

Лабораторная работа №1

ВВЕДЕНИЕ В ГЕНЕТИЧЕСКИЕ АЛГОРИТМЫ: ЗНАКОМСТВО С БАЗОВЫМИ ПРИНЦИПАМИ НА ПРОСТЫХ ПРИМЕРАХ

Цель работы: *Ознакомление с базовыми принципами и концепциями генетических алгоритмов через теоретическое изучение и практическую реализацию.*

Аннотация: *Генетические алгоритмы и эволюционные методы обеспечения безопасности автоматизированных систем имеют взаимосвязь через применение принципов эволюции для решения сложных задач. В основе обоих подходов лежат идеи селекции, мутации и кроссовера, которые могут быть применены для оптимизации параметров безопасности.*

В работе "Введение в генетические алгоритмы" мы рассмотрим базовые принципы этих алгоритмов, которые можно прямо применить в контексте обеспечения безопасности. Например, можно использовать генетические алгоритмы для автоматического поиска наиболее эффективных параметров в системах обнаружения вторжений, минимизации уязвимостей или для адаптивного управления доступом.

Таким образом, базовые механизмы генетических алгоритмов, изученные в данной работе, предоставляют инструментарий, который может быть адаптирован и применен для улучшения методов обеспечения безопасности в автоматизированных системах. Эта связь делает текущую работу релевантной для исследований в области эволюционных методов обеспечения безопасности, показывая, как базовые принципы могут быть применены в этом конкретном контексте.

ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

Генетические алгоритмы — это методы оптимизации и поиска решений, которые имитируют процесс естественной эволюции. Эти алгоритмы применяют принципы генетики и естественного отбора для "эволюции" популяции возможных решений задачи в направлении оптимального или приемлемого решения.

Генетические алгоритмы являются весьма гибкими и могут быть адаптированы для различных задач, но у всех их имеются общие компоненты: хромосомы, гены, популяция и функция фитнесса.

Основные компоненты:

1. **Хромосома:** Это одно из возможных решений задачи, закодированное определенным образом. Например, если задача заключается в поиске оптимального маршрута для курьера, хромосома может представлять собой последовательность точек маршрута.
2. **Ген:** Элемент хромосомы, который представляет собой конкретный атрибут решения. В примере с курьером генами будут отдельные точки маршрута.
3. **Популяция:** Это множество хромосом. Популяция эволюционирует со временем, "скрещивая" хромосомы и "мутируя" их, чтобы создать новое поколение возможных решений.
4. **Функция фитнесса:** Это метрика, которая оценивает, насколько хорошо хромосома решает задачу. В случае с курьером это может быть общее время или расходы на доставку по заданному маршруту.

Генетические алгоритмы впервые были предложены в 1960-х и 1970-х годах и получили широкое распространение в 1980-х. Один из основателей этой области — Джон Холланд, который внёс значительный вклад в формализацию и разработку методов, используемых в генетических алгоритмах. Он также является автором одной из первых книг на эту тему, "Adaptation in Natural and Artificial Systems" (1975).

Этапы работы алгоритма:

