Running AGP2 iteration 1/20

Author Annotation: Elitism

```
def selection(population, fitnesses):
    """
    Selects parents for crossover based on their fitness.

    Parameters:
    population (list): A list of individuals.
    fitnesses (list): A list of fitness values corresponding to the individuals in the population.

    Returns:
    individual: An individual from the population selected as a parent.
    """

    # Combine the population and fitnesses into a numpy array for efficient indexing
    individuals = np.array(population)

    # Select the index of the best individual based on its fitness (minimum error)
    idx_best = np.argmin(fitnesses, axis=0)[0]

    # Return the best individual as a parent
    return individuals[idx_best]
```

Running AGP2 iteration 2/20

Author Annotation: Elitism

```
def selection(population, fitnesses):
    """
    Selects an individual from a population based on its fitness.

    Parameters:
    - population (list): A list of individuals in the current population.
    - fitnesses (list): A list of fitness values corresponding to each individual in the population.

    Returns:
    - selected_individual (individual): An individual chosen for reproduction based on its fitness value.
    """
    # Combine the population and their fitness values into a list of tuples
    population_fitness = list(zip(population, fitnesses))

    # Sort the population by fitness in ascending order
    sorted_population = sorted(population_fitness, key=lambda x: np.mean(x[1]))

    # Select the individual with the best fitness (i.e., minimum error)
    selected_individual = sorted_population[0][0]
```

```
    return selected_individual
```

Running AGP2 iteration 3/20

Author Annotation: Elitism

```
def selection(population, fitnesses):
    # Calculate the total errors for each individual in the population
    total_errors = [sum(fit) for fit in fitnesses]

    # Choose the index of the fittest individual with the lowest total error
    best_index = np.argmin(total_errors)

    return population[best_index]
```

Running AGP2 iteration 4/20

Author Annotation: Elitism

```
def selection(population, fitnesses):
    # Calculate the total errors of each individual in the population
    total_errors = [sum(f) for f in fitnesses]

    # Select the indices of individuals with the lowest total error
    min_error_idx = np.argmin(total_errors)
```

```
    # Select one individual from those with the lowest total error based on their original order of appearance in the population
    # (to avoid favoring some over others due to random initialization or early convergence)
    return np.random.choice([individual for idx, individual in enumerate(population) if idx == min_error_idx])
```

Running AGP2 iteration 5/20

Author Annotation: Elitism

```
import numpy as np
def selection(population, fitnesses):
    # Convert lists to numpy arrays for efficient computation
    population = np.array(population)
    fitnesses = np.array(fitnesses)

    # Select the individual with the best fitness (lowest error)
    selected_individual_index = np.argmin(fitnesses, axis=0)[0]

    return population[selected_individual_index]
```

Running AGP2 iteration 6/20

Author Annotation: Elitism

```
def selection(population, fitnesses):
    """
```

```
    Select parent from population based on fitness.

    Args:
        population (list): List of individual in the population.
        fitnesses (list): List of fitness values corresponding to individuals in the population.

    Returns:
        The selected parent as an individual from the population.
    """

    # Combine population and fitnesses into a list of tuples for easier manipulation
    pop_with_fitness = list(zip(population, fitnesses))

    # Sort the population based on fitness (lower is better)
    sorted_pop_with_fitness = sorted(pop_with_fitness, key=lambda x: np.mean(x[1]))

    # Select first 10 individuals with lowest fitness as parents
    parents = [x[0] for x in sorted_pop_with_fitness[:10]]

    # Select one parent randomly from the selected parents
    parent = np.random.choice(parents)

    return parent
```

Running AGP2 iteration 7/20

Author Annotation: Elitism

```
def selection(population, fitnesses):
    """
    Selects the fittest individual from a population based on their fitness.
    Args:
        population (list): A list of individuals in the GP system.
        fitnesses (list): A list of fitness values corresponding to each individual in the population.
    Returns:
        Individual: The fittest individual in the population.
    """

    # Combine the population and fitnesses into a single data structure for easier access
    combined = [(population[i], fitnesses[i]) for i in range(len(population))]

    # Sort the population by their fitness (lower is better)
    combined.sort(key=lambda x: np.mean(x[1]))

    # Select the fittest individual as the parent for reproduction
    return combined[0][0]
```

