Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»

ОТЧЕТ

По лабораторной работе №2

«Вариации генетических алгоритмов»

По дисциплине

«Методы эволюционной оптимизаци»

Выполнили: Шевченко Алексей ИУ1-52Б

Шемет Александра ИУ1-51Б

Работа выполнена: 15.10.2023

Отчет сдан: 15.10.2023

Москва 2023

Оглавление

Цель работы:

В соответсвии с вариантом реализовать генетический алгоритм для задачи поиска локального минимума в заданной области поиска на основе функции Розенброка с определёнными параметрами.

Вариант работы: 222242

	1	2	3	4
Критерий сходимости	Спустя n поколений	Функция приспособленности не меняется	Дисперсия достигла порога	
Элитарность	Способ1	Способ2		
Отбор родителей	Ниширование	Очистка	Скучивание	
Рулетка	Турнирный отбор	Избыточный отбор	Ранговый отбор	Сигма-шкалирование
Скрещивание	Трехточечное	Сегментированное	Равномерное	Симулированное бинарное скрещивание
Мутация	Равномерная мутация с центром в x	Равномерная мутация с центром в поисковой области		

Таблица 1 – начальные параметры алгоритма

Теоретическая часть

На основе нашего варианта разберём теоретическую часть

Отбор родителей. Очистка.

Очистка аналогична обмену информацией о приспособленности, но, вместо того чтобы делиться значениями приспособленности между особями, которые разделяют одну и ту же нишу, мы уменьшаем приспособленность некоторых из этих особей [Pétrowski, 1996], [Sareni и Krahenbiihl, 1998]. Данную идею можно реализовать несколькими способами, в том числе следующими. Во-первых, мы определяем множество ниш Di каждой особи в популяции:

$$D_i = \{x_j : d_{ij} < \sigma\},$$

где dij – это такое же расстояние, как и в уравнении (8.8), а σ – определяемый пользователем параметр. Затем мы ранжируем особей в каждой нише в соответствии с их приспособленностью:

$$r_{ki} = \text{rank of } x_k \text{ in } D_i,$$

где лучшая особь в каждой нише имеет ранг 1, вторая лучшая особь имеет ранг 2 и т. д

Рис. 2 - Отбор родителей - очистка (часть 1)

Наконец, мы определяем параметр R как число особей, которые по нашему желанию должны выжить в каждой нише, и получаем модифицированные значения приспособленности следующим образом, где N – это размер популяци

For $i = 1$ **to** N
  For $k = 1$ **to** $|D_i|$
    If $r_{ki} \le R$ **then**
      $f'_k \leftarrow f_k$
    else
      $f'_k \leftarrow -\infty$
    End if
  Next ниша
Next особь

Приведенный выше алгоритм гарантирует, что наименее приспособленные особи в каждой нише недоступны для отбора или рекомбинации. Тем не менее это не гарантирует, что наиболее приспособленные особи будут доступны для отбора, потому что особи могут принадлежать более чем к одной нише.

Рис. 3 Отбор родителей – очистка (часть 2)

Отбор родителей. Избыточный отбор.

Избыточный отбор (over-selection) – это метод, первоначально предложенный Дж. Козой в контексте генетического программирования [Koza, 1992, глава 6]. Избыточный выбор модифицирует рулеточный отбор путем непропорционального взвешивания значений приспособленности высоко приспособленных особей, для того чтобы увеличить их шансы на отбор. В версии Дж. Козы избыточного отбора лучшие 32 % популяции имеют 80%-ный шанс быть отобранными, и худшие 68 % популяции имеют 20%-ный шанс быть отобранными. Точные проценты не слишком важны; ключевой особенностью избыточного отбора является то, что приспособленные особи имеют значительно более высокую вероятность быть отобранными. Это один из типов шкалирования по значению приспособленности.

Рис. 4 – Рулетка - избыточный отбор

Сегментированное скрещивание (в бинарных или непрерывных эволюционных алгоритмах

Можно представить как обобщение многоточечного скрещивания. Ребенок 1 получает свой первый признак от родителя 1. Затем, с вероятностью p, мы переключаемся на родителя 2 и получаем второй для ребенка 1 признак; и с вероятностью (1 – p) мы получаем второй для ребенка 1 признак от родителя 1. Каждый раз, когда мы получаем признак для ребенка 1, то переключаемся на другого родителя, чтобы получить следующий признак с вероятностью p. Ребенок 1 и ребенок 2 получают свои признаки от разных родителей, поэтому если ребенок 1 получает признак k от родителя 1, то ребенок 2 получает признак k от родителя 2, для $k \in [1, n]$. Аналогичным образом, если ребенок 1 получает признак k от родителя 2, то ребенок 2 получает признак k от родителя 1. Сегментированное скрещивание эквивалентно многоточечному скрещиванию, где число точек скрещивания является случайным числом.