1. **Инициализация популяции:** Случайным образом создается начальная популяция хромосом. На начальном этапе работы генетического алгоритма создается популяция, состоящая из хромосом — наборов параметров, которые представляют возможные решения задачи. Обычно эта популяция инициализируется случайным образом, хотя в некоторых случаях можно использовать и предварительно известные «хорошие» решения. Размер популяции и структура хромосом зависят от конкретной задачи и могут быть подобраны эмпирически или на основе экспертных знаний.
2. **Оценка популяции:** С помощью функции фитнеса оценивается "качество" каждой хромосомы.
3. **Селекция:** Хромосомы выбираются для создания нового поколения на основе их "пригодности", оцененной функцией фитнеса. После инициализации следует этап селекции, на котором из текущей популяции выбираются родители для создания следующего поколения. Цель этого этапа — определить наилучшие хромосомы, которые затем будут использоваться для порождения новых особей. Часто для этого используют функцию фитнеса — метрику, которая показывает, насколько хорошо данная хромосома решает поставленную задачу. Существуют различные методы селекции, включая рулеточное колесо, турнирную селекцию и другие.
4. **Кроссовер (скрещивание):** Выбранные хромосомы "скрещиваются" друг с другом, обмениваясь частями своих генов. Кроссовер, или скрещивание, — это процесс комбинирования генов двух или более родителей для создания потомства. В результате получаются новые хромосомы, которые могут наследовать хорошие свойства обоих родителей. Существует множество методов кроссовера, и выбор конкретного метода зависит от типа задачи и структуры хромосом. В простых случаях можно использовать одноточечный или двухточечный кроссовер, а для более сложных задач — более продвинутые методы, например, униформный кроссовер или арифметический.
5. **Мутация:** С небольшой вероятностью гены в хромосомах изменяются. После кроссовера следует этап мутации, на котором некоторые гены в хромосомах изменяются случайным образом. Мутация необходима для внесения разнообразия в популяцию и избегания преждевременной сходимости к локальным оптимумам. Как и в случае с кроссовером, существует множество методов мутации, и их выбор зависит от конкретной задачи.
6. **Новое поколение и повторение процесса:** Полученные хромосомы формируют новое поколение, и процесс повторяется с шага 2. После того как новое поколение было создано с помощью кроссовера и мутации, каждая хромосома в новой популяции оценивается с помощью функции фитнеса. Затем на основе этих оценок выбираются хромосомы, которые будут составлять новую популяцию. Этот процесс может включать в себя не только потомство, но и лучших родителей из предыдущего поколения, чтобы сохранить хорошие решения.

Все эти этапы повторяются в течение заданного числа поколений или до тех пор, пока не будет достигнута желаемая точность решения. Таким образом, генетический алгоритм является итеративным методом, который сочетает в себе элементы случайного поиска, отбора и наследственности для нахождения наилучших решений сложных задач.

Применение:

1. Генетические алгоритмы имеют широкий спектр применения в различных областях:
2. Оптимизация: Поиск наилучших параметров для определенной задачи, например, настройка параметров нейронных сетей или оптимизация расписания.
3. Робототехника: Адаптация и "обучение" роботов выполнению сложных задач.
4. Биоинформатика: Анализ генетических последовательностей, поиск схожих участков, филогенетическое моделирование.
5. Экономика и финансы: Оптимизация торговых стратегий, риск-менеджмент.
6. Графика и дизайн: Автоматическое создание сложных дизайнов и анимаций.
7. Сетевая безопасность: Оптимизация правил для систем обнаружения вторжений или файерволов.
8. Телекоммуникации: Оптимизация маршрутов передачи данных.
9. Медицина: Анализ медицинских данных для диагностики и прогнозирования заболеваний.
10. Логистика: Оптимизация маршрутов доставки, распределение ресурсов.

Функция фитнеса

Функция фитнеса является центральным элементом в генетическом алгоритме, играя ключевую роль в определении "качества" или "пригодности" каждого решения, представленного хромосомой. Эта функция оценивает, насколько хорошо конкретное решение соответствует заданным критериям или целям. В других словах, функция фитнеса служит мерой успешности хромосомы в контексте решаемой задачи.

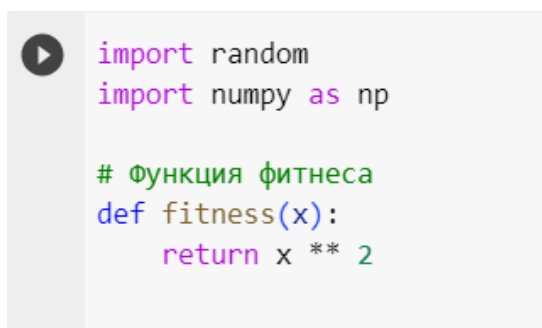
Выбор подходящей функции фитнеса — это чрезвычайно важный этап в разработке генетического алгоритма, потому что от неё напрямую зависит эффективность всего процесса. Например, если задача заключается в минимизации расходов на производство, функция фитнеса может быть таковой, чтобы чем меньше расходы, тем выше значение фитнеса.

Часто функция фитнеса основывается на уже существующих методах оценки решений. В задачах оптимизации это может быть просто значением целевой функции, которую необходимо минимизировать или максимизировать. В других случаях функция фитнеса может быть более сложной и включать в себя несколько различных метрик, например, время выполнения задачи, затраты ресурсов и уровень ошибок.

