دانشکدهٔ مهندسی کامپیوتر

عنوان: گزارش پروژهٔ الگوریتم ژنتیک درس هوش مصنوعی و سیستم های خبره

توضیحات: هدف این پروژه این است که برنامه ای بنویسیم که فقط با استفاده از الگوریتم ژنتیک، دستگاه های چند معادله چند مجهول با شرایط مطرح شده در هر بخش را حل کند و پاسخ آن را خروجی بدهد.

نام استاد: دکتر عبدی

نام دانشجو: امیرحسین اسلامی

بهار ۱۴۰۴

چکیده

در این پروژه با بهره‌گیری از الگوریتم ژنتیک، مسئله‌ی حل دستگاه معادلات خطی و غیرخطی چندمجهولی مورد بررسی و پیاده‌سازی قرار گرفته است. هدف اصلی، طراحی سیستمی است که تنها با اتکا به مفاهیم الگوریتم ژنتیک، بدون استفاده از روش‌های مستقیم عددی، قادر به یافتن پاسخ تقریبی معادلات ورودی باشد. روند حل به‌صورت مرحله‌ای شامل دریافت معادلات، تولید جمعیت اولیه، محاسبه میزان خطا (تابع برازش)، انتخاب والدین، انجام عمل تقاطع و جهش، و در نهایت تولید نسل‌های جدید تا رسیدن به جواب مناسب پیاده‌سازی شده است. پروژه شامل سه بخش است که در آن به ترتیب حل معادلات دو، سه و چهار مجهولی بررسی شده است. همچنین تلاش‌هایی جهت بهینه‌سازی الگوریتم برای پرهیز از گیر افتادن در کمینه‌های محلی انجام شده است. در نهایت، دقت الگوریتم با نمونه‌های ارائه‌شده در جزوه درسی مقایسه شده و پیشنهاداتی جهت بهبود ارائه شده است.

واژه‌های کلیدی: الگوریتم ژنتیک، دستگاه معادلات، تابع برازش، تقاطع، جهش ژنتیکی

فهرست مطالب

فهرست اشکال

فهرست اشکال

فصل ۱:

بخش اول

برای این بخش ابتدا ۴ تا ورودی را از کاربر می گیریم که شامل ضرایب مجهولات ما (x,y) هست و البته c1 و c2 که در طرف دیگر تساوی هستند.

```
def get_equation_coefficients() -> tuple[tuple[float, float, float], tuple…:
    print("(a1 * x + b1 * y = c1):")
    a1 = float(x=input(prompt="input a1: "))
    b1 = float(x=input(prompt="input b1: "))
    c1 = float(x=input(prompt="input c1: "))

    print("\n(a2 * x + b2 * y = c2):")
    a2 = float(x=input(prompt="input a2: "))
    b2 = float(x=input(prompt="input b2: "))
    c2 = float(x=input(prompt="input c2: "))

    return (a1, b1, c1), (a2, b2, c2)
```

شکل (۱-۱) تابع دریافت ضرایب معادلات

ایجاد جمعیت اولیه:

```
def generate_initial_population(pop_size=100, lower_bound=-100, upper_bound=100) -> list:
    population: list = []
    for _ in range(stop/pop_size):
        x: float = random.uniform(a=lower_bound, b=upper_bound)
        y: float = random.uniform(a=lower_bound, b=upper_bound)
        chromosome: list[float] = [x, y]
        population.append(object/chromosome)
    return population
```

شکل (۲-۱) تابع ایجاد جمعیت اولیه

محاسبه خطای هر کروموزوم:

```python
def fitness(chromosome, eq1, eq2) -> Any:
    x: Any,y: Any = chromosome
    a1: Any, b1: Any, c1: Any = eq1
    a2: Any, b2: Any, c2: Any = eq2

    error1: Any = (a1*x + b1*y) - c1
    error2: Any = (a2*x + b2*y) - c2

    totalSqueredError: Any = error1**2 + error2**2

    return totalSqueredError
```

شکل (۳-۱) تابع محاسبه خطا

تابع انتخاب والدین:

```python
def select_parents(population, eq1, eq2, num_parents=NUM_PARENTS_SELECTION) -> list:
    population_sorted: list = sorted(iterable/population, key=lambda chromo: fitness(chromosome=chromo, eq1=eq1, eq2=
    return population_sorted[:num_parents]
```

شکل (۴-۱) تابع انتخاب والدین

در این تابع بر مبنای خطای هر کروموزوم آن ها را مرتب کرده سپس آن هایی که بهتر هستند به عنوان والد انتخاب می شوند.

ایجاد نسل جدید:

```python
def cross_over(parent1, parent2) -> list:
    x: Any = (parent1[0] + parent2[0])/2
    y: Any = (parent1[1] + parent2[1])/2
    return [x,y]
```

شکل (۵-۱) تابع ایجاد کروموزوم

جهش:

```python
def mutate(chromosome, mutation_rate = MUTATION_RATE) -> Any:

    if(random.random < mutation_rate):
        chromosome[0] += random.uniform(a=MUT_MIN_BOUND, b=MUT_MAX_BOUND)

    if(random.random < mutation_rate):
        chromosome[1] += random.uniform(a=MUT_MIN_BOUND,b=MUT_MAX_BOUND)

    return chromosome
```

شکل (۶-۱) تابع جهش

برای آنکه کمی تصادفی تر کار کنیم و همه ی کروموزوم ها به یک سمت همسو نشوند با یک تصادف جهش را انجام می دهیم.