Рис. 5 – Скрещивание – симулированное бинарное скрещивание

Равномерная мутация с центром в середине поисковой области

Равномерная мутация с центром в середине поисковой области может быть записана как

$$r \leftarrow U[0, 1]$$

$$x_i(k) \leftarrow \begin{cases} x_i(k) & \text{если } r \geq \rho \\ U[x_{\min}(k),\ x_{\max}(k)] & \text{если } r < \rho \end{cases} \qquad (8.60)$$

для $i \in [1, N]$ и $k \in [1, n]$.

Рис. 6 – Мутация – равномерная мутация с центром в поисковой области

Принцип работы генетического алгоритма:

1. Инициализация популяции: Начальная популяция создается случайным образом. Каждый индивидуум в популяции представляет собой потенциальное решение задачи оптимизации и называется хромосомой. В данном коде хромосома представляет собой вектор чисел.

```
# Инициализация начальной популяции
def initialize_population():
    population = []
    for _ in range(population_size):
        chromosome = np.random.uniform(search_range[0], search_range[1], chromosome_length)
        population.append(chromosome)
    return population
```

Рис. 7 – инициализация начальной популяции

2. Оценка приспособленности: Каждая хромосома оценивается на основе значения целевой функции (в данном случае функции Розенброка). Чем меньше значение функции, тем лучше приспособленность хромосомы.

```
# Определение функции Розенброка
def rosenbrock_function(x):
    return sum(100 * (x[i+1] - x[i]**2)**2 + (1 - x[i])**2 for i in range(len(x)-1))
```

Рис. 8 – функция Розенброка

3. Выбор родителей: Родители выбираются из популяции для создания потомства. В данном коде используется метод очистки, при котором каждый родитель выбирается случайно из популяции.

```
# Выбор родителей методом очистки
def select_parents(population, fitness):
    parents = []
    for _ in range(population_size):
        # Выбор случайного родителя из популяции
        parent_index = random.randint(0, population_size-1)
        parents.append(population[parent_index])
    return parents
```

Рис. 9 – выбор родителей методом очистки

4. Скрещивание: Родители комбинируют свои генетические материалы, чтобы создать потомство. В данном коде используется симулированное бинарное скрещивание, при котором каждый ген потомка формируется путем смешивания генов родителей с помощью случайного коэффициента.

```python
# Симулированное бинарное скрещивание
def crossover(parents):
    offspring = []
    for i in range(0, population_size, 2):
        parent1 = parents[i]
        parent2 = parents[i+1]
        child1 = np.zeros(chromosome_length)
        child2 = np.zeros(chromosome_length)
        for j in range(chromosome_length):
            alpha = random.uniform(0, 1)
            child1[j] = alpha * parent1[j] + (1 - alpha) * parent2[j]
            child2[j] = alpha * parent2[j] + (1 - alpha) * parent1[j]
        offspring.append(child1)
        offspring.append(child2)
    return offspring
```

Рис. 10 - Скрещивание

5. Мутация: Потомство может подвергаться случайным изменениям, называемым мутациями. В данном коде используется равномерная мутация, при которой каждый ген потомка имеет небольшую вероятность быть измененным случайным образом.

```python
# Равномерная мутация
def mutate(offspring):
    for i in range(population_size):
        for j in range(chromosome_length):
            if random.uniform(0, 1) < mutation_probability:
                offspring[i][j] += random.uniform(-1, 1) * (search_range[1] - search_range[0])
                offspring[i][j] = np.clip(offspring[i][j], search_range[0], search_range[1])
    return offspring
```

Рис. 11 – Мутация

6. Обновление популяции: Новое поколение создается из потомства и заменяет предыдущую популяцию.

```
# Обновление лучшего решения
best_index = np.argmin(fitness)
best_fitness = fitness[best_index]
best_solution = population[best_index]

parents = select_parents(population, fitness)
offspring = crossover(parents)
offspring = mutate(offspring)

population = offspring
```

Рис. 12 – Обновление популяции

7. Проверка критерия сходимости: Проверяется условие остановки алгоритма. В данном коде алгоритм останавливается, если лучшая приспособленность в популяции не улучшается в течение нескольких поколений.

```
# Оценка приспособленности особей в популяции
def evaluate_fitness(population):
    fitness = []
    for chromosome in population:
        fitness.append(rosenbrock_function(chromosome))
    return fitness
```

Рис. 13 – функция оценки приспособленности

```
for generation in range(max_generations):
    fitness = evaluate_fitness(population)

    # Проверка критерия сходимости
    if min(fitness) >= best_fitness:
        break
```

Рис. 14 – проверка критерия сходимости

8. Возврат результата: После окончания работы алгоритма возвращается лучшее найденное решение (хромосома) и его приспособленность (значение функции).

```
    return best_solution, best_fitness

# Запуск генетического алгоритма
best_solution, best_fitness = genetic_algorithm()

print("Лучшее решение:", best_solution)
print("Лучшая приспособленность:", best_fitness)
```

```
Лучшее решение: [ 0.02282048  0.10246321 -0.17322983 -0.17062501  0.00045072  0.37269309
 -0.08250486  0.16535364 -0.13047275 -0.12193773]
Лучшая приспособленность: 43.2959697230402
```

Рис. 15 – вывод результата

Генетический алгоритм продолжает итеративно выполнять шаги с 3 по 7 до тех пор, пока не будет достигнут критерий сходимости или максимальное количество поколений. В конечном итоге, алгоритм находит локальный оптимум функции в заданной области поиска.