Также стоит учитывать, что функция фитнеса должна быть выбрана так, чтобы она была согласована с натурой задачи. Например, в задаче о коммивояжёре, где необходимо найти кратчайший путь через ряд городов, функция фитнеса может быть просто обратной величиной общей длины маршрута — таким образом, чем короче маршрут, тем выше его "фитнес".

Но выбор функции фитнеса — это не только научный, но и искусственный процесс. Иногда приходится проводить ряд экспериментов, чтобы понять, какая функция фитнеса наиболее эффективно решает поставленную задачу. Это может потребовать времени и тщательного анализа результатов каждого эксперимента, чтобы довести функцию до оптимального вида.

Для демонстрации принципов генетического алгоритма рассмотрим простую задачу поиска максимума функции: $f(x) = x^2$ на интервале от -10 до 10. В этом случае, функция фитнеса будет просто квадратом значения x , так как мы ищем его максимум. Вначале импортируем необходимые библиотеки и определим функцию фитнеса (рис. 1):

A screenshot of a code editor showing Python code. On the left, there is a play button icon. The code defines the fitness function as the square of x.

```
import random
import numpy as np

# Функция фитнеса
def fitness(x):
    return x ** 2
```

Рисунок 1. Определение функции фитнеса на языке python

Теперь инициализируем популяцию. Допустим, популяция состоит из 10 хромосом, и каждая хромосома является числом от -10 до 10 (рис. 2):

A screenshot of a code editor showing Python code. On the left, there is a play button icon. The code imports random and numpy, defines the fitness function, and then initializes a population of 10 random values between -10 and 10.

```
import random
import numpy as np

# Функция фитнеса
def fitness(x):
    return x ** 2

population_size = 10
population = [random.uniform(-10, 10) for _ in range(population_size)]
```

Рисунок 2. Инициализация популяции.

Затем выполним несколько итераций алгоритма. На каждой итерации проведем селекцию, кроссовер и мутацию (рис. 3):

```

num_generations = 20 # количество поколений
mutation_rate = 0.2 # вероятность мутации

for generation in range(num_generations):
    # Оценка фитнеса каждой хромосомы
    fitness_values = [fitness(x) for x in population]

    # Селекция: выбираем двух лучших родителей
    sorted_indices = np.argsort(fitness_values)
    best_parents = [population[i] for i in sorted_indices[-2:]]

    # Кроссовер: среднее арифметическое родителей
    offspring = [(best_parents[0] + best_parents[1]) / 2 for _ in range(population_size - 2)]

    # Мутация: добавляем случайное маленькое число к некоторым особям
    for i in range(len(offspring)):
        if random.random() < mutation_rate:
            offspring[i] += random.uniform(-1, 1)

    # Новое поколение: два лучших родителя + потомство
    population = best_parents + offspring

    # Выводим информацию о текущем поколении
    print(f"Generation {generation + 1}, Best value: {max(fitness_values)}")

# Выводим финальную популяцию и лучшее решение
print(f"Final population: {population}")
print(f"Best solution: {max(population, key=fitness)}")

```

Рисунок 3. Итерации алгоритма

Этот простой пример демонстрирует базовые принципы работы генетических алгоритмов, включая функцию фитнеса, селекцию, кроссовер и мутацию. Конечно, это очень упрощенная модель, и в реальных задачах генетические алгоритмы часто значительно сложнее и тоньше настроены. Но основные принципы остаются теми же.

ПРАКТИЧЕСКАЯ ЧАСТЬ

Теперь, когда мы разобрались с теоретической частью и основами генетических алгоритмов, давайте перейдем к более практическим аспектам. В качестве примера мы возьмем задачу минимизации функции:

$$f(x) = x^2 - 4x + 4$$

Хотя данная функция имеет аналитическое решение, применение генетических алгоритмов позволит нам понять, насколько эффективно этот метод может аппроксимировать решение в случаях, когда аналитический метод неприменим или слишком сложен. Таким образом, на практическом примере мы увидим, как реализовать все этапы генетического алгоритма: инициализацию, оценку фитнеса, селекцию, кроссовер и мутацию.

