ГУАП

КАФЕДРА № 43

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

Дожность		Скобцов Ю.А
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

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.

2. Индивидуальное задание

15	Cos(2x)/abs(x-2)	$x \in [-10,2),(2,10]$, max

3. 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 P_m — ключевой параметр, влияющий на разнообразие решений.

- При слишком малом P_m возможна преждевременная сходимость (популяция быстро застревает в локальном оптимуме).
- При слишком большом P_m алгоритм превращается в случайный поиск, теряя эволюционные преимущества.

  Поэтому важно исследовать зависимость работы ГА от значения вероятности мутации.

4. Программа Листинг

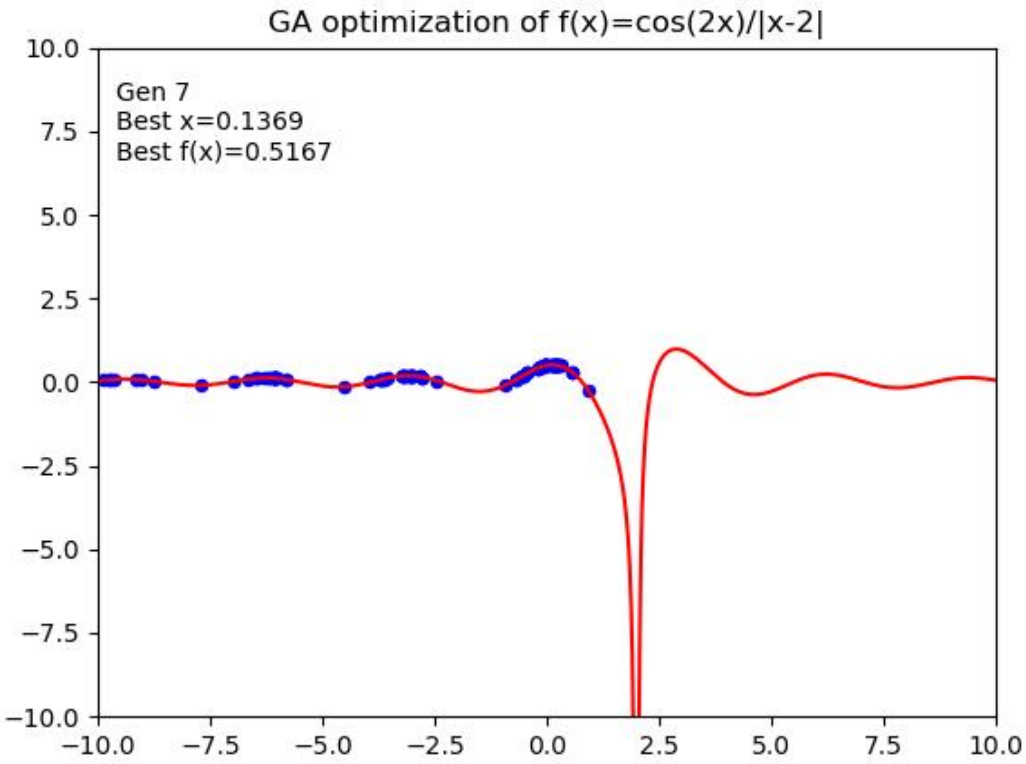
```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation

# 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)
```

