student

April 19, 2025

1 CA2 - Genetic & Game

Ali Borzoozadeh - 810102410

1.0.1 Matplotlib – Data Visualization in Python

matplotlib is a python library that is mainly used for data visualization. This library allows you to plot different type of figures including scatters and histograms. In the first part of this project you are supposed to implement a genetic algorithm. To visualize plots that are required in the project description use plotting as much as you can because it gives a great insight on what is happening during each run. It also helps you to compare your results whenever you want to understand effect of different parameters during different runs. For more information, check this notebook and visit the website.

[1]:
```python
import matplotlib.pyplot as plt
```

2 Genetic Algorithm

[2]:
```python
import random
import itertools
import numpy as np
```

[3]:
```python
# algorithm parameters
numCoeffs = 41
populationSize = 100
generations = 100
mutationRate = 0.15
functionRange = (-np.pi, np.pi)
sampleCount = 100
```

[4]:
```python
# These functions are given as samples to use in the algorithm
def getTargetFunction(functionName="sin_cos"):
    def sinCosFunction(t):
        """Target function: sin(2 t) + 0.5*cos(4 t)."""
        return np.sin(2 * np.pi * t) + 0.5 * np.cos(4 * np.pi * t)

    def linearFunction(t):
        """Simple linear function: y = 2t + 1."""
        return 2 * t + 1

    def quadraticFunction(t):
        """Quadratic function: y = 4t^2 - 4t + 2."""
        return 4 * (t**2) - 4 * t + 2

    def cubicFunction(t):
        """Cubic function: y = 8t^3 - 12t^2 + 6t."""
        return 8 * (t**3) - 12 * (t**2) + 6 * t

    def gaussianFunction(t):
        """Gaussian function centered at t=0.5."""
        mu = 0.5
        sigma = 0.1  # Adjust sigma to control the width of the peak
        return np.exp(-((t - mu) ** 2) / (2 * sigma**2))

    def squareWaveFunction(t):
        """Approximation of a square wave. Smoothed for better Fourier␣
↪approximation."""
        return 0.5 * (np.sign(np.sin(2 * np.pi * t)) + 1)

    def sawtoothFunction(t):
        """Sawtooth wave, normalized to [0, 1]."""
        return (t * 5) % 1

    def complexFourierFunction(t):
        return (
            np.sin(2 * np.pi * t)
            + 0.3 * np.cos(4 * np.pi * t)
            + 0.2 * np.sin(6 * np.pi * t)
            + 0.1 * np.cos(8 * np.pi * t)
        )

    def polynomialFunction(t):
        return 10 * (t**5) - 20 * (t**4) + 15 * (t**3) - 4 * (t**2) + t + 0.5

    functionOptions = {
        "sin_cos": sinCosFunction,
        "linear": linearFunction,
        "quadratic": quadraticFunction,
        "cubic": cubicFunction,
        "gaussian": gaussianFunction,
        "square_wave": squareWaveFunction,
        "sawtooth": sawtoothFunction,
        "complex_fourier": complexFourierFunction,
        "polynomial": polynomialFunction,
    }

    selectedFunction = functionOptions.get(functionName.lower())
    if selectedFunction:
        return selectedFunction
```

[5]:
```
# generate samples
tSamples = np.linspace(functionRange[0], functionRange[1], sampleCount)
fSamples = getTargetFunction()(tSamples)
```

Important Note:

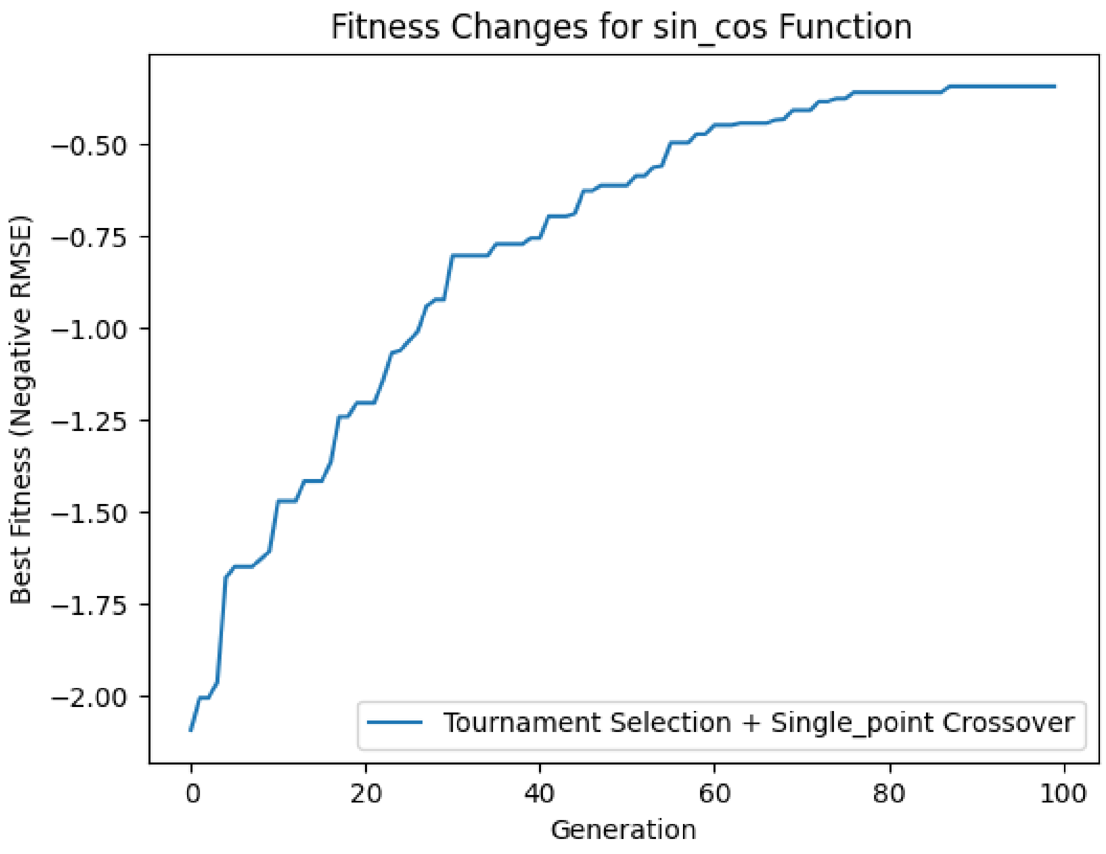
Using **NumPy arrays** allows you to perform operations on vectors **more efficiently** and **faster**.

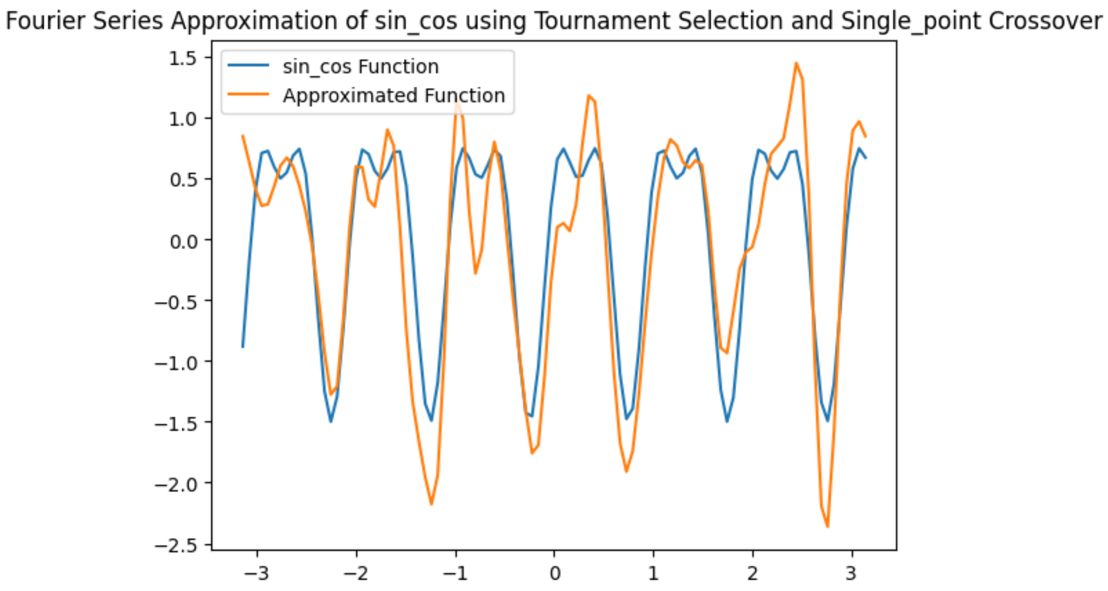
Avoid using `for` loops whenever possible, as vectorized operations in NumPy are **optimized for performance** and significantly reduce execution time.

[6]:
```
def fourier_series(t, coeffs):
    a0 = coeffs[0]
    series = a0 / 2
    for n in range(1, 21):
        series += coeffs[n] * np.cos(n * t) + coeffs[20 + n] * np.sin(n * t)
    return series

# Using RMSE as the fitness function
def fitness(coeffs):
    predicted = fourier_series(tSamples, coeffs)
    rmse = np.sqrt(np.mean((fSamples - predicted) ** 2))
    return -rmse  # Negative because we want to maximize fitness

def initialize_population():
    return [np.random.uniform(-1, 1, numCoeffs) for _ in range(populationSize)]

def roulette_wheel_selection(population, fitnesses):
    min_fitness = min(fitnesses)
    offset = abs(min_fitness) + 1e-6
    adjusted_fitnesses = [f + offset for f in fitnesses]

    total_fitness = sum(adjusted_fitnesses)
    probabilities = [f / total_fitness for f in adjusted_fitnesses]

    selected_index = random.choices(range(len(population)),␣
 ↪weights=probabilities, k=1)[0]
    return population[selected_index]

def rank_selection(population, fitnesses):
    sorted_indices = np.argsort(fitnesses)
    ranks = np.arange(1, len(fitnesses) + 1)
    probabilities = ranks / ranks.sum()
    sorted_population = [population[i] for i in sorted_indices]
    return random.choices(sorted_population, weights=probabilities, k=1)[0]

def tournament_selection(population, fitnesses):
    idx1, idx2 = random.sample(range(len(population)), 2)
    return population[idx1] if fitnesses[idx1] > fitnesses[idx2] else␣
↪population[idx2]

def single_point_crossover(parent1, parent2):
    point = random.randint(1, len(parent1) - 1)
    child1 = np.concatenate((parent1[:point], parent2[point:]))
    child2 = np.concatenate((parent2[:point], parent1[point:]))

    return child1, child2

def two_point_crossover(parent1, parent2):
    point1, point2 = sorted(random.sample(range(1, len(parent1)), 2))
    child1 = np.concatenate((parent1[:point1], parent2[point1:point2],␣
↪parent1[point2:]))
    child2 = np.concatenate((parent2[:point1], parent1[point1:point2],␣
↪parent2[point2:]))

    return child1, child2

def uniform_crossover(parent1, parent2):
    mask = np.random.randint(0, 2, size=numCoeffs)
    child1 = np.where(mask == 1, parent1, parent2)
    child2 = np.where(mask == 1, parent2, parent1)

    return child1, child2

def mutate(individual):
    for i in range(numCoeffs):
        if random.random() < mutationRate:
            individual[i] += np.random.normal(0, 0.1)
    return individual

def genetic_algorithm(selection_method="roulette", crossover_method="uniform"):
    selection_function = selection_methods.get(selection_method)
    crossover_function = crossover_methods.get(crossover_method)
    population = initialize_population()
    best_solution = None
    best_fitness = float('-inf')
    fitness_history = []

    for generation in range(generations):
        fitnesses = [fitness(ind) for ind in population]
        new_population = []

        for _ in range(populationSize // 2):
            parent1 = selection_function(population, fitnesses)
            parent2 = selection_function(population, fitnesses)
            child1, child2 = crossover_function(parent1, parent2)
            new_population.append(mutate(child1))
            new_population.append(mutate(child2))

        population = new_population

        max_fitness_idx = np.argmax(fitnesses)
        if fitnesses[max_fitness_idx] > best_fitness:
            best_fitness = fitnesses[max_fitness_idx]
            best_solution = population[max_fitness_idx]

        print(f"Generation {generation + 1}: Best Fitness = {best_fitness:.6f}")
        fitness_history.append(best_fitness)

    return best_solution, fitness_history

functions = ["linear", "quadratic", "cubic", "gaussian", "square_wave",␣
 ↪"sawtooth", "complex_fourier", "polynomial"]

selection_methods = {
    "tournament": tournament_selection,
    "roulette": roulette_wheel_selection,
    "rank": rank_selection
}

crossover_methods = {
    "single_point": single_point_crossover,
    "two_point": two_point_crossover,
    "uniform": uniform_crossover
}

print(f"\nRunning Genetic Algorithm for sin_cos Function:")
fSamples = getTargetFunction("sin_cos")(tSamples)
for selection in selection_methods:
    for crossover in crossover_methods:
        print(f"\nUsing {selection.capitalize()} Selection and {crossover.
 ↪capitalize()} Crossover:")
        best_coeffs, fitness_history = genetic_algorithm(selection, crossover)

        plt.plot(fitness_history, label=f"{selection.capitalize()} Selection +␣
 ↪{crossover.capitalize()} Crossover")
```

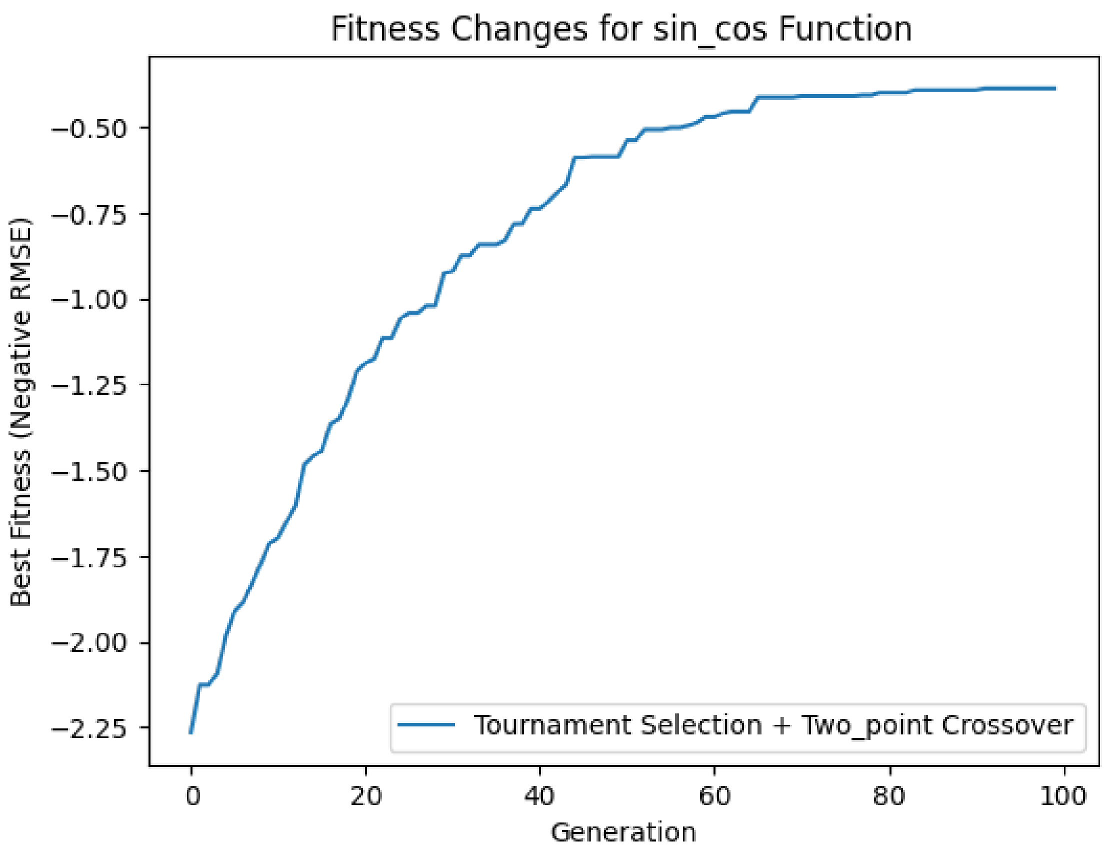
```
        plt.xlabel("Generation")
        plt.ylabel("Best Fitness (Negative RMSE)")
        plt.title("Fitness Changes for sin_cos Function")
        plt.legend()
        plt.show()

        plt.plot(tSamples, fSamples, label="sin_cos Function")
        plt.plot(tSamples, fourier_series(tSamples, best_coeffs),␣
↪label="Approximated Function")
        plt.legend()
        plt.title(f"Fourier Series Approximation of sin_cos using {selection.
↪capitalize()} Selection and {crossover.capitalize()} Crossover")
        plt.show()

for function_name in functions:
    print(f"\nRunning Genetic Algorithm for {function_name} Function:")
    fSamples = getTargetFunction(function_name)(tSamples)

    selection = "rank"
    crossover = "uniform"

    print(f"\nUsing {selection.capitalize()} Selection and {crossover.
↪capitalize()} Crossover:")
    best_coeffs, fitness_history = genetic_algorithm(selection, crossover)

    plt.plot(fitness_history, label=f"{selection.capitalize()} Selection +␣
↪{crossover.capitalize()} Crossover")
    plt.xlabel("Generation")
    plt.ylabel("Best Fitness (Negative RMSE)")
    plt.title(f"Fitness Changes for {function_name} Function")
    plt.legend()
    plt.show()

    plt.plot(tSamples, fSamples, label=f"{function_name} Function")
    plt.plot(tSamples, fourier_series(tSamples, best_coeffs),␣
↪label="Approximated Function")
    plt.legend()
    plt.title(f"Fourier Series Approximation of {function_name} using␣
↪{selection.capitalize()} Selection and {crossover.capitalize()} Crossover")
    plt.show()
```

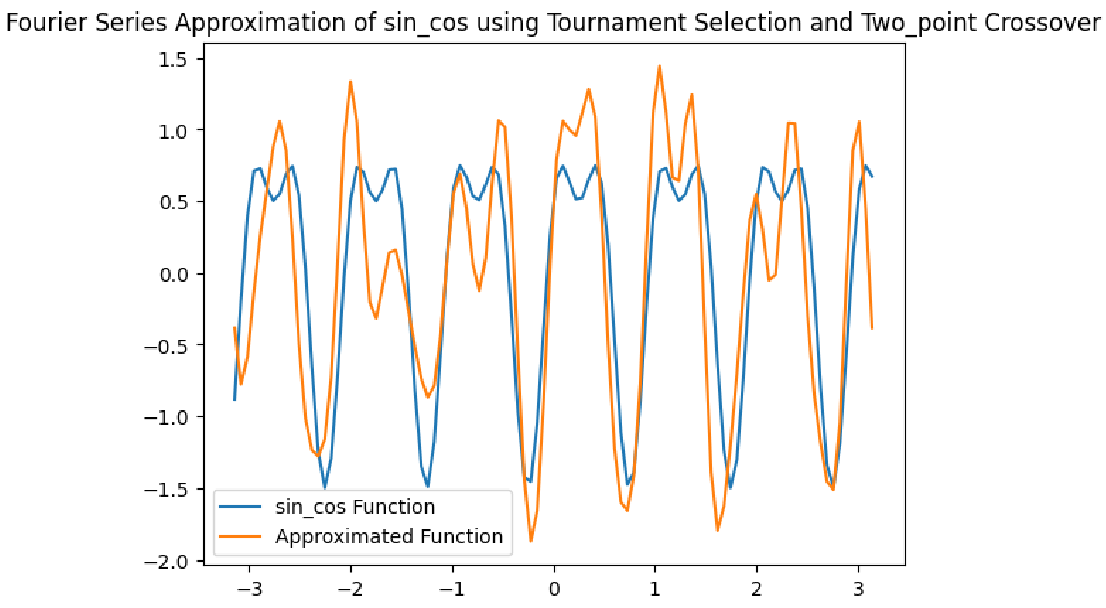
```
Running Genetic Algorithm for sin_cos Function:

Using Tournament Selection and Single_point Crossover:
Generation 1: Best Fitness = -2.094122
Generation 2: Best Fitness = -2.007098
Generation 3: Best Fitness = -2.007098
Generation 4: Best Fitness = -1.965451
Generation 5: Best Fitness = -1.679890
Generation 6: Best Fitness = -1.650440
Generation 7: Best Fitness = -1.650440
Generation 8: Best Fitness = -1.650440
Generation 9: Best Fitness = -1.629875
Generation 10: Best Fitness = -1.609008
Generation 11: Best Fitness = -1.472097
Generation 12: Best Fitness = -1.472097
Generation 13: Best Fitness = -1.472097
Generation 14: Best Fitness = -1.417410
Generation 15: Best Fitness = -1.417410
Generation 16: Best Fitness = -1.417273
Generation 17: Best Fitness = -1.367517
Generation 18: Best Fitness = -1.242471
Generation 19: Best Fitness = -1.242471
Generation 20: Best Fitness = -1.204814
Generation 21: Best Fitness = -1.204814
Generation 22: Best Fitness = -1.204814
Generation 23: Best Fitness = -1.144791
Generation 24: Best Fitness = -1.069677
Generation 25: Best Fitness = -1.061813
Generation 26: Best Fitness = -1.035824
Generation 27: Best Fitness = -1.010484
Generation 28: Best Fitness = -0.942626
Generation 29: Best Fitness = -0.923722
Generation 30: Best Fitness = -0.923722
Generation 31: Best Fitness = -0.804196
Generation 32: Best Fitness = -0.804196
Generation 33: Best Fitness = -0.804196
Generation 34: Best Fitness = -0.804196
Generation 35: Best Fitness = -0.804196
Generation 36: Best Fitness = -0.772705
Generation 37: Best Fitness = -0.772705
Generation 38: Best Fitness = -0.772705
Generation 39: Best Fitness = -0.772705
Generation 40: Best Fitness = -0.756364
Generation 41: Best Fitness = -0.756364
Generation 42: Best Fitness = -0.697334
Generation 43: Best Fitness = -0.697334
Generation 44: Best Fitness = -0.697334
Generation 45: Best Fitness = -0.690138
Generation 46: Best Fitness = -0.627918
Generation 47: Best Fitness = -0.627918
Generation 48: Best Fitness = -0.613298
Generation 49: Best Fitness = -0.613298
Generation 50: Best Fitness = -0.613298
Generation 51: Best Fitness = -0.613298
Generation 52: Best Fitness = -0.587037
Generation 53: Best Fitness = -0.587037
Generation 54: Best Fitness = -0.564425
Generation 55: Best Fitness = -0.560093
Generation 56: Best Fitness = -0.496921
Generation 57: Best Fitness = -0.496921
Generation 58: Best Fitness = -0.496921
Generation 59: Best Fitness = -0.473665
Generation 60: Best Fitness = -0.473665
Generation 61: Best Fitness = -0.449514
Generation 62: Best Fitness = -0.449514
Generation 63: Best Fitness = -0.449514
Generation 64: Best Fitness = -0.443799
Generation 65: Best Fitness = -0.443799
Generation 66: Best Fitness = -0.443799
Generation 67: Best Fitness = -0.443799
Generation 68: Best Fitness = -0.435342
Generation 69: Best Fitness = -0.432918
Generation 70: Best Fitness = -0.408565
Generation 71: Best Fitness = -0.408565
Generation 72: Best Fitness = -0.408565
Generation 73: Best Fitness = -0.384917
Generation 74: Best Fitness = -0.384917
Generation 75: Best Fitness = -0.376856
Generation 76: Best Fitness = -0.376856
Generation 77: Best Fitness = -0.360359
Generation 78: Best Fitness = -0.360359
Generation 79: Best Fitness = -0.360359
Generation 80: Best Fitness = -0.360359
Generation 81: Best Fitness = -0.360359
Generation 82: Best Fitness = -0.360359
Generation 83: Best Fitness = -0.360359
Generation 84: Best Fitness = -0.360359
Generation 85: Best Fitness = -0.360359
Generation 86: Best Fitness = -0.360359
Generation 87: Best Fitness = -0.360359
Generation 88: Best Fitness = -0.343355
Generation 89: Best Fitness = -0.343355
Generation 90: Best Fitness = -0.343355
Generation 91: Best Fitness = -0.343355
Generation 92: Best Fitness = -0.343355
Generation 93: Best Fitness = -0.343355
Generation 94: Best Fitness = -0.343355
Generation 95: Best Fitness = -0.343355
Generation 96: Best Fitness = -0.343355
Generation 97: Best Fitness = -0.343355
Generation 98: Best Fitness = -0.343355
Generation 99: Best Fitness = -0.343355
Generation 100: Best Fitness = -0.343355
```