پیاده سازی الگوریتم ژنتیک با استفاده از توابع پیشین:

```python
def genetic_algorithm(eq1, eq2, generations=1000, pop_size=100) -> Any:
    lower: Any, upper: Any = estimate_bounds(eq1=eq1, eq2=eq2)
    population: list = generate_initial_population(pop_size=pop_size, lower_bound=lower, upper_bound=upper)

    for generation in range(stop/generations):
        population: list = sorted(iterable/population, key=lambda chromo: fitness(chromosome=chromo, eq1=eq1, eq2=eq2))
        best_fitness: Any = fitness(chromosome=population[0], eq1=eq1, eq2=eq2)

        if best_fitness < 1e-6:
            print(f"Solution found in generation {generation}")
            return population[0]

        parents: list = select_parents(population=population, eq1=eq1, eq2=eq2, num_parents=20)

        # Generate new population
        new_population: list = []
        while len(obj/new_population) < pop_size:
            parent1: Any = random.choice(seq=parents)
            parent2: Any = random.choice(seq=parents)
            child: list = crossover(parent1=parent1, parent2=parent2)
            child: list = mutate(chromosome=child)
            new_population.append(object/child)

        population: list = new_population

    print("No exact solution found. Best approximation:")
    return sorted(iterable/population, key=lambda chromo: fitness(chromosome=chromo, eq1=eq1, eq2=eq2))[0]
```

شکل (۷-۱) تابع الگوریتم ژنتیک

با استفاده از توابعی که پیش از این نوشتیم حالا می توانیم یک نسل اولیه بسازیم سپس از بین آن ها بهترین ها را جدا کرده و دوباره نسل جدیدی ایجاد کرده و جهش را انجام داده و سپس تا زمانی این فرایند

را تکرار کنیم که به دقت مورد نظر خودمان برسیم.

تست الگوریتم:

```
eq1: tuple[float, float, float], eq2: tuple[float, … = get_equation_coefficients()
solution: Any = genetic_algorithm(eq1=eq1, eq2=eq2)
print(f"Approximate solution: x = {solution[0]}, y = {solution[1]}")
```

شکل (۸-۱) کد تست الگوریتم

```
(a1 * x + b1 * y = c1):
input a1: 1
input b1: 2
input c1: 4
```

شکل (۹-۱) ورودی اول داده شده

```
(a2 * x + b2 * y = c2):
input a2: 4
input b2: 4
input c2: 12
Solution found in generation 8
Approximate solution: x = 2.0002460012081356, y = 0.9998368463088123
```

شکل (۱۰-۱) ورودی دوم داده شده و جواب نهایی

$$\begin{cases} x + 2y = 4 \\ 4x + 4y = 12 \end{cases} \Longrightarrow \begin{cases} x = 2 \\ y = 1 \end{cases}$$

شکل (۱۱-۱) مثال جزوه

همانطور که مشاهده می شود با دقت بسیار خوبی توانستیم به جواب برسیم.

فصل ۲: بخش دوم

برای بخش دوم هم همین مســـیر را باید برویم ولی ابتدا باید تابعی که کاربر داده را دریافت کنیم و از آن معادله را استخراج کنیم.

```python
def parse_equation(equation_str) -> Callable[..., Any]:
    """
    Parses a user-provided equation string into a lambda function.
    :param equation_str: The equation string (e.g., "3x + 5/y + 3.39z^2 = 194").
    :return: A lambda function that takes x, y, z as inputs.
    """
    # Split the equation by '='
    if "=" not in equation_str:
        raise ValueError("Equation must contain an '=' sign.")

    left_expr: Any, right_expr: Any = equation_str.split("=")

    # Clean and prepare the expression for safe evaluation
    allowed_chars: Pattern[str] = re.compile(pattern=r"[^0-9xyz\+\-\*/\.\^\(\)\s]")
    if allowed_chars.search(string=left_expr):
        raise ValueError("Invalid characters in the equation.")

    # Replace ^ with ** for Python exponentiation
    left_expr: Any = left_expr.replace("^", "**")

    # Convert to a lambda function
    equation_func: Callable[..., Any] = lambda x, y, z: eval(source/left_expr, globals={"x": x, "y": y, "z": z}) - float(x=right_expr)

    return equation_func
```

شکل (۱-۲) تابع بررسی رشته ورودی به عنوان تابع

```python
# Example for parsing equation
equation = "3*x + 5/y + 3.39*z**2 = 194"
parsed_func: Callable[..., Any] = parse_equation(equation_str=equation)
print("Parsed Function Test:", parsed_func(x=1, y=1, z=1))
```

شکل (۲-۲) بررسی درستی عملکرد تابع

```
PS C:\Home\University\AI\Work
Parsed Function Test: -182.61
```

