ГУАП

КАФЕДРА № 43

ОТЧЕТ   
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Дожность |  |  |  | [Скобцов Ю.А](https://pro.guap.ru/inside/profile/11959) |
| д-р техн. наук, профессор |  | подпись, дата |  | инициалы, фамилия |

|  |  |
| --- | --- |
| ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №1 |  |
| Простой генетический алгоритм |  |
| по дисциплине: [Эволюционные методы проектирования программно-информационных систем](https://pro.guap.ru/inside/students/subjects/3391378) |  |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ ГР. | 4236 |  |  |  | Л. Мвале |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург

2025

**Часть 1. Модели линейного и нелинейного программирования.**

1. **Цель Работы**

Целью работы является изучение принципов построения и функционирования простого генетического алгоритма (ПГА), освоение основных генетических операторов (репродукция, кроссинговер, мутация) и исследование зависимости эффективности работы алгоритма от параметров. В индивидуальном задании необходимо исследовать влияние вероятности мутации PmP\_mPm​ на процесс поиска оптимума заданной функции.

1. **Индивидуальное задание**



1. **Краткие теоретические сведения**

Генетический алгоритм (ГА) — это эвристический метод поиска и оптимизации, основанный на принципах естественного отбора и эволюции. Основные идеи ГА заимствованы из биологии:

* Особь — потенциальное решение задачи, представленное в виде хромосомы.
* Хромосома — закодированное решение задачи (чаще всего в двоичном виде).
* Ген — элемент хромосомы (бит).
* Популяция — набор особей, рассматриваемых в текущем поколении.
* Фитнесс-функция — целевая функция, которая определяет качество решения.

Работа ПГА основана на следующих операторах:

1. Репродукция (селекция) — отбор наиболее приспособленных особей для воспроизводства.
2. Кроссинговер (скрещивание) — обмен частями хромосом родителей для формирования потомков.
3. Мутация — случайное изменение генов в хромосоме для поддержания разнообразия популяции.

Алгоритм работает итеративно: создается начальная популяция, оцениваются ее особи, затем применяются генетические операторы, формируется новое поколение, и процесс повторяется до достижения критерия остановки (например, числа поколений или точности решения).

Вероятность мутации Pm — ключевой параметр, влияющий на разнообразие решений.

* При слишком малом Pm возможна преждевременная сходимость (популяция быстро застревает в локальном оптимуме).
* При слишком большом Pm алгоритм превращается в случайный поиск, теряя эволюционные преимущества.  
  Поэтому важно исследовать зависимость работы ГА от значения вероятности мутации.

1. **Программа Листинг**

import numpy as np

import matplotlib.pyplot as plt

import matplotlib.animation as animation

# Function to maximize

def f(x):

    if np.any(x == 2):

        return -np.inf

    return np.cos(2\*x) / np.abs(x - 2)

# GA parameters

POP\_SIZE = 100

GENES = 16

GENERATIONS = 100

CROSSOVER\_RATE = 0.7

MUTATION\_RATE = 0.01

intervals = [(-10, 2-1e-5), (2+1e-5, 10)]

def binary\_to\_real(chrom, x\_min, x\_max):

    integer = int("".join(str(int(b)) for b in chrom), 2)

    max\_int = 2\*\*GENES - 1

    return x\_min + (integer / max\_int) \* (x\_max - x\_min)

def init\_population():

    return np.random.randint(0, 2, size=(POP\_SIZE, GENES))

def evaluate(pop, x\_min, x\_max):

    return np.array([f(binary\_to\_real(chrom, x\_min, x\_max)) for chrom in pop])

def select(pop, fitness):

    total\_fit = np.sum(fitness - np.min(fitness) + 1e-6)

    probs = (fitness - np.min(fitness) + 1e-6) / total\_fit

    indices = np.random.choice(np.arange(POP\_SIZE), size=POP\_SIZE, p=probs)

    return pop[indices]

def crossover(pop):

    for i in range(0, POP\_SIZE, 2):

        if np.random.rand() < CROSSOVER\_RATE:

            point = np.random.randint(1, GENES)

            pop[i, point:], pop[i+1, point:] = pop[i+1, point:].copy(), pop[i, point:].copy()