Этап 1:

```
import random

# Функция фитнеса (в данном случае, чем меньше, тем лучше)
def fitness(x):
    return x ** 2 - 4 * x + 4
```

В этом блоке мы импортировали модуль *random* для генерации случайных чисел и определили функцию фитнеса *fitness(x)*, которая вычисляет значение функции: $x^2 - 4x + 4$

Этап 2:

```
# Инициализация популяции
population_size = 10
population = [random.uniform(-10, 10) for _ in range(population_size)]

# Параметры алгоритма
num_generations = 20
mutation_rate = 0.2
```

На этом этапе инициализируется популяция из 10 случайных чисел в диапазоне от -10 до 10 и устанавливаются параметры алгоритма: количество поколений и вероятность мутации.

Этап 3:

```
# Генетический алгоритм
for generation in range(num_generations):
    # Оценка фитнеса каждой хромосомы
    fitness_values = [fitness(x) for x in population]

    # Селекция: выбираем двух лучших родителей
    best_parents = sorted(population, key=fitness)[:2]

    # Кроссовер: среднее арифметическое родителей
    offspring = [(best_parents[0] + best_parents[1]) / 2 for _ in range(population_size - 2)]
```

Цикл для каждого поколения. Внутри цикла первое, что делается, — это оценка фитнеса каждого элемента популяции.

Селекция родителей. Мы сортируем популяцию по фитнесу и выбираем двух лучших родителей.

Кроссовер. Для создания потомства мы просто берём среднее арифметическое родителей.

Этап 4.

```
# Мутация
for i in range(len(offspring)):
    if random.random() < mutation_rate:
        offspring[i] += random.uniform(-0.5, 0.5)

# Новое поколение
population = best_parents + offspring
```

Мутация. С некоторой вероятностью (в данном случае, 0.2 или 20%) мы изменяем значение потомка, добавляя к нему случайное число в диапазоне от -0.5 до 0.5.

Формирование нового поколения из двух лучших родителей и их потомства.

Этап 5.

```
# Выводим информацию о текущем поколении
print(f"Generation {generation + 1}, Best value: {min(fitness_values)}")

# Выводим финальную популяцию и лучшее решение
print(f"Final population: {population}")
print(f"Best solution: {min(population, key=fitness)}")
```

Вывод информации о текущем поколении и наилучшем значении фитнеса.

В конце программы выводится финальная популяция и лучшее найденное решение.

Результат выполнения программы представлен на рисунке 4.

```
Generation 1, Best value: 0.003048809917738815
Generation 2, Best value: 0.0003383310264775119
Generation 3, Best value: 0.0003383310264775119
Generation 4, Best value: 0.0003383310264775119
Generation 5, Best value: 0.0003383310264775119
Generation 6, Best value: 0.0003383310264775119
Generation 7, Best value: 0.0003383310264775119
Generation 8, Best value: 0.0003383310264775119
Generation 9, Best value: 0.0003383310264775119
Generation 10, Best value: 0.0003383310264775119
Generation 11, Best value: 0.0003383310264775119
Generation 12, Best value: 0.0003383310264775119
Generation 13, Best value: 0.0003383310264775119
Generation 14, Best value: 0.0003383310264775119
Generation 15, Best value: 0.0003383310264775119
Generation 16, Best value: 0.0003383310264775119
Generation 17, Best value: 0.0003383310264775119
Generation 18, Best value: 5.392695847472595e-05
Generation 19, Best value: 3.052716727180993e-05
Generation 20, Best value: 3.052716727180993e-05
Final population: [1.9944748604296547, 1.9944748604296547, 1.9944748604296547, 1.9944748604296547,
Best solution: 1.9944748604296547
```

Рисунок 4. Результат выполнения программы по решению квадратного уравнения

Этот вывод программы сообщает нам о том, как алгоритм находит оптимальное решение на каждом этапе (поколении).

Best value представляет наилучшее значение функции фитнеса, найденное в текущем поколении. В вашем примере это значение уменьшается, начиная с 0.003 до около 0.00003, что указывает на то, что алгоритм находит всё более "хорошие" решения (в контексте вашей функции фитнеса).