شکل (۳-۲) خروجی دریافتی از بررسی عملکرد تابع بررسی رشته ورودی

خب همانطور که روشن است این تابع روند درستی را طی می کند و به درستی مقدار هر معادله را می تواند ارزیابی کند و در ادامه از این اندازه ی خطا استفاده می کنیم و سعی داریم تا آن را کمینه کنیم.

```python
def generate_initial_population(population_size=POP_SIZE_INIT_POP, value_range=(LOWER_BOUND_INIT_POP,UPPER_BOUND_INIT
    """
    Generates the initial population for the genetic algorithm.
    :param population_size: Number of chromosomes in the population.
    :param value_range: A tuple (min, max) for the range of x, y, z values.
    :return: A numpy array of shape (population_size, 3).
    """
    min_val: Any, max_val: Any = value_range
    # Generate population with random values within the given range
    population: NDArray[float64] = np.random.uniform(low=min_val, high=max_val, size=(population_size, 3))
    return population
```

شکل (۴-۲) تابع ایجاد جمعیت نخست

```
[ 86.97729318  55.52272624 -63.15352858]
[ 89.38986747 -54.7790876   20.51801454]
[ 75.41624617  70.6048285   32.71599405]
[ 10.38029788 -59.1987186   80.97408002]
[ 57.81295752 -83.49314924  21.3168993 ]
[ 53.58523012  51.50790263  88.58781311]
[  8.9172475  -28.59502087  -3.64102061]
[  5.11679739 -94.12217514   1.81424585]]
```

شکل (۵-۲) نمونه جمعیت اولیه ایجاد شده

```python
def calculate_fitness(population, equations) -> NDArray:
    """
    Calculates the fitness of each chromosome in the population.
    :param population: A numpy array of shape (population_size, 3).
    :param equations: A list of lambda functions representing the equations.
    :return: A numpy array of fitness values.
    """
    fitness_values: list = []

    for chromosome in population:
        x: Any, y: Any, z: Any = chromosome
        total_error: int = sum(iterable/abs(x/eq(x, y, z)) for eq in equations)
        fitness_values.append(object/total_error)

    return np.array(object=fitness_values)
```

شکل (۶-۲) تابع برازش

با استفاده از تابع برازش انگیزه داریم تا خطای هرکدام از کروموزوم ها را بسنجیم.

```python
def tournament_selection(population, fitness_values, tournament_size=POP_SIZE_INIT_POP) -> NDArray:
    """
    Performs tournament selection on the population.
    :param population: The population of chromosomes.
    :param fitness_values: Array of fitness values for the population.
    :param tournament_size: Number of chromosomes in each tournament.
    :return: A new population after selection.
    """
    selected_population: list = []
    population_size: int = len(obj/population)

    for _ in range(stop/population_size):
        # Randomly select chromosomes for the tournament
        indices: NDArray[long] = np.random.choice(a=population_size, size=tournament_size, replace=False)
        tournament: Any = population[indices]
        tournament_fitness: Any = fitness_values[indices]

        # Select the best chromosome (minimum fitness)
        winner_index: intp = np.argmin(a=tournament_fitness)
        selected_population.append(object/tournament[winner_index])

    return np.array(object=selected_population)
```

شکل (۷-۲) تابع انتخاب کروموزوم های برتر

```python
def single_point_crossover(parents) -> NDArray:
    """
    Performs single-point crossover on the selected parents.
    :param parents: A numpy array of selected parents.
    :return: A new population after crossover.
    """
    offspring: list = []
    for i in range(start/0, stop/len(obj/parents), step=2):
        parent1: Any = parents[i]
        parent2: Any = parents[i + 1] if (i + 1) < len(obj/parents) else parents[i]

        # Random crossover point
        crossover_point: int = np.random.randint(low=1, high=len(obj/parent1))

        # Generate offspring
        child1: NDArray = np.concatenate(arrays/(parent1[:crossover_point], parent2[crossover_point:]))
        child2: NDArray = np.concatenate(arrays/(parent2[:crossover_point], parent1[crossover_point:]))

        offspring.append(object/child1)
        offspring.append(object/child2)

    return np.array(object=offspring)
```

شکل (۸-۲) تابع ایجاد کروموزوم جدید

```python
def mutation(offspring, mutation_rate, value_range) -> Any:
    """
    Performs mutation on the offspring population.
    :param offspring: The population after crossover.
    :param mutation_rate: The probability of each gene mutating.
    :param value_range: A tuple (min, max) for the range of mutation.
    :return: The mutated offspring population.
    """
    min_val: Any, max_val: Any = value_range

    for i in range(stop/len(obj/offspring)):
        for j in range(stop/len(obj/offspring[i])):
            if np.random.rand() < mutation_rate:
                # Apply mutation by adding a random value within the range
                offspring[i][j] += np.random.uniform(min_val * 0.1, max_val * 0.1)

                # Ensure it stays within the specified range
                offspring[i][j] = np.clip(offspring[i][j], min_val, max_val)

    return offspring
```

