МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по учебной практике

Тема: Генетические алгоритмы

Соц Е.А.

Студенты гр. 2384 _____ Поглазов Н.В.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург
2024

Цель работы.

Частично реализовать генетический алгоритм и создать частично работающий GUI.

Задание.

Для заданного полинома f(x) (степень не больше 8) необходимо найти параметры ступенчатой функции g(x) (высота “ступеней”), которая приближает полиномиальную функцию, то есть минимизировать расстояние |f(x) - g(x)| между функциями на заданном интервале [l, r]. Количество и длина ступеней вводятся пользователем.

Выполнение работы.

Основные методы алгоритма.

Для реализации основного функционала алгроитма был создан класс GeneticAlgoritm (src/algorithms/genetic.py):

- Конструктор *(__init__)*:

population_size (int): Размер популяции.

mutation_rate (float): Вероятность мутации.

crossover_rate (float): Вероятность скрещивания.

max_iterations (int): Максимальное количество итераций.

search_space (tuple[float, float]): Пространство поиска значений для ступенчатой функции.

session_settings (SettingsData): Параметры сессии.

crossover_strategy (CrossoverStrategy): Оператор скрещивания.

mutation_strategy (MutationStrategy): Оператор мутации.

selection_strategy (SelectionStrategy): Оператор селекции.

- Атрибуты:

chromosome_length: Длина хромосомы, определяется количеством ступеней.

function: Многочлен, заданный в настройках сессии.

left_bound и right_bound: Левые и правые границы поиска.

current_population: Текущая популяция, начально инициализированная случайными значениями.

current_quality_function_values: Значения целевой функции для текущей популяции.

max_iterations: Максимальное количество итераций.

current_iteration: Текущая итерация.

best_quality_function_values: Лучшие значения целевой функции на каждой итерации.

crossover_strategy, mutation_strategy, selection_strategy: Операторы скрещивания, мутации и селекции соответственно.

points_per_step: Количество точек на шаг.

- Методы

1) *quality_function_vectorized(self, population: np.ndarray) -> np.ndarray*:

Вычисляет отклонения многочлена от ступенчатой функции и возвращает среднее отклонение для всей популяции.

2) *step(self) -> bool:*

Выполняет одну итерацию алгоритма.

Проверяет, достигнуто ли максимальное количество итераций.

Селекционирует родителей, выполняет кроссовер и мутацию, создавая новую популяцию.

Обновляет текущую популяцию и значения целевой функции.

Возвращает True, если итерации продолжаются, и False, если достигнут максимум итераций.

Был вынесен в отдельный метод для того, чтобы GUI взаимодействовал с этим классом.

3) *run(self) -> None:*

Запускает алгоритм до достижения максимального количества итераций.

В цикле вызывает метод step().

Необходим для запуска генетического алгоритма с любой итерации через GUI.

4) *get_top_individuals(self, n: int) -> np.ndarray:*

Возвращает n лучших индивидуумов текущей популяции.

Сортирует популяцию по значению целевой функции и возвращает верхние n.

Необходим для отрисовки лучших особей на графике.

5) *get_best_quality_function_values(self) -> np.ndarray:*

Возвращает массив лучших значений целевой функции до текущей итерации.

Необходим для отрисовки графика функции целевой функции.

Для реализации отбора на основе ранжирования особей был реализован класс *RankBasedSelection*.

В методе *select(self, population: np.ndarray, quality_function_values: np.ndarray) -> np.ndarray* происходит расчет рангов: меньшему значению целевой функции присваивается больший ранг; рассчитываются вероятности для каждого ранга: чем больше ранг - тем больше вероятность выбора особи; выбираются индексы двух особей, гарантируется, что особь не может выбираться дважды. Метод возвращает два выбранных родителя для дальнейшего скрещивания.

Для реализации равномерного скрещивания был реализован класс *UniformCrossover*.

В методе *crossover(self, parent1: np.ndarray, parent2: np.ndarray) -> tuple[np.ndarray, np.ndarray]* происходит генерация маски (0 либо 1), размер этой маски равен размеру векторов родителей, это означает, что каждый элемент хромосомы будет проверяться на принадлежность к одному из родителей. Затем создаются две новые особи, для каждой из которых проверяется условие: если значение в маске равно 1 (что происходит в вероятностью 50%), то элемент в потомке берется из первого родителя, иначе - из второго. Это обеспечивает равновероятное наследование генов обоих родителей. Таким образом, метод возвращает кортеж из двух новых особей.

Класс *IntermediateRecombination* предназначен для промежуточной рекомбинации.

Конструктор класса принимает один необязательный параметр d, который является определяющим числом и определяет степень

изменяемости между родительскими хромосомами при скрещивании. По умолчанию d установлено равным 0,25.

Метод *_generate_alpha* создает массив значений alpha, каждый из которых является случайным числом, распределенным равномерно между -d и 1 + d.