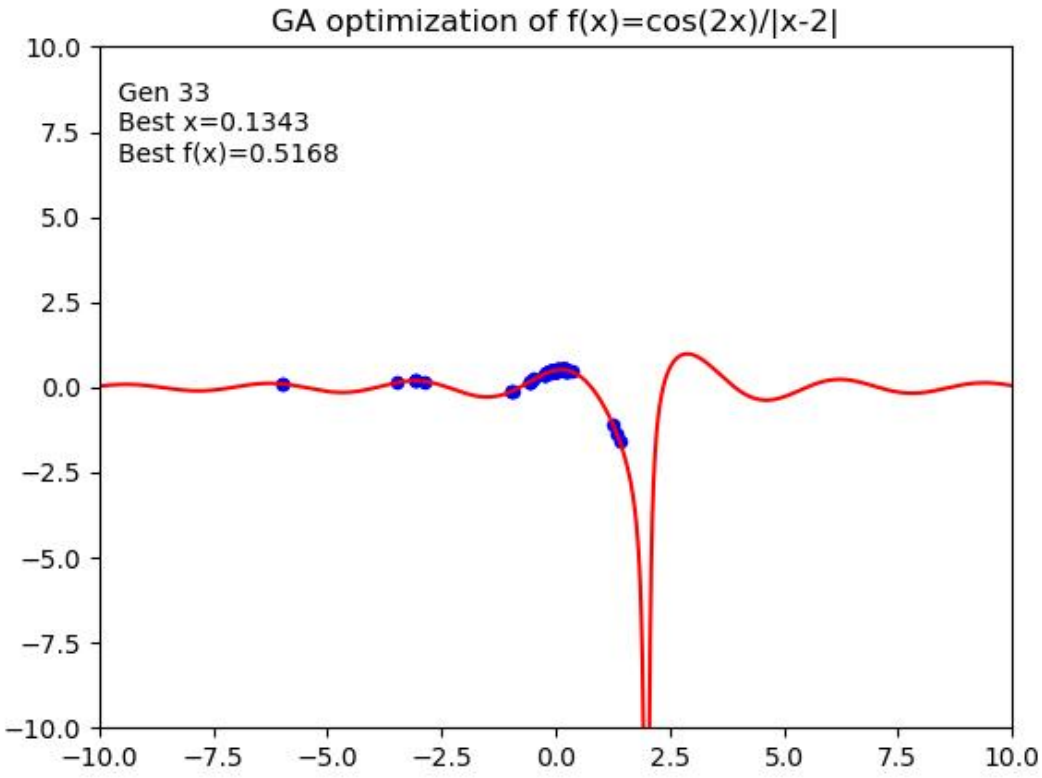
```
    probs = (fitness - np.min(fitness) + 1e-6) / total_fit
    indices = np.random.choice(np.arange(POP_SIZE), size=POP_SIZE, p=probs)
    return pop[indices]

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))
```

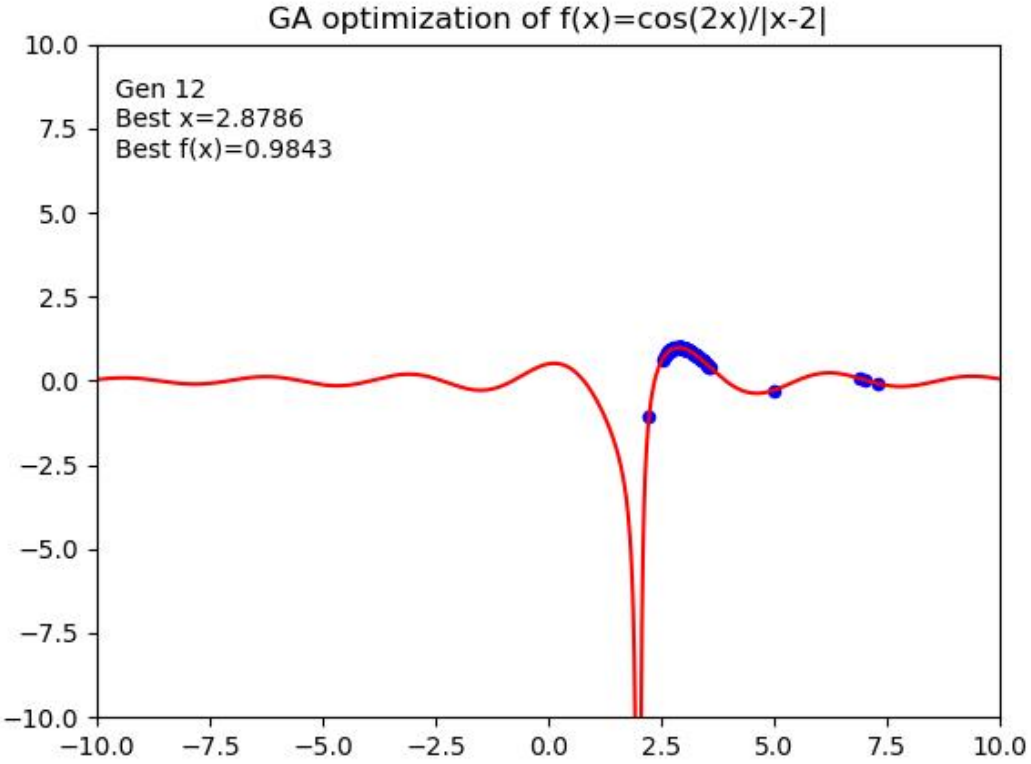
```
        if gen_best_f > best_f:
            best_f, best_x = gen_best_f, gen_best_x

        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)
```