Running AGP2 iteration 8/20

Author Annotation: Elitism

```
import numpy as np
```

```
def selection(population, fitnesses):
    """
    Parent selection for Genetic Programming.

    Args:
        population (list): A list of individuals.
        fitnesses (list): A list of fitness values corresponding to each individual in the population.

    Returns:
        Individual: The selected parent.
    """

    # Combine individuals and their fitnesses into a list of tuples
    individuals = list(zip(population, fitnesses))

    # Sort individuals based on their fitness values
    sorted_individuals = sorted(individuals, key=lambda x: np.sum(x[1]))

    # Select the individual with the lowest fitness value (i.e., highest probability)
    selected_parent = sorted_individuals[0][0]

    return selected_parent
```

Running AGP2 iteration 9/20

Author Annotation: Elitism

```
def selection(population, fitnesses):
    """
    Parent Selection Function for Genetic Programming.

    Parameters:
    population (list): A list of individuals in the current generation.
    fitnesses (list): A list of fitness values corresponding to each individual in the population.

    Returns:
    individual: An individual from the population selected for reproduction.
    """

    # Combine population and fitnesses into a single data structure
    combined = np.column_stack((population, fitnesses))

    # Select the index of the best individual based on minimum total error
    best_index = np.argmin(combined[:, 1])

    # Return the best individual
    return population[best_index]
```

Running AGP2 iteration 10/20

Author Annotation: Fitness-proportionate

```
def selection(population, fitnesses):
    # Calculate the cumulative probability distribution of the population fitnesses.
    probabilities = np.cumsum(fitnesses)

    # Generate a random number between 0 and 1 to simulate a roulette wheel spin.
    select_prob = np.random.uniform(0.0, 1.0)

    # Perform binary search on the cumulative probability distribution to find the individual that would be selected by this spin of the roulette wheel.
    lo = 0
    hi = len(population) - 1
    while lo < hi:
        mid = (lo + hi) // 2
        if probabilities[mid] <= select_prob:
            lo = mid + 1
        else:
            hi = mid

    # Return the individual that was selected by this spin of the roulette wheel.
    return population[lo-1]
```

Running AGP2 iteration 11/20

Author Annotation: Elitism

```
import numpy as np
```

```
def selection(population, fitnesses):
    # Combine population and fitnesses into a list of tuples
    combined = list(zip(population, fitnesses))

    # Sort the combined list based on fitness
    sorted_combined = sorted(combined, key=lambda x: sum(x[1]), reverse=False)

    # Select the best individual (with lowest fitness) as the parent
    best_individual = sorted_combined[0][0]

    return best_individual
```

Running AGP2 iteration 12/20

Author Annotation: Fitness-proportionate

```
def selection(population, fitnesses):
    """
    Selects parents from a population based on their fitness.

    Args:
        population: A list of individuals, each represented as a tree-based structure.
        fitnesses: A list of fitness values corresponding to the individuals in the population.

    Returns:
```

```
    An individual selected for breeding.
    """

    # Calculate the total fitness for each individual (assuming we want to maximize it)
    total_fitness = [sum(fit) for fit in fitnesses]

    # Calculate the cumulative sum of the total fitness
    cum_sum = np.cumsum(total_fitness)

    # Generate a random number between 0 and the sum of all total fitness values
    rand_num = np.random.uniform(0, cum_sum[-1])

    # Find the index of the first cumulative sum that is greater than or equal to the random number
    idx = np.where(cum_sum >= rand_num)[0][0]

    return population[idx]
```

Running AGP2 iteration 13/20

Author Annotation: Elitism

```
def selection(population, fitnesses):
    """
    Selects an individual for reproduction based on its fitness.

    Parameters:
```

```
    population (list): List of individuals.
    fitnesses (list): Corresponding list of fitness values for each individual.

    Returns:
    individual: One of the elements in the population.
    """

    # Calculate the total number of errors across all test cases
    num_errors = [sum(f) for f in fitnesses]

    # Find the index of the individual with the lowest error (i.e., best fitness)
    min_error_idx = np.argmin(num_errors)

    # Select this individual for reproduction
    return population[min_error_idx]
```