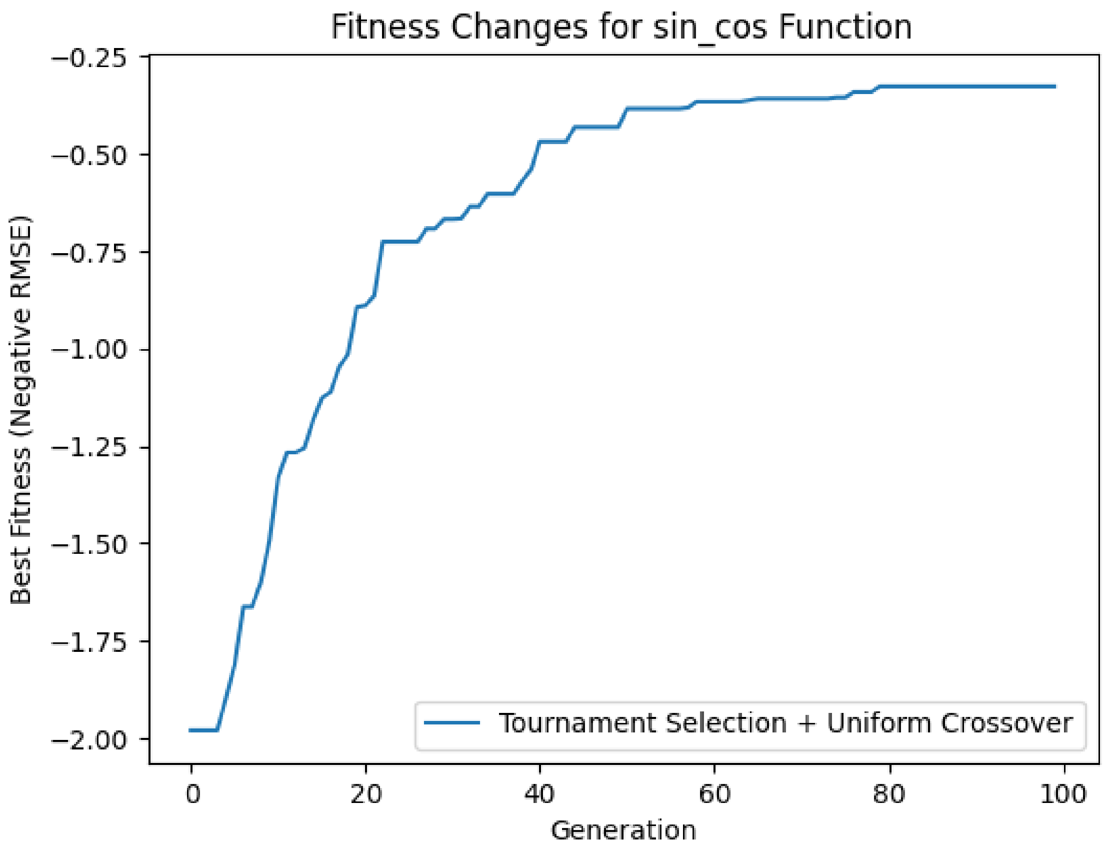


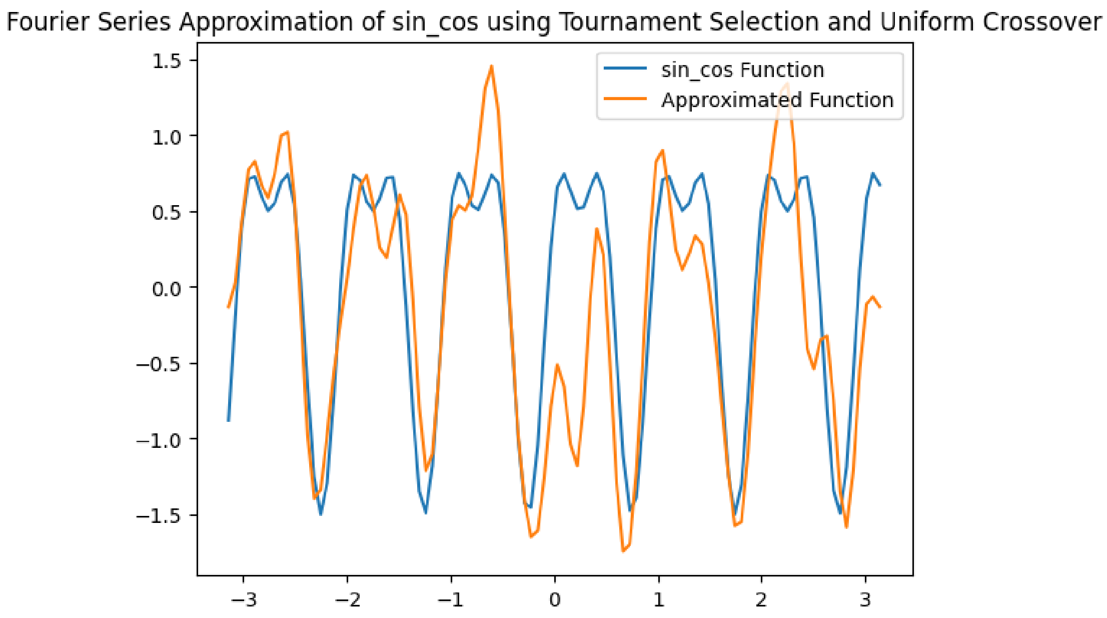
```
Using Tournament Selection and Two_point Crossover:
Generation 1: Best Fitness = -2.265083
Generation 2: Best Fitness = -2.127296
Generation 3: Best Fitness = -2.127296
Generation 4: Best Fitness = -2.093337
Generation 5: Best Fitness = -1.984130
Generation 6: Best Fitness = -1.911379
Generation 7: Best Fitness = -1.884140
Generation 8: Best Fitness = -1.830765
Generation 9: Best Fitness = -1.772898
Generation 10: Best Fitness = -1.713825
Generation 11: Best Fitness = -1.696506
Generation 12: Best Fitness = -1.648166
Generation 13: Best Fitness = -1.603483
Generation 14: Best Fitness = -1.483783
Generation 15: Best Fitness = -1.459664
Generation 16: Best Fitness = -1.442590
Generation 17: Best Fitness = -1.364542
Generation 18: Best Fitness = -1.349926
Generation 19: Best Fitness = -1.293429
Generation 20: Best Fitness = -1.212394
Generation 21: Best Fitness = -1.187826
Generation 22: Best Fitness = -1.174891
Generation 23: Best Fitness = -1.113864
Generation 24: Best Fitness = -1.113864
Generation 25: Best Fitness = -1.057776
Generation 26: Best Fitness = -1.041062
Generation 27: Best Fitness = -1.041062
Generation 28: Best Fitness = -1.020542
Generation 29: Best Fitness = -1.020424
Generation 30: Best Fitness = -0.925499
Generation 31: Best Fitness = -0.920069
Generation 32: Best Fitness = -0.874276
Generation 33: Best Fitness = -0.874276
Generation 34: Best Fitness = -0.841229
Generation 35: Best Fitness = -0.841229
Generation 36: Best Fitness = -0.841229
Generation 37: Best Fitness = -0.829326
Generation 38: Best Fitness = -0.781530
Generation 39: Best Fitness = -0.780050
Generation 40: Best Fitness = -0.737669
Generation 41: Best Fitness = -0.737669
Generation 42: Best Fitness = -0.715143
Generation 43: Best Fitness = -0.689732
Generation 44: Best Fitness = -0.667710
Generation 45: Best Fitness = -0.587872
Generation 46: Best Fitness = -0.587872
Generation 47: Best Fitness = -0.585479
Generation 48: Best Fitness = -0.585479
Generation 49: Best Fitness = -0.585479
Generation 50: Best Fitness = -0.585479
Generation 51: Best Fitness = -0.537893
Generation 52: Best Fitness = -0.537893
Generation 53: Best Fitness = -0.506058
Generation 54: Best Fitness = -0.506058
Generation 55: Best Fitness = -0.506058
Generation 56: Best Fitness = -0.500663
Generation 57: Best Fitness = -0.500663
Generation 58: Best Fitness = -0.494717
Generation 59: Best Fitness = -0.486555
Generation 60: Best Fitness = -0.469387
Generation 61: Best Fitness = -0.469387
Generation 62: Best Fitness = -0.459019
Generation 63: Best Fitness = -0.453170
Generation 64: Best Fitness = -0.453170
Generation 65: Best Fitness = -0.453170
Generation 66: Best Fitness = -0.412179
Generation 67: Best Fitness = -0.412179
Generation 68: Best Fitness = -0.412179
Generation 69: Best Fitness = -0.412179
Generation 70: Best Fitness = -0.412179
Generation 71: Best Fitness = -0.408499
Generation 72: Best Fitness = -0.408499
Generation 73: Best Fitness = -0.408499
Generation 74: Best Fitness = -0.408499
Generation 75: Best Fitness = -0.408499
Generation 76: Best Fitness = -0.408499
Generation 77: Best Fitness = -0.408499
Generation 78: Best Fitness = -0.406110
Generation 79: Best Fitness = -0.406110
Generation 80: Best Fitness = -0.398400
Generation 81: Best Fitness = -0.398400
Generation 82: Best Fitness = -0.398400
Generation 83: Best Fitness = -0.398400
Generation 84: Best Fitness = -0.390869
Generation 85: Best Fitness = -0.390869
Generation 86: Best Fitness = -0.390869
Generation 87: Best Fitness = -0.390869
Generation 88: Best Fitness = -0.390869
Generation 89: Best Fitness = -0.390869
Generation 90: Best Fitness = -0.390869
Generation 91: Best Fitness = -0.390869
Generation 92: Best Fitness = -0.385672
Generation 93: Best Fitness = -0.385672
Generation 94: Best Fitness = -0.385672
Generation 95: Best Fitness = -0.385672
Generation 96: Best Fitness = -0.385672
Generation 97: Best Fitness = -0.385672
Generation 98: Best Fitness = -0.385672
Generation 99: Best Fitness = -0.385672
Generation 100: Best Fitness = -0.385672
```



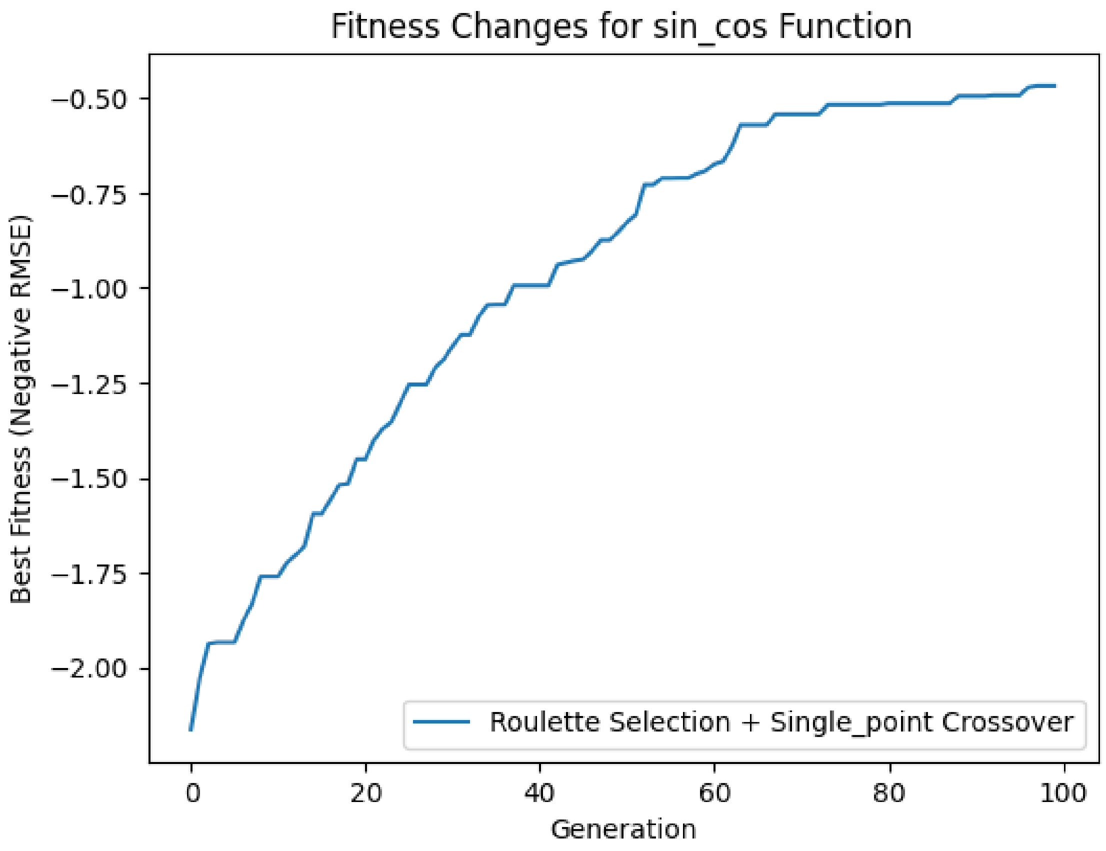


```
Using Tournament Selection and Uniform Crossover:
Generation 1: Best Fitness = -1.979523
Generation 2: Best Fitness = -1.979523
Generation 3: Best Fitness = -1.979523
Generation 4: Best Fitness = -1.979523
Generation 5: Best Fitness = -1.897817
Generation 6: Best Fitness = -1.811842
Generation 7: Best Fitness = -1.662412
Generation 8: Best Fitness = -1.662412
Generation 9: Best Fitness = -1.600806
Generation 10: Best Fitness = -1.493641
Generation 11: Best Fitness = -1.331441
Generation 12: Best Fitness = -1.266616
Generation 13: Best Fitness = -1.266616
Generation 14: Best Fitness = -1.255309
Generation 15: Best Fitness = -1.183788
Generation 16: Best Fitness = -1.127307
Generation 17: Best Fitness = -1.111734
Generation 18: Best Fitness = -1.046682
Generation 19: Best Fitness = -1.014507
Generation 20: Best Fitness = -0.894319
Generation 21: Best Fitness = -0.889448
Generation 22: Best Fitness = -0.864835
Generation 23: Best Fitness = -0.725916
Generation 24: Best Fitness = -0.725916
Generation 25: Best Fitness = -0.725916
Generation 26: Best Fitness = -0.725916
Generation 27: Best Fitness = -0.725916
Generation 28: Best Fitness = -0.692568
Generation 29: Best Fitness = -0.692568
Generation 30: Best Fitness = -0.667850
Generation 31: Best Fitness = -0.667850
Generation 32: Best Fitness = -0.666418
Generation 33: Best Fitness = -0.635556
Generation 34: Best Fitness = -0.635556
Generation 35: Best Fitness = -0.603309
Generation 36: Best Fitness = -0.603309
Generation 37: Best Fitness = -0.603309
Generation 38: Best Fitness = -0.603309
Generation 39: Best Fitness = -0.568963
Generation 40: Best Fitness = -0.540661
Generation 41: Best Fitness = -0.468869
Generation 42: Best Fitness = -0.468869
Generation 43: Best Fitness = -0.468869
Generation 44: Best Fitness = -0.468869
Generation 45: Best Fitness = -0.432020
Generation 46: Best Fitness = -0.432020
Generation 47: Best Fitness = -0.432020
Generation 48: Best Fitness = -0.432020
Generation 49: Best Fitness = -0.432020
Generation 50: Best Fitness = -0.432020
Generation 51: Best Fitness = -0.384018
Generation 52: Best Fitness = -0.384018
Generation 53: Best Fitness = -0.384018
Generation 54: Best Fitness = -0.384018
Generation 55: Best Fitness = -0.384018
Generation 56: Best Fitness = -0.384018
Generation 57: Best Fitness = -0.384018
Generation 58: Best Fitness = -0.381387
Generation 59: Best Fitness = -0.366286
Generation 60: Best Fitness = -0.366286
Generation 61: Best Fitness = -0.366286
Generation 62: Best Fitness = -0.366286
Generation 63: Best Fitness = -0.366286
Generation 64: Best Fitness = -0.366286
Generation 65: Best Fitness = -0.362542
Generation 66: Best Fitness = -0.358558
Generation 67: Best Fitness = -0.358558
Generation 68: Best Fitness = -0.358558
Generation 69: Best Fitness = -0.358558
Generation 70: Best Fitness = -0.358558
Generation 71: Best Fitness = -0.358558
Generation 72: Best Fitness = -0.358558
Generation 73: Best Fitness = -0.358558
Generation 74: Best Fitness = -0.358558
Generation 75: Best Fitness = -0.355425
Generation 76: Best Fitness = -0.355425
Generation 77: Best Fitness = -0.341778
Generation 78: Best Fitness = -0.341778
Generation 79: Best Fitness = -0.341778
Generation 80: Best Fitness = -0.326870
Generation 81: Best Fitness = -0.326870
Generation 82: Best Fitness = -0.326870
Generation 83: Best Fitness = -0.326870
Generation 84: Best Fitness = -0.326870
Generation 85: Best Fitness = -0.326870
Generation 86: Best Fitness = -0.326870
Generation 87: Best Fitness = -0.326870
Generation 88: Best Fitness = -0.326870
Generation 89: Best Fitness = -0.326870
Generation 90: Best Fitness = -0.326870
Generation 91: Best Fitness = -0.326870
Generation 92: Best Fitness = -0.326870
Generation 93: Best Fitness = -0.326870
Generation 94: Best Fitness = -0.326870
Generation 95: Best Fitness = -0.326870
Generation 96: Best Fitness = -0.326870
Generation 97: Best Fitness = -0.326870
Generation 98: Best Fitness = -0.326870
Generation 99: Best Fitness = -0.326870
Generation 100: Best Fitness = -0.326870
```

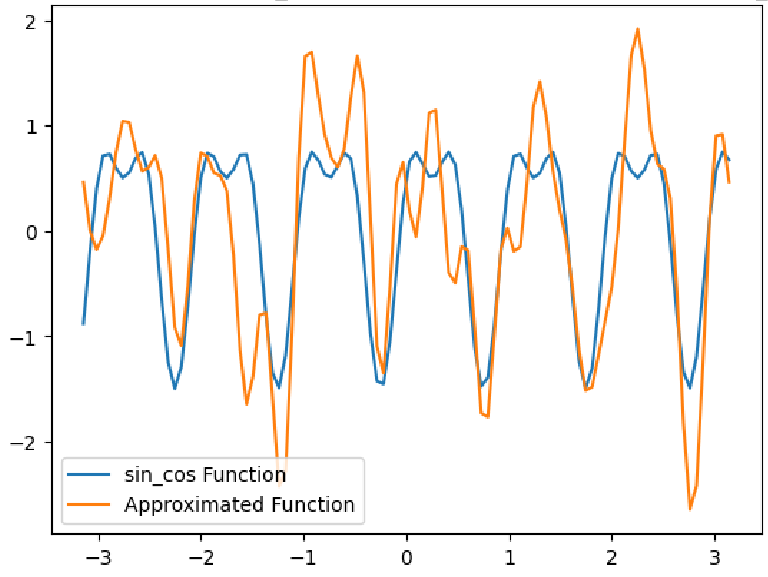

Fitness Changes for sin_cos Function
Best Fitness (Negative RMSE)
−0.25
−0.50
−0.75
−1.00
−1.25
−1.50
−1.75
−2.00
0
20
40
60
80
100
Generation
Tournament Selection + Uniform Crossover


Fourier Series Approximation of sin_cos using Tournament Selection and Uniform Crossover
sin_cos Function
Approximated Function
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−3
−2
−1
0
1
2
3

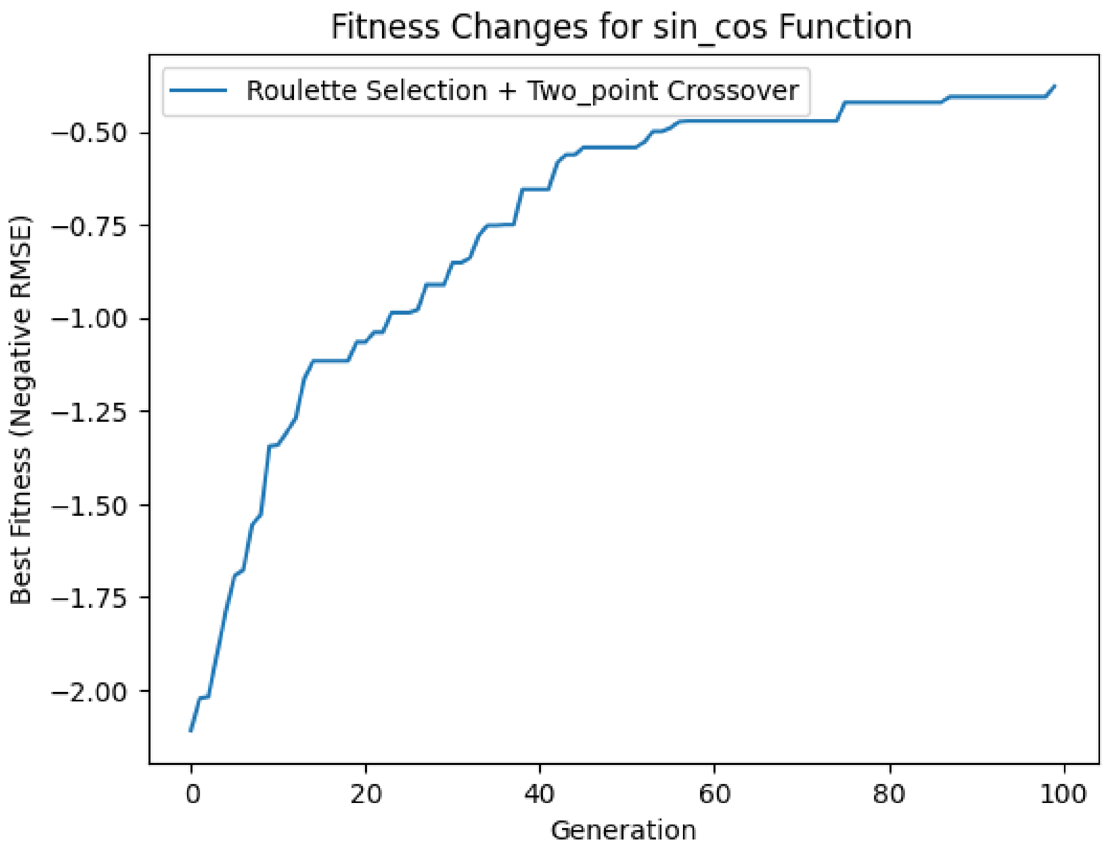
```
Using Roulette Selection and Single_point Crossover:
Generation 1: Best Fitness = -2.162589
Generation 2: Best Fitness = -2.028314
Generation 3: Best Fitness = -1.937256
Generation 4: Best Fitness = -1.933549
Generation 5: Best Fitness = -1.933549
Generation 6: Best Fitness = -1.933549
Generation 7: Best Fitness = -1.876316
Generation 8: Best Fitness = -1.832288
Generation 9: Best Fitness = -1.760046
Generation 10: Best Fitness = -1.760046
Generation 11: Best Fitness = -1.760046
Generation 12: Best Fitness = -1.723238
Generation 13: Best Fitness = -1.703744
Generation 14: Best Fitness = -1.680786
Generation 15: Best Fitness = -1.594824
Generation 16: Best Fitness = -1.594824
Generation 17: Best Fitness = -1.557259
Generation 18: Best Fitness = -1.518775
Generation 19: Best Fitness = -1.516087
Generation 20: Best Fitness = -1.450860
Generation 21: Best Fitness = -1.450860
Generation 22: Best Fitness = -1.399501
Generation 23: Best Fitness = -1.370827
Generation 24: Best Fitness = -1.352265
Generation 25: Best Fitness = -1.302333
Generation 26: Best Fitness = -1.254843
Generation 27: Best Fitness = -1.254843
Generation 28: Best Fitness = -1.254843
Generation 29: Best Fitness = -1.210434
Generation 30: Best Fitness = -1.187665
Generation 31: Best Fitness = -1.152938
Generation 32: Best Fitness = -1.123024
Generation 33: Best Fitness = -1.123024
Generation 34: Best Fitness = -1.074863
Generation 35: Best Fitness = -1.044987
Generation 36: Best Fitness = -1.043876
Generation 37: Best Fitness = -1.043876
Generation 38: Best Fitness = -0.993828
Generation 39: Best Fitness = -0.993828
Generation 40: Best Fitness = -0.993828
Generation 41: Best Fitness = -0.993828
Generation 42: Best Fitness = -0.993828
Generation 43: Best Fitness = -0.939048
Generation 44: Best Fitness = -0.933716
Generation 45: Best Fitness = -0.928478
Generation 46: Best Fitness = -0.924791
Generation 47: Best Fitness = -0.902107
Generation 48: Best Fitness = -0.874079
Generation 49: Best Fitness = -0.874079
Generation 50: Best Fitness = -0.852121
Generation 51: Best Fitness = -0.826659
Generation 52: Best Fitness = -0.807468
Generation 53: Best Fitness = -0.728109
Generation 54: Best Fitness = -0.728109
Generation 55: Best Fitness = -0.711106
Generation 56: Best Fitness = -0.711106
Generation 57: Best Fitness = -0.710546
Generation 58: Best Fitness = -0.710546
Generation 59: Best Fitness = -0.699501
Generation 60: Best Fitness = -0.691691
Generation 61: Best Fitness = -0.673736
Generation 62: Best Fitness = -0.666714
Generation 63: Best Fitness = -0.628496
Generation 64: Best Fitness = -0.571151
Generation 65: Best Fitness = -0.571151
Generation 66: Best Fitness = -0.571151
Generation 67: Best Fitness = -0.571151
Generation 68: Best Fitness = -0.542662
Generation 69: Best Fitness = -0.542662
Generation 70: Best Fitness = -0.542662
Generation 71: Best Fitness = -0.542662
Generation 72: Best Fitness = -0.542662
Generation 73: Best Fitness = -0.542662
Generation 74: Best Fitness = -0.517433
Generation 75: Best Fitness = -0.517433
Generation 76: Best Fitness = -0.517433
Generation 77: Best Fitness = -0.517433
Generation 78: Best Fitness = -0.517433
Generation 79: Best Fitness = -0.517433
Generation 80: Best Fitness = -0.517433
Generation 81: Best Fitness = -0.513986
Generation 82: Best Fitness = -0.513986
Generation 83: Best Fitness = -0.513986
Generation 84: Best Fitness = -0.513986
Generation 85: Best Fitness = -0.513986
Generation 86: Best Fitness = -0.513986
Generation 87: Best Fitness = -0.513986
Generation 88: Best Fitness = -0.513986
Generation 89: Best Fitness = -0.494295
Generation 90: Best Fitness = -0.494295
Generation 91: Best Fitness = -0.494295
Generation 92: Best Fitness = -0.494295
Generation 93: Best Fitness = -0.492354
Generation 94: Best Fitness = -0.492354
Generation 95: Best Fitness = -0.492354
Generation 96: Best Fitness = -0.492354
Generation 97: Best Fitness = -0.471674
Generation 98: Best Fitness = -0.467336
Generation 99: Best Fitness = -0.467336
Generation 100: Best Fitness = -0.467336
```

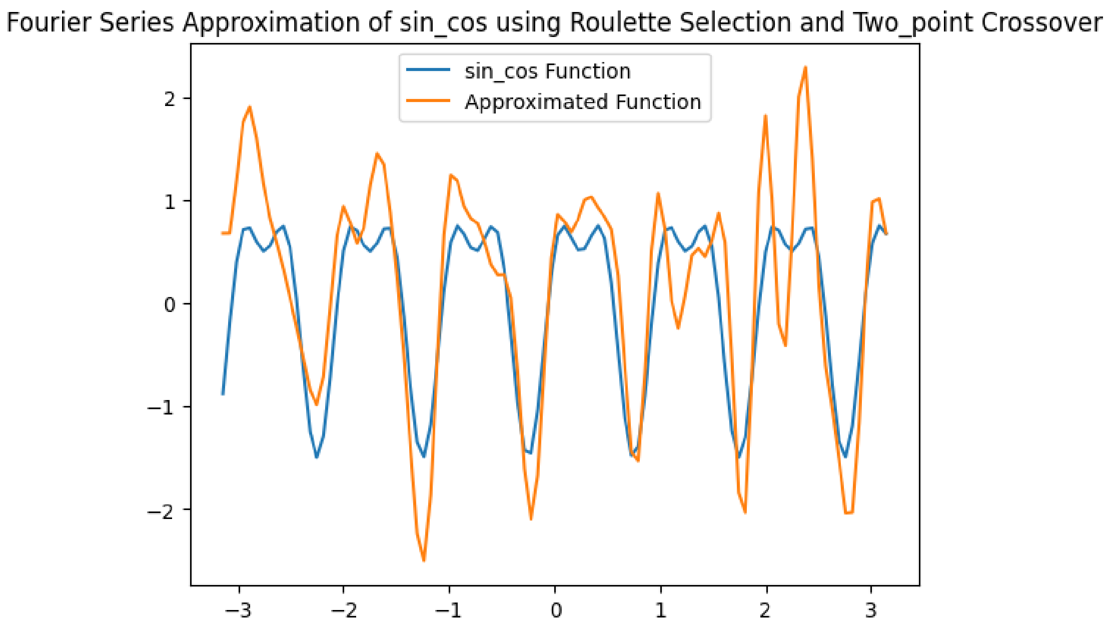


Fourier Series Approximation of sin_cos using Roulette Selection and Single_point Crossover

