів**»

Виконала: студентка 3 курсу

ТЕФ групи ТІ-01

Круть Каткрина

Перевірив: д.т.н. Мусієнко А. П.

**КИЇВ 2023**

**Завдання:**

Написати програму оптимізації функції з використанням генетичного алгоритму, що буде здатна обраховувати екстремуми функції на заданому проміжку.

**Набір даних:**

*Варіант 12*

Функція:

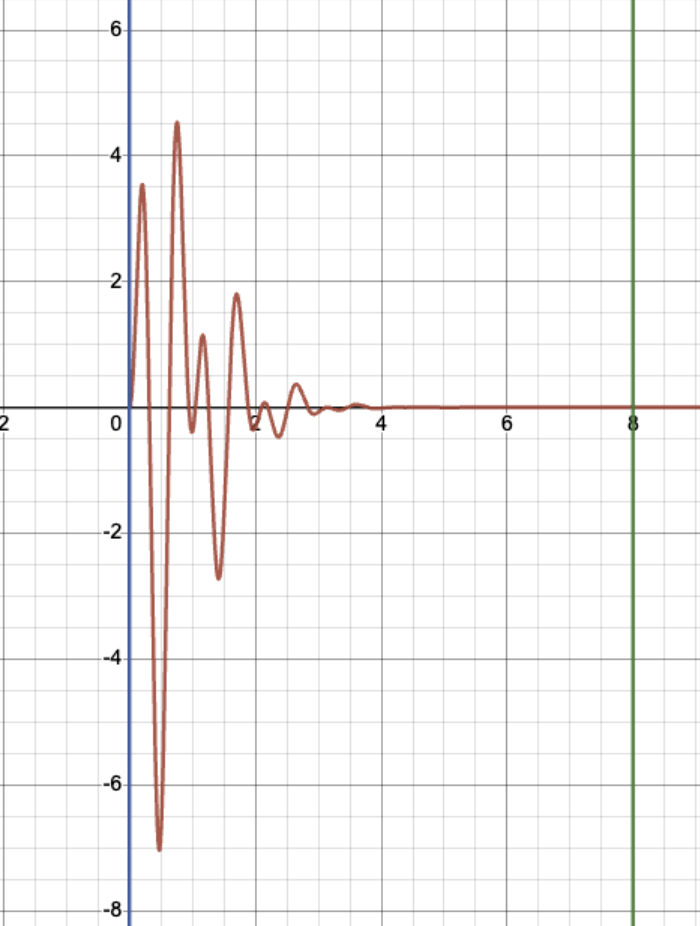
Y(x)=5\*sin(10\*x)\*sin(3\*x)/(x^x),

Проміжок:

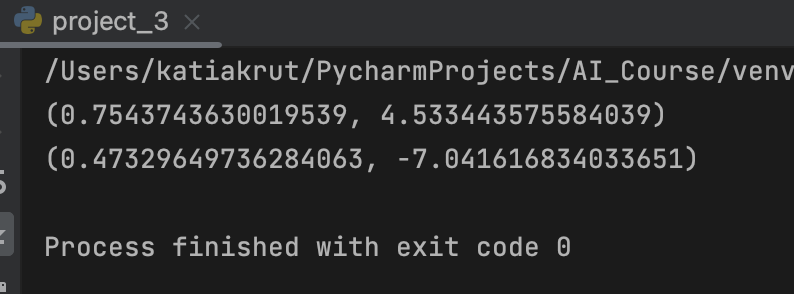
x=[0...8]

**Графік:**

Графік функції *Y(x)=5\*sin(10\*x)\*sin(3\*x)/(x^x)* на проміжку [0;8]



**Результат роботи програми:**



**Код програми**

import random

import numpy as np

import numpy.random as npr

class Chromosome:

def \_\_init\_\_(self, min\_value, max\_value, value=None, mut\_rate=0.1):

self.fitness = 0

self.min\_value = min\_value

self.max\_value = max\_value

self.value = random.uniform(min\_value, max\_value) if value is None else value

self.fitness = self.fitness\_function(self.value)

self.mutation\_rate = mut\_rate

def get\_value(self):

return self.value

def get\_fitness(self):

return self.fitness

@staticmethod

def fitness\_function(x):

return 5 \* np.sin(10 \* x) \* np.sin(3 \* x) / x \*\* x

def mutate(self):

rate = random.uniform(0, 1)

if rate < self.mutation\_rate:

delta = self.get\_sign() \* 0.1

if self.min\_value <= self.value + delta <= self.max\_value:

self.value += delta

self.fitness = self.fitness\_function(self.value)

@staticmethod

def get\_sign():

sign = random.uniform(0, 1) \* 2 - 1

return 1 if sign >= 0 else -1

def \_\_str\_\_(self):

return f"Chromosome: ({self.value}, {self.fitness})\n"

class Population:

def \_\_init\_\_(self, population\_n, reversed, min\_value, max\_value):

self.population\_n = population\_n

self.reversed = reversed

self.population = [Chromosome(min\_value, max\_value) for \_ in range(population\_n)]

self.min\_value = min\_value

self.max\_value = max\_value

self.ranging\_rate = 0.125

self.step = 2

def sort\_chromosomes(self):

self.population = sorted(self.population, reverse=self.reversed, key=lambda x: x.fitness)

def get\_min\_parents(self, index):

return min(self.population[index].value, self.population[index + 1].value)

def get\_max\_parents(self, index):

return max(self.population[index].value, self.population[index + 1].value)

def check\_range(self, value, range='min'):

return eval(f'self.{range}\_value if {value} {"<" if range == "min" else ">"} self.{range}\_value else {value}')

def check\_min(self, value):

return self.min\_value if value < self.min\_value else value

def check\_max(self, value):

return self.max\_value if value > self.max\_value else value

# get ranges for children

def get\_ranges(self, index):

min\_p\_value = self.get\_min\_parents(index)

max\_p\_value = self.get\_max\_parents(index)

correction = self.ranging\_rate \* (max\_p\_value - min\_p\_value)

return self.check\_min(min\_p\_value - correction), self.check\_max(max\_p\_value + correction)

# random value for children

@staticmethod

def rand\_value(min\_range, max\_range):

return npr.random() \* (max\_range - min\_range) + min\_range

def crossover(self):

for i in range(0, self.population\_n, self.step):

ranges = self.get\_ranges(i)

for child in range(self.step):

self.population.append(Chromosome(ranges[0], ranges[1], self.rand\_value(ranges[0], ranges[1])))

self.sort\_chromosomes()

self.population = self.population[:self.population\_n]

def mutate(self):

for chromosome in self.population:

chromosome.mutate()

def get\_population(self):

return self.population

def get\_best(self):

return self.get\_population()[0].value, self.get\_population()[0].fitness

# return f"Best: ({round(self.get\_population()[0].value, 4)}, {round(self.get\_population()[0].fitness, 4)})"

def evolution(self):

self.crossover()

self.mutate()

def \_\_str\_\_(self):

return f"Population:\npopulation number: {self.population\_n}\n\n{''.join([str(x) for x in self.population])}\n"

population = Population(40, True, 0, 8)

for i in range(50):

population.evolution()

print(population.get\_best())

population = Population(40, False, 0, 8)

for i in range(50):

population.evolution()

print(population.get\_best())

**Висновки:** Під час виконання лабораторної роботи було ознайомлено з основним видом генетичного алгоритму, відомостями про сфери використання генетичних алгоритмів, переваги над іншими алгоритмами, було розроблено програму для визначення екстремумів функції на проміжку без використання заздалегідь відомих значень для коригування роботи.