В методе *crossover(self, parent1: np.ndarray, parent2: np.ndarray) -> tuple[np.ndarray, np.ndarray]* генерируются два массива с alpha с размером родителей. Затем для каждого потомка вычисляется их значение по заданной формуле: $Offspring = Parent1 + \alpha * (Parent2 - Parent1)$. Метод возвращает двух новых особей.

Класс *RealNumberMutation* создан для мутации особей с вещественными генами.

Конструктор класса принимает два параметра: *search_space*, который определяет диапазон возможных значений для мутации, и *m*, количество различных уровней мутации. По умолчанию *m* установлено равным 20.

Метод *_calculate_delta* генерирует массив *delta*, который представляет собой $\delta = \sum_{i=1}^{m} \frac{a(i)}{2^i}$, где a(i) = 1 с вероятностью 1/m, в противном случае a(i) = 0.

В методе *mutate(self, individual: np.ndarray) -> np.ndarray* происходит сама мутация по заданному правилу: новая переменная = старая переменная ± α · δ, где знаки + или - выбираются с равной вероятностью, α = 0.5 * поисковое пространство. Метод возвращает измененную (или прежнюю) особь.

Создание пользовательского графического интерфейса (GUI).

В процессе работы GUI не подвергся сильным изменениям относительно прошлой итерации.

Был добавлен отдельный график для отображения динамики изменения лучших значений целевой функции, а также были добавлены подсказки при наведении на виджеты (кнопки, текстовые поля и др.).

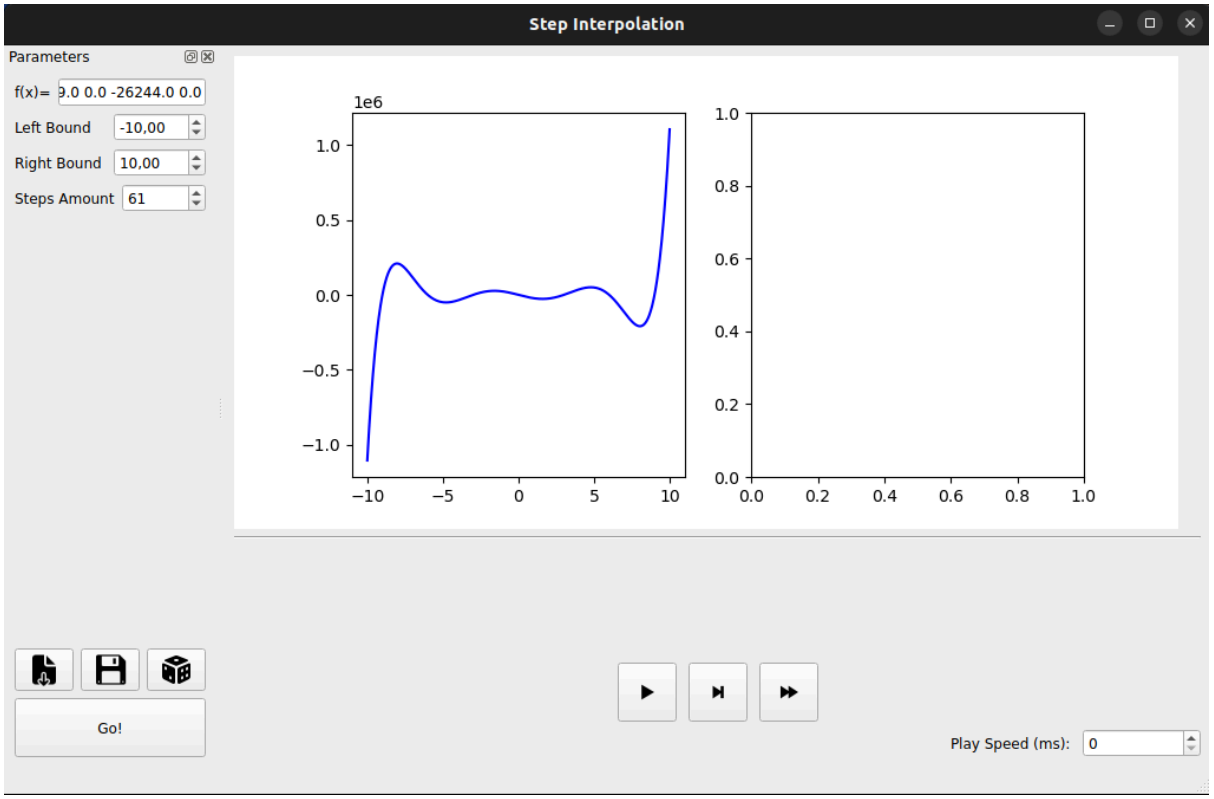

Рисунок 1. Изменения в GUI.

Вывод.

В результате выполнения задания были реализованы методы для работы генетического алгоритма. В GUI был добавлен отдельный график для отображения динамики изменения лучший значений целевой функции.