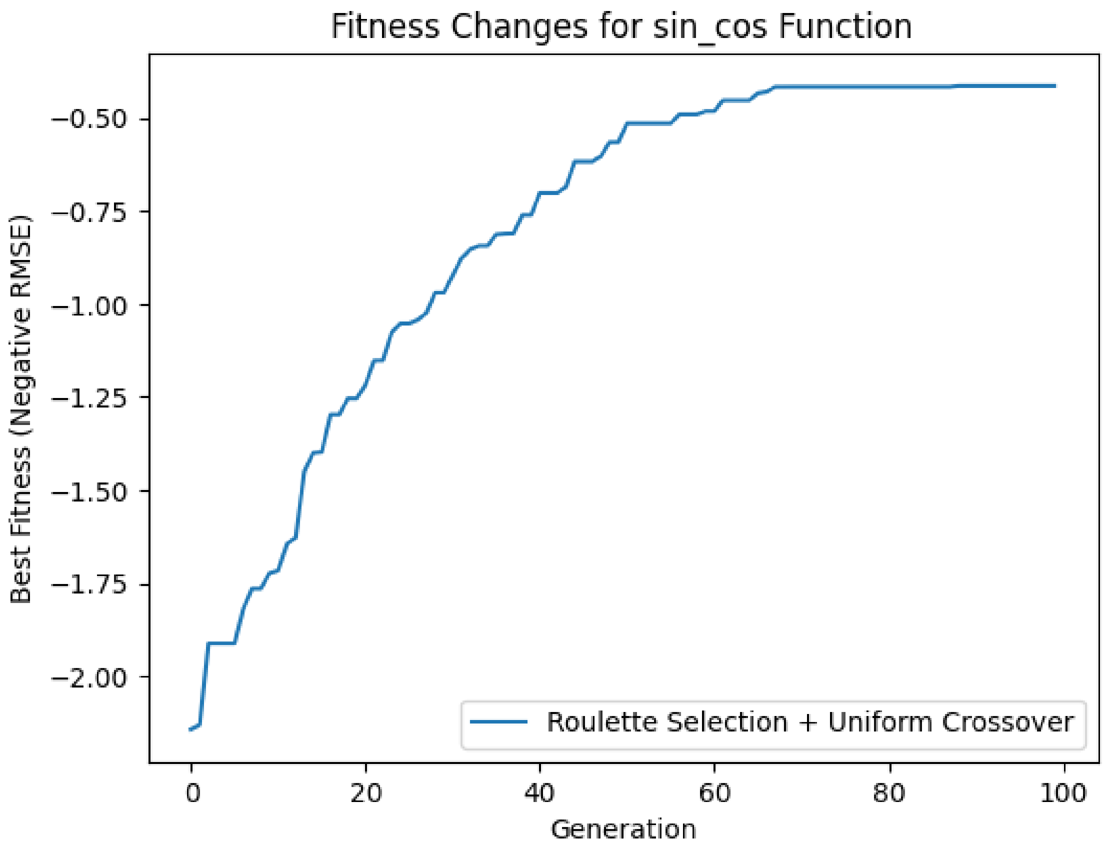


```
Using Roulette Selection and Two_point Crossover:
Generation 1: Best Fitness = -2.108392
Generation 2: Best Fitness = -2.022611
Generation 3: Best Fitness = -2.018019
Generation 4: Best Fitness = -1.901760
Generation 5: Best Fitness = -1.788229
Generation 6: Best Fitness = -1.692993
Generation 7: Best Fitness = -1.677050
Generation 8: Best Fitness = -1.557149
Generation 9: Best Fitness = -1.528982
Generation 10: Best Fitness = -1.345081
Generation 11: Best Fitness = -1.341238
Generation 12: Best Fitness = -1.306224
Generation 13: Best Fitness = -1.270272
Generation 14: Best Fitness = -1.163483
Generation 15: Best Fitness = -1.115745
Generation 16: Best Fitness = -1.115745
Generation 17: Best Fitness = -1.115745
Generation 18: Best Fitness = -1.115745
Generation 19: Best Fitness = -1.115745
Generation 20: Best Fitness = -1.065038
Generation 21: Best Fitness = -1.065038
Generation 22: Best Fitness = -1.038631
Generation 23: Best Fitness = -1.038631
Generation 24: Best Fitness = -0.985573
Generation 25: Best Fitness = -0.985573
Generation 26: Best Fitness = -0.985573
Generation 27: Best Fitness = -0.977971
Generation 28: Best Fitness = -0.911183
Generation 29: Best Fitness = -0.911183
Generation 30: Best Fitness = -0.911183
Generation 31: Best Fitness = -0.851761
Generation 32: Best Fitness = -0.851761
Generation 33: Best Fitness = -0.838627
Generation 34: Best Fitness = -0.778160
Generation 35: Best Fitness = -0.751267
Generation 36: Best Fitness = -0.751267
Generation 37: Best Fitness = -0.749525
Generation 38: Best Fitness = -0.749525
Generation 39: Best Fitness = -0.654566
Generation 40: Best Fitness = -0.654566
Generation 41: Best Fitness = -0.654566
Generation 42: Best Fitness = -0.654566
Generation 43: Best Fitness = -0.582726
Generation 44: Best Fitness = -0.561567
Generation 45: Best Fitness = -0.561567
Generation 46: Best Fitness = -0.542072
Generation 47: Best Fitness = -0.542072
Generation 48: Best Fitness = -0.542072
Generation 49: Best Fitness = -0.542072
Generation 50: Best Fitness = -0.542072
Generation 51: Best Fitness = -0.542072
Generation 52: Best Fitness = -0.542072
Generation 53: Best Fitness = -0.526737
Generation 54: Best Fitness = -0.498661
Generation 55: Best Fitness = -0.498661
Generation 56: Best Fitness = -0.488938
Generation 57: Best Fitness = -0.472524
Generation 58: Best Fitness = -0.470955
Generation 59: Best Fitness = -0.470955
Generation 60: Best Fitness = -0.470955
Generation 61: Best Fitness = -0.470955
Generation 62: Best Fitness = -0.470955
Generation 63: Best Fitness = -0.470955
Generation 64: Best Fitness = -0.470955
Generation 65: Best Fitness = -0.470955
Generation 66: Best Fitness = -0.470955
Generation 67: Best Fitness = -0.470955
Generation 68: Best Fitness = -0.470955
Generation 69: Best Fitness = -0.470955
Generation 70: Best Fitness = -0.470955
Generation 71: Best Fitness = -0.470955
Generation 72: Best Fitness = -0.470955
Generation 73: Best Fitness = -0.470955
Generation 74: Best Fitness = -0.470955
Generation 75: Best Fitness = -0.470955
Generation 76: Best Fitness = -0.420688
Generation 77: Best Fitness = -0.420688
Generation 78: Best Fitness = -0.420688
Generation 79: Best Fitness = -0.420688
Generation 80: Best Fitness = -0.420688
Generation 81: Best Fitness = -0.420688
Generation 82: Best Fitness = -0.420688
Generation 83: Best Fitness = -0.420688
Generation 84: Best Fitness = -0.420688
Generation 85: Best Fitness = -0.420688
Generation 86: Best Fitness = -0.420688
Generation 87: Best Fitness = -0.420688
Generation 88: Best Fitness = -0.405437
Generation 89: Best Fitness = -0.405437
Generation 90: Best Fitness = -0.405437
Generation 91: Best Fitness = -0.405437
Generation 92: Best Fitness = -0.405437
Generation 93: Best Fitness = -0.405437
Generation 94: Best Fitness = -0.405437
Generation 95: Best Fitness = -0.405437
Generation 96: Best Fitness = -0.405437
Generation 97: Best Fitness = -0.405437
Generation 98: Best Fitness = -0.405437
Generation 99: Best Fitness = -0.405437
Generation 100: Best Fitness = -0.377366
```

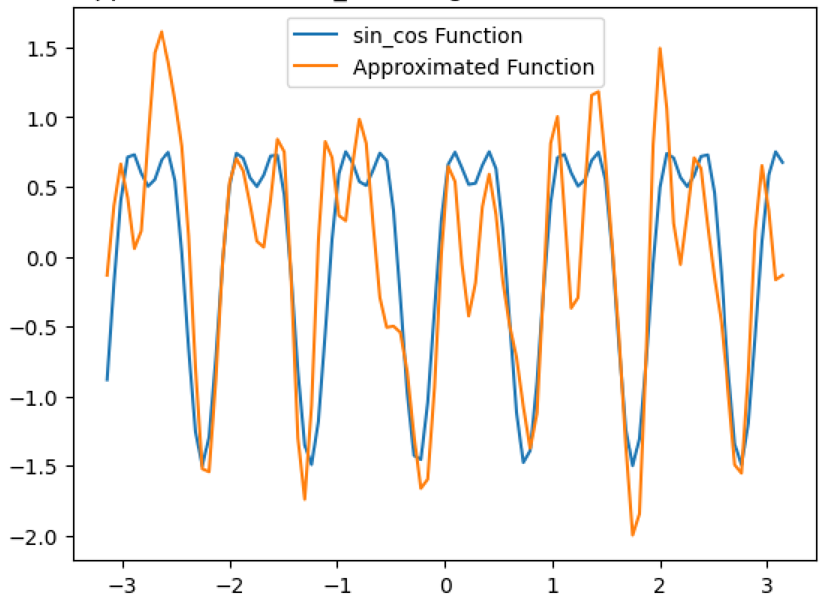

Fitness Changes for sin_cos Function
Roulette Selection + Two_point Crossover
Best Fitness (Negative RMSE)
−0.50
−0.75
−1.00
−1.25
−1.50
−1.75
−2.00
0
20
40
60
80
100
Generation


Fourier Series Approximation of sin_cos using Roulette Selection and Two_point Crossover
sin_cos Function
Approximated Function
2
1
0
−1
−2
−3
−2
−1
0
1
2
3

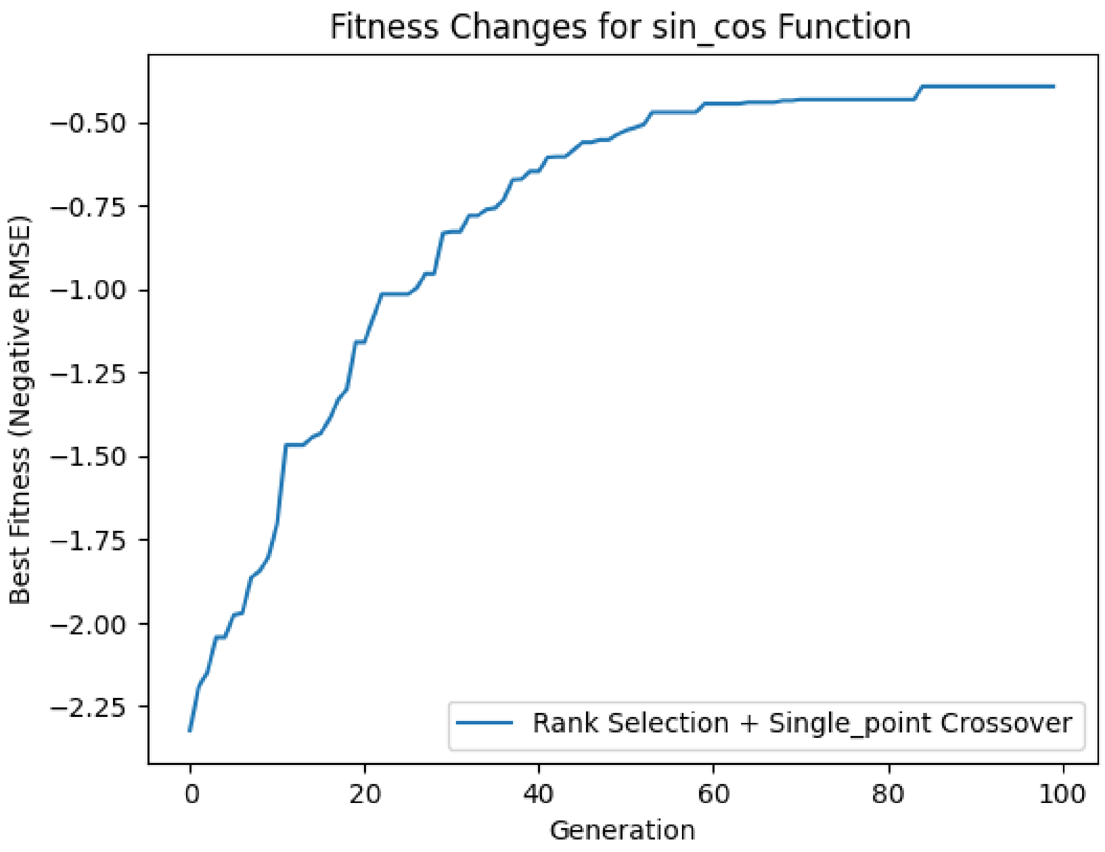
```
Using Roulette Selection and Uniform Crossover:
Generation 1: Best Fitness = -2.143155
Generation 2: Best Fitness = -2.131180
Generation 3: Best Fitness = -1.912401
Generation 4: Best Fitness = -1.912401
Generation 5: Best Fitness = -1.912401
Generation 6: Best Fitness = -1.912401
Generation 7: Best Fitness = -1.818585
Generation 8: Best Fitness = -1.765196
Generation 9: Best Fitness = -1.765196
Generation 10: Best Fitness = -1.723656
Generation 11: Best Fitness = -1.716254
Generation 12: Best Fitness = -1.645097
Generation 13: Best Fitness = -1.628147
Generation 14: Best Fitness = -1.449868
Generation 15: Best Fitness = -1.400552
Generation 16: Best Fitness = -1.397911
Generation 17: Best Fitness = -1.297439
Generation 18: Best Fitness = -1.297439
Generation 19: Best Fitness = -1.254123
Generation 20: Best Fitness = -1.254123
Generation 21: Best Fitness = -1.217992
Generation 22: Best Fitness = -1.152014
Generation 23: Best Fitness = -1.152014
Generation 24: Best Fitness = -1.077329
Generation 25: Best Fitness = -1.052396
Generation 26: Best Fitness = -1.052396
Generation 27: Best Fitness = -1.042872
Generation 28: Best Fitness = -1.024368
Generation 29: Best Fitness = -0.969642
Generation 30: Best Fitness = -0.969642
Generation 31: Best Fitness = -0.923946
Generation 32: Best Fitness = -0.878075
Generation 33: Best Fitness = -0.854007
Generation 34: Best Fitness = -0.843904
Generation 35: Best Fitness = -0.843904
Generation 36: Best Fitness = -0.813051
Generation 37: Best Fitness = -0.811031
Generation 38: Best Fitness = -0.810108
Generation 39: Best Fitness = -0.761048
Generation 40: Best Fitness = -0.761048
Generation 41: Best Fitness = -0.702206
Generation 42: Best Fitness = -0.702206
Generation 43: Best Fitness = -0.702206
Generation 44: Best Fitness = -0.684925
Generation 45: Best Fitness = -0.617610
Generation 46: Best Fitness = -0.617610
Generation 47: Best Fitness = -0.617610
Generation 48: Best Fitness = -0.603286
Generation 49: Best Fitness = -0.565525
Generation 50: Best Fitness = -0.565525
Generation 51: Best Fitness = -0.515007
Generation 52: Best Fitness = -0.515007
Generation 53: Best Fitness = -0.515007
Generation 54: Best Fitness = -0.515007
Generation 55: Best Fitness = -0.515007
Generation 56: Best Fitness = -0.515007
Generation 57: Best Fitness = -0.490868
Generation 58: Best Fitness = -0.490868
Generation 59: Best Fitness = -0.490868
Generation 60: Best Fitness = -0.482127
Generation 61: Best Fitness = -0.482127
Generation 62: Best Fitness = -0.452938
Generation 63: Best Fitness = -0.452938
Generation 64: Best Fitness = -0.452938
Generation 65: Best Fitness = -0.452938
Generation 66: Best Fitness = -0.433919
Generation 67: Best Fitness = -0.430049
Generation 68: Best Fitness = -0.416655
Generation 69: Best Fitness = -0.416655
Generation 70: Best Fitness = -0.416498
Generation 71: Best Fitness = -0.416498
Generation 72: Best Fitness = -0.416498
Generation 73: Best Fitness = -0.416498
Generation 74: Best Fitness = -0.416498
Generation 75: Best Fitness = -0.416498
Generation 76: Best Fitness = -0.416498
Generation 77: Best Fitness = -0.416498
Generation 78: Best Fitness = -0.416498
Generation 79: Best Fitness = -0.416498
Generation 80: Best Fitness = -0.416498
Generation 81: Best Fitness = -0.416498
Generation 82: Best Fitness = -0.416498
Generation 83: Best Fitness = -0.416498
Generation 84: Best Fitness = -0.416498
Generation 85: Best Fitness = -0.416498
Generation 86: Best Fitness = -0.416498
Generation 87: Best Fitness = -0.416498
Generation 88: Best Fitness = -0.416498
Generation 89: Best Fitness = -0.413686
Generation 90: Best Fitness = -0.413686
Generation 91: Best Fitness = -0.413686
Generation 92: Best Fitness = -0.413686
Generation 93: Best Fitness = -0.413686
Generation 94: Best Fitness = -0.413686
Generation 95: Best Fitness = -0.413686
Generation 96: Best Fitness = -0.413686
Generation 97: Best Fitness = -0.413686
Generation 98: Best Fitness = -0.413686
Generation 99: Best Fitness = -0.413686
Generation 100: Best Fitness = -0.413686
```



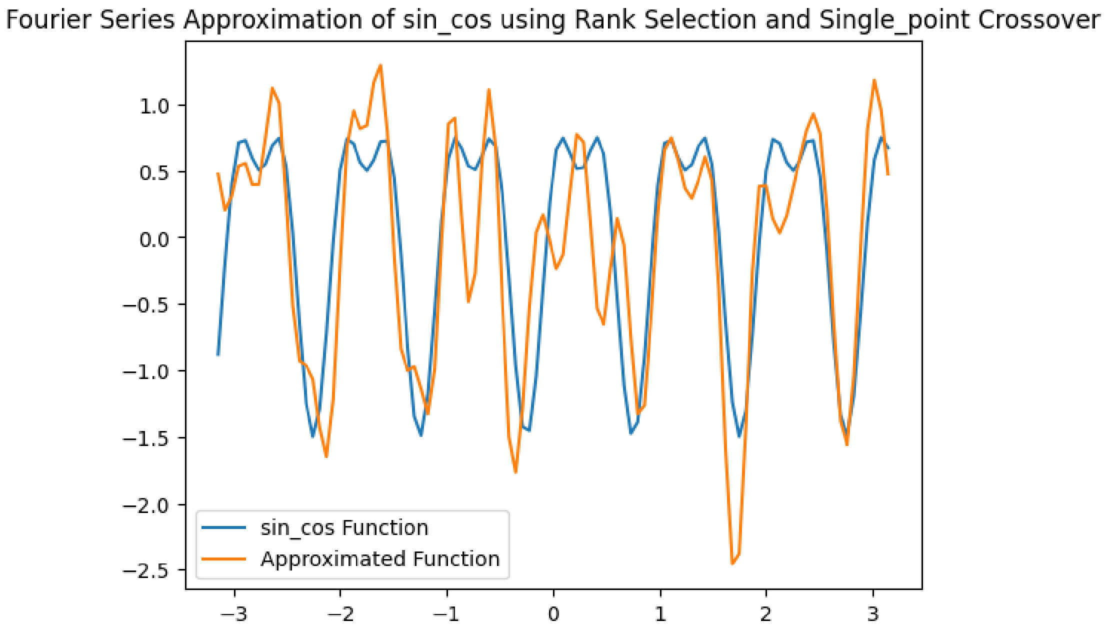
Fourier Series Approximation of sin_cos using Roulette Selection and Uniform Crossover



```
Using Rank Selection and Single_point Crossover:
Generation 1: Best Fitness = -2.322382
Generation 2: Best Fitness = -2.192075
Generation 3: Best Fitness = -2.148258
Generation 4: Best Fitness = -2.044137
Generation 5: Best Fitness = -2.044137
Generation 6: Best Fitness = -1.977657
Generation 7: Best Fitness = -1.971217
Generation 8: Best Fitness = -1.865437
Generation 9: Best Fitness = -1.844772
Generation 10: Best Fitness = -1.803453
Generation 11: Best Fitness = -1.702240
Generation 12: Best Fitness = -1.467832
Generation 13: Best Fitness = -1.467832
Generation 14: Best Fitness = -1.467832
Generation 15: Best Fitness = -1.444364
Generation 16: Best Fitness = -1.432463
Generation 17: Best Fitness = -1.389547
Generation 18: Best Fitness = -1.331507
Generation 19: Best Fitness = -1.300236
Generation 20: Best Fitness = -1.159582
Generation 21: Best Fitness = -1.159582
Generation 22: Best Fitness = -1.086933
Generation 23: Best Fitness = -1.015507
Generation 24: Best Fitness = -1.015507
Generation 25: Best Fitness = -1.015507
Generation 26: Best Fitness = -1.015507
Generation 27: Best Fitness = -0.996984
Generation 28: Best Fitness = -0.954751
Generation 29: Best Fitness = -0.954751
Generation 30: Best Fitness = -0.832672
Generation 31: Best Fitness = -0.828175
Generation 32: Best Fitness = -0.828175
Generation 33: Best Fitness = -0.780062
Generation 34: Best Fitness = -0.780062
Generation 35: Best Fitness = -0.761430
Generation 36: Best Fitness = -0.757487
Generation 37: Best Fitness = -0.730108
Generation 38: Best Fitness = -0.672703
Generation 39: Best Fitness = -0.671016
Generation 40: Best Fitness = -0.647519
Generation 41: Best Fitness = -0.647519
Generation 42: Best Fitness = -0.605539
Generation 43: Best Fitness = -0.603719
Generation 44: Best Fitness = -0.603719
Generation 45: Best Fitness = -0.583215
Generation 46: Best Fitness = -0.560451
Generation 47: Best Fitness = -0.560451
Generation 48: Best Fitness = -0.553028
Generation 49: Best Fitness = -0.553028
Generation 50: Best Fitness = -0.537162
Generation 51: Best Fitness = -0.524658
Generation 52: Best Fitness = -0.516701
Generation 53: Best Fitness = -0.507936
Generation 54: Best Fitness = -0.470366
Generation 55: Best Fitness = -0.470366
Generation 56: Best Fitness = -0.470366
Generation 57: Best Fitness = -0.470366
Generation 58: Best Fitness = -0.470366
Generation 59: Best Fitness = -0.470366
Generation 60: Best Fitness = -0.445027
Generation 61: Best Fitness = -0.445027
Generation 62: Best Fitness = -0.445027
Generation 63: Best Fitness = -0.445027
Generation 64: Best Fitness = -0.445027
Generation 65: Best Fitness = -0.440515
Generation 66: Best Fitness = -0.440515
Generation 67: Best Fitness = -0.440515
Generation 68: Best Fitness = -0.440515
Generation 69: Best Fitness = -0.435377
Generation 70: Best Fitness = -0.435377
Generation 71: Best Fitness = -0.432235
Generation 72: Best Fitness = -0.432235
Generation 73: Best Fitness = -0.432235
Generation 74: Best Fitness = -0.432235
Generation 75: Best Fitness = -0.432235
Generation 76: Best Fitness = -0.432235
Generation 77: Best Fitness = -0.432235
Generation 78: Best Fitness = -0.432235
Generation 79: Best Fitness = -0.432235
Generation 80: Best Fitness = -0.432235
Generation 81: Best Fitness = -0.432235
Generation 82: Best Fitness = -0.432235
Generation 83: Best Fitness = -0.432235
Generation 84: Best Fitness = -0.432235
Generation 85: Best Fitness = -0.392492
Generation 86: Best Fitness = -0.392492
Generation 87: Best Fitness = -0.392492
Generation 88: Best Fitness = -0.392492
Generation 89: Best Fitness = -0.392492
Generation 90: Best Fitness = -0.392492
Generation 91: Best Fitness = -0.392492
Generation 92: Best Fitness = -0.392492
Generation 93: Best Fitness = -0.392492
Generation 94: Best Fitness = -0.392492
Generation 95: Best Fitness = -0.392492
Generation 96: Best Fitness = -0.392492
Generation 97: Best Fitness = -0.392492
Generation 98: Best Fitness = -0.392492
Generation 99: Best Fitness = -0.392492
Generation 100: Best Fitness = -0.392492
```

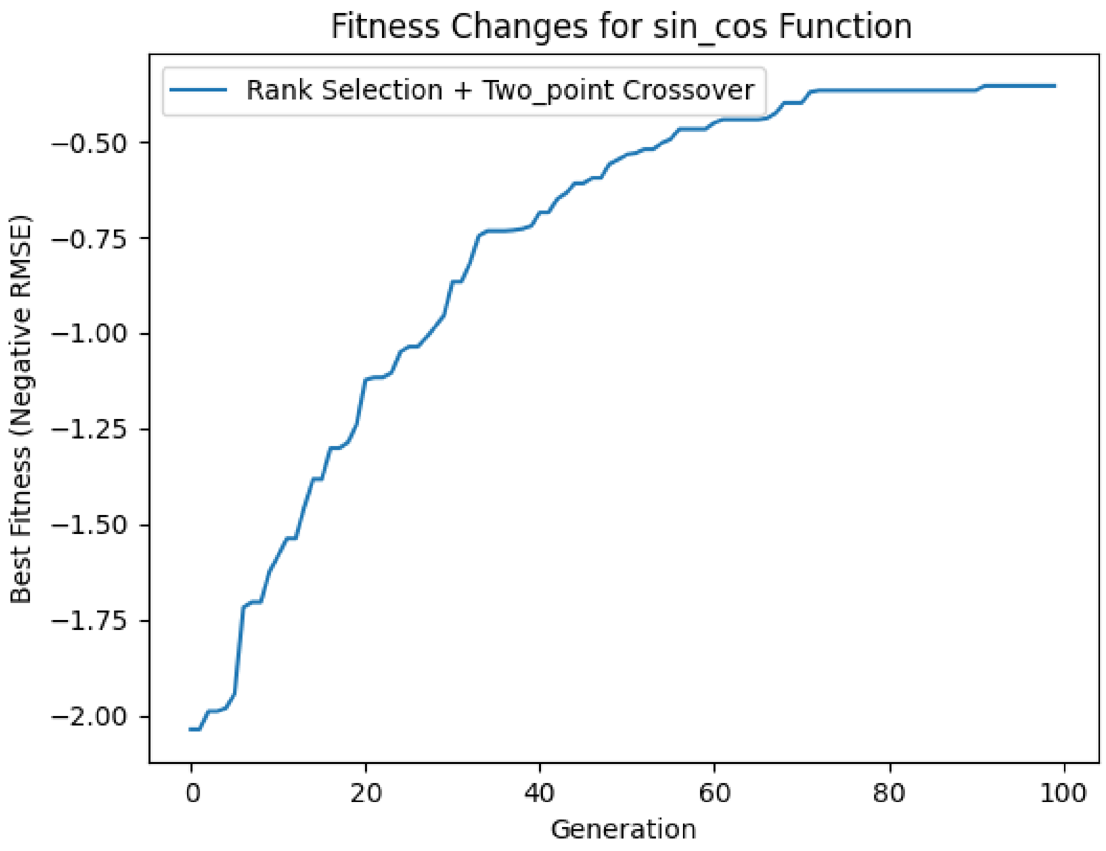


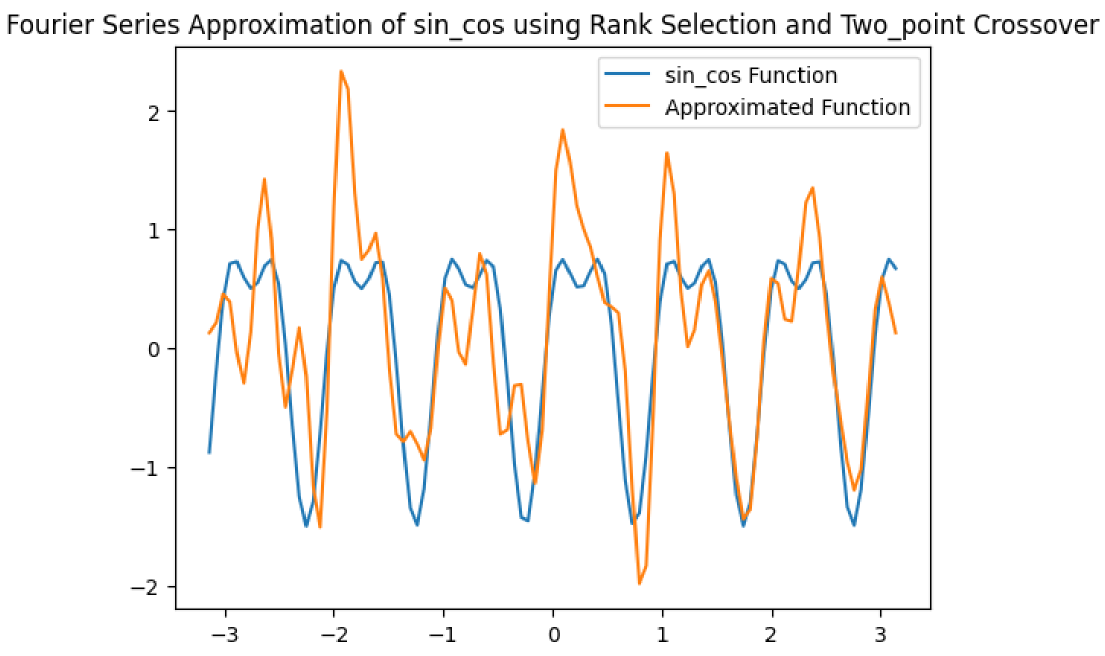
Fitness Changes for sin_cos Function
Best Fitness (Negative RMSE)
−0.50
−0.75
−1.00
−1.25
−1.50
−1.75
−2.00
−2.25
0
20
40
60
80
100
Generation
Rank Selection + Single_point Crossover