شکل (۹-۲) تابع جهش

جهت گیر نکردن در کمینه های محلی به این تابع در این پرسش هم نیازمندیم.

```python
def genetic_algorithm(equations, population_size=100, generations=1000, value_range=(-10, 10),
                      tournament_size=3, mutation_rate=0.05) -> tuple[Any, Any]:
    population: NDArray[float64] = generate_initial_population(population_size=population_size, value_ran…value_range)

    for generation in range(stop/generations):
        fitness_values: NDArray = calculate_fitness(population=population, equations=equations)

        if generation % 100 == 0:
            best_fitness: Any = np.min(a=fitness_values)
            print(f"Generation {generation}, Best Fitness: {best_fitness}")

        selected_population: NDArray = tournament_selection(population=population, fitness_values=fitness_values, tourname…tournament_size)
        offspring: NDArray = single_point_crossover(parents=selected_population)
        population: NDArray = mutation(offspring=offspring, mutation_rate=mutation_rate, value_rang…value_range)

    # Final evaluation
    fitness_values: NDArray = calculate_fitness(population=population, equations=equations)
    best_index: intp = np.argmin(a=fitness_values)
    best_solution: Any = population[best_index]
    best_fitness: Any = fitness_values[best_index]

    print(f"\nBest Solution: {best_solution}")
    print(f"Fitness: {best_fitness}")

    return best_solution, best_fitness
```

شکل (۱۰-۲) تابع الگوریتم ژنتیک

با بکارگیری توابع نوشته شده ی پیشین حالا الگوریتم ژنتیک را پیاده سازی کرده ایم در این تابع و اقدام به تست آن می کنیم.

```python
def get_equations_from_user() -> list[Callable[..., Any]]:
    equations
    for i in range(stop/3):
        equations.append(object/parse_equation(equation_str=input(prompt=f"Write equation({i+1}): ")))
    return equations
```

شکل (۱۱-۲) تابع دریافت معادلات از کاربر

```python
174
175  equations: list[Callable[..., Any]] = [
176      parse_equation(equation_str="3*x + 5/y + 3.39*z**2 = 194"),
177      parse_equation(equation_str="2*x - 4*y + z = 5"),
178      parse_equation(equation_str="x**2 + y**2 - z = 10")
179  ]
180  best_solution: Any, best_fitness: Any = genetic_algorithm(equations=equations)
181
```

```
PROBLEMS 1  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

Generation 0, Best Fitness: 13.92328611521287
Generation 100, Best Fitness: 5.124826760159124
Generation 200, Best Fitness: 5.119607244014457
Generation 300, Best Fitness: 5.11832928898067
Generation 400, Best Fitness: 5.1174632866116525
Generation 500, Best Fitness: 5.1174632866116525
Generation 600, Best Fitness: 5.117034115377589
Generation 700, Best Fitness: 5.117034115377589
Generation 800, Best Fitness: 5.117034115377589
Generation 900, Best Fitness: 5.117034115377589

Best Solution: [ 0.96564344 -1.97903975 -7.55763302]
Fitness: 5.117034115377589
```

شکل (۱۲-۲) سنجش الگوریتم

خب الگوریتم ما در یک کمینه محلی گیر کرده و پاسخی که به ما داده است زیاد خوب نیست بنابراین باید الگوریتم را بهبود دهیم.

اندازه های هایپر پارامتر هارا بزرگتر کردیم.

```
INIT_POPULATION_SIZE = 500
LOWER_BOUND_INIT_POP = -1000
UPPER_BOUND_INIT_POP = 1000
NUM_PARENTS_SELECTION = 20
MUTATION_RATE = 0.1
MUT_MIN_BOUND = -20
MUT_MAX_BOUND = 20
BEST_FITNESS_MAX_ERROR = 1e-8
GENERATIONS_DEFAULT_AMOUNT = 10000
TOURNAMENT_DEFAULT_SIZE = 3
```

شکل (۱۳-۲) تغییر هایپر پارامتر ها

```
 [ 392.83605492  370.15703729  670.32935503]
 [-765.73278693  752.7258173  -748.71097317]]
Generation 0, Best Fitness: 23957.622829662978
Generation 100, Best Fitness: 8.116951563698402
Generation 200, Best Fitness: 8.116951563698402
Generation 300, Best Fitness: 8.081124016561908
Generation 400, Best Fitness: 8.081124016561908
Generation 500, Best Fitness: 8.081124016561908
Generation 600, Best Fitness: 8.070218854185423
Generation 700, Best Fitness: 8.070218854185423
Generation 800, Best Fitness: 8.070218854185423
Generation 900, Best Fitness: 8.070218854185423
Generation 1000, Best Fitness: 8.021573426092981
Generation 1100, Best Fitness: 7.9078840534370265
Generation 1200, Best Fitness: 7.9078840534370265
Generation 1300, Best Fitness: 7.9078840534370265
Generation 1400, Best Fitness: 7.9078840534370265
Generation 1500, Best Fitness: 7.9078840534370265
Generation 1600, Best Fitness: 7.895614243534521
Generation 1700, Best Fitness: 7.895614243534521
Generation 1800, Best Fitness: 7.895614243534521
Generation 1900, Best Fitness: 7.895614243534521
Generation 2000, Best Fitness: 7.895614243534521
Generation 2100, Best Fitness: 7.895614243534521
Generation 2200, Best Fitness: 7.895614243534521
```