Running AGP2 iteration 14/20

Author Annotation: Elitism

```
def selection(population, fitnesses):
    # Calculate the error for each individual in the population
    errors = [sum(f) for f in fitnesses]

    # Use tournament selection to select an individual from the population
    best_individual_index = np.argmin(errors)
```

```
    return population[best_individual_index]
```

Running AGP2 iteration 15/20

Author Annotation: Elitism

```
def selection(population, fitnesses):
    """
    Select parents for crossover and mutation based on their fitness.

    Parameters:
    population (list): List of individuals.
    fitnesses (list): Corresponding list of fitness values.

    Returns:
    individual: Chosen parent from the population.
    """

    # Combine individuals and their fitnesses into a list of tuples
    ind_fit = list(zip(population, fitnesses))

    # Sort the list in ascending order based on fitness
    ind_fit.sort(key=lambda x: np.mean(x[1]))

    # Select the best individual (with the minimum mean error)
    chosen_parent = ind_fit[0][0]
```

```
    return chosen_parent
```

Running AGP2 iteration 16/20

Author Annotation: Elitism

```
def selection(population, fitnesses):
    """
    Selects a parent for crossover based on the tournament selection method.

    Parameters:
    population (list): A list of individuals in the current generation.
    fitnesses (list): A list of lists where each sublist contains the errors
                on test cases for an individual in the population.

    Returns:
    individual: The selected parent from the population.
    """
    # Initialize variables to keep track of the best individual and its fitness
    best_of_run_f = float('inf')
    best_individual = None

    # Iterate over each individual in the population
    for i, individual in enumerate(population):
        # Get the fitness of the current individual
        individual_fitness = fitnesses[i]

        # Check if this is a better individual than the current best
```

```
        if np.mean(individual_fitness) < np.mean(best_of_run_f):
            best_individual = individual
            best_of_run_f = np.mean(individual_fitness)

    return best_individual
```

Running AGP2 iteration 17/20

Author Annotation: Elitism

```
def selection(population, fitnesses):
    # Combine population and fitness into a list of tuples for easier manipulation
    individuals = list(zip(population, fitnesses))

    # Sort the individuals based on their fitness (lower is better)
    individuals.sort(key=lambda x: np.sum(x[1]))

    # Select the best individual
    selected_individual = individuals[0]

    return selected_individual[0]  # Return the individual (not its fitness)
```

Running AGP2 iteration 18/20

Author Annotation: Elitism

```
def selection(population, fitnesses):
```

```
    """
    Selects parents for the next generation based on their fitness.

    Parameters:
    population (list): A list of individuals in the current population.
    fitnesses (list): A list of fitness values corresponding to each individual in the population.

    Returns:
    Individual: The selected parent, i.e., one of the elements in the population.
    """
    # Combine population and fitness into a numpy array for easier manipulation
    pop_fit = np.column_stack((population, fitnesses))

    # Get indices of individuals with the best fitness (smallest error)
    best_of_run_f = min(fitnesses)
    best_indices = np.where(pop_fit[:, 1:] == best_of_run_f)[0]

    # Select one parent at random from the ones with the best fitness
    parent_idx = np.random.choice(best_indices, size=1)[0]

    return population[parent_idx]
```

Running AGP2 iteration 19/20

Author Annotation: Elitism

```
def selection(population, fitnesses):
    """
```

```
    Selects a parent from the population based on their fitness.

    Args:
        population (list): A list of individuals in the current generation.
        fitnesses (list): A list of corresponding fitness values for each individual in the population.

    Returns:
        The selected individual with the best fitness.
    """

    # Combine the population and fitness lists into a list of tuples
    population_fitness = list(zip(population, fitnesses))

    # Sort the combined list based on the fitness value (lower is better)
    sorted_population_fitness = sorted(population_fitness, key=lambda x: np.mean(x[1]))

    # Select the individual with the best fitness (i.e., the first element in the sorted list)
    selected_individual, _ = sorted_population_fitness[0]

    return selected_individual
```

Running AGP2 iteration 20/20

Author Annotation: Elitism

```
def selection(population, fitnesses):
    # Combine population and fitnesses into a list of tuples for easier manipulation
```

```
individuals = list(zip(population, fitnesses))

# Sort the population by their fitness values in ascending order
individuals.sort(key=lambda x: np.sum(x[1]))

# Select the individual with the best fitness
return individuals[0][0]
```