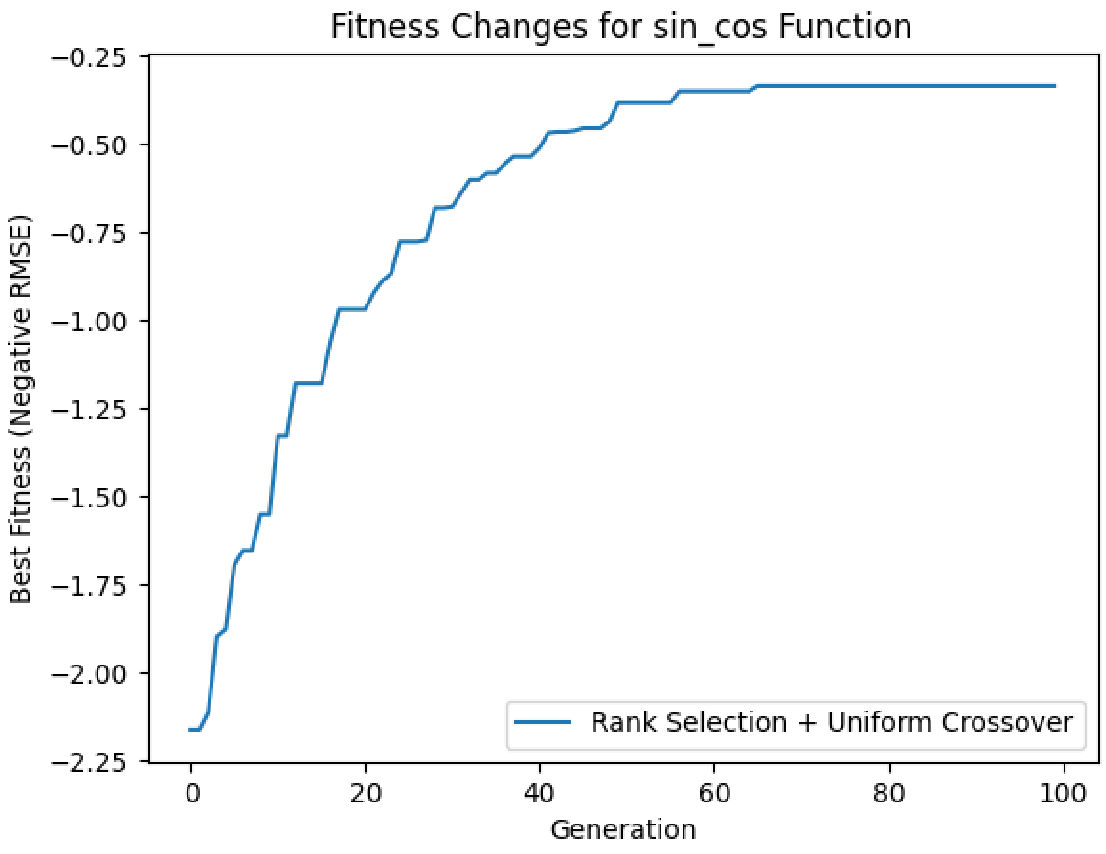
Fourier Series Approximation of sin_cos using Rank Selection and Single_point Crossover
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−2.5
−3
−2
−1
0
1
2
3
sin_cos Function
Approximated Function

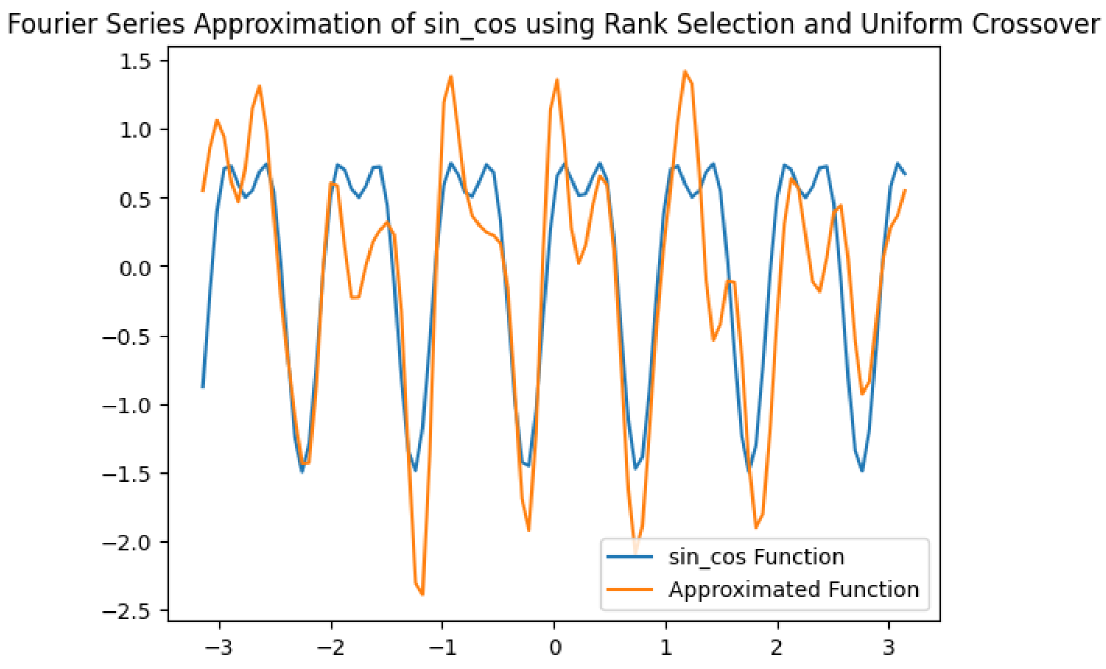
```
Using Rank Selection and Two_point Crossover:
Generation 1: Best Fitness = -2.036504
Generation 2: Best Fitness = -2.036504
Generation 3: Best Fitness = -1.989305
Generation 4: Best Fitness = -1.989305
Generation 5: Best Fitness = -1.981233
Generation 6: Best Fitness = -1.943525
Generation 7: Best Fitness = -1.717657
Generation 8: Best Fitness = -1.704436
Generation 9: Best Fitness = -1.704436
Generation 10: Best Fitness = -1.624924
Generation 11: Best Fitness = -1.582944
Generation 12: Best Fitness = -1.537775
Generation 13: Best Fitness = -1.537775
Generation 14: Best Fitness = -1.455980
Generation 15: Best Fitness = -1.383052
Generation 16: Best Fitness = -1.383052
Generation 17: Best Fitness = -1.301747
Generation 18: Best Fitness = -1.301747
Generation 19: Best Fitness = -1.287540
Generation 20: Best Fitness = -1.239427
Generation 21: Best Fitness = -1.123606
Generation 22: Best Fitness = -1.117406
Generation 23: Best Fitness = -1.117241
Generation 24: Best Fitness = -1.105245
Generation 25: Best Fitness = -1.050753
Generation 26: Best Fitness = -1.036723
Generation 27: Best Fitness = -1.036723
Generation 28: Best Fitness = -1.012676
Generation 29: Best Fitness = -0.984969
Generation 30: Best Fitness = -0.955760
Generation 31: Best Fitness = -0.867499
Generation 32: Best Fitness = -0.867499
Generation 33: Best Fitness = -0.817872
Generation 34: Best Fitness = -0.746942
Generation 35: Best Fitness = -0.734639
Generation 36: Best Fitness = -0.734639
Generation 37: Best Fitness = -0.734639
Generation 38: Best Fitness = -0.732785
Generation 39: Best Fitness = -0.729352
Generation 40: Best Fitness = -0.722035
Generation 41: Best Fitness = -0.686479
Generation 42: Best Fitness = -0.686479
Generation 43: Best Fitness = -0.650830
Generation 44: Best Fitness = -0.636265
Generation 45: Best Fitness = -0.610507
Generation 46: Best Fitness = -0.610507
Generation 47: Best Fitness = -0.596328
Generation 48: Best Fitness = -0.596328
Generation 49: Best Fitness = -0.559354
Generation 50: Best Fitness = -0.548378
Generation 51: Best Fitness = -0.534989
Generation 52: Best Fitness = -0.531178
Generation 53: Best Fitness = -0.521368
Generation 54: Best Fitness = -0.521368
Generation 55: Best Fitness = -0.504882
Generation 56: Best Fitness = -0.494634
Generation 57: Best Fitness = -0.468706
Generation 58: Best Fitness = -0.468706
Generation 59: Best Fitness = -0.468706
Generation 60: Best Fitness = -0.468706
Generation 61: Best Fitness = -0.452250
Generation 62: Best Fitness = -0.443887
Generation 63: Best Fitness = -0.443887
Generation 64: Best Fitness = -0.443887
Generation 65: Best Fitness = -0.443887
Generation 66: Best Fitness = -0.443887
Generation 67: Best Fitness = -0.440556
Generation 68: Best Fitness = -0.428156
Generation 69: Best Fitness = -0.400510
Generation 70: Best Fitness = -0.400315
Generation 71: Best Fitness = -0.400315
Generation 72: Best Fitness = -0.372003
Generation 73: Best Fitness = -0.367634
Generation 74: Best Fitness = -0.367634
Generation 75: Best Fitness = -0.367634
Generation 76: Best Fitness = -0.367634
Generation 77: Best Fitness = -0.367634
Generation 78: Best Fitness = -0.367634
Generation 79: Best Fitness = -0.367634
Generation 80: Best Fitness = -0.367634
Generation 81: Best Fitness = -0.367634
Generation 82: Best Fitness = -0.367634
Generation 83: Best Fitness = -0.367634
Generation 84: Best Fitness = -0.367634
Generation 85: Best Fitness = -0.367634
Generation 86: Best Fitness = -0.367634
Generation 87: Best Fitness = -0.367634
Generation 88: Best Fitness = -0.367634
Generation 89: Best Fitness = -0.367634
Generation 90: Best Fitness = -0.367634
Generation 91: Best Fitness = -0.367634
Generation 92: Best Fitness = -0.355343
Generation 93: Best Fitness = -0.355343
Generation 94: Best Fitness = -0.355343
Generation 95: Best Fitness = -0.355343
Generation 96: Best Fitness = -0.355343
Generation 97: Best Fitness = -0.355343
Generation 98: Best Fitness = -0.355343
Generation 99: Best Fitness = -0.355343
Generation 100: Best Fitness = -0.355343
```



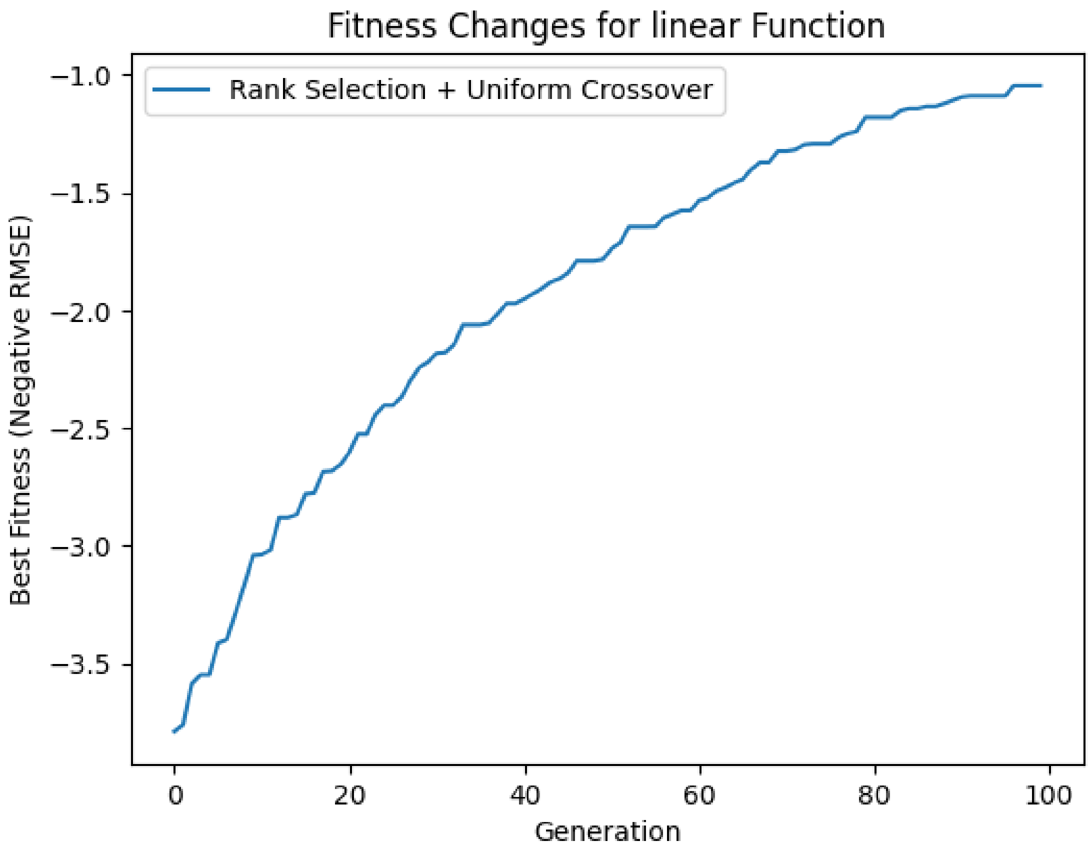


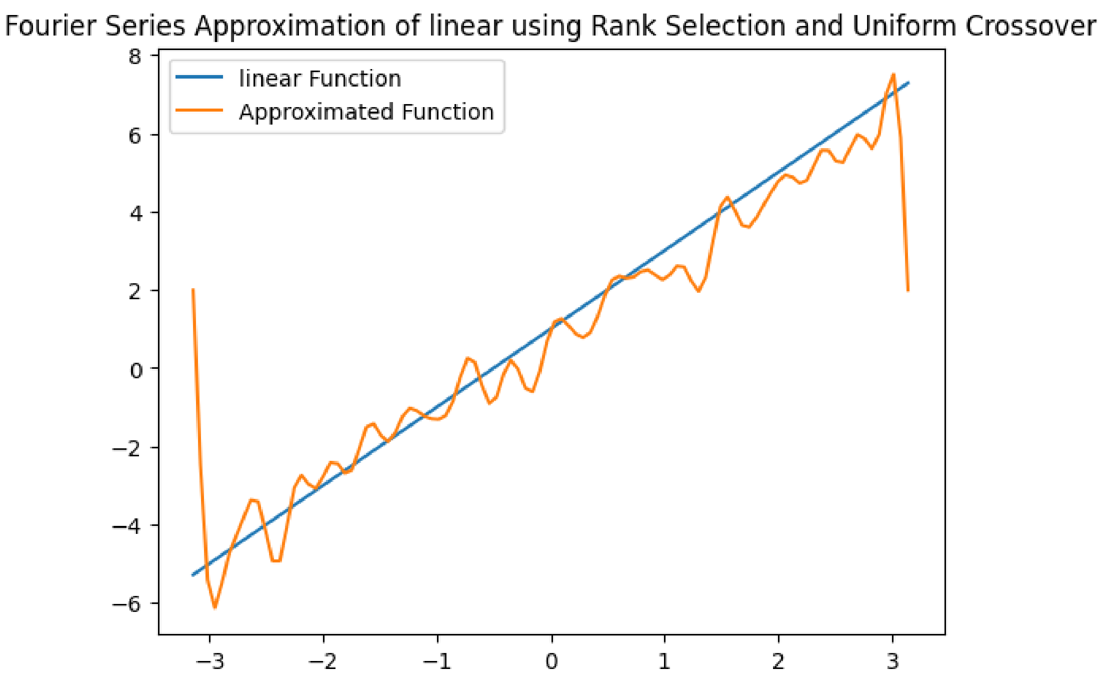
```
Using Rank Selection and Uniform Crossover:
Generation 1: Best Fitness = -2.162980
Generation 2: Best Fitness = -2.162980
Generation 3: Best Fitness = -2.114075
Generation 4: Best Fitness = -1.898868
Generation 5: Best Fitness = -1.877831
Generation 6: Best Fitness = -1.695401
Generation 7: Best Fitness = -1.654284
Generation 8: Best Fitness = -1.654284
Generation 9: Best Fitness = -1.552371
Generation 10: Best Fitness = -1.552371
Generation 11: Best Fitness = -1.327877
Generation 12: Best Fitness = -1.327877
Generation 13: Best Fitness = -1.179624
Generation 14: Best Fitness = -1.179624
Generation 15: Best Fitness = -1.179624
Generation 16: Best Fitness = -1.179624
Generation 17: Best Fitness = -1.064640
Generation 18: Best Fitness = -0.969766
Generation 19: Best Fitness = -0.969766
Generation 20: Best Fitness = -0.969766
Generation 21: Best Fitness = -0.969766
Generation 22: Best Fitness = -0.921800
Generation 23: Best Fitness = -0.888167
Generation 24: Best Fitness = -0.867563
Generation 25: Best Fitness = -0.777946
Generation 26: Best Fitness = -0.777946
Generation 27: Best Fitness = -0.777946
Generation 28: Best Fitness = -0.773690
Generation 29: Best Fitness = -0.681017
Generation 30: Best Fitness = -0.681017
Generation 31: Best Fitness = -0.677496
Generation 32: Best Fitness = -0.639367
Generation 33: Best Fitness = -0.601508
Generation 34: Best Fitness = -0.601508
Generation 35: Best Fitness = -0.582843
Generation 36: Best Fitness = -0.582843
Generation 37: Best Fitness = -0.556088
Generation 38: Best Fitness = -0.535442
Generation 39: Best Fitness = -0.535442
Generation 40: Best Fitness = -0.535442
Generation 41: Best Fitness = -0.510420
Generation 42: Best Fitness = -0.468870
Generation 43: Best Fitness = -0.466207
Generation 44: Best Fitness = -0.466207
Generation 45: Best Fitness = -0.463096
Generation 46: Best Fitness = -0.455503
Generation 47: Best Fitness = -0.455503
Generation 48: Best Fitness = -0.455503
Generation 49: Best Fitness = -0.436013
Generation 50: Best Fitness = -0.382844
Generation 51: Best Fitness = -0.382844
Generation 52: Best Fitness = -0.382844
Generation 53: Best Fitness = -0.382844
Generation 54: Best Fitness = -0.382844
Generation 55: Best Fitness = -0.382844
Generation 56: Best Fitness = -0.382844
Generation 57: Best Fitness = -0.350377
Generation 58: Best Fitness = -0.350377
Generation 59: Best Fitness = -0.350377
Generation 60: Best Fitness = -0.350377
Generation 61: Best Fitness = -0.350377
Generation 62: Best Fitness = -0.350377
Generation 63: Best Fitness = -0.350377
Generation 64: Best Fitness = -0.350377
Generation 65: Best Fitness = -0.350377
Generation 66: Best Fitness = -0.335310
Generation 67: Best Fitness = -0.335310
Generation 68: Best Fitness = -0.335310
Generation 69: Best Fitness = -0.335310
Generation 70: Best Fitness = -0.335310
Generation 71: Best Fitness = -0.335310
Generation 72: Best Fitness = -0.335310
Generation 73: Best Fitness = -0.335310
Generation 74: Best Fitness = -0.335310
Generation 75: Best Fitness = -0.335310
Generation 76: Best Fitness = -0.335310
Generation 77: Best Fitness = -0.335310
Generation 78: Best Fitness = -0.335310
Generation 79: Best Fitness = -0.335310
Generation 80: Best Fitness = -0.335310
Generation 81: Best Fitness = -0.335310
Generation 82: Best Fitness = -0.335310
Generation 83: Best Fitness = -0.335310
Generation 84: Best Fitness = -0.335310
Generation 85: Best Fitness = -0.335310
Generation 86: Best Fitness = -0.335310
Generation 87: Best Fitness = -0.335310
Generation 88: Best Fitness = -0.335310
Generation 89: Best Fitness = -0.335310
Generation 90: Best Fitness = -0.335310
Generation 91: Best Fitness = -0.335310
Generation 92: Best Fitness = -0.335310
Generation 93: Best Fitness = -0.335310
Generation 94: Best Fitness = -0.335310
Generation 95: Best Fitness = -0.335310
Generation 96: Best Fitness = -0.335310
Generation 97: Best Fitness = -0.335310
Generation 98: Best Fitness = -0.335310
Generation 99: Best Fitness = -0.335310
Generation 100: Best Fitness = -0.335310
```


Fitness Changes for sin_cos Function
Best Fitness (Negative RMSE)
−0.25
−0.50
−0.75
−1.00
−1.25
−1.50
−1.75
−2.00
−2.25
0
20
40
60
80
100
Generation
Rank Selection + Uniform Crossover

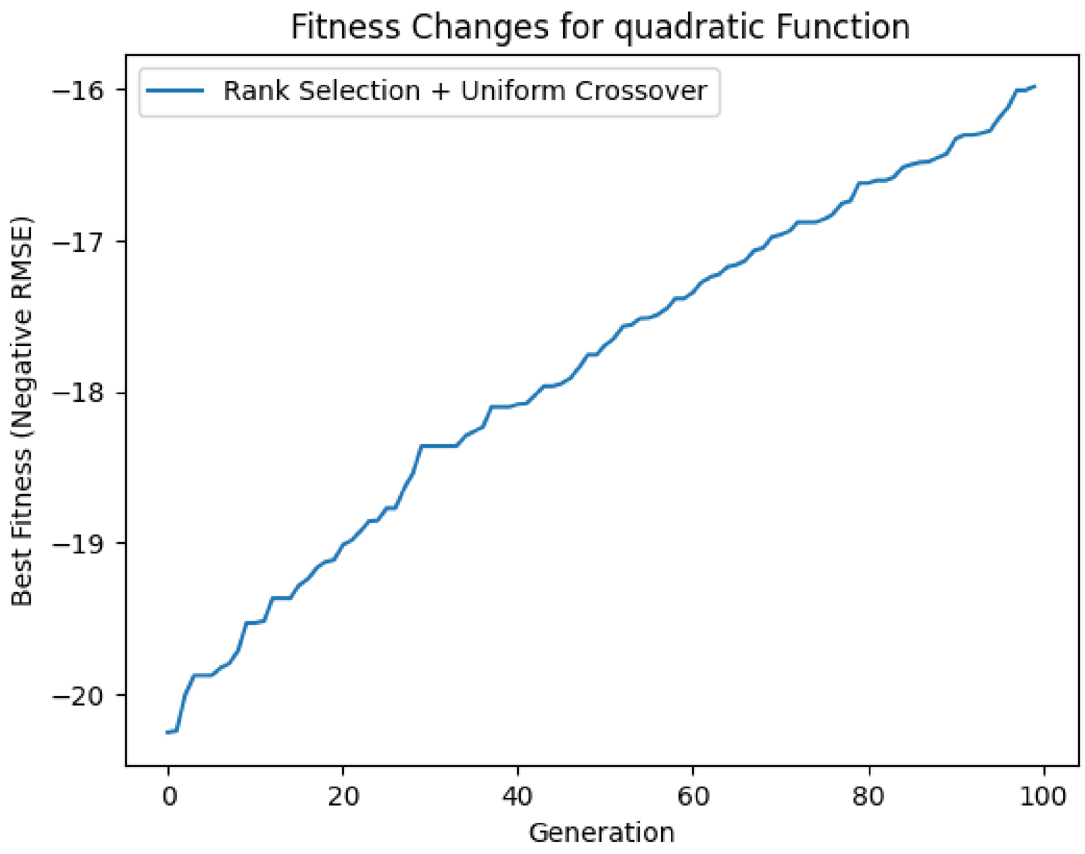

Fourier Series Approximation of sin_cos using Rank Selection and Uniform Crossover
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−2.5
−3
−2
−1
0
1
2
3
sin_cos Function
Approximated Function

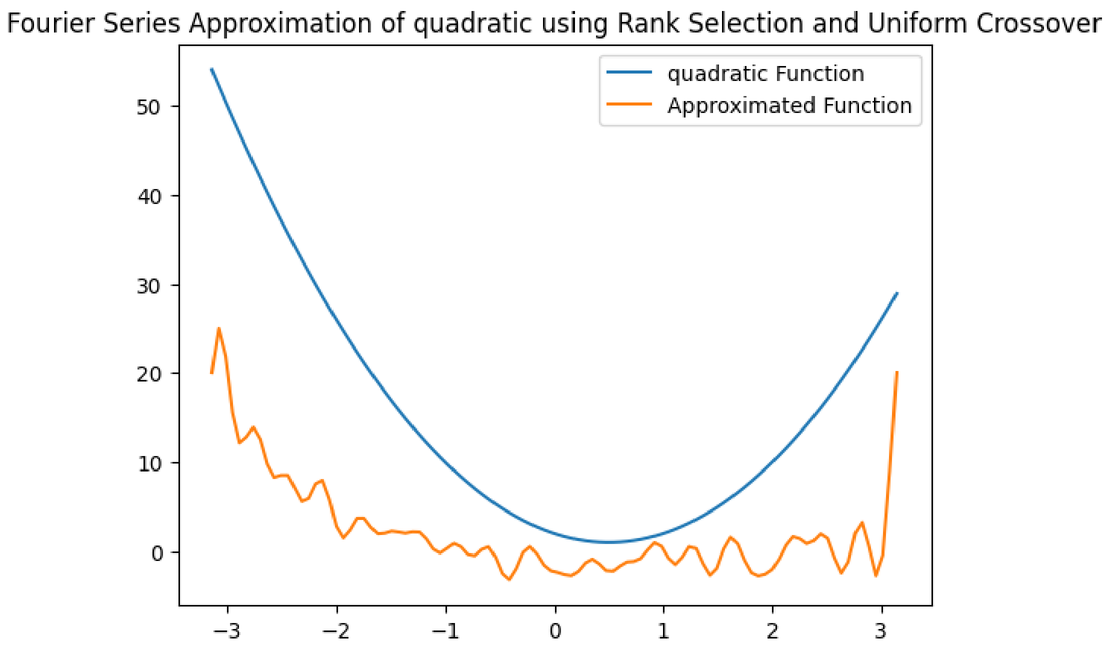
```
Running Genetic Algorithm for linear Function:

Using Rank Selection and Uniform Crossover:
Generation 1: Best Fitness = -3.788070
Generation 2: Best Fitness = -3.761158
Generation 3: Best Fitness = -3.586543
Generation 4: Best Fitness = -3.548104
Generation 5: Best Fitness = -3.548104
Generation 6: Best Fitness = -3.413881
Generation 7: Best Fitness = -3.398261
Generation 8: Best Fitness = -3.285982
Generation 9: Best Fitness = -3.167539
Generation 10: Best Fitness = -3.041495
Generation 11: Best Fitness = -3.037887
Generation 12: Best Fitness = -3.018132
Generation 13: Best Fitness = -2.881012
Generation 14: Best Fitness = -2.881012
Generation 15: Best Fitness = -2.867498
Generation 16: Best Fitness = -2.780593
Generation 17: Best Fitness = -2.776359
Generation 18: Best Fitness = -2.686128
Generation 19: Best Fitness = -2.682729
Generation 20: Best Fitness = -2.655776
Generation 21: Best Fitness = -2.604610
Generation 22: Best Fitness = -2.525980
Generation 23: Best Fitness = -2.525980
Generation 24: Best Fitness = -2.443092
Generation 25: Best Fitness = -2.403765
Generation 26: Best Fitness = -2.403765
Generation 27: Best Fitness = -2.368080
Generation 28: Best Fitness = -2.299170
Generation 29: Best Fitness = -2.243148
Generation 30: Best Fitness = -2.220270
Generation 31: Best Fitness = -2.182944
Generation 32: Best Fitness = -2.179486
Generation 33: Best Fitness = -2.145969
Generation 34: Best Fitness = -2.062098
Generation 35: Best Fitness = -2.062098
Generation 36: Best Fitness = -2.062098
Generation 37: Best Fitness = -2.054351
Generation 38: Best Fitness = -2.014028
Generation 39: Best Fitness = -1.971444
Generation 40: Best Fitness = -1.971444
Generation 41: Best Fitness = -1.952382
Generation 42: Best Fitness = -1.930501
Generation 43: Best Fitness = -1.908828
Generation 44: Best Fitness = -1.881281
Generation 45: Best Fitness = -1.866821
Generation 46: Best Fitness = -1.842823
Generation 47: Best Fitness = -1.791012
Generation 48: Best Fitness = -1.791012
Generation 49: Best Fitness = -1.791012
Generation 50: Best Fitness = -1.784627
Generation 51: Best Fitness = -1.738789
Generation 52: Best Fitness = -1.712724
Generation 53: Best Fitness = -1.645349
Generation 54: Best Fitness = -1.645349
Generation 55: Best Fitness = -1.645349
Generation 56: Best Fitness = -1.644625
Generation 57: Best Fitness = -1.606733
Generation 58: Best Fitness = -1.592319
Generation 59: Best Fitness = -1.576769
Generation 60: Best Fitness = -1.576769
Generation 61: Best Fitness = -1.534884
Generation 62: Best Fitness = -1.523626
Generation 63: Best Fitness = -1.493394
Generation 64: Best Fitness = -1.479595
Generation 65: Best Fitness = -1.459924
Generation 66: Best Fitness = -1.443693
Generation 67: Best Fitness = -1.401188
Generation 68: Best Fitness = -1.372899
Generation 69: Best Fitness = -1.372899
Generation 70: Best Fitness = -1.324653
Generation 71: Best Fitness = -1.324653
Generation 72: Best Fitness = -1.318789
Generation 73: Best Fitness = -1.297037
Generation 74: Best Fitness = -1.293218
Generation 75: Best Fitness = -1.293218
Generation 76: Best Fitness = -1.293218
Generation 77: Best Fitness = -1.266022
Generation 78: Best Fitness = -1.250249
Generation 79: Best Fitness = -1.241752
Generation 80: Best Fitness = -1.180807
Generation 81: Best Fitness = -1.180807
Generation 82: Best Fitness = -1.180807
Generation 83: Best Fitness = -1.180807
Generation 84: Best Fitness = -1.153696
Generation 85: Best Fitness = -1.143875
Generation 86: Best Fitness = -1.143875
Generation 87: Best Fitness = -1.135233
Generation 88: Best Fitness = -1.135233
Generation 89: Best Fitness = -1.123111
Generation 90: Best Fitness = -1.109308
Generation 91: Best Fitness = -1.094757
Generation 92: Best Fitness = -1.090253
Generation 93: Best Fitness = -1.090253
Generation 94: Best Fitness = -1.090253
Generation 95: Best Fitness = -1.090253
Generation 96: Best Fitness = -1.090253
Generation 97: Best Fitness = -1.046307
Generation 98: Best Fitness = -1.046307
Generation 99: Best Fitness = -1.046307
Generation 100: Best Fitness = -1.046307
```