شکل (۱۴-۲) سنجش درستی کارکرد الگوریتم

شوربختانه دوباره این گیر کردن در کمینه محلی رخ داد و باید نگاه دوباره ای بر الگوریتم داشته باشیم و آن را بهبود دهیم.

```
equations: list[Callable[..., Any]] = [
    parse_equation(equation_str="6*x - 2*y + 8*z = 20"),
    parse_equation(equation_str="y + 8*x * z = -1"),
    parse_equation(equation_str="2*z * y/x + 3/2 * y = 6")
]
best_solution: Any, best_fitness: Any = genetic_algorithm(equations=equations, population_size=500, generations=10000,
                                        value_range=(-100,100), tournament_size=3, mutation_rate=0.2)
```

شکل (۱۵-۲) سنجش مثال آورده شده نمونه در جزوه

```
Generation 9500, Best Fitness: 1.205577234771816
Generation 9600, Best Fitness: 1.285217861146415
Generation 9700, Best Fitness: 1.285217861146415
Generation 9800, Best Fitness: 1.285217861146415
Generation 9900, Best Fitness: 1.285217861146415

Best Solution: [ 4.662001    3.52231979 -0.11626379]
Fitness: 1.0810921744374653
```

شکل (۱۶-۲) دستاورد نهایی

این الگوریتم خوب کار نمی کند و باید اصلاح شود به هیچ عنوان با نتیجه ی ما هم خوانی ندارد اما نزدیک هست و می توانیم نزدیک شویم.

```
179 ]
180
181 equations: list[Callable[..., Any]] = [
182     parse_equation(equation_str="6*x - 2*y + 8*z = 20"),
183     parse_equation(equation_str="y + 8*x * z = -1"),
184     parse_equation(equation_str="2*z * y/x + 3/2 * y = 6")
185 ]
186
187 best_solution: Any, best_fitness: Any = genetic_algorithm(equations=equations, population_size=1000, generations=100,
188                                         value_range=(-100,100), tournament_size=100, mutation_rate=0.2)
189
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

Generation 50, Best Fitness: 0.05294108002987841
Generation 60, Best Fitness: 0.0509299146966411
Generation 70, Best Fitness: 0.02795922426795627 3
Generation 80, Best Fitness: 0.024876614035847844
Generation 90, Best Fitness: 0.024876614035847844

Best Solution: [ 4.89363314  4.14983378 -0.1316719 ]
Fitness: 0.015187340743541888
```

شکل (۱۷-۲) سنجش نمونه معادله داده شده در جزوه

همان گونه که روشن است به گمان زیادی معادلات را اشتباه متوجه می شود بنابراین باید تابع parse_equation را اصلاح کنیم.

```
Tabnine | Edit | Test | Explain | Document
def parse_equation(equation_str) -> Callable[..., float]:
    if "=" not in equation_str:
        raise ValueError("Equation must contain an '=' sign.")

    left_expr: Any, right_expr: Any = equation_str.split("=")

    try:
        left_expr: Any = eval(source/left_expr, globals={"x": x, "y": y, "z": z})
        right_expr: Any = eval(source/right_expr, globals={"x": x, "y": y, "z": z})
    except Exception as e:
        raise ValueError(f"Invalid equation format: {e}")

    def eq_func(x_val, y_val, z_val) -> float:
        try:
            if abs(x/x_val) < X_AVOID_ZERO_THRESHOLD:
                return float(x="inf")
            result: Any = left_expr.subs({x: x_val, y: y_val, z: z_val}) - right_expr
            return float(x=result)
        except Exception:
            return float(x="inf")  # Penalize invalid math
    return eq_func
```

شکل (۱۸-۲) ویرایش تابع parse_equation

مشـکل تابع این بود که در مبدا مقدار طول صـفر داشـت و همین باعث خطا در الگوریتم می شـد بنابراین با استفاده از try/catch این چالش هم درست شد.

تست:

```
Generation 260: Fitness=0.000398 | Solution=[ 0.66666731 -4.99996553  0.75001814]
Generation 270: Fitness=0.000398 | Solution=[ 0.66666731 -4.99996553  0.75001814]
Generation 280: Fitness=0.000398 | Solution=[ 0.66666731 -4.99996553  0.75001814]

Converged at generation 282

Best Solution Found: [ 0.66666731 -4.99996553  0.75001814] | Fitness: 0.00039753
```

شکل (۱۹-۲) آزمایش نمونه مثال درون جزوه

دیده می شود که با موشکافی خوبی به پاسخ درست برسیم .

فصل ۳: بخش سوم

برای این بخش ما باید چهار معادله چهار مجهول حل کنیم تا آســـتانه بســـیار خوبی می توانیم از تابع های پیشین استفاده نماییم.