    return pop

def mutate(pop):

    for i in range(POP\_SIZE):

        for j in range(GENES):

            if np.random.rand() < MUTATION\_RATE:

                pop[i, j] = 1 - pop[i, j]

    return pop

# Run GA and keep history of best per generation

def run\_ga(x\_min, x\_max):

    pop = init\_population()

    history = []

    best\_x, best\_f = None, -np.inf

    for gen in range(GENERATIONS):

        fitness = evaluate(pop, x\_min, x\_max)

        xs = np.array([binary\_to\_real(chrom, x\_min, x\_max) for chrom in pop])

        idx = np.argmax(fitness)

        gen\_best\_f, gen\_best\_x = fitness[idx], xs[idx]

        # 🔑 Save \*all candidates\* and the generation-best

        history.append((xs, fitness, gen\_best\_x, gen\_best\_f, gen))

        if gen\_best\_f > best\_f:

            best\_f, best\_x = gen\_best\_f, gen\_best\_x

        pop = select(pop, fitness)

        pop = crossover(pop)

        pop = mutate(pop)

    return history, best\_x, best\_f

# Combine histories from both intervals

history = []

best\_global\_x, best\_global\_f = None, -np.inf

for (x\_min, x\_max) in intervals:

    h, x, fx = run\_ga(x\_min, x\_max)

    history += h

    if fx > best\_global\_f:

        best\_global\_f, best\_global\_x = fx, x

print(f"\n✅ Global maximum found: f(x)={best\_global\_f:.5f} at x={best\_global\_x:.5f}")

# Plot setup

fig, ax = plt.subplots()

x\_vals\_left = np.linspace(-10, 2-1e-3, 1000)

x\_vals\_right = np.linspace(2+1e-3, 10, 1000)

ax.plot(x\_vals\_left, f(x\_vals\_left), 'r', label="f(x)")

ax.plot(x\_vals\_right, f(x\_vals\_right), 'r')

scat = ax.scatter([], [], c='b', s=20)

text = ax.text(0.02, 0.95, '', transform=ax.transAxes, fontsize=10,

               verticalalignment='top')

ax.set\_xlim(-10, 10)

ax.set\_ylim(-10, 10)

ax.set\_title("GA optimization of f(x)=cos(2x)/|x-2|")

# 🔑 Update function with alternating best values

def update(frame):

    xs, fitness, gen\_best\_x, gen\_best\_f, gen = history[frame]

    scat.set\_offsets(np.c\_[xs, fitness])

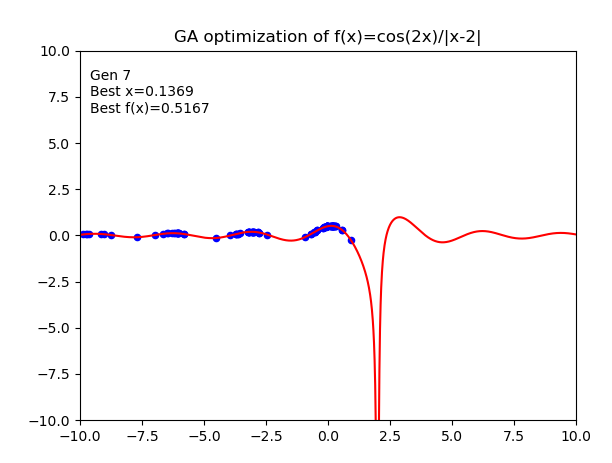
    text.set\_text(f"Gen {gen+1}\nBest x={gen\_best\_x:.4f}\nBest f(x)={gen\_best\_f:.4f}")