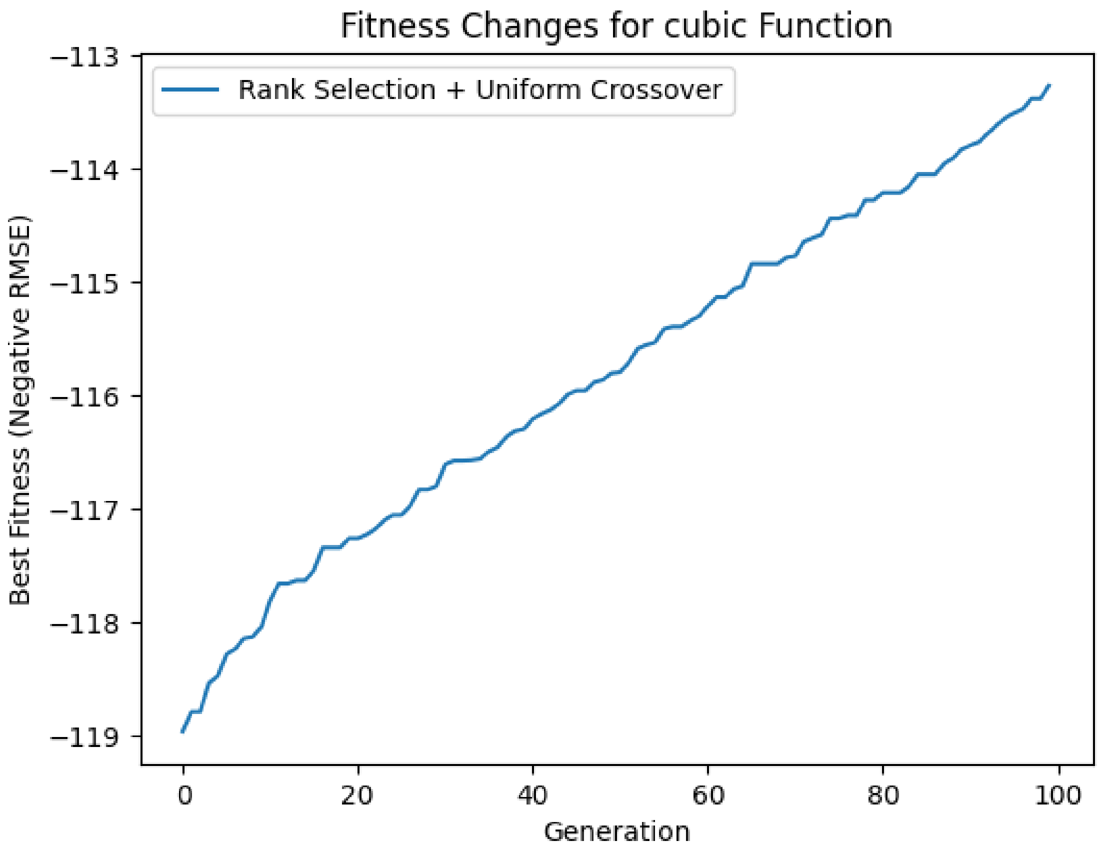


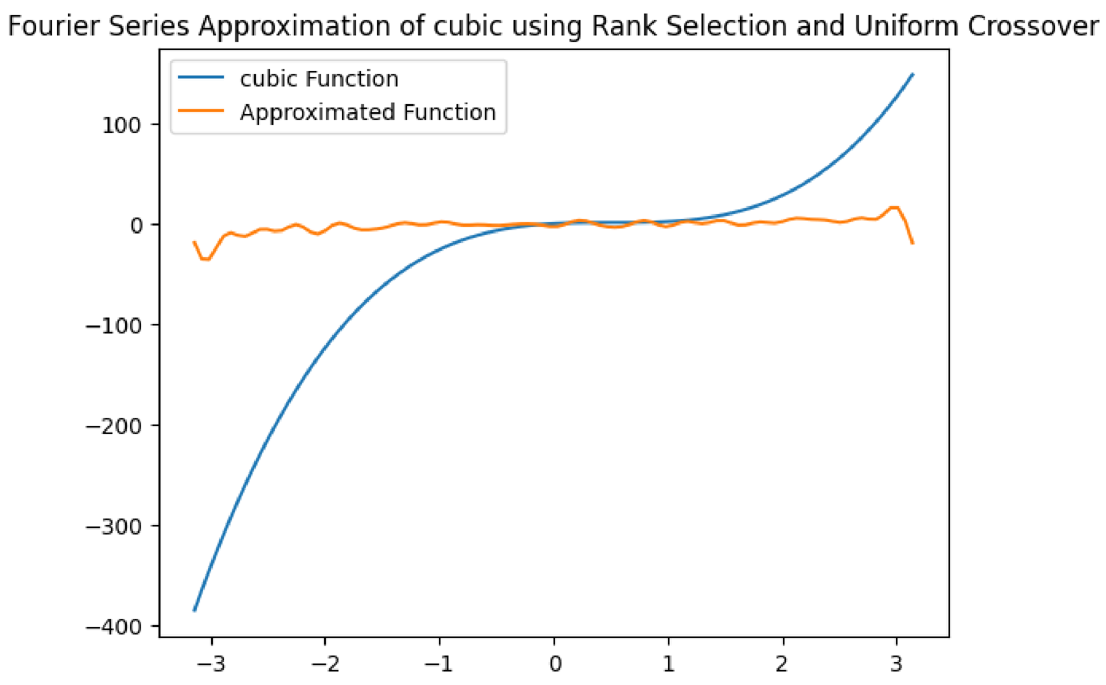
```
Running Genetic Algorithm for quadratic Function:

Using Rank Selection and Uniform Crossover:
Generation 1: Best Fitness = -20.252513
Generation 2: Best Fitness = -20.243021
Generation 3: Best Fitness = -20.005652
Generation 4: Best Fitness = -19.876853
Generation 5: Best Fitness = -19.876853
Generation 6: Best Fitness = -19.876853
Generation 7: Best Fitness = -19.828602
Generation 8: Best Fitness = -19.797509
Generation 9: Best Fitness = -19.718542
Generation 10: Best Fitness = -19.530149
Generation 11: Best Fitness = -19.528790
Generation 12: Best Fitness = -19.516271
Generation 13: Best Fitness = -19.365458
Generation 14: Best Fitness = -19.365458
Generation 15: Best Fitness = -19.365458
Generation 16: Best Fitness = -19.280014
Generation 17: Best Fitness = -19.240570
Generation 18: Best Fitness = -19.166933
Generation 19: Best Fitness = -19.125178
Generation 20: Best Fitness = -19.112504
Generation 21: Best Fitness = -19.014911
Generation 22: Best Fitness = -18.982136
Generation 23: Best Fitness = -18.924218
Generation 24: Best Fitness = -18.856962
Generation 25: Best Fitness = -18.852227
Generation 26: Best Fitness = -18.771125
Generation 27: Best Fitness = -18.771125
Generation 28: Best Fitness = -18.642395
Generation 29: Best Fitness = -18.539415
Generation 30: Best Fitness = -18.360460
Generation 31: Best Fitness = -18.360460
Generation 32: Best Fitness = -18.360460
Generation 33: Best Fitness = -18.360460
Generation 34: Best Fitness = -18.360460
Generation 35: Best Fitness = -18.293127
Generation 36: Best Fitness = -18.263506
Generation 37: Best Fitness = -18.233758
Generation 38: Best Fitness = -18.101298
Generation 39: Best Fitness = -18.101298
Generation 40: Best Fitness = -18.101298
Generation 41: Best Fitness = -18.083607
Generation 42: Best Fitness = -18.078049
Generation 43: Best Fitness = -18.021172
Generation 44: Best Fitness = -17.964087
Generation 45: Best Fitness = -17.964087
Generation 46: Best Fitness = -17.947245
Generation 47: Best Fitness = -17.911109
Generation 48: Best Fitness = -17.841610
Generation 49: Best Fitness = -17.755007
Generation 50: Best Fitness = -17.755007
Generation 51: Best Fitness = -17.691776
Generation 52: Best Fitness = -17.647034
Generation 53: Best Fitness = -17.567980
Generation 54: Best Fitness = -17.557637
Generation 55: Best Fitness = -17.515765
Generation 56: Best Fitness = -17.512880
Generation 57: Best Fitness = -17.489436
Generation 58: Best Fitness = -17.451954
Generation 59: Best Fitness = -17.384760
Generation 60: Best Fitness = -17.384760
Generation 61: Best Fitness = -17.343169
Generation 62: Best Fitness = -17.277125
Generation 63: Best Fitness = -17.241816
Generation 64: Best Fitness = -17.223681
Generation 65: Best Fitness = -17.173147
Generation 66: Best Fitness = -17.160830
Generation 67: Best Fitness = -17.133302
Generation 68: Best Fitness = -17.066800
Generation 69: Best Fitness = -17.051359
Generation 70: Best Fitness = -16.979591
Generation 71: Best Fitness = -16.963064
Generation 72: Best Fitness = -16.939861
Generation 73: Best Fitness = -16.880137
Generation 74: Best Fitness = -16.880137
Generation 75: Best Fitness = -16.880137
Generation 76: Best Fitness = -16.859454
Generation 77: Best Fitness = -16.826934
Generation 78: Best Fitness = -16.755639
Generation 79: Best Fitness = -16.739267
Generation 80: Best Fitness = -16.621224
Generation 81: Best Fitness = -16.621224
Generation 82: Best Fitness = -16.602641
Generation 83: Best Fitness = -16.602641
Generation 84: Best Fitness = -16.580956
Generation 85: Best Fitness = -16.515244
Generation 86: Best Fitness = -16.497805
Generation 87: Best Fitness = -16.483724
Generation 88: Best Fitness = -16.479640
Generation 89: Best Fitness = -16.451595
Generation 90: Best Fitness = -16.427271
Generation 91: Best Fitness = -16.329621
Generation 92: Best Fitness = -16.302714
Generation 93: Best Fitness = -16.302714
Generation 94: Best Fitness = -16.291561
Generation 95: Best Fitness = -16.272838
Generation 96: Best Fitness = -16.187087
Generation 97: Best Fitness = -16.122254
Generation 98: Best Fitness = -16.007623
Generation 99: Best Fitness = -16.007623
Generation 100: Best Fitness = -15.982465
```


Fitness Changes for quadratic Function
Rank Selection + Uniform Crossover
Best Fitness (Negative RMSE)
−16
−17
−18
−19
−20
0
20
40
60
80
100
Generation

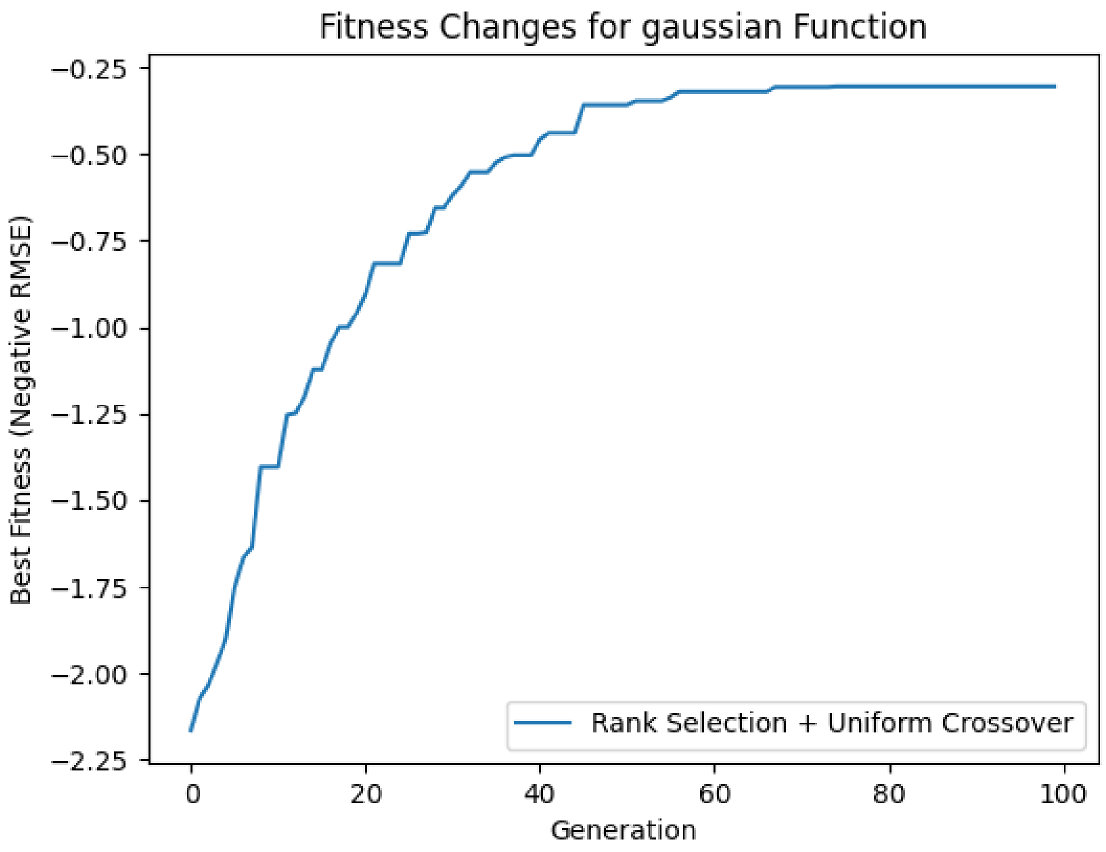

Fourier Series Approximation of quadratic using Rank Selection and Uniform Crossover
quadratic Function
Approximated Function
50
40
30
20
10
0
−3
−2
−1
0
1
2
3

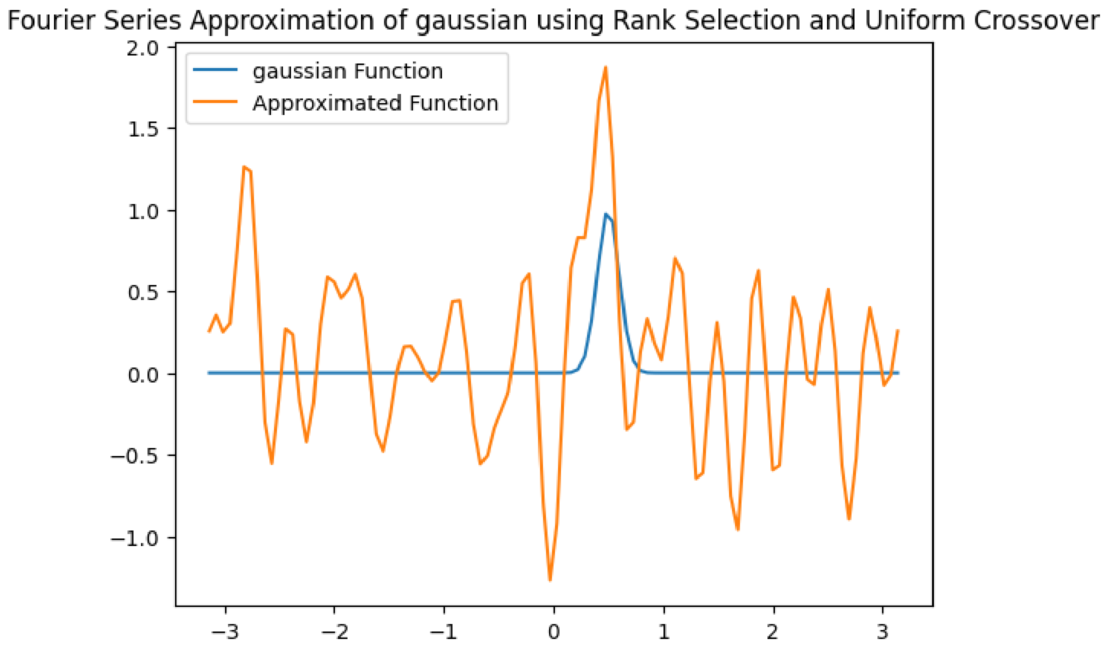
```
Running Genetic Algorithm for cubic Function:

Using Rank Selection and Uniform Crossover:
Generation 1: Best Fitness = -118.962438
Generation 2: Best Fitness = -118.789404
Generation 3: Best Fitness = -118.789404
Generation 4: Best Fitness = -118.535755
Generation 5: Best Fitness = -118.475312
Generation 6: Best Fitness = -118.284043
Generation 7: Best Fitness = -118.235865
Generation 8: Best Fitness = -118.143232
Generation 9: Best Fitness = -118.128798
Generation 10: Best Fitness = -118.040660
Generation 11: Best Fitness = -117.805107
Generation 12: Best Fitness = -117.659901
Generation 13: Best Fitness = -117.659901
Generation 14: Best Fitness = -117.632179
Generation 15: Best Fitness = -117.632179
Generation 16: Best Fitness = -117.542092
Generation 17: Best Fitness = -117.341457
Generation 18: Best Fitness = -117.341457
Generation 19: Best Fitness = -117.341457
Generation 20: Best Fitness = -117.260703
Generation 21: Best Fitness = -117.260703
Generation 22: Best Fitness = -117.228494
Generation 23: Best Fitness = -117.179960
Generation 24: Best Fitness = -117.104233
Generation 25: Best Fitness = -117.055114
Generation 26: Best Fitness = -117.055114
Generation 27: Best Fitness = -116.974870
Generation 28: Best Fitness = -116.831322
Generation 29: Best Fitness = -116.831322
Generation 30: Best Fitness = -116.798972
Generation 31: Best Fitness = -116.608304
Generation 32: Best Fitness = -116.575484
Generation 33: Best Fitness = -116.575484
Generation 34: Best Fitness = -116.571486
Generation 35: Best Fitness = -116.558764
Generation 36: Best Fitness = -116.494781
Generation 37: Best Fitness = -116.458384
Generation 38: Best Fitness = -116.363923
Generation 39: Best Fitness = -116.314388
Generation 40: Best Fitness = -116.297087
Generation 41: Best Fitness = -116.209019
Generation 42: Best Fitness = -116.162248
Generation 43: Best Fitness = -116.129568
Generation 44: Best Fitness = -116.075516
Generation 45: Best Fitness = -115.991857
Generation 46: Best Fitness = -115.958183
Generation 47: Best Fitness = -115.958183
Generation 48: Best Fitness = -115.886447
Generation 49: Best Fitness = -115.863219
Generation 50: Best Fitness = -115.805863
Generation 51: Best Fitness = -115.795624
Generation 52: Best Fitness = -115.708049
Generation 53: Best Fitness = -115.587126
Generation 54: Best Fitness = -115.555659
Generation 55: Best Fitness = -115.531673
Generation 56: Best Fitness = -115.413177
Generation 57: Best Fitness = -115.393857
Generation 58: Best Fitness = -115.393857
Generation 59: Best Fitness = -115.344271
Generation 60: Best Fitness = -115.301844
Generation 61: Best Fitness = -115.215488
Generation 62: Best Fitness = -115.134792
Generation 63: Best Fitness = -115.134408
Generation 64: Best Fitness = -115.061716
Generation 65: Best Fitness = -115.037236
Generation 66: Best Fitness = -114.840255
Generation 67: Best Fitness = -114.840255
Generation 68: Best Fitness = -114.840255
Generation 69: Best Fitness = -114.840255
Generation 70: Best Fitness = -114.784081
Generation 71: Best Fitness = -114.771361
Generation 72: Best Fitness = -114.642969
Generation 73: Best Fitness = -114.612650
Generation 74: Best Fitness = -114.583616
Generation 75: Best Fitness = -114.440434
Generation 76: Best Fitness = -114.440434
Generation 77: Best Fitness = -114.412287
Generation 78: Best Fitness = -114.412287
Generation 79: Best Fitness = -114.278497
Generation 80: Best Fitness = -114.278497
Generation 81: Best Fitness = -114.214165
Generation 82: Best Fitness = -114.214165
Generation 83: Best Fitness = -114.214165
Generation 84: Best Fitness = -114.158184
Generation 85: Best Fitness = -114.052078
Generation 86: Best Fitness = -114.052078
Generation 87: Best Fitness = -114.052078
Generation 88: Best Fitness = -113.957350
Generation 89: Best Fitness = -113.911109
Generation 90: Best Fitness = -113.832766
Generation 91: Best Fitness = -113.795437
Generation 92: Best Fitness = -113.765521
Generation 93: Best Fitness = -113.691574
Generation 94: Best Fitness = -113.621355
Generation 95: Best Fitness = -113.552746
Generation 96: Best Fitness = -113.511580
Generation 97: Best Fitness = -113.477612
Generation 98: Best Fitness = -113.384162
Generation 99: Best Fitness = -113.384162
Generation 100: Best Fitness = -113.268312
```