برای مثال تابع پردازش معادلات ورودی کاربر از بخش دوم می تواند استفاده شود.

```
Tabnine | Edit | Test | Explain | Document
def parse_equation(equation_str) -> Callable[..., float]:
    if "=" not in equation_str:
        raise ValueError("Equation must contain an '=' sign.")

    left_expr: Any, right_expr: Any = equation_str.split("=")

    print(f"Parsing equation: {left_expr.strip()} = {right_expr.strip()}")

    try:
        left_expr: Any = eval(source/left_expr, globals={"x": x, "y": y, "z": z})
        right_expr: Any = eval(source/right_expr, globals={"x": x, "y": y, "z": z})
    except Exception as e:
        raise ValueError(f"Invalid equation format: {e}")

    def eq_func(x_val, y_val, z_val) -> float:
        try:
            if abs(x/x_val) < X_AVOID_ZERO_THRESHOLD:
                return float(x="inf")
            result: Any = left_expr.subs({x: x_val, y: y_val, z: z_val}) - right_expr
            return float(x=result)
        except Exception:
            return float(x="inf")
    return eq_func
```

OUTPUT DEBUG CONSOLE TERMINAL PORTS

شکل (۱-۳) تابع پردازش معادلات

```
Tabnine | Edit | Test | Explain | Document
def generate_initial_population(population_size=INIT_POPULATION_SIZE, value_range=(LOWER_BOUND_INIT_POP, UPPER_BOUND_INIT_P
    min_val: Any, max_val: Any = value_range
    population: NDArray[float64] = np.random.uniform(low=min_val, high=max_val, size=(population_size, 4))
    for i in range(stop/len(obj/population)):
        while abs(x/population[i][0]) < X_AVOID_ZERO_THRESHOLD:
            population[i][0] = np.random.uniform(min_val, max_val)
    return population
```

شکل (۲-۳) ویرایش تابع ایجاد جمعیت اولیه

```
Tabnine | Edit | Test | Explain | Document
def calculate_fitness(population, equations) -> NDArray[Any]:
    return np.array(object=[
        np.sqrt(sum(iterable/(eq(*chrom))**2 for eq in equations)) for chrom in population
    ])
```

شکل (۳-۳) تابع بدست آوردن خطا

```python
def tournament_selection(population, fitness_values, tournament_size=TOURNAMENT_DEFAULT_SIZE) -> NDArray:
    selected: list = []
    for _ in range(stop/len(obj/population)):
        indices: NDArray[long] = np.random.choice(a=len(obj/population), size=tournament_size, replace=False)
        winner: Any = population[indices[np.argmin(a=fitness_values[indices])]]
        selected.append(object/winner)
    return np.array(object=selected)
```

شکل (۴-۳) تابع انتخاب والدین

```python
# Single-point crossover
Tabnine | Edit | Test | Explain | Document
def single_point_crossover(parents) -> NDArray:
    offspring: list = []
    for i in range(start/0, stop/len(obj/parents), step=2):
        p1: Any = parents[i]
        p2: Any = parents[i+1] if i+1 < len(obj/parents) else parents[i]
        point: int = np.random.randint(low=1, high=len(obj/p1))
        offspring.append(object/np.concatenate(arrays/(p1[:point], p2[point:])))
        offspring.append(object/np.concatenate(arrays/(p2[:point], p1[point:])))
    return np.array(object=offspring)
```

شکل (۵-۳) ترکیب دو نقطه و ایجاد فرزند جدید

همچنین الگوریتم ژنتیک را هم دوباره استفاده می کنیم دقیقا از همان کدی که برای بخش پیش نوشتیم تا ببینیم روند حل آن به چه روی است.

```
  File "c:\Home\University\AI\WorkShops\Gen\400411099\part3.py", line 31, in parse_equation
    raise ValueError(f"Invalid equation format: {e}")
ValueError: Invalid equation format: name 't' is not defined
```

شکل (۶-۳) خطای اجرا

همانطور که در خطای داده شده مشخص است ما متغیر t را به تابع قبل اضافه نکرده بودیم بنابراین ویرایش می کنیم.

```python
x: Any, y: Any, z: Any = symbols(names="x y z t")

# Parse and return a callable for the equation
Tabnine | Edit | Test | Explain | Document
def parse_equation(equation_str) -> Callable[..., float]:
    if "=" not in equation_str:
        raise ValueError("Equation must contain an '=' sign.")

    left_expr: Any, right_expr: Any = equation_str.split("=")

    print(f"Parsing equation: {left_expr.strip()} = {right_expr.strip()}")

    try:
        left_expr: Any = eval(source/left_expr, globals={"x": x, "y": y, "z": z})
        right_expr: Any = eval(source/right_expr, globals={"x": x, "y": y, "z": z})
    except Exception as e:
        raise ValueError(f"Invalid equation format: {e}")
```

شکل (۳-۷) ویرایش متغیر های معادلات

```python
def parse_equation(equation_str) -> Callable[..., float]:
    if "=" not in equation_str:
        raise ValueError("Equation must contain an '=' sign.")

    left_expr: Any, right_expr: Any = equation_str.split("=")

    print(f"Parsing equation: {left_expr.strip()} = {right_expr.strip()}")

    try:
        left_expr: Any = eval(source/left_expr, globals={"x": x, "y": y, "z": z, "t":t})
        right_expr: Any = eval(source/right_expr, globals={"x": x, "y": y, "z": z, "t":t})
    except Exception as e:
        raise ValueError(f"Invalid equation format: {e}")

    def eq_func(x_val, y_val, z_val, t_val) -> float:
        try:
            if abs(x/x_val) < X_AVOID_ZERO_THRESHOLD:
                return float(x="inf")
            result: Any = left_expr.subs({x: x_val, y: y_val, z: z_val, t:t_val}) - right_expr
            return float(x=result)
        except Exception:
            return float(x="inf")
    return eq_func
```