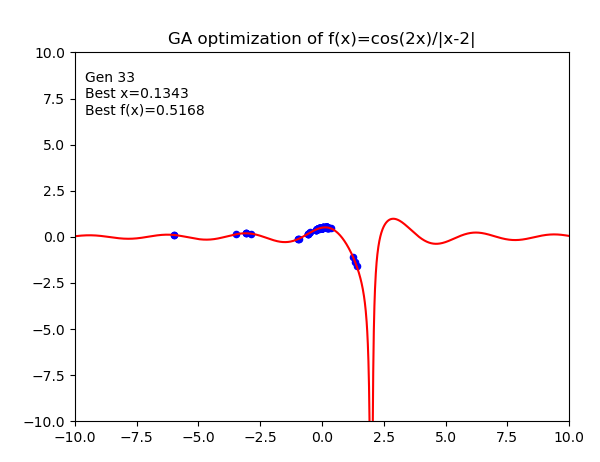
    return scat, text

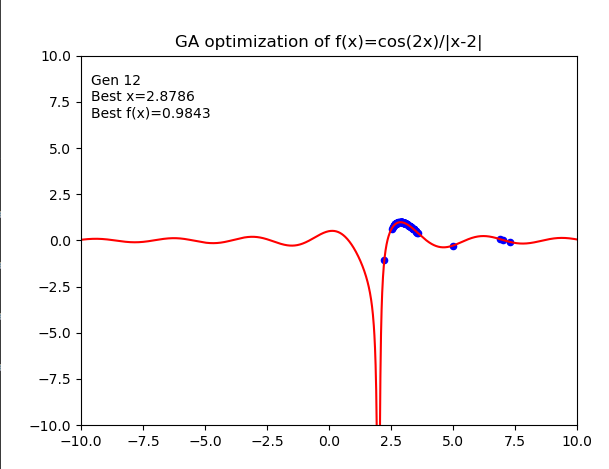
ani = animation.FuncAnimation(fig, update, frames=len(history), interval=200, repeat=False)

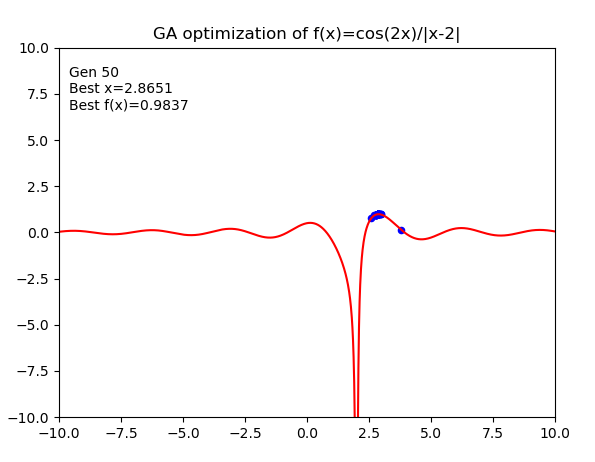
plt.show()

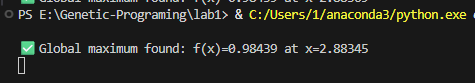
1. **результаты выполнения**











1. **Письменный ответ на контрольный вопрос №15**

Вопрос 15: Исследуйте зависимость работы ПГА от значения вероятности мутации Pm.

Ответ:  
Вероятность мутации Pm является важным параметром генетического алгоритма, определяющим вероятность случайного изменения гена в хромосоме. Ее влияние можно описать следующим образом:

1. Малое значение PmP\_mPm​ (например, 0.001–0.01):

* Популяция быстро теряет разнообразие.
* Увеличивается вероятность преждевременной сходимости к локальному максимуму или минимуму.
* Скорость работы алгоритма высокая, но точность нахождения глобального оптимума снижается.

1. Среднее значение Pm (например, 0.05–0.1):

* Поддерживается достаточное генетическое разнообразие.
* Сохраняется баланс между скоростью сходимости и возможностью выхода из локальных экстремумов.
* Обычно такие значения считаются оптимальными для большинства задач.

Большое значение Pm (например, >0.2):

* Алгоритм начинает вести себя как случайный поиск.
* Теряется накопленная информация о предыдущих поколениях.
* Сходимость замедляется, а качество найденного решения снижается.

**Выводы**

Оптимальное значение Pm зависит от конкретной задачи и структуры фитнесс-функции. Для большинства задач поиска экстремума рекомендуется использовать малую вероятность мутации (в диапазоне 0.01–0.1). Слишком малые значения приводят к стагнации, а слишком большие — к потере эволюционных преимуществ. В ходе экспериментов с функцией, заданной по варианту, необходимо варьировать Pm и фиксировать скорость сходимости и качество решения для нахождения оптимального диапазона.