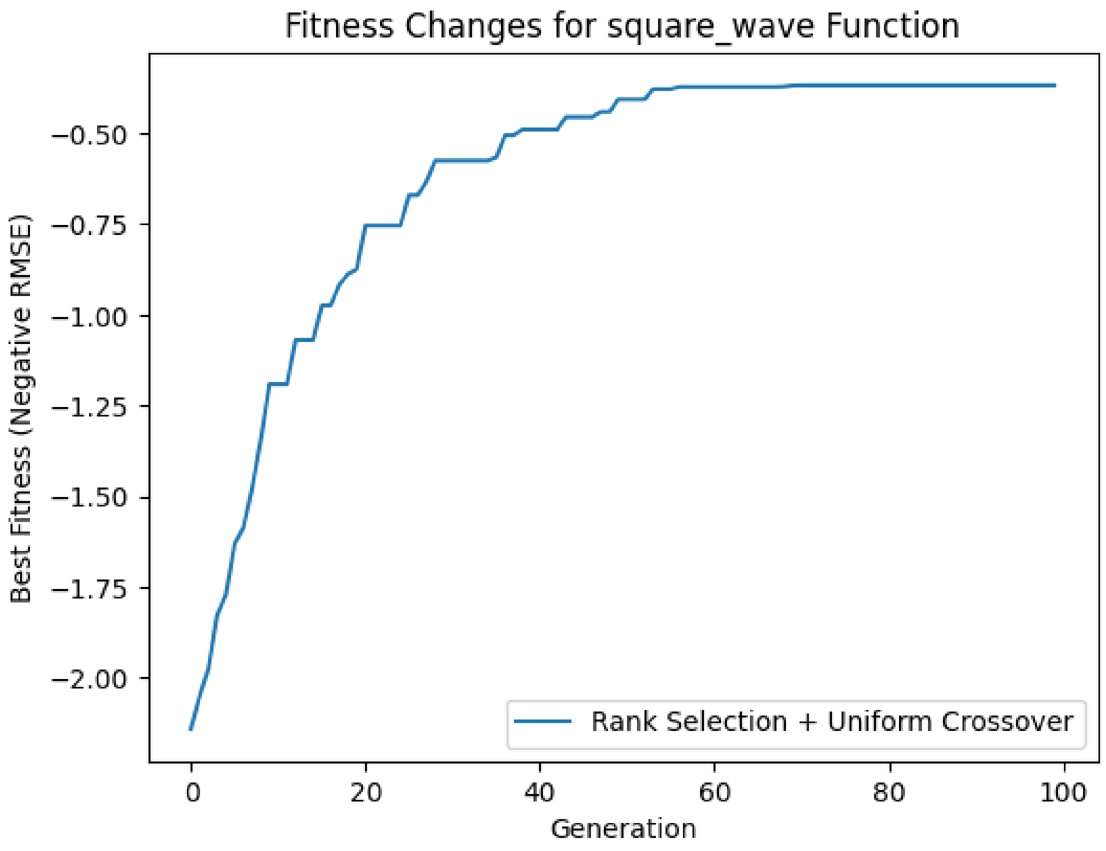


```
Running Genetic Algorithm for gaussian Function:

Using Rank Selection and Uniform Crossover:
Generation 1: Best Fitness = -2.166318
Generation 2: Best Fitness = -2.073978
Generation 3: Best Fitness = -2.035218
Generation 4: Best Fitness = -1.970763
Generation 5: Best Fitness = -1.900490
Generation 6: Best Fitness = -1.752875
Generation 7: Best Fitness = -1.666467
Generation 8: Best Fitness = -1.637633
Generation 9: Best Fitness = -1.404074
Generation 10: Best Fitness = -1.404074
Generation 11: Best Fitness = -1.403781
Generation 12: Best Fitness = -1.254838
Generation 13: Best Fitness = -1.249412
Generation 14: Best Fitness = -1.202490
Generation 15: Best Fitness = -1.123270
Generation 16: Best Fitness = -1.123270
Generation 17: Best Fitness = -1.047221
Generation 18: Best Fitness = -1.000369
Generation 19: Best Fitness = -1.000369
Generation 20: Best Fitness = -0.959196
Generation 21: Best Fitness = -0.906419
Generation 22: Best Fitness = -0.816255
Generation 23: Best Fitness = -0.816255
Generation 24: Best Fitness = -0.816255
Generation 25: Best Fitness = -0.816255
Generation 26: Best Fitness = -0.730245
Generation 27: Best Fitness = -0.730245
Generation 28: Best Fitness = -0.726615
Generation 29: Best Fitness = -0.655613
Generation 30: Best Fitness = -0.655613
Generation 31: Best Fitness = -0.617495
Generation 32: Best Fitness = -0.592219
Generation 33: Best Fitness = -0.552163
Generation 34: Best Fitness = -0.552163
Generation 35: Best Fitness = -0.552163
Generation 36: Best Fitness = -0.524214
Generation 37: Best Fitness = -0.509131
Generation 38: Best Fitness = -0.503146
Generation 39: Best Fitness = -0.503146
Generation 40: Best Fitness = -0.503146
Generation 41: Best Fitness = -0.458102
Generation 42: Best Fitness = -0.438960
Generation 43: Best Fitness = -0.438960
Generation 44: Best Fitness = -0.438960
Generation 45: Best Fitness = -0.438960
Generation 46: Best Fitness = -0.357927
Generation 47: Best Fitness = -0.357927
Generation 48: Best Fitness = -0.357927
Generation 49: Best Fitness = -0.357927
Generation 50: Best Fitness = -0.357927
Generation 51: Best Fitness = -0.357927
Generation 52: Best Fitness = -0.346453
Generation 53: Best Fitness = -0.346453
Generation 54: Best Fitness = -0.346453
Generation 55: Best Fitness = -0.346453
Generation 56: Best Fitness = -0.335800
Generation 57: Best Fitness = -0.319633
Generation 58: Best Fitness = -0.319633
Generation 59: Best Fitness = -0.319633
Generation 60: Best Fitness = -0.319633
Generation 61: Best Fitness = -0.319633
Generation 62: Best Fitness = -0.319633
Generation 63: Best Fitness = -0.319633
Generation 64: Best Fitness = -0.319633
Generation 65: Best Fitness = -0.319633
Generation 66: Best Fitness = -0.319633
Generation 67: Best Fitness = -0.319633
Generation 68: Best Fitness = -0.305822
Generation 69: Best Fitness = -0.305822
Generation 70: Best Fitness = -0.305822
Generation 71: Best Fitness = -0.305822
Generation 72: Best Fitness = -0.305822
Generation 73: Best Fitness = -0.305822
Generation 74: Best Fitness = -0.305822
Generation 75: Best Fitness = -0.303860
Generation 76: Best Fitness = -0.303860
Generation 77: Best Fitness = -0.303860
Generation 78: Best Fitness = -0.303860
Generation 79: Best Fitness = -0.303860
Generation 80: Best Fitness = -0.303860
Generation 81: Best Fitness = -0.303860
Generation 82: Best Fitness = -0.303860
Generation 83: Best Fitness = -0.303860
Generation 84: Best Fitness = -0.303860
Generation 85: Best Fitness = -0.303860
Generation 86: Best Fitness = -0.303860
Generation 87: Best Fitness = -0.303860
Generation 88: Best Fitness = -0.303860
Generation 89: Best Fitness = -0.303860
Generation 90: Best Fitness = -0.303860
Generation 91: Best Fitness = -0.303860
Generation 92: Best Fitness = -0.303860
Generation 93: Best Fitness = -0.303860
Generation 94: Best Fitness = -0.303860
Generation 95: Best Fitness = -0.303860
Generation 96: Best Fitness = -0.303860
Generation 97: Best Fitness = -0.303860
Generation 98: Best Fitness = -0.303860
Generation 99: Best Fitness = -0.303860
Generation 100: Best Fitness = -0.303860
```

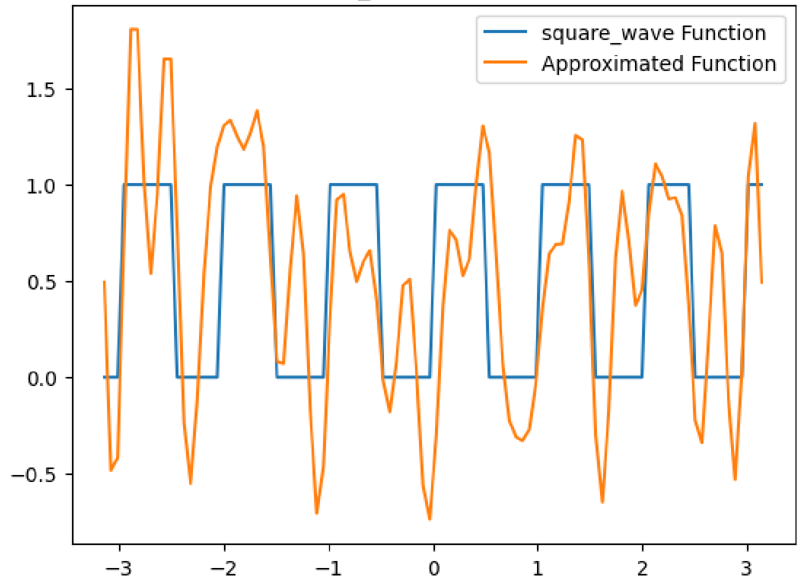

Fitness Changes for gaussian Function
Best Fitness (Negative RMSE)
Generation
Rank Selection + Uniform Crossover
−0.25
−0.50
−0.75
−1.00
−1.25
−1.50
−1.75
−2.00
−2.25
0
20
40
60
80
100


Fourier Series Approximation of gaussian using Rank Selection and Uniform Crossover
gaussian Function
Approximated Function
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−3
−2
−1
0
1
2
3

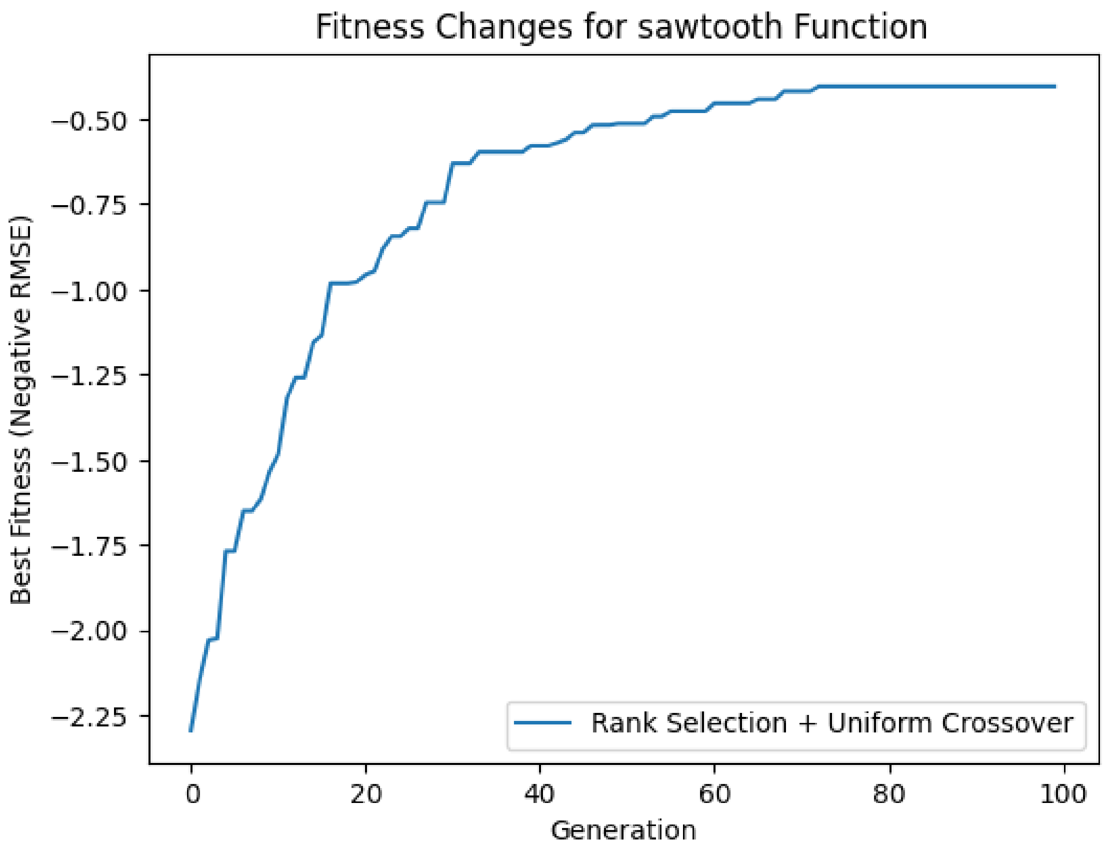
```
Running Genetic Algorithm for square_wave Function:

Using Rank Selection and Uniform Crossover:
Generation 1: Best Fitness = -2.139627
Generation 2: Best Fitness = -2.048991
Generation 3: Best Fitness = -1.973671
Generation 4: Best Fitness = -1.826232
Generation 5: Best Fitness = -1.770494
Generation 6: Best Fitness = -1.630911
Generation 7: Best Fitness = -1.585767
Generation 8: Best Fitness = -1.478873
Generation 9: Best Fitness = -1.344631
Generation 10: Best Fitness = -1.191176
Generation 11: Best Fitness = -1.191176
Generation 12: Best Fitness = -1.191176
Generation 13: Best Fitness = -1.069297
Generation 14: Best Fitness = -1.069297
Generation 15: Best Fitness = -1.069297
Generation 16: Best Fitness = -0.973956
Generation 17: Best Fitness = -0.973956
Generation 18: Best Fitness = -0.917084
Generation 19: Best Fitness = -0.887961
Generation 20: Best Fitness = -0.874784
Generation 21: Best Fitness = -0.753987
Generation 22: Best Fitness = -0.753987
Generation 23: Best Fitness = -0.753987
Generation 24: Best Fitness = -0.753987
Generation 25: Best Fitness = -0.753987
Generation 26: Best Fitness = -0.670204
Generation 27: Best Fitness = -0.670204
Generation 28: Best Fitness = -0.633212
Generation 29: Best Fitness = -0.575434
Generation 30: Best Fitness = -0.575434
Generation 31: Best Fitness = -0.575434
Generation 32: Best Fitness = -0.575434
Generation 33: Best Fitness = -0.575434
Generation 34: Best Fitness = -0.575434
Generation 35: Best Fitness = -0.575434
Generation 36: Best Fitness = -0.566754
Generation 37: Best Fitness = -0.505858
Generation 38: Best Fitness = -0.505858
Generation 39: Best Fitness = -0.489396
Generation 40: Best Fitness = -0.489396
Generation 41: Best Fitness = -0.489396
Generation 42: Best Fitness = -0.489396
Generation 43: Best Fitness = -0.489396
Generation 44: Best Fitness = -0.455398
Generation 45: Best Fitness = -0.455398
Generation 46: Best Fitness = -0.455398
Generation 47: Best Fitness = -0.455398
Generation 48: Best Fitness = -0.440659
Generation 49: Best Fitness = -0.440659
Generation 50: Best Fitness = -0.406586
Generation 51: Best Fitness = -0.406586
Generation 52: Best Fitness = -0.406586
Generation 53: Best Fitness = -0.406586
Generation 54: Best Fitness = -0.378218
Generation 55: Best Fitness = -0.378218
Generation 56: Best Fitness = -0.378218
Generation 57: Best Fitness = -0.372317
Generation 58: Best Fitness = -0.372317
Generation 59: Best Fitness = -0.372317
Generation 60: Best Fitness = -0.372317
Generation 61: Best Fitness = -0.372317
Generation 62: Best Fitness = -0.372317
Generation 63: Best Fitness = -0.372317
Generation 64: Best Fitness = -0.372317
Generation 65: Best Fitness = -0.372317
Generation 66: Best Fitness = -0.372317
Generation 67: Best Fitness = -0.372317
Generation 68: Best Fitness = -0.372317
Generation 69: Best Fitness = -0.371538
Generation 70: Best Fitness = -0.368080
Generation 71: Best Fitness = -0.367646
Generation 72: Best Fitness = -0.367646
Generation 73: Best Fitness = -0.367646
Generation 74: Best Fitness = -0.367646
Generation 75: Best Fitness = -0.367646
Generation 76: Best Fitness = -0.367646
Generation 77: Best Fitness = -0.367646
Generation 78: Best Fitness = -0.367646
Generation 79: Best Fitness = -0.367646
Generation 80: Best Fitness = -0.367646
Generation 81: Best Fitness = -0.367646
Generation 82: Best Fitness = -0.367646
Generation 83: Best Fitness = -0.367646
Generation 84: Best Fitness = -0.367646
Generation 85: Best Fitness = -0.367646
Generation 86: Best Fitness = -0.367646
Generation 87: Best Fitness = -0.367646
Generation 88: Best Fitness = -0.367646
Generation 89: Best Fitness = -0.367646
Generation 90: Best Fitness = -0.367646
Generation 91: Best Fitness = -0.367646
Generation 92: Best Fitness = -0.367646
Generation 93: Best Fitness = -0.367646
Generation 94: Best Fitness = -0.367646
Generation 95: Best Fitness = -0.367646
Generation 96: Best Fitness = -0.367646
Generation 97: Best Fitness = -0.367646
Generation 98: Best Fitness = -0.367646
Generation 99: Best Fitness = -0.367646
Generation 100: Best Fitness = -0.367646
```

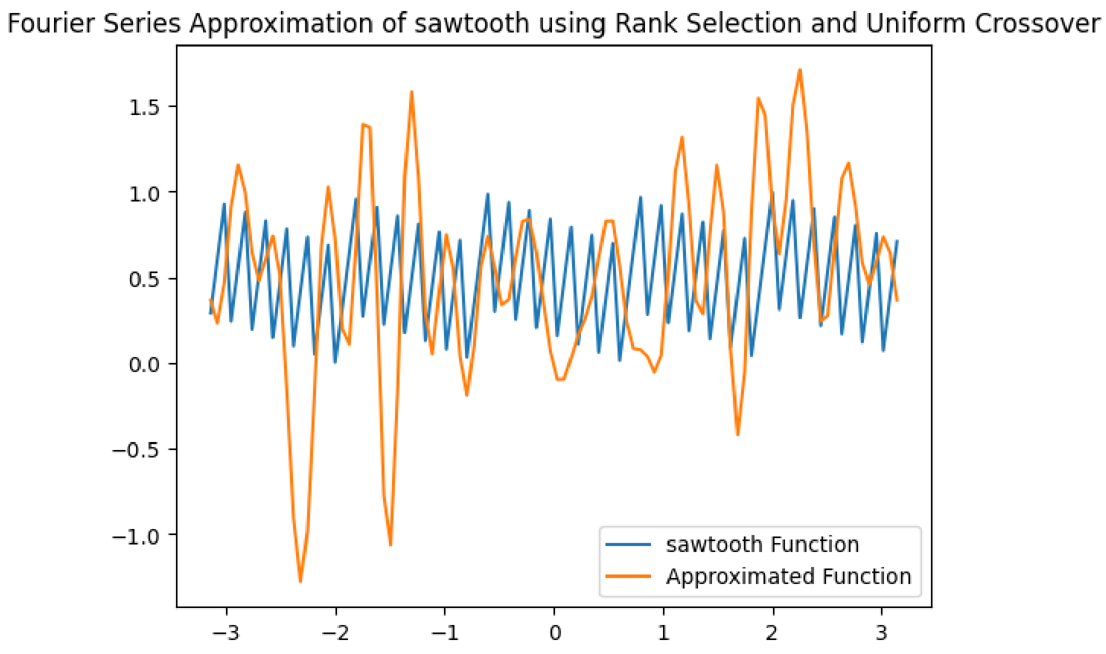


Fourier Series Approximation of square_wave using Rank Selection and Uniform Crossover

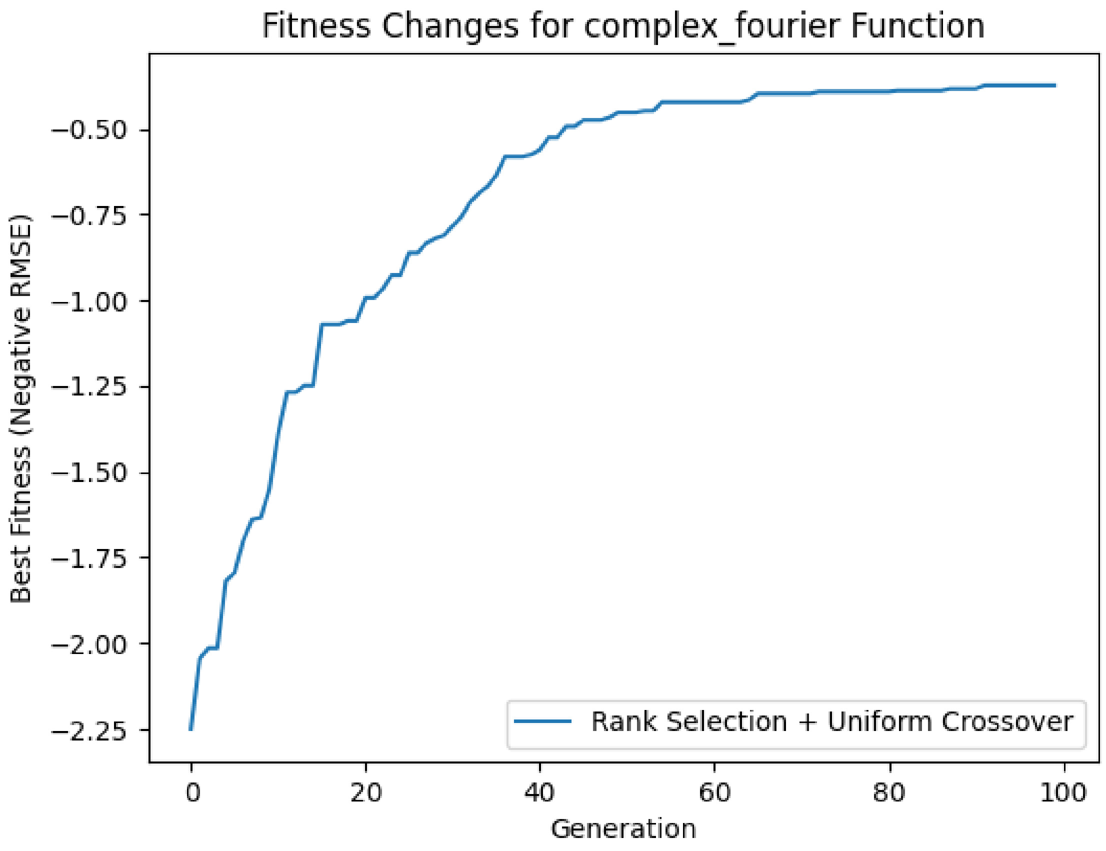


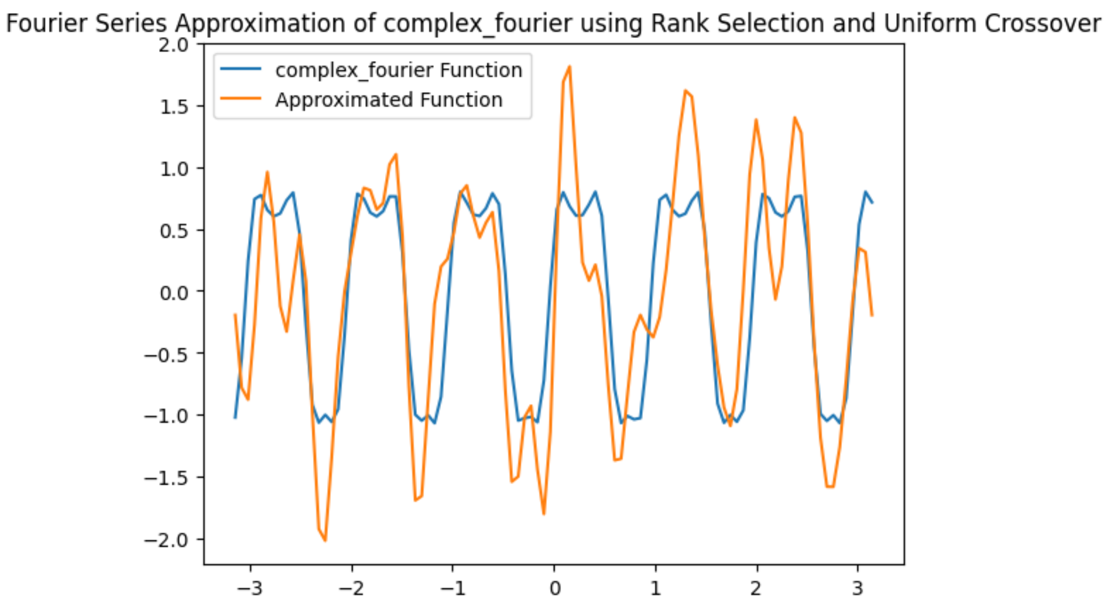
```
Running Genetic Algorithm for sawtooth Function:

Using Rank Selection and Uniform Crossover:
Generation 1: Best Fitness = -2.293825
Generation 2: Best Fitness = -2.145479
Generation 3: Best Fitness = -2.029532
Generation 4: Best Fitness = -2.024268
Generation 5: Best Fitness = -1.767971
Generation 6: Best Fitness = -1.767971
Generation 7: Best Fitness = -1.650312
Generation 8: Best Fitness = -1.650312
Generation 9: Best Fitness = -1.616343
Generation 10: Best Fitness = -1.535508
Generation 11: Best Fitness = -1.483549
Generation 12: Best Fitness = -1.319730
Generation 13: Best Fitness = -1.258903
Generation 14: Best Fitness = -1.258903
Generation 15: Best Fitness = -1.156769
Generation 16: Best Fitness = -1.133482
Generation 17: Best Fitness = -0.982572
Generation 18: Best Fitness = -0.982572
Generation 19: Best Fitness = -0.982572
Generation 20: Best Fitness = -0.978279
Generation 21: Best Fitness = -0.957518
Generation 22: Best Fitness = -0.947512
Generation 23: Best Fitness = -0.880973
Generation 24: Best Fitness = -0.844111
Generation 25: Best Fitness = -0.844111
Generation 26: Best Fitness = -0.821080
Generation 27: Best Fitness = -0.821080
Generation 28: Best Fitness = -0.744899
Generation 29: Best Fitness = -0.744899
Generation 30: Best Fitness = -0.744899
Generation 31: Best Fitness = -0.630084
Generation 32: Best Fitness = -0.630084
Generation 33: Best Fitness = -0.630084
Generation 34: Best Fitness = -0.596415
Generation 35: Best Fitness = -0.596415
Generation 36: Best Fitness = -0.596415
Generation 37: Best Fitness = -0.596415
Generation 38: Best Fitness = -0.596415
Generation 39: Best Fitness = -0.596415
Generation 40: Best Fitness = -0.578503
Generation 41: Best Fitness = -0.578503
Generation 42: Best Fitness = -0.578503
Generation 43: Best Fitness = -0.569931
Generation 44: Best Fitness = -0.560292
Generation 45: Best Fitness = -0.540046
Generation 46: Best Fitness = -0.540046
Generation 47: Best Fitness = -0.518363
Generation 48: Best Fitness = -0.517889
Generation 49: Best Fitness = -0.517889
Generation 50: Best Fitness = -0.513468
Generation 51: Best Fitness = -0.513468
Generation 52: Best Fitness = -0.513468
Generation 53: Best Fitness = -0.513468
Generation 54: Best Fitness = -0.492660
Generation 55: Best Fitness = -0.492429
Generation 56: Best Fitness = -0.477172
Generation 57: Best Fitness = -0.477172
Generation 58: Best Fitness = -0.477172
Generation 59: Best Fitness = -0.477172
Generation 60: Best Fitness = -0.477172
Generation 61: Best Fitness = -0.453943
Generation 62: Best Fitness = -0.453943
Generation 63: Best Fitness = -0.453943
Generation 64: Best Fitness = -0.453943
Generation 65: Best Fitness = -0.453943
Generation 66: Best Fitness = -0.441826
Generation 67: Best Fitness = -0.441826
Generation 68: Best Fitness = -0.441826
Generation 69: Best Fitness = -0.418523
Generation 70: Best Fitness = -0.418523
Generation 71: Best Fitness = -0.418523
Generation 72: Best Fitness = -0.418523
Generation 73: Best Fitness = -0.403901
Generation 74: Best Fitness = -0.403901
Generation 75: Best Fitness = -0.403901
Generation 76: Best Fitness = -0.403901
Generation 77: Best Fitness = -0.403901
Generation 78: Best Fitness = -0.403901
Generation 79: Best Fitness = -0.403901
Generation 80: Best Fitness = -0.403901
Generation 81: Best Fitness = -0.403901
Generation 82: Best Fitness = -0.403901
Generation 83: Best Fitness = -0.403901
Generation 84: Best Fitness = -0.403901
Generation 85: Best Fitness = -0.403901
Generation 86: Best Fitness = -0.403901
Generation 87: Best Fitness = -0.403901
Generation 88: Best Fitness = -0.403901
Generation 89: Best Fitness = -0.403901
Generation 90: Best Fitness = -0.403901
Generation 91: Best Fitness = -0.403901
Generation 92: Best Fitness = -0.403901
Generation 93: Best Fitness = -0.403901
Generation 94: Best Fitness = -0.403901
Generation 95: Best Fitness = -0.403901
Generation 96: Best Fitness = -0.403901
Generation 97: Best Fitness = -0.403901
Generation 98: Best Fitness = -0.403901
Generation 99: Best Fitness = -0.403901
Generation 100: Best Fitness = -0.403901
```


Fitness Changes for sawtooth Function
Best Fitness (Negative RMSE)
Generation
Rank Selection + Uniform Crossover
−0.50
−0.75
−1.00
−1.25
−1.50
−1.75
−2.00
−2.25
0
20
40
60
80
100

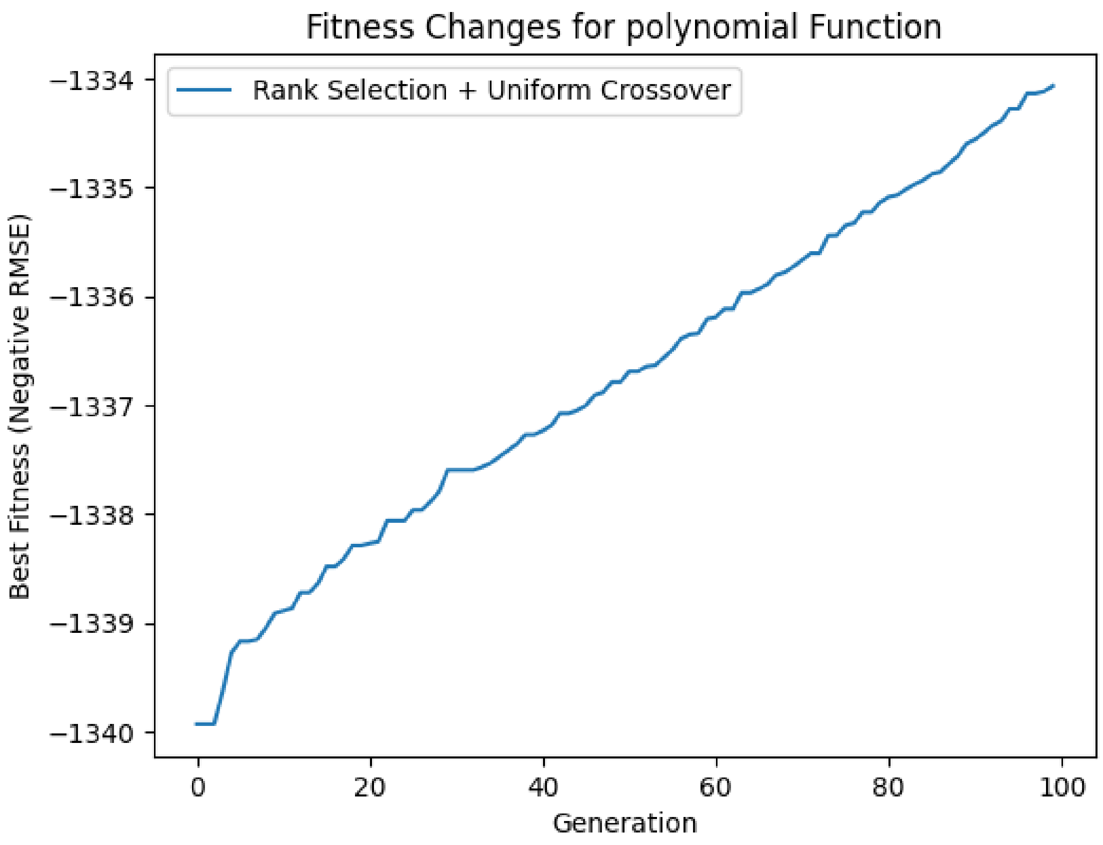

Fourier Series Approximation of sawtooth using Rank Selection and Uniform Crossover
sawtooth Function
Approximated Function
1.5
1.0
0.5
0.0
−0.5
−1.0
−3
−2
−1
0
1
2
3