شکل (۳-۸) ویرایش دوباره تابع دریافت معادلات

```
    print(f"Parsing equation: {left_expr.strip()} = {right_expr.strip()}")

    try:
        left_expr: Any = eval(source/left_expr, globals={"x": x, "y": y, "z": z, "t":t})
        right_expr: Any = eval(source/right_expr, globals={"x": x, "y": y, "z": z, "t":t})
    except Exception as e:
        raise ValueError(f"Invalid equation format: {e}")
                                        Double-click to insert
    def eq_func(x_val, y_val, z_val, t_val) -> float:
        try:
            if abs(x/x_val) < X_AVOID_ZERO_THRESHOLD:
                return float(x="inf")
            result: Any = left_expr.subs({x: x_val, y: y_val, z: z_val, t:t_val}) - right_expr
            return float(x=result)
        except Exception:
            return float(x="inf")
    return eq_func


# Generate population (avoid x ≈ 0)
    Tabnine | Edit | Test | Explain | Document
def generate_initial_population(population_size=INIT_POPULATION_SIZE, value_range=(LOWER_BOUND_INIT_POP, UPPER_BOUND
    min_val: Any, max_val: Any = value_range

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

equation4:1/2 * x + 2*y + 7/4 * z + 4/3 * t = -9
Parsing equation: 1/15 * x - 2*y -15*z -4/5 *t = 3
Parsing equation: -5/2 * x - 9/4 * y + 12*z - t = 17
Parsing equation: -13*x + 3/10 * y - 6*z - 2/5 * t = 17
Generation 0: Fitness=183.780149 | Solution=[ -3.96223297 -17.34299319   9.17711246 -13.89424486]
Generation 10: Fitness=0.651323 | Solution=[ -2.90509873 -18.08869329   0.33106627  35.00232222]
Generation 20: Fitness=0.040321 | Solution=[ -2.91811556 -17.68488092   0.33106627  34.04091928]
Generation 30: Fitness=0.028357 | Solution=[ -2.91456078 -17.68488092   0.33060673  34.04701619]
```

شکل (۹-۳) سرانجام اجرای کد

همانگونه که مشخص است در اینجا یک ایراد وارد است و آن هم این است که معادله ی سوم وارد نشده است و آن را در کد درست نکرده ایم.

$$\begin{cases} \frac{1}{15}x - 2y - 15z - \frac{4}{5}t = 3 \\ -\frac{5}{2}x - \frac{9}{4}y + 12z - t = 17 \\ -13x + \frac{3}{10}y - 6z - \frac{2}{5}t = 17 \\ \frac{1}{2}x + 2y + \frac{7}{4}z + \frac{4}{3}t = -9 \end{cases} \Rightarrow \begin{cases} x = -\frac{3}{2} \\ y = -\frac{7}{2} \\ z = \frac{1}{3} \\ t = -\frac{11}{8} \end{cases}$$

شکل (۱۰-۳) نمونه پرسش درون جزوه

حالا با این نمونه کد خود را راستی آزمایی می کنیم.

```
Generation 450: Fitness=0.071310 | Solution=[-1.50631039 -3.58116331  0.33279771 -1.20937392]
Generation 460: Fitness=0.069184 | Solution=[-1.50631039 -3.57729725  0.33279771 -1.22949652]
Generation 470: Fitness=0.063313 | Solution=[-1.50631039 -3.57140963  0.33279771 -1.22949652]
Generation 480: Fitness=0.063186 | Solution=[-1.50631039 -3.57140963  0.33279771 -1.23321487]
Generation 490: Fitness=0.063186 | Solution=[-1.50631039 -3.57140963  0.33279771 -1.23321487]

Best Solution Found: [-1.50631039 -3.56552839  0.33279771 -1.82891225] | Fitness: 0.06008997
```