В конце выводится финальная популяция, которая в вашем случае состоит из чисел, близких к 1.9945. Это и есть значения, которые минимизируют вашу функцию фитнеса.

`Best solution` — это наилучшее найденное решение. В вашем случае это примерно 1.9945.

Визуализация процесса работы алгоритма является важным этапом для понимания его эффективности и для дальнейшего анализа. Графическое представление динамики изменения функции фитнеса, например, может быть очень полезным для оценки скорости сходимости алгоритма к оптимальному решению. Кроме того, визуализация может помочь выявить возможные проблемы, такие как застревание в локальных минимумах или максимумах.

Для решения задачи максимизации функции $f(x) = x^2 - 4x + 4$ мы можем визуализировать изменение лучшего значения функции фитнеса в каждом поколении. Это даст нам представление о том, как быстро алгоритм сходится и как стабилен этот процесс. В зависимости от используемого языка программирования, такая визуализация может быть выполнена с использованием различных библиотек для построения графиков. Например, в Python для этой цели часто используют библиотеку `Matplotlib`.

Добавление визуализации в наш код могло бы быть следующим этапом для глубокого понимания механизма работы генетического алгоритма и его эффективности для данной задачи. Через графическое представление мы можем увидеть, достигает ли алгоритм оптимального решения, и если да, то как быстро это происходит. Также это может быть полезным для сравнения эффективности разных вариантов алгоритма или для демонстрации результатов в научных исследованиях.

Для визуализации работы алгоритма с использованием библиотеки `Matplotlib` необходимо добавить следующее к уже реализованному коду:

1. Импортировать библиотеку:

```
import random
import matplotlib.pyplot as plt
```

2. Создание пустого списка `best_values`, который будет хранить лучшие (минимальные) значения функции фитнеса на каждом этапе:

```
# Сюда будем складывать лучшие значения функции фитнеса на каждом этапе
best_values = []
```

3. В цикле генетического алгоритма, сохранение минимального значения функции фитнеса в список `best_values`:

```
# Сохраняем лучшее значение функции фитнеса
best_value = min(fitness_values)
best_values.append(best_value)
```

4. В конце кода, добавление строки для создания графика с использованием `Matplotlib`:


```
# Строим график
plt.plot(range(1, num_generations + 1), best_values)
plt.xlabel('Generation')
plt.ylabel('Best Fitness Value')
plt.title('Fitness Evolution')
plt.show()
```

Результат работы визуализации представлен на рисунке 5.

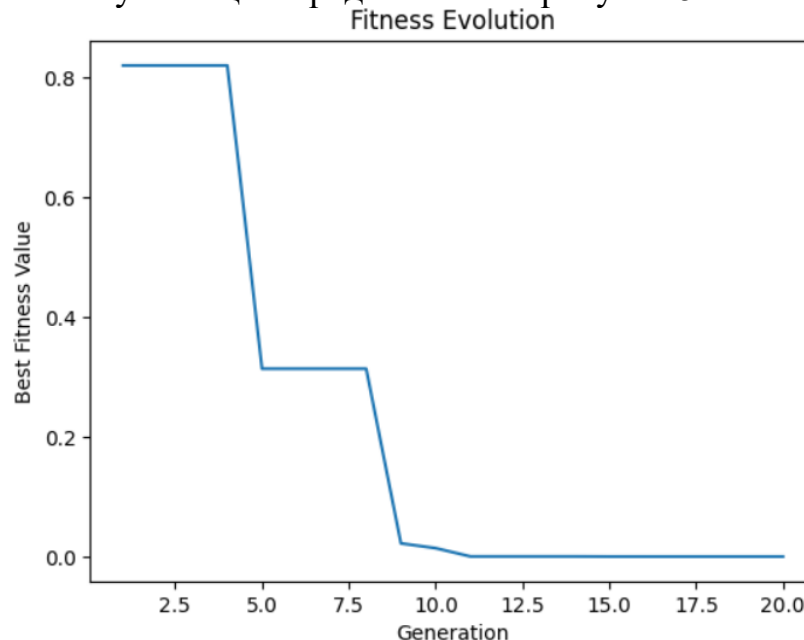


Рисунок 5. График для визуализации работы фитнес функции

В предыдущем разделе мы успешно применили генетический алгоритм для решения задачи минимизации функции:

$$f(x) = x^2 - 4x + 4$$

Этот опыт показал нам, как алгоритм может быть эффективно применен для решения задач оптимизации. Теперь давайте углубим наше понимание этого метода, переключившись на задачу максимизации. Конкретно, рассмотрим функцию:

$$f(x) = -x^2 - 4x - 4,$$

попробуем найти её максимальное значение с помощью генетического алгоритма.