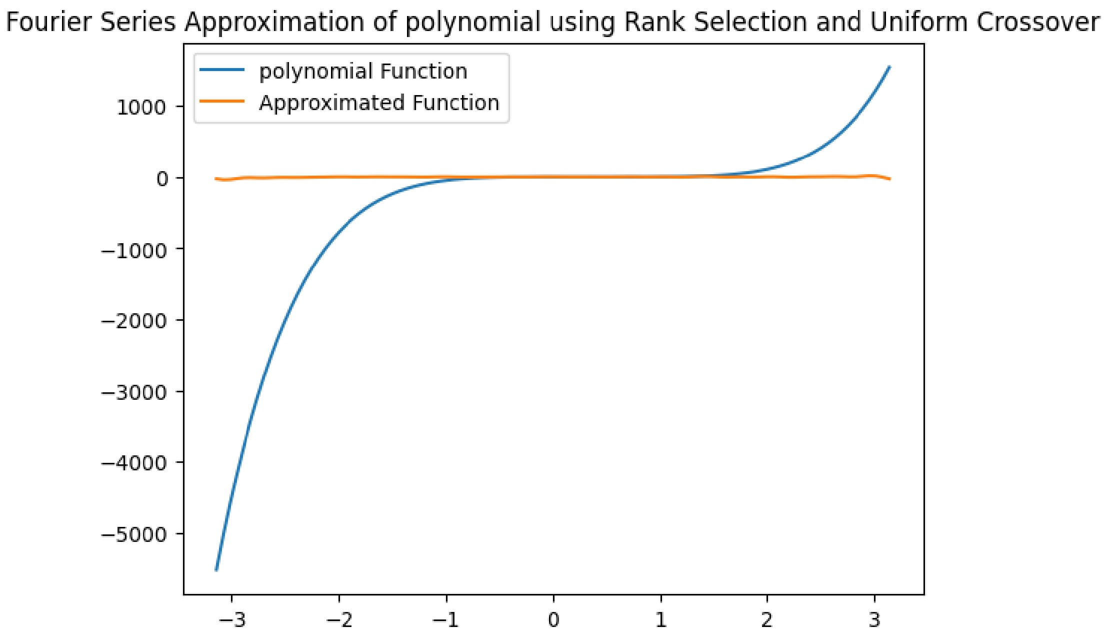
```
Running Genetic Algorithm for complex_fourier Function:

Using Rank Selection and Uniform Crossover:
Generation 1: Best Fitness = -2.250599
Generation 2: Best Fitness = -2.046470
Generation 3: Best Fitness = -2.015284
Generation 4: Best Fitness = -2.015284
Generation 5: Best Fitness = -1.819544
Generation 6: Best Fitness = -1.794803
Generation 7: Best Fitness = -1.700707
Generation 8: Best Fitness = -1.640340
Generation 9: Best Fitness = -1.635033
Generation 10: Best Fitness = -1.551468
Generation 11: Best Fitness = -1.390342
Generation 12: Best Fitness = -1.269427
Generation 13: Best Fitness = -1.269427
Generation 14: Best Fitness = -1.250073
Generation 15: Best Fitness = -1.250073
Generation 16: Best Fitness = -1.071425
Generation 17: Best Fitness = -1.071425
Generation 18: Best Fitness = -1.071425
Generation 19: Best Fitness = -1.060800
Generation 20: Best Fitness = -1.060800
Generation 21: Best Fitness = -0.993630
Generation 22: Best Fitness = -0.993630
Generation 23: Best Fitness = -0.967956
Generation 24: Best Fitness = -0.927698
Generation 25: Best Fitness = -0.927698
Generation 26: Best Fitness = -0.862527
Generation 27: Best Fitness = -0.862527
Generation 28: Best Fitness = -0.833850
Generation 29: Best Fitness = -0.820641
Generation 30: Best Fitness = -0.811372
Generation 31: Best Fitness = -0.784165
Generation 32: Best Fitness = -0.758254
Generation 33: Best Fitness = -0.714806
Generation 34: Best Fitness = -0.689122
Generation 35: Best Fitness = -0.668662
Generation 36: Best Fitness = -0.636428
Generation 37: Best Fitness = -0.582018
Generation 38: Best Fitness = -0.582018
Generation 39: Best Fitness = -0.582018
Generation 40: Best Fitness = -0.576218
Generation 41: Best Fitness = -0.562435
Generation 42: Best Fitness = -0.526313
Generation 43: Best Fitness = -0.526313
Generation 44: Best Fitness = -0.493950
Generation 45: Best Fitness = -0.493950
Generation 46: Best Fitness = -0.475657
Generation 47: Best Fitness = -0.475657
Generation 48: Best Fitness = -0.475657
Generation 49: Best Fitness = -0.467372
Generation 50: Best Fitness = -0.452970
Generation 51: Best Fitness = -0.452970
Generation 52: Best Fitness = -0.452970
Generation 53: Best Fitness = -0.448434
Generation 54: Best Fitness = -0.448434
Generation 55: Best Fitness = -0.423218
Generation 56: Best Fitness = -0.423218
Generation 57: Best Fitness = -0.423218
Generation 58: Best Fitness = -0.423218
Generation 59: Best Fitness = -0.423218
Generation 60: Best Fitness = -0.423218
Generation 61: Best Fitness = -0.423218
Generation 62: Best Fitness = -0.423218
Generation 63: Best Fitness = -0.423218
Generation 64: Best Fitness = -0.423218
Generation 65: Best Fitness = -0.416249
Generation 66: Best Fitness = -0.398162
Generation 67: Best Fitness = -0.398162
Generation 68: Best Fitness = -0.398162
Generation 69: Best Fitness = -0.398162
Generation 70: Best Fitness = -0.398162
Generation 71: Best Fitness = -0.398162
Generation 72: Best Fitness = -0.398162
Generation 73: Best Fitness = -0.392679
Generation 74: Best Fitness = -0.392679
Generation 75: Best Fitness = -0.392679
Generation 76: Best Fitness = -0.392679
Generation 77: Best Fitness = -0.392679
Generation 78: Best Fitness = -0.392679
Generation 79: Best Fitness = -0.392679
Generation 80: Best Fitness = -0.392679
Generation 81: Best Fitness = -0.392679
Generation 82: Best Fitness = -0.389645
Generation 83: Best Fitness = -0.389645
Generation 84: Best Fitness = -0.389645
Generation 85: Best Fitness = -0.389645
Generation 86: Best Fitness = -0.389645
Generation 87: Best Fitness = -0.389645
Generation 88: Best Fitness = -0.384069
Generation 89: Best Fitness = -0.384069
Generation 90: Best Fitness = -0.384069
Generation 91: Best Fitness = -0.384069
Generation 92: Best Fitness = -0.373794
Generation 93: Best Fitness = -0.373794
Generation 94: Best Fitness = -0.373794
Generation 95: Best Fitness = -0.373794
Generation 96: Best Fitness = -0.373794
Generation 97: Best Fitness = -0.373794
Generation 98: Best Fitness = -0.373794
Generation 99: Best Fitness = -0.373794
Generation 100: Best Fitness = -0.373794
```





```
Running Genetic Algorithm for polynomial Function:

Using Rank Selection and Uniform Crossover:
Generation 1: Best Fitness = -1339.927381
Generation 2: Best Fitness = -1339.927381
Generation 3: Best Fitness = -1339.927381
Generation 4: Best Fitness = -1339.626455
Generation 5: Best Fitness = -1339.276134
Generation 6: Best Fitness = -1339.167681
Generation 7: Best Fitness = -1339.167681
Generation 8: Best Fitness = -1339.149893
Generation 9: Best Fitness = -1339.043476
Generation 10: Best Fitness = -1338.911859
Generation 11: Best Fitness = -1338.888808
Generation 12: Best Fitness = -1338.864791
Generation 13: Best Fitness = -1338.721903
Generation 14: Best Fitness = -1338.721903
Generation 15: Best Fitness = -1338.638947
Generation 16: Best Fitness = -1338.483358
Generation 17: Best Fitness = -1338.483358
Generation 18: Best Fitness = -1338.411296
Generation 19: Best Fitness = -1338.289822
Generation 20: Best Fitness = -1338.289822
Generation 21: Best Fitness = -1338.269549
Generation 22: Best Fitness = -1338.252558
Generation 23: Best Fitness = -1338.062594
Generation 24: Best Fitness = -1338.062594
Generation 25: Best Fitness = -1338.062594
Generation 26: Best Fitness = -1337.963538
Generation 27: Best Fitness = -1337.963538
Generation 28: Best Fitness = -1337.888889
Generation 29: Best Fitness = -1337.797639
Generation 30: Best Fitness = -1337.597851
Generation 31: Best Fitness = -1337.597851
Generation 32: Best Fitness = -1337.597851
Generation 33: Best Fitness = -1337.597851
Generation 34: Best Fitness = -1337.568703
Generation 35: Best Fitness = -1337.531125
Generation 36: Best Fitness = -1337.473839
Generation 37: Best Fitness = -1337.416041
Generation 38: Best Fitness = -1337.358191
Generation 39: Best Fitness = -1337.272740
Generation 40: Best Fitness = -1337.272740
Generation 41: Best Fitness = -1337.235927
Generation 42: Best Fitness = -1337.187309
Generation 43: Best Fitness = -1337.076702
Generation 44: Best Fitness = -1337.076702
Generation 45: Best Fitness = -1337.048644
Generation 46: Best Fitness = -1337.003635
Generation 47: Best Fitness = -1336.909636
Generation 48: Best Fitness = -1336.882508
Generation 49: Best Fitness = -1336.788847
Generation 50: Best Fitness = -1336.788847
Generation 51: Best Fitness = -1336.690259
Generation 52: Best Fitness = -1336.690259
Generation 53: Best Fitness = -1336.646471
Generation 54: Best Fitness = -1336.636233
Generation 55: Best Fitness = -1336.564299
Generation 56: Best Fitness = -1336.493200
Generation 57: Best Fitness = -1336.390609
Generation 58: Best Fitness = -1336.353432
Generation 59: Best Fitness = -1336.342618
Generation 60: Best Fitness = -1336.210219
Generation 61: Best Fitness = -1336.193975
Generation 62: Best Fitness = -1336.117606
Generation 63: Best Fitness = -1336.117606
Generation 64: Best Fitness = -1335.968776
Generation 65: Best Fitness = -1335.968776
Generation 66: Best Fitness = -1335.934362
Generation 67: Best Fitness = -1335.891231
Generation 68: Best Fitness = -1335.805659
Generation 69: Best Fitness = -1335.781485
Generation 70: Best Fitness = -1335.728589
Generation 71: Best Fitness = -1335.669633
Generation 72: Best Fitness = -1335.607600
Generation 73: Best Fitness = -1335.607600
Generation 74: Best Fitness = -1335.445820
Generation 75: Best Fitness = -1335.441977
Generation 76: Best Fitness = -1335.351541
Generation 77: Best Fitness = -1335.330853
Generation 78: Best Fitness = -1335.229723
Generation 79: Best Fitness = -1335.229723
Generation 80: Best Fitness = -1335.139881
Generation 81: Best Fitness = -1335.090722
Generation 82: Best Fitness = -1335.072972
Generation 83: Best Fitness = -1335.019911
Generation 84: Best Fitness = -1334.975066
Generation 85: Best Fitness = -1334.936688
Generation 86: Best Fitness = -1334.879735
Generation 87: Best Fitness = -1334.859769
Generation 88: Best Fitness = -1334.784745
Generation 89: Best Fitness = -1334.715475
Generation 90: Best Fitness = -1334.601257
Generation 91: Best Fitness = -1334.562700
Generation 92: Best Fitness = -1334.502326
Generation 93: Best Fitness = -1334.434984
Generation 94: Best Fitness = -1334.393083
Generation 95: Best Fitness = -1334.281785
Generation 96: Best Fitness = -1334.281785
Generation 97: Best Fitness = -1334.140651
Generation 98: Best Fitness = -1334.140651
Generation 99: Best Fitness = -1334.119894
Generation 100: Best Fitness = -1334.071194
```


Fitness Changes for polynomial Function
Rank Selection + Uniform Crossover
Best Fitness (Negative RMSE)
Generation
−1334
−1335
−1336
−1337
−1338
−1339
−1340
0
20
40
60
80
100


Fourier Series Approximation of polynomial using Rank Selection and Uniform Crossover
polynomial Function
Approximated Function
1000
0
−1000
−2000
−3000
−4000
−5000
−3
−2
−1
0
1
2
3

2.1 Questions

2.1.1 Question 1:

Considering that the initial value of each coefficient is between -1 and 1 with eight decimal places, the number of possible states for a coefficient becomes:

$(1-(-1)) \times 10^8 = 2 \times 10^8 = 200{,}000{,}000$

Since there are 41 coefficients, the total state space is:

$(200{,}000{,}000)^{41}$

$\log_{10}(\text{State space}) = 41 \times \log_{10}(200{,}000{,}000)$
$\approx 41 \times 8.30103$
≈ 340.342

State space $\approx 10^{340.342}$

The state space is astronomically large, approximately 10^{340}. This demonstrates the infeasibility of brute force approaches and the necessity of optimization techniques like genetic algorithms to explore this vast search space efficiently.

2.1.2 Question 2:

Ideas that can help the genetic algorithm converge faster:

1. Elitism:

- **Explanation**: Elitism ensures that the best individuals from the current generation are carried over to the next generation without modification. This prevents the loss of high-quality solutions due to random mutations or crossover.
- **Implementation**: At the end of each generation, we will identify the top `k` individuals (e.g., the top 5% of the population) based on fitness and directly we'll copy them into the next generation. The remaining population can be filled using the usual genetic operations (selection, crossover, and mutation).
- **Benefit**: By preserving the best solutions, the algorithm avoids "forgetting" good solutions and ensures steady progress toward convergence.

2. Adaptive Mutation Rate:

- **Explanation**: Instead of using a fixed mutation rate, we can adapt the mutation rate dynamically based on the diversity of the population or the progress of the algorithm. For example:
  - We use a **higher mutation rate** in the early generations to explore the search space more broadly.
  - Gradually we reduce the mutation rate in later generations to fine-tune the solutions.
- **Implementation**: The mutation rate can be adjusted based on the generation number or the fitness improvement rate. For instance: `python     mutationRate = initialRate * (1 - (currentGeneration / totalGenerations))`
- **Benefit**: This approach balances exploration and exploitation, allowing the algorithm to explore diverse solutions early on and focus on refining the best solutions later.

2.1.3 Question 3:

Two common strategies for selecting the next generation:

1. Roulette Wheel Selection:

- **Explanation**: This is a probabilistic selection method where individuals are selected based on their fitness values. Each individual is assigned a probability proportional to its fitness, and a "roulette wheel" is spun to determine which individual is selected.
- **How It Works**:
  1. Calculating the fitness of all individuals in the population.
  2. Normalizing the fitness values to create a probability distribution.
  3. Using a random number to select an individual based on the probabilities.
- **Advantages**:
  - Individuals with higher fitness have a greater chance of being selected, promoting the survival of the fittest.
  - Maintains diversity by giving all individuals a chance to be selected.
- **Disadvantages**:
  - Can lead to premature convergence if one individual dominates the population (fitness proportionate bias).

2. Tournament Selection:

- **Explanation**: In this method, a subset of individuals is randomly chosen from the population, and the fittest individual from this subset is selected. This process is repeated to fill the next generation.
- **How It Works**:
  1. Randomly selecting a group of individuals (tournament size).
  2. Comparing their fitness values.
  3. Selecting the individual with the highest fitness from the group.
- **Advantages**:
  - Simple to implement and computationally efficient.
  - Allows control over selection pressure by adjusting the tournament size (larger tournaments increase selection pressure).
- **Disadvantages**:
  - Can reduce diversity if the tournament size is too large, as weaker individuals are less likely to be selected.

2.1.4 Question 4:

Methods to prevent premature convergence and how they increase genetic diversity:

1. Diversity-Preserving Selection (e.g., Fitness Sharing):

- **Explanation**: Fitness sharing reduces the fitness of individuals that are too similar to others in the population. This encourages the algorithm to maintain diversity by promoting individuals that are different from the majority.
- **How It Works**:
  1. Calculating the similarity between individuals based on their genetic makeup.

2. Reducing the fitness of individuals that are too similar to others by dividing their fitness by a "sharing factor."
3. Selecting individuals for the next generation based on the adjusted fitness values.

- **How It Increases Diversity**:
  - By penalizing similar individuals, the algorithm discourages the population from converging to a single solution too quickly.
  - It promotes exploration of different areas of the search space, increasing the chances of finding the global optimum.

2. Adaptive Mutation Rate:

- **Explanation**: Instead of using a fixed mutation rate, the mutation rate is dynamically adjusted based on the diversity of the population or the progress of the algorithm.
- **How It Works**:
  1. Using a **higher mutation rate** when the population becomes too homogeneous (low diversity) to introduce new genetic material.
  2. Gradually reducing the mutation rate as the population approaches convergence to fine-tune the solutions.
- **How It Increases Diversity**:
  - A higher mutation rate introduces more random changes in the population, preventing individuals from becoming too similar.
  - This helps the algorithm escape local optima and explore new areas of the search space.

2.1.5 Question 5:

The **R-squared (R^2)** function, also known as the **coefficient of determination**, is a statistical measure that indicates how well a regression model fits the data. It quantifies the proportion of the variance in the dependent variable that is predictable from the independent variables.

The formula for R-squared is:

`R² = 1 - (SS_res / SS_tot)`

Where: - **SS__res** (Residual Sum of Squares): The sum of squared differences between the observed values and the predicted values. `SS_res = Σ(y_i - ŷ_i)²` - **SS__tot** (Total Sum of Squares): The sum of squared differences between the observed values and their mean. `SS_tot = Σ(y_i - ȳ)²` - **y__i**: Observed values. - **ŷ__i**: Predicted values from the model. - **ȳ**: Mean of the observed values.

How R-squared Works:

- **$R^2 = 1$**: Indicates a perfect fit, where the model explains all the variance in the data.
- **$R^2 = 0$**: Indicates that the model does not explain any of the variance in the data.
- **$R^2 < 0$**: Indicates that the model performs worse than a simple horizontal line at the mean of the observed values.

In this problem, the goal is to approximate a target function (e.g., Fourier series coefficients) using a genetic algorithm. The fitness function currently uses **Root Mean Squared Error (RMSE)** to evaluate how well the predicted function matches the target function. While R-squared could theoretically be used as an alternative, it may not be the most suitable choice for the following reasons:

1. **Interpretability**:
    - R-squared is more commonly used in regression problems to evaluate the goodness of fit of a model. In this problem, the focus is on minimizing the error (e.g., RMSE) rather than explaining variance.
2. **Sensitivity to Data Distribution**:
    - R-squared depends on the variance of the observed data. If the target function has a low variance, R-squared may give misleadingly high values, even if the approximation is poor.
3. **Optimization Challenges**:
    - Genetic algorithms work by maximizing or minimizing a fitness function. Since R-squared is bounded between 0 and 1 (or less than 0 for poor fits), it may not provide enough granularity for the algorithm to differentiate between solutions, especially in early generations.

While R-squared is a useful metric in regression problems, it is less suitable for this problem. The current use of RMSE as the fitness function is more appropriate because it directly measures the error between the predicted and target functions, providing a clear objective for the genetic algorithm to minimize.

3 Minimax Algorithm

[7]:
```
import random
import numpy as np
from math import inf
import time
import pygame
```

```
pygame 2.6.1 (SDL 2.28.4, Python 3.13.2)
Hello from the pygame community. https://www.pygame.org/contribute.html
```

[8]:
```
class PentagoGame:
    def __init__(self, ui=False, print=False, depth=2):
        self.board = np.zeros((6, 6), dtype=int)
        self.current_player = 1
        self.ui = ui
        self.depth = depth
        self.nodes_visited = 0
        self.game_over = False
        self.result = None
        self.selected_block = None
        self.move_stage = 0  # 0: place piece, 1: select block, 2: rotate
        self.temp_piece = None
        self.print = print

        self.WIN_SCORE = 100000 # For evaluation
        self.FOUR_SCORE = 1000
        self.THREE_SCORE = 100
        self.TWO_SCORE = 10

        if ui:
            pygame.font.init()
            self.screen = pygame.display.set_mode((800, 600))
            pygame.display.set_caption("Pygame Board")
            # self.font = pygame.font.SysFont("Arial", 20)
            self.show_buttons = False
            self.buttons = {
                "rotate_cw": pygame.Rect(650, 200, 100, 50),
                "rotate_ccw": pygame.Rect(650, 300, 100, 50),
            }
            self.setup_controls()
            self.draw_board()

    def setup_controls(self):
        if self.show_buttons:
            pygame.draw.rect(self.screen, (144, 238, 144), self.
↪buttons["rotate_cw"])   # Light Green
            pygame.draw.rect(self.screen, (173, 216, 230), self.
↪buttons["rotate_ccw"])  # Light Blue

            self.screen.draw_text("CLOCKWISE", self.buttons["rotate_cw"].center)
            self.screen.draw_text("COUNTER-CLOCKWISE", self.
↪buttons["rotate_ccw"].center)



    def hide_rotation_buttons(self):
        self.show_buttons = False

    def show_rotation_buttons(self):
        self.show_buttons = True

    def copy_board(self, board):
        return np.copy(board)

    def rotate_block(self, board, block, direction):
        row_start = (block // 2) * 3
        col_start = (block % 2) * 3
        sub = board[row_start : row_start + 3, col_start : col_start + 3]
        rotated = np.rot90(sub, 3 if direction == "cw" else 1)
        board[row_start : row_start + 3, col_start : col_start + 3] = rotated

    def get_possible_moves(self, board, player):
        moves = []
        for i in range(6):
            for j in range(6):
                if board[i][j] == 0:
                    for block in range(4):
                        for dir in ["cw", "ccw"]:
                            moves.append((i, j, block, dir))
        return moves

    def apply_move(self, board, move, player):
        new_board = self.copy_board(board)
        row, col, block, direction = move
        if new_board[row][col] != 0:
            return None
        new_board[row][col] = player
        self.rotate_block(new_board, block, direction)
        return new_board

    def check_winner(self, board):
        for i in range(6):
            for j in range(6):
                if board[i][j] == 0:
                    continue

                # Horizontal
                if j <= 1 and np.all(board[i, j : j + 5] == board[i][j]):
                    return board[i][j]

                # Vertical
                if i <= 1 and np.all(board[i : i + 5, j] == board[i][j]):
                    return board[i][j]

                # Diagonal
                if (
                    i <= 1
                    and j <= 1
                    and all(board[i + k][j + k] == board[i][j] for k in␣
↪range(5))
                ):
                    return board[i][j]

                # Anti-diagonal
                if (
                    i <= 1
                    and j >= 4
                    and all(board[i + k][j - k] == board[i][j] for k in␣
↪range(5))
                ):
                    return board[i][j]
        if np.all(board != 0):
            return 0
        return None

    def evaluate_line(self, line, player):
        score = 0
        opponent = -player
        player_pieces = np.count_nonzero(line == player)
        opponent_pieces = np.count_nonzero(line == opponent)
        empty_cells = len(line) - player_pieces - opponent_pieces

        if player_pieces == 5:
            score += self.WIN_SCORE
        elif player_pieces == 4 and empty_cells == 1:
            score += self.FOUR_SCORE
        elif player_pieces == 3 and empty_cells == 2:
            score += self.THREE_SCORE
        elif player_pieces == 2 and empty_cells == 3:
            score += self.TWO_SCORE

        if opponent_pieces == 5:
             score -= self.WIN_SCORE * 0.9
        elif opponent_pieces == 4 and empty_cells == 1:
            score -= self.FOUR_SCORE * 0.9
        elif opponent_pieces == 3 and empty_cells == 2:
            score -= self.THREE_SCORE * 0.9
        elif opponent_pieces == 2 and empty_cells == 3:
            score -= self.TWO_SCORE * 0.9

        return score

    def evaluate_board(self, board, player):
        score = 0
        for i in range(6):
            for j in range(2):
                score += self.evaluate_line(board[i, j:j+5], player) # Rows
                score += self.evaluate_line(board[j:j+5, i], player) # Columns

        # Diagonals (top-left to bottom-right)
        for i in range(2):
            for j in range(2):
                diag = [board[i+k][j+k] for k in range(5)]
                score += self.evaluate_line(np.array(diag), player)

        # Diagonals (top-right to bottom-left)
        for i in range(2):
            for j in range(4, 6):
                anti_diag = [board[i+k][j-k] for k in range(5)]
                score += self.evaluate_line(np.array(anti_diag), player)

        # Small bonus for center control
        center_mask = np.zeros((6, 6), dtype=bool)
        center_mask[1:5, 1:5] = True
        score += np.sum(board[center_mask] == player)
        score -= np.sum(board[center_mask] == -player)

        return score

    def minimax(self, board, depth, alpha, beta, maximizing_player):
        self.nodes_visited += 1
        winner = self.check_winner(board)

        if winner is not None:
            if winner == -1: return self.WIN_SCORE
            if winner == 1: return -self.WIN_SCORE
            if winner == 0: return 0

        if depth == 0:
            return self.evaluate_board(board, -1)

        possible_moves = self.get_possible_moves(board, -1 if maximizing_player␣
↪else 1)
        random.shuffle(possible_moves)

        if maximizing_player:
            max_eval = -inf
            for move in possible_moves:
                child_board = self.apply_move(board, move, -1)
                if child_board is None: continue
                eval_score = self.minimax(child_board, depth - 1, alpha, beta,␣
↪False)
                max_eval = max(max_eval, eval_score)
                alpha = max(alpha, eval_score)
                if beta <= alpha:
                    break
            return max_eval
        else:
            min_eval = inf
            for move in possible_moves:
                child_board = self.apply_move(board, move, 1)
                if child_board is None: continue
                eval_score = self.minimax(child_board, depth - 1, alpha, beta,␣
↪True)
                min_eval = min(min_eval, eval_score)
                beta = min(beta, eval_score)
                if beta <= alpha:
                    break
            return min_eval

    def get_computer_move(self):
        start_time = time.time()
        best_move = None
        best_value = -inf
        alpha = -inf
        beta = inf

        moves = self.get_possible_moves(self.board, -1)
        if not moves:
            return None
        best_move = moves[0]

        random.shuffle(moves)

        for move in moves:
            if self.game_over:
                break
            new_board = self.apply_move(self.board, move, -1)
            if new_board is None:
                continue

            value = self.minimax(new_board, self.depth - 1, alpha, beta, False)

            if value > best_value:
                best_value = value
                best_move = move

            alpha = max(alpha, value)

        if self.print == True:
            print(f"Move took {time.time()-start_time:.2f}s, nodes visited:␣
↪{self.nodes_visited}")
        self.nodes_visited = 0
        return best_move

    def draw_text(self, text, center_pos, max_width):
        font_size = 24
        font = pygame.font.Font(None, font_size)
        text_surface = font.render(text, True, (0, 0, 0))

        text_width = text_surface.get_width()
        if text_width > max_width:
            scale_factor = max_width / text_width
            new_font_size = int(font_size * scale_factor)
            font = pygame.font.Font(None, new_font_size)
            text_surface = font.render(text, True, (0, 0, 0))

        text_rect = text_surface.get_rect(center=center_pos)
        self.screen.blit(text_surface, text_rect)

    def draw_board(self):
        self.screen.fill((0, 0, 0))

        for i in range(6):
            for j in range(6):
                x0 = j * 100
                y0 = i * 100

                if self.board[i][j] == 1:
                    pygame.draw.circle(self.screen, (255, 0, 0), (x0 + 50, y0 +␣
↪50), 40)
                elif self.board[i][j] == -1:
                    pygame.draw.circle(self.screen, (0, 0, 255), (x0 + 50, y0 +␣
↪50), 40)

                pygame.draw.rect(self.screen, (255, 255, 255), (x0, y0, 100,␣
↪100), 1)

        for i in [3, 6]:
            pygame.draw.line(self.screen, (255, 255, 255), (0, i * 100), (600,␣
↪i * 100), 3)  # Horizontal
            pygame.draw.line(self.screen, (255, 255, 255), (i * 100, 0), (i *␣
↪100, 600), 3)  # Vertical

        # Show rotation buttons if in move_stage 2
        if self.move_stage == 2:
            self.highlight_selected_block()
            self.show_rotation_buttons()

        if self.show_buttons:
            pygame.draw.rect(self.screen, (144, 238, 144), self.
↪buttons["rotate_cw"])  # Light Green
            pygame.draw.rect(self.screen, (173, 216, 230), self.
↪buttons["rotate_ccw"])  # Light Blue

            self.draw_text(
                "CLOCKWISE",
                self.buttons["rotate_cw"].center,
```

```python
                self.buttons["rotate_cw"].width,
            )
            self.draw_text(
                "COUNTER-CLOCKWISE",
                self.buttons["rotate_ccw"].center,
                self.buttons["rotate_ccw"].width,
            )

    def click_handler(self, event):
        if self.game_over or self.current_player != 1:
            return

        x, y = event.pos
        if self.move_stage == 0:  # Place piece
            if x > 600:
                return  # clicks on control area
            col = x // 100
            row = y // 100
            if 0 <= row < 6 and 0 <= col < 6 and self.board[row][col] == 0:
                self.temp_piece = (row, col)
                self.board[row][col] = 1
                self.move_stage = 1
                self.draw_board()

        elif self.move_stage == 1:  # Select block
            if x > 600:
                return
            # which block was clicked
            block_x = 0 if x < 300 else 1
            block_y = 0 if y < 300 else 1
            self.selected_block = block_y * 2 + block_x
            self.move_stage = 2
            self.show_rotation_buttons()
            self.highlight_selected_block()

        elif self.move_stage == 2:  # Rotate
            if self.buttons["rotate_cw"].collidepoint(event.pos):
                self.apply_rotation("cw")
            if self.buttons["rotate_ccw"].collidepoint(event.pos):
                self.apply_rotation("ccw")

    def apply_rotation(self, direction):
        self.rotate_block(self.board, self.selected_block, direction)
        self.current_player = -1
        self.move_stage = 0
        self.selected_block = None
        self.temp_piece = None
        self.hide_rotation_buttons()
        self.draw_board()
        pygame.display.flip()
        self.check_game_over()
        pygame.time.delay(1000)
        self.play_computer_move()

    def highlight_selected_block(self):
        colors = [
            (255, 153, 153),
            (153, 255, 153),
            (153, 153, 255),
            (255, 255, 153),
        ]  # RGB colors

        row_start = (self.selected_block // 2) * 3
        col_start = (self.selected_block % 2) * 3

        pygame.draw.rect(
            self.screen,
            colors[self.selected_block],
            (col_start * 100, row_start * 100, 300, 300),
            5,
        )

    def play_computer_move(self):
        move = self.get_computer_move()
        if move and not self.game_over:
            new_board = self.apply_move(self.board, move, -1)
            if new_board is not None:
                self.board = new_board
                self.current_player = 1
                self.draw_board()
                pygame.display.flip()
                self.check_game_over()
            else:
                print("Invalid computer move!")

    def check_game_over(self):
        winner = self.check_winner(self.board)
        if winner is not None:
            self.game_over = True
            self.result = winner
            print("Game over! Result:", winner)
            if self.ui:
                self.show_game_over_message()
    def show_game_over_message(self):
        self.screen.fill((200, 200, 200))
        pygame.draw.rect(self.screen, (255, 255, 255), (100, 200, 500, 200))
        pygame.draw.rect(self.screen, (0, 0, 0), (100, 200, 500, 200), 3)

        result_text = f"Player {self.result} wins!" if self.result != 0 else␣
↪"Draw!"
        text_surface = self.font_large.render(result_text, True, (255, 0, 0))
        self.screen.blit(text_surface, (250, 250))

        exit_text = self.font_small.render("Click anywhere to exit", True, (0,␣
↪0, 0))
        self.screen.blit(exit_text, (230, 350))
        pygame.display.flip()

    def play(self):
        if self.ui:
            running = True
            while running:
                for event in pygame.event.get():
                    if event.type == pygame.QUIT:
                        running = False
                    elif event.type == pygame.MOUSEBUTTONDOWN:
                        self.click_handler(event)
                self.draw_board()
                pygame.display.flip()
            pygame.quit()
            return self.result
        else:
            while not self.game_over:
                self.print_board()
                winner = self.check_winner(self.board)
                if winner is not None:
                    return winner

                if self.current_player == 1:
                    move = random.choice(self.get_possible_moves(self.board, 1))
                else:
                    move = self.get_computer_move()

                self.board = self.apply_move(self.board, move, self.
↪current_player)
                self.current_player *= -1
            return self.result

    def print_board(self):
        if self.print == False:
            return
        print("-" * 25)
        for row in self.board:
            print(" ".join(f"{x:2}" for x in row))
        print("-" * 25)
```