شکل (۱۱-۳) سرانجام آزمایش با نمونه ی سوم درون جزوه

همان گونه که دیده می شود با تعداد نسل های بسیار بیشتری نسبت به بخش قبلی به پاسخ رسیدیم.

```
Generation 0: Fitness=127.004842 | Solution=[ 6.16553091 16.16945556 -2.15658104 -2.21036724]
Generation 10: Fitness=19.530861 | Solution=[ -3.38730352 -25.48260198   0.30490369  44.36894754]
Generation 20: Fitness=17.734695 | Solution=[ -3.25552233 -23.58713551   0.30490369  40.0093181 ]
Generation 30: Fitness=15.804868 | Solution=[ -3.00343208 -21.396381     0.30490369  35.63696285]
Generation 40: Fitness=13.963144 | Solution=[ -2.86966231 -19.34966321   0.30490369  31.22518035]
Generation 50: Fitness=12.185051 | Solution=[ -2.71653881 -17.26514116   0.30490369  27.05279947]
Generation 60: Fitness=10.541389 | Solution=[ -2.61584543 -15.18854255   0.30490369  22.92924996]
Generation 70: Fitness=8.915023 | Solution=[ -2.39846992 -13.68131644   0.30490369  18.96829001]
Generation 80: Fitness=7.024927 | Solution=[ -2.178662   -11.51478753   0.30490369  14.99521919]
Generation 90: Fitness=5.675548 | Solution=[-1.96316009 -9.96722624  0.30490369 11.68997507]
Generation 100: Fitness=4.443619 | Solution=[-1.96930911 -8.42904099  0.30490369  8.34435485]
Generation 110: Fitness=3.463299 | Solution=[-1.78012915 -7.4584595   0.30490369  6.3945639 ]
Generation 120: Fitness=2.707981 | Solution=[-1.78012915 -5.98230302  0.30490369  4.3280128 ]
Generation 130: Fitness=2.075754 | Solution=[-1.69943568 -5.86872632  0.33210244  3.46091749]
Generation 140: Fitness=1.791165 | Solution=[-1.69399958 -5.29630913  0.33210244  2.5992575 ]
Generation 150: Fitness=0.972814 | Solution=[-1.59689126 -4.61441552  0.33210244  0.84216495]
Generation 160: Fitness=0.750487 | Solution=[-1.5741883  -4.31151642  0.33210244  0.36968075]
Generation 170: Fitness=0.706840 | Solution=[-1.56917638 -4.31120867  0.33210244  0.25358975]
Generation 180: Fitness=0.637297 | Solution=[-1.56283654 -4.22204948  0.33210244  0.11303156]
Generation 190: Fitness=0.305283 | Solution=[-1.52535303 -3.85114949  0.33210244 -0.69067063]
Generation 200: Fitness=0.230173 | Solution=[-1.52535303 -3.74908846  0.33210244 -0.88105148]
Generation 210: Fitness=0.216269 | Solution=[-1.51886519 -3.74908846  0.33210244 -0.88105148]
Generation 220: Fitness=0.186251 | Solution=[-1.51886519 -3.70867263  0.33210244 -0.94337448]
Generation 230: Fitness=0.177784 | Solution=[-1.51886519 -3.6960978   0.33210244 -0.96940369]
Generation 240: Fitness=0.172078 | Solution=[-1.51682874 -3.6960978   0.33210244 -0.97938776]
Generation 250: Fitness=0.169025 | Solution=[-1.51682874 -3.68805657  0.33210244 -1.01415944]
Generation 260: Fitness=0.166817 | Solution=[-1.51598519 -3.68805657  0.33210244 -0.98607502]
Generation 270: Fitness=0.163448 | Solution=[-1.51598519 -3.6862653   0.33210244 -1.00507325]
Generation 280: Fitness=0.155574 | Solution=[-1.51598519 -3.67575464  0.33279771 -1.01539342]
Generation 290: Fitness=0.150896 | Solution=[-1.51598519 -3.66878945  0.33279771 -1.02942929]
Generation 300: Fitness=0.134583 | Solution=[-1.51316211 -3.647512    0.33279771 -1.09923119]
Generation 310: Fitness=0.121490 | Solution=[-1.51316211 -3.63394698  0.33279771 -1.09923119]
Generation 320: Fitness=0.119422 | Solution=[-1.50841471 -3.63394698  0.33279771 -1.10835896]
Generation 330: Fitness=0.113487 | Solution=[-1.50841471 -3.62511255  0.33279771 -1.11559108]
Generation 340: Fitness=0.095443 | Solution=[-1.50841471 -3.6084463   0.33279771 -1.15239723]
Generation 350: Fitness=0.094143 | Solution=[-1.50841471 -3.6084463   0.33279771 -1.16243583]
Generation 360: Fitness=0.090322 | Solution=[-1.50841471 -3.60146772  0.33279771 -1.16345611]
Generation 370: Fitness=0.085354 | Solution=[-1.50777777 -3.59753991  0.33279771 -1.176509  ]
Generation 380: Fitness=0.081243 | Solution=[-1.50777777 -3.59312712  0.33279771 -1.18984082]
Generation 390: Fitness=0.079363 | Solution=[-1.50777777 -3.58861645  0.33279771 -1.18984082]
Generation 400: Fitness=0.078794 | Solution=[-1.50777777 -3.58861645  0.33279771 -1.19162429]
Generation 410: Fitness=0.077344 | Solution=[-1.50631039 -3.58861645  0.33279771 -1.20232895]
Generation 420: Fitness=0.074585 | Solution=[-1.50631039 -3.58030492  0.33279771 -1.20232895]
Generation 430: Fitness=0.073424 | Solution=[-1.50631039 -3.58435532  0.33279771 -1.20937392]
Generation 440: Fitness=0.073424 | Solution=[-1.50631039 -3.58435532  0.33279771 -1.20937392]
Generation 450: Fitness=0.071310 | Solution=[-1.50631039 -3.58116331  0.33279771 -1.20937392]
```

شکل (۱۲-۳) روند رسیدن به پاسخ نهایی

همانگونه که پیش بینی هم می شد این روند برای چهار متغیر و چهار مجهول کند تر بود اما در نهایت پاسخ به پاسخ درست نزدیک بود و موشکافی به نسبت خوبی داشت.

فصل ۴:

مراجع

مراجع

[۱] جزوه درس

[۲] جزوه صورت سوال

[۳] دستیار هوش مصنوعی Chat GPT