Этап 1.

```
import random
import numpy as np

# Функция фитнеса (такая же, как искомая функция)
def fitness(x):
    return -x ** 2 + 4 * x - 4
```

Этап 2.

```
# Инициализация популяции
population_size = 10
population = [random.uniform(0, 4) for _ in range(population_size)]

# Параметры генетического алгоритма
num_generations = 50
mutation_rate = 0.1
```

Этап 3.

```
# Главный цикл генетического алгоритма
for generation in range(num_generations):
    # Оценка фитнеса
    fitness_values = [fitness(x) for x in population]

    # Селекция: выбор двух лучших родителей
    best_parents = sorted(population, key=fitness, reverse=True)[:2]

    # Кроссовер и мутация для создания нового поколения
    offspring = []
    while len(offspring) < population_size - 2:
        parent1, parent2 = random.sample(best_parents, 2)
        child = (parent1 + parent2) / 2
        if random.random() < mutation_rate:
            child += random.uniform(-0.5, 0.5)
        offspring.append(child)
```

Этап 4.

```
# Формирование нового поколения
population = best_parents + offspring

# Вывод информации о текущем поколении
print(f"Generation {generation + 1}, Best value: {max(fitness_values)}")

# Вывод финальной популяции и лучшего решения
print(f"Final population: {population}")
print(f"Best solution: {max(population, key=fitness)}")
```

Этот код просто модифицирует ранее описанный алгоритм, чтобы максимизировать функцию:

$$f(x) = -x^2 - 4x - 4,$$

Как и раньше, алгоритм начинает с случайной популяции кандидатов и применяет операторы селекции, кроссовера и мутации, чтобы создать новое поколение кандидатов. Затем процесс повторяется, пока не будет достигнуто заданное количество поколений. Результат работы программы представлен на рисунке 5.

Задания для самостоятельной работы:

1. Изменение функции фитнеса: Модифицируйте текущий код, чтобы он мог минимизировать или максимизировать функцию $f(x) = -x^2 - 4x - 4$.
2. Проанализируйте результаты и сравните их с предыдущим примером.
3. Сравнительный анализ: Реализуйте генетический алгоритм для функции $f(x) = x^2 + 4x - 4$,
4. и сравните результаты с первоначальной функцией. Объясните, как различные коэффициенты влияют на поведение алгоритма.
5. Адаптивная мутация: Измените алгоритм так, чтобы вероятность мутации изменялась в зависимости от поколения или среднего фитнеса популяции. Исследуйте, как это влияет на сходимость алгоритма.
6. Введение кроссовера с несколькими точками: Замените простой кроссовер среднего арифметического на кроссовер с двумя или более точками. Анализируйте, как это влияет на скорость сходимости и качество решения.
7. Визуализация: Добавьте графическую визуализацию для отслеживания изменения лучшего и худшего значения функции фитнеса в каждом поколении. Это может быть реализовано, например, с помощью библиотеки `matplotlib` в Python.
8. Производительность и сходимость: Запустите алгоритм несколько раз с различными начальными популяциями и проанализируйте, насколько результаты стабильны. Попробуйте объяснить, почему алгоритм сходится к определенному решению в каждом случае.
9. Параметризация алгоритма: Исследуйте влияние различных параметров алгоритма (размер популяции, вероятность мутации, количество поколений и так далее) на эффективность решения задачи.
10. Расширение на многомерные функции: Модифицируйте код для работы с функциями нескольких переменных. Оцените, как увеличение размерности влияет на эффективность алгоритма.
11. Сравнение с другими методами оптимизации: Попробуйте решить ту же задачу с помощью других методов оптимизации, например, градиентного спуска, и сравните результаты.