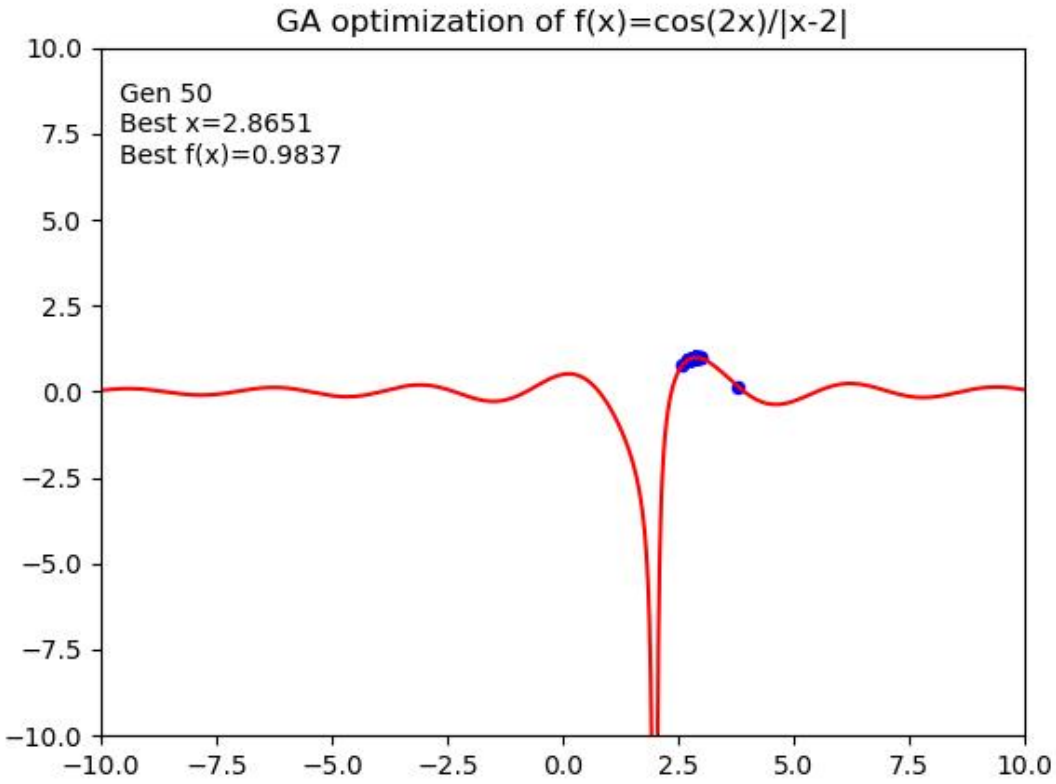
```
ax.set_ylim(-10, 10)
ax.set_title("GA optimization of f(x)=cos(2x)/|x-2|")

#     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()
```

5. результаты выполнения




GA optimization of f(x)=cos(2x)/|x-2|
Gen 33
Best x=0.1343
Best f(x)=0.5168
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0


GA optimization of f(x)=cos(2x)/|x-2|
Gen 12
Best x=2.8786
Best f(x)=0.9843
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0




```
PS E:\Genetic-Programing\lab1> & C:/Users/1/anaconda3/python.exe
✅ Global maximum found: f(x)=0.98439 at x=2.88345
```

6. Письменный ответ на контрольный вопрос №15

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

Ответ:

Вероятность мутации P_m 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.

2. Среднее значение P_m (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 P_m (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 P_m 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 P_m и фиксировать скорость сходимости и качество решения для нахождения оптимального диапазона.