Оценка результата

Для того чтобы оценить работу генетического алгоритма – построим с помощью библиотеки matplotlib 3D график поверхности Розенброка и отметим на ней наше решение.

Для сравнения так же отметим значение истинного локального минимума для этой поверхности – локальный минимум для этой поверхности в области исследования находится в точке X = 1, Y = 1 и равен Z = 0.

```python
import matplotlib.pyplot as plt

# Функция Розенброка
def rosenbrock(x, y):
    return (1 - x)**2 + 100 * (y - x**2)**2

# Построение графика поверхности Розенброка
x = np.linspace(-2, 2, 100)
y = np.linspace(-1, 3, 100)
X, Y = np.meshgrid(x, y)
Z = rosenbrock(X, Y)

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.plot_surface(X, Y, Z, cmap='viridis')

# Найденный локальный минимум (по формуле)
best_solution_math = [1, 1]
best_fitness_math = rosenbrock(*best_solution_math)

ax.scatter(best_solution_math[0], best_solution_math[1], best_fitness_math, color='red', s=400)
ax.scatter(best_solution[0], best_solution[1], best_fitness, color='green', s=400)

print(f"Best X: {best_solution[0]}, Best Y: {best_solution[1]}")
print(f"Best Fitness: {best_fitness}")

print(f"Math X: {best_solution_math[0]}, Math Y: {best_solution_math[1]}")
print(f"Math Fitness: {best_fitness_math}")

ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')

plt.show()
```

Рис. 16 – функция построения графика поверхности Розенброка

На данном графике красным цветом отмечена точка истинного локального минимума в области наблюдения для поверхнорсти Розенброка, а зелёным цветом – точка локального минимума, которая была найдена с помощью нашего генетического алгоритма.

```
Best X: 0.2862727798812744, Best Y: -0.17801680146289894
Best Fitness: 49.56117804902543
Math X: 1, Math Y: 1
Math Fitness: 0
```

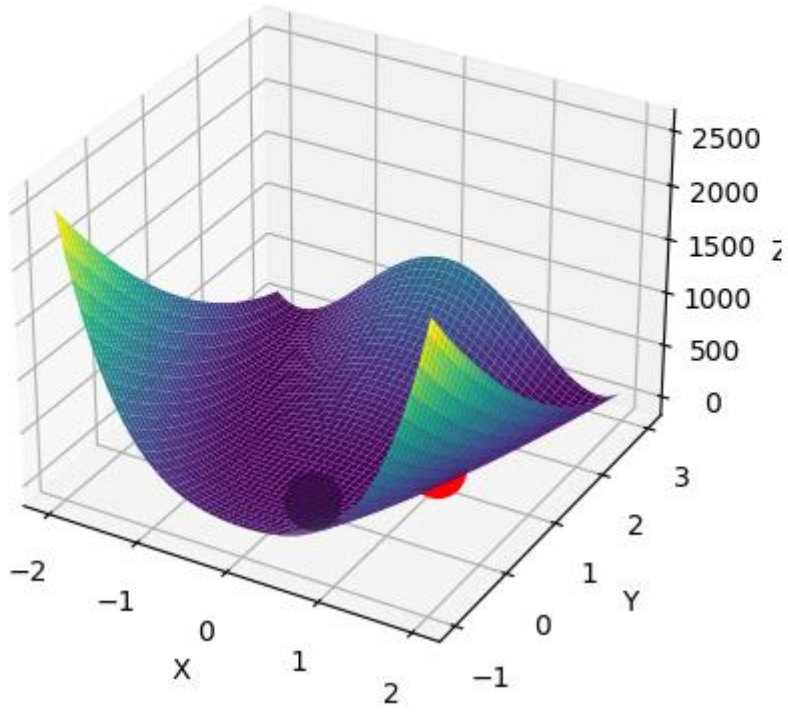

Рис. График поверхности Розенброка

Вывод:

Фактически, генетические алгоритмы обладают уникальной по своей простоте и гибкости структурой. При этом, они позволяют легко учитывать большое количество факторов и критериев. Выдают множество субоптимальных решений, могут легко включать в себя другие математические методы.