Список рекомендованных источников

1. Bäck, Thomas. "Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms". — Oxford University Press, 1996. — 368;
2. Buontempo, Frances. "Genetic Algorithms and Machine Learning for Programmers". — Pragmatic Bookshelf, 2019. — 200 с.;
3. Deb, Kalyanmoy. "Genetic Algorithms: Concepts and Designs". — Springer, 1999. — 400 с.;
4. Goldberg, David E. "Genetic Algorithms in Search, Optimization, and Machine Learning". — Addison-Wesley, 1989. — 432 с. ;
5. Haupt, Randy L., Haupt, Sue Ellen. "Practical Genetic Algorithms". — Wiley, 2004. — 253 с.;
6. Holland, John. "Adaptation in Natural and Artificial Systems". — University of Michigan Press, 1975. — 228 с.;
7. Luke, Sean. "Essentials of Metaheuristics". — Lulu, 2011. — 230 с.;
8. Mitchell, Melanie. "An Introduction to Genetic Algorithms". — MIT Press, 1998. — 162 с.;
9. Simon, Dan. "Evolutionary Optimization Algorithms". — Wiley, 2013. — 772 с.;
10. Yu, Xinjie, Gen, Mitsuo. "Introduction to Evolutionary Algorithms". — Springer, 2010. — 427 с.

ПРИЛОЖЕНИЕ

Код полностью для работы с квадратным уравнением: $f(x) = x^2 - 4x + 4$

```
import random

# Функция фитнеса (в данном случае, чем меньше, тем лучше)
def fitness(x):
    return x ** 2 - 4 * x + 4

# Инициализация популяции
population_size = 10
population = [random.uniform(-10, 10) for _ in range(population_size)]

# Параметры алгоритма
num_generations = 20
mutation_rate = 0.2

# Генетический алгоритм
for generation in range(num_generations):
    # Оценка фитнеса каждой хромосомы
    fitness_values = [fitness(x) for x in population]

    # Селекция: выбираем двух лучших родителей
    best_parents = sorted(population, key=fitness)[:2]

    # Кроссовер: среднее арифметическое родителей
    offspring = [(best_parents[0] + best_parents[1]) / 2 for _ in
range(population_size - 2)]

    # Мутация
    for i in range(len(offspring)):
        if random.random() < mutation_rate:
            offspring[i] += random.uniform(-0.5, 0.5)

    # Новое поколение
    population = best_parents + offspring

    # Выводим информацию о текущем поколении
    print(f"Generation {generation + 1}, Best value: {min(fitness_values)}")

# Выводим финальную популяцию и лучшее решение
print(f"Final population: {population}")
print(f"Best solution: {min(population, key=fitness)}")
```

Код полностью для работы с уравнением: $f(x) = -x^2 - 4x - 4$:

```
import random
import numpy as np

# Функция фитнеса (такая же, как искомая функция)
def fitness(x):
    return -x ** 2 + 4 * x - 4

# Инициализация популяции
population_size = 10
population = [random.uniform(0, 4) for _ in range(population_size)]

# Параметры генетического алгоритма
num_generations = 50
mutation_rate = 0.1

# Главный цикл генетического алгоритма
for generation in range(num_generations):
    # Оценка фитнеса
    fitness_values = [fitness(x) for x in population]

    # Селекция: выбор двух лучших родителей
    best_parents = sorted(population, key=fitness, reverse=True)[:2]

    # Кроссовер и мутация для создания нового поколения
    offspring = []
    while len(offspring) < population_size - 2:
        parent1, parent2 = random.sample(best_parents, 2)
        child = (parent1 + parent2) / 2
        if random.random() < mutation_rate:
            child += random.uniform(-0.5, 0.5)
        offspring.append(child)

    # Формирование нового поколения
    population = best_parents + offspring

    # Вывод информации о текущем поколении
    print(f"Generation {generation + 1}, Best value: {max(fitness_values)}")

# Вывод финальной популяции и лучшего решения
print(f"Final population: {population}")
print(f"Best solution: {max(population, key=fitness)}")
```