[9]:
```
if __name__ == "__main__":
    numGames = 6
    numWins, numTies, numLosses = 0, 0, 0
    for i in range(numGames):
        game = PentagoGame(ui=False, print=True, depth=1)
        result = game.play()
        if result == -1:
            numWins += 1
        elif result == 0:
            numTies += 1
        else:
            numLosses += 1

    print(f"{numWins} wins, {numTies} ties, {numLosses} losses")

    numGames = 1
    numWins, numTies, numLosses = 0, 0, 0
    for i in range(numGames):
        game = PentagoGame(ui=False, print=True, depth=2)
        result = game.play()
        if result == -1:
            numWins += 1
        elif result == 0:
            numTies += 1
        else:
            numLosses += 1

    print(f"{numWins} wins, {numTies} ties, {numLosses} losses")
```

```
-------------------------
 0  0  0  0  0  0
 0  0  0  0  0  0
 0  0  0  0  0  0
 0  0  0  0  0  0
 0  0  0  0  0  0
 0  0  0  0  0  0
-------------------------
-------------------------
 0  0  0  0  0  0
 0  0  0  0  0  0
 0  0  0  0  0  0
 0  0  0  0  0  0
 0  0  0  0  0  0
 0  0  0  0  0  1
-------------------------
Move took 0.16s, nodes visited: 280
-------------------------
 0  0  0  0  0  0
 0  0  0  0  0  0
 0  0  0  0  0  0
 0 -1  0  0  0  0
 0  0  0  0  0  0
 0  0  0  0  0  1
-------------------------
-------------------------
 0  0  0  0  0  0
 0  0  0  0  0  0
 0  0  0  0  0  0
 0 -1  0  0  0  1
 0  0  0  0  0  0
 0  0  0  0  0  1
-------------------------
Move took 0.15s, nodes visited: 264
-------------------------
 0  0  0  0  0  0
 0  0  0  0  0  0
 0  0  0  0  0  0
 0 -1  0  1  0  1
 0 -1  0  0  0  0
 0  0  0  0  0  0
-------------------------
-------------------------
 0  0  0  0  0  0
 0  0  0  0  1  0
 0  0  0  0  0  0
 0 -1  0  0  0  1
 0 -1  0  0  0  0
 0  0  0  0  0  1
-------------------------
Move took 0.13s, nodes visited: 248
-------------------------
 0  0  0  0  0  0
 0 -1  0  0  1  0
 0  0  0  0  0  0
 0 -1  0  1  0  1
 0 -1  0  0  0  0
 0  0  0  0  0  0
-------------------------
-------------------------
```

```
 0  0  0  0  0  0
 0 -1  0  0  1  0
 0  0  0  0  1  0
 0 -1  0  1  0  1
 0 -1  0  0  0  0
 0  0  0  0  0  0
-------------------------
Move took 0.14s, nodes visited: 232
-------------------------
 0  0  0  0  0  0
 0 -1  0  1  1  0
 0 -1  0  0  0  0
 0 -1  0  1  0  1
 0 -1  0  0  0  0
 0  0  0  0  0  0
-------------------------
-------------------------
 0  0  0  0  1  0
 0 -1  0  0  1  0
 0 -1  0  0  1  0
 0 -1  0  1  0  1
 0 -1  0  0  0  0
 0  0  0  0  0  0
-------------------------
Move took 0.13s, nodes visited: 216
-------------------------
 0 -1  0  0  1  0
 0 -1  0  0  1  0
 0 -1  0  0  1  0
 0 -1  0  0  0  1
 0 -1  0  0  0  0
 0  0  0  0  0  1
-------------------------
-------------------------
 0  0  0  0  0  0
 0  0  0  0  0  0
 0  0  0  0  0  0
 0  0  0  0  0  0
 0  0  0  0  0  0
 0  0  0  0  0  0
-------------------------
-------------------------
 0  0  0  0  0  0
 0  0  0  0  0  0
 0  0  0  0  0  0
 1  0  0  0  0  0
 0  0  0  0  0  0
 0  0  0  0  0  0
```

```
-------------------------
Move took 0.14s, nodes visited: 280
-------------------------
 0  0  0  0  0  0
 0 -1  0  0  0  0
 0  0  0  0  0  0
 1  0  0  0  0  0
 0  0  0  0  0  0
 0  0  0  0  0  0
-------------------------
-------------------------
 0  0  0  0  0  0
 0 -1  0  0  0  0
 0  0  0  0  0  0
 1  0  0  0  0  0
 0  1  0  0  0  0
 0  0  0  0  0  0
-------------------------
Move took 0.16s, nodes visited: 264
-------------------------
 0  0  0  0  0  0
 0 -1  0  0  0  0
 0  0 -1  0  0  0
 1  0  0  0  0  0
 0  1  0  0  0  0
 0  0  0  0  0  0
-------------------------
-------------------------
 0  0  0  0  0  0
 0 -1  0  0  0  0
 0  0 -1  0  0  0
 1  0  0  0  0  0
 0  1  0  0  0  0
 0  0  0  1  0  0
-------------------------
Move took 0.13s, nodes visited: 248
-------------------------
 0  0  0  0  0  0
 0 -1  0  0  0  0
 0  0 -1  0  0  0
 1  0  0  0  0  0
 0  1  0  0 -1  0
 0  0  0  1  0  0
-------------------------
-------------------------
 0  0  0  0  0  1
 0 -1  0  0  0  0
 0  0 -1  0  0  0
 1  0  0  0  0  0
 0  1  0  0 -1  0
 0  0  0  1  0  0
-------------------------
Move took 0.14s, nodes visited: 232
-------------------------
 0  0  0  0  0  0
 0 -1  0  0  0  0
 0  0 -1  0  0  1
 1  0  0 -1  0  0
 0  1  0  0 -1  0
 0  0  0  1  0  0
-------------------------
-------------------------
 0  0 -1  0  0  0
 0 -1  0  0  0  0
 0  0  0  0  0  1
 1  1  0 -1  0  0
 0  1  0  0 -1  0
 0  0  0  1  0  0
-------------------------
Move took 0.13s, nodes visited: 216
-------------------------
 0  0  0  0  0  0
 0 -1  0  0  0  0
 0  0 -1  0  0  1
 1  1  0 -1  0  0
 0  1  0  0 -1  0
 0  0  0  1  0 -1
-------------------------
-------------------------
 0  0  0  0  0  0
 0  0  0  0  0  0
 0  0  0  0  0  0
 0  0  0  0  0  0
 0  0  0  0  0  0
 0  0  0  0  0  0
-------------------------
-------------------------
 0  0  0  0  0  0
 0  0  1  0  0  0
 0  0  0  0  0  0
 0  0  0  0  0  0
 0  0  0  0  0  0
 0  0  0  0  0  0
-------------------------
Move took 0.14s, nodes visited: 280
-------------------------
```

```
 0  1  0  0  0  0
 0 -1  0  0  0  0
 0  0  0  0  0  0
 0  0  0  0  0  0
 0  0  0  0  0  0
 0  0  0  0  0  0
-------------------------
-------------------------
 0  0  0  0  0  0
 0 -1  1  0  0  0
 0  0  0  0  0  0
 0  1  0  0  0  0
 0  0  0  0  0  0
 0  0  0  0  0  0
-------------------------
Move took 0.14s, nodes visited: 264
-------------------------
 0  1  0  0  0  0
 0 -1 -1  0  0  0
 0  0  0  0  0  0
 0  1  0  0  0  0
 0  0  0  0  0  0
 0  0  0  0  0  0
-------------------------
-------------------------
 0  0  0  0  0  1
 0 -1  1  0  0  0
 0 -1  0  0  0  0
 0  1  0  0  0  0
 0  0  0  0  0  0
 0  0  0  0  0  0
-------------------------
Move took 0.14s, nodes visited: 248
-------------------------
 0  0  0  0  0  1
 0 -1  1  0  0  0
 0 -1  0  0  0  0
 0  0  0  0  0  0
 1 -1  0  0  0  0
 0  0  0  0  0  0
-------------------------
-------------------------
 0  0  0  0  0  1
-1 -1  0  0  0  0
 0  1  0  0  0  0
 0  0  0  0  0  0
 1 -1  0  0  0  0
 0  0  0  0  0  1
```

```
-------------------------
Move took 0.15s, nodes visited: 232
-------------------------
 0  0  0  0  0  1
 0 -1  1  0  0  0
 0 -1  0  0  0  0
 0 -1  0  0  0  0
 1 -1  0  0  0  0
 0  0  0  0  0  1
-------------------------
-------------------------
 0  1  0  0  0  1
 0 -1  1  0  0  0
 0 -1  0  0  0  0
 0  0  0  0  0  0
-1 -1  0  0  0  0
 0  1  0  0  0  1
-------------------------
Move took 0.14s, nodes visited: 216
-------------------------
 0  1  0  0  0  1
 0 -1  1  0  0  0
 0 -1  0  0  0  0
 0 -1  0  0  0  0
 1 -1  0  0  0  0
 0 -1  0  0  0  1
-------------------------
-------------------------
 0  0  0  0  0  0
 0  0  0  0  0  0
 0  0  0  0  0  0
 0  0  0  0  0  0
 0  0  0  0  0  0
 0  0  0  0  0  0
-------------------------
-------------------------
 0  0  0  0  0  0
 0  0  0  0  0  0
 0  0  0  0  0  0
 0  0  1  0  0  0
 0  0  0  0  0  0
 0  0  0  0  0  0
-------------------------
Move took 0.14s, nodes visited: 280
-------------------------
 0  0  0  0  0  0
 0  0  0  0  0  0
 0  0  0  0  0  0
 1  0  0  0  0  0
 0  0  0  0 -1  0
 0  0  0  0  0  0
-------------------------
-------------------------
 0  0  0  0  0  0
 0  0  0  0  0  0
 0  0  0  0  0  0
 1  0  0  1  0  0
 0  0  0  0 -1  0
 0  0  0  0  0  0
-------------------------
Move took 0.14s, nodes visited: 264
-------------------------
 0  0  0  0  0  0
 0  0  0  0  0  0
 0  0  0  0  0  0
 1  0  0  0  0  0
 0 -1  0  0 -1  0
 0  0  0  1  0  0
-------------------------
-------------------------
 0  0  0  0  0  0
 0  0  1  0  0  0
 0  0  0  0  0  0
 0  0  1  0  0  0
 0 -1  0  0 -1  0
 0  0  0  1  0  0
-------------------------
Move took 0.13s, nodes visited: 248
-------------------------
 0  1  0  0  0  0
 0  0  0  0  0  0
 0  0  0  0  0  0
 0  0  1  0  0  0
 0 -1  0 -1 -1  0
 0  0  0  1  0  0
-------------------------
-------------------------
 0  0  0  0  0  0
 1  0  0  0  0  0
 0  0  0  0  0  0
 0  0  1  0  0  0
 0 -1  1 -1 -1  0
 0  0  0  1  0  0
-------------------------
Move took 0.13s, nodes visited: 232
-------------------------
```

```
 0  0  0  0  0  0
 1  0  0  0  0  0
 0  0  0  0  0  0
 1  1  0  0  0  0
 0 -1 -1 -1 -1  0
 0  0  0  1  0  0
-------------------------
-------------------------
 0  0  0  0  0  0
 1  0  0  0  0  0
 0  0  0  0  0  1
 0 -1  0  0  0  0
 1 -1  0 -1 -1  0
 1  0  0  1  0  0
-------------------------
Move took 0.13s, nodes visited: 216
-------------------------
 0  0  0  0  0  0
 1  0  0  0  0  0
 0  0  0  0  0  1
 0  0  0  0  0  0
-1 -1 -1 -1 -1  0
 0  1  1  1  0  0
-------------------------
-------------------------
 0  0  0  0  0  0
 0  0  0  0  0  0
 0  0  0  0  0  0
 0  0  0  0  0  0
 0  0  0  0  0  0
 0  0  0  0  0  0
-------------------------
-------------------------
 0  0  0  0  0  0
 0  0  0  0  0  0
 0  0  0  0  0  0
 0  0  0  0  0  0
 0  0  0  0  0  0
 0  0  0  1  0  0
-------------------------
Move took 0.13s, nodes visited: 280
-------------------------
 0  0  0  0  0  0
 0  0  0  0  0  0
 0  0  0  0  0  0
 0 -1  0  0  0  0
 0  0  0  0  0  0
 0  0  0  1  0  0
```

```
------------------------
------------------------
 0  0  0  0  0  0
 0  0  0  0  0  0
 0  0  0  0  0  0
 0 -1  0  0  0  1
 0  0  0  0  0  0
 0  0  0  1  0  0
------------------------
Move took 0.14s, nodes visited: 264
------------------------
 0  0  0  0  0  0
 0  0  0  0  0  0
 0  0 -1  0  0  0
 0  0  0  0  0  1
 0  0 -1  0  0  0
 0  0  0  1  0  0
------------------------
------------------------
 0  0  0  0  0  0
 0  0  0  0  0  0
 0  0 -1  0  0  1
 0  0  0  0  0  1
 0  0 -1  0  0  0
 0  0  0  1  0  0
------------------------
Move took 0.12s, nodes visited: 248
------------------------
 0  0  0  0  0  1
 0  0 -1  0  0  0
 0  0 -1  0  0  0
 0  0  0  0  0  1
 0  0 -1  0  0  0
 0  0  0  1  0  0
------------------------
------------------------
 0  0  0  1  0  0
 0  0 -1  0  0  0
 0  0 -1  0  0  0
 0  0  0  0  0  1
 0  1 -1  0  0  0
 0  0  0  1  0  0
------------------------
Move took 0.12s, nodes visited: 232
------------------------
 0  0  0  0  0  1
 0  0 -1  0  0  0
 0  0 -1  0  0  0
 0  0 -1  0  0  1
 0  1 -1  0  0  0
 0  0  0  1  0  0
------------------------
------------------------
 1  0  0  0  0  1
 0  0  0  0  0  0
-1 -1  0  0  0  0
 0  0 -1  0  0  1
 0  1 -1  0  0  0
 0  0  0  1  0  0
------------------------
Move took 0.13s, nodes visited: 216
------------------------
 0  0  0  0  0  1
 0  0 -1  0  0  0
 1  0 -1  0  0  0
 0  0 -1  0  0  1
 0  1 -1  0  0  0
 0  0 -1  1  0  0
------------------------
------------------------
 0  0  0  0  0  0
 0  0  0  0  0  0
 0  0  0  0  0  0
 0  0  0  0  0  0
 0  0  0  0  0  0
 0  0  0  0  0  0
------------------------
------------------------
 0  0  0  0  0  0
 0  0  0  0  0  0
 0  0  0  0  1  0
 0  0  0  0  0  0
 0  0  0  0  0  0
 0  0  0  0  0  0
------------------------
Move took 0.14s, nodes visited: 280
------------------------
 0  0  0  0  0  0
 0  0  0  0  0  1
 0 -1  0  0  0  0
 0  0  0  0  0  0
 0  0  0  0  0  0
 0  0  0  0  0  0
------------------------
------------------------
 0  0  0  0  0  0
 0  0  0  0  0  0
 0 -1  0  0  1  0
 1  0  0  0  0  0
 0  0  0  0  0  0
 0  0  0  0  0  0
-------------------------
Move took 0.13s, nodes visited: 264
-------------------------
 0  0  0  0  0  0
 0  0  0  0  0  1
 0 -1 -1  0  0  0
 1  0  0  0  0  0
 0  0  0  0  0  0
 0  0  0  0  0  0
-------------------------
-------------------------
 0  0  0  0  0  0
 0  0  0  0  0  1
 0 -1 -1  0  0  0
 0  0  1  0  0  0
 0  0  0  1  0  0
 0  0  0  0  0  0
-------------------------
Move took 0.12s, nodes visited: 248
-------------------------
 0  0  0  0  0  0
 0  0  0  0  0  1
 0 -1 -1  0 -1  0
 0  0  1  0  0  0
 0  0  0  0  0  0
 0  0  0  0  1  0
-------------------------
-------------------------
 0  0  0  0  0  0
 0  0  0  0  0  1
 0 -1 -1  0 -1  0
 0  0  1  0  0  0
 0  0  0  1  0  0
 1  0  0  0  0  0
-------------------------
Move took 0.13s, nodes visited: 232
-------------------------
 0  0  0  0  0  0
 0  0  0  0  0  1
 0 -1 -1 -1 -1  0
 1  0  0  0  0  0
 0  0  0  1  0  0
 0  0  1  0  0  0
```

```
-------------------------
-------------------------
 0  0 -1  0  0  0
 0  0 -1  0  0  1
 0  0  1 -1 -1  0
 1  0  0  0  0  0
 0  0  0  1  0  0
 0  0  1  0  0  0
-------------------------
Move took 0.12s, nodes visited: 216
-------------------------
 0  0  0  0  0  0
 0  0  0  0  0  1
 1 -1 -1 -1 -1 -1
 1  0  0  0  0  0
 0  0  0  1  0  0
 0  0  1  0  0  0
-------------------------
6 wins, 0 ties, 0 losses
-------------------------
 0  0  0  0  0  0
 0  0  0  0  0  0
 0  0  0  0  0  0
 0  0  0  0  0  0
 0  0  0  0  0  0
 0  0  0  0  0  0
-------------------------
-------------------------
 0  0  0  0  0  0
 0  0  0  0  0  0
 0  0  0  0  0  0
 0  0  0  0  0  0
 1  0  0  0  0  0
 0  0  0  0  0  0
-------------------------
Move took 6.65s, nodes visited: 12360
-------------------------
 0  0  0  0  0  0
 0  0  0  0  0  0
 0  0  0 -1  0  0
 0  0  0  0  0  0
 1  0  0  0  0  0
 0  0  0  0  0  0
-------------------------
-------------------------
 0  0  0  0  0  0
 0  0  0  0  0  0
 1  0  0 -1  0  0
 0  1  0  0  0  0
 0  0  0  0  0  0
 0  0  0  0  0  0
-------------------------
Move took 4.42s, nodes visited: 7913
-------------------------
 0  0  0  0  0  0
 0  0  0  0  0  0
 1  0  0 -1  0  0
 0  0 -1  0  0  0
 0  0  1  0  0  0
 0  0  0  0  0  0
-------------------------
-------------------------
 0  0  0  0  0  0
 0  0  0  0  0  0
 1  0  0 -1  0  0
-1  1  0  0  0  0
 0  0  1  0  0  0
 0  0  0  0  0  0
-------------------------
Move took 5.43s, nodes visited: 9882
-------------------------
 0  0  0  0  0  0
 0  0  0  0  0  0
 1  0  0 -1  0  0
-1  1 -1  0  0  0
 0  0  1  0  0  0
 0  0  0  0  0  0
-------------------------
-------------------------
 0  0  0  0  0  0
 0  0  0  0  0  0
 0  0  1 -1  0  0
-1  1 -1  0  0  0
 0  0  1  0  0  1
 0  0  0  0  0  0
-------------------------
Move took 1.69s, nodes visited: 2877
-------------------------
 0  0  0  0  0  0
 0  0  0  0  0  0
 1  0  0 -1  0  0
-1  1 -1  0  0  0
 0 -1  1  0  0  1
 0  0  0  0  0  0
-------------------------
-------------------------
```

```
 0  0  1 -1  0  0
 0  0  0  0  0  0
 1  0  0  0  0  0
-1  1 -1  0  0  0
 0 -1  1  0  0  1
 0  0  0  0  0  0
-------------------------
Move took 2.37s, nodes visited: 3946
-------------------------
 0  0  1  0  0  0
 0  0  0  0  0  0
 1  0  0 -1  0  0
-1  1 -1  0  0  0
 0 -1  1 -1  0  1
 0  0  0  0  0  0
-------------------------
-------------------------
 0  0  1  0  0  0
 0  0  0  0  0  0
 1  0  0  0  0 -1
-1  1 -1  0  0  0
 0 -1  1 -1  0  1
 0  1  0  0  0  0
-------------------------
Move took 3.10s, nodes visited: 3654
-------------------------
 0  0  1  0  0  0
 0  0  0  0  0  0
 1  0  0 -1  0  0
-1  1 -1 -1  0  0
 0 -1  1 -1  0  1
 0  1  0  0  0  0
-------------------------
-------------------------
 0  0  1  0  0  0
 0  0  0  0  0  0
 1  0  0 -1  0  0
-1  1  0 -1  0  0
 1 -1  1 -1  0  1
-1  0  1  0  0  0
-------------------------
Move took 2.03s, nodes visited: 3138
-------------------------
 0  0  1  0  0  0
 0  0  0  0  0  0
 1  0  0 -1  0  0
-1  1 -1 -1  0  0
 0 -1  1 -1  0  1
 1  1 -1  0  0  0
-------------------------
-------------------------
 0  0  1 -1  0  0
 0  0  0  0  0  0
 1  0  0  0  0  1
-1  1 -1 -1  0  0
 0 -1  1 -1  0  1
 1  1 -1  0  0  0
-------------------------
Move took 1.89s, nodes visited: 2640
-------------------------
 0  0  1  0 -1  1
 0  0  0  0  0  0
 1  0  0 -1  0  0
-1  1 -1 -1  0  0
 0 -1  1 -1  0  1
 1  1 -1  0  0  0
-------------------------
-------------------------
 0  0  1  0 -1  1
 0  0  0  1  0  0
 1  0  0 -1  0  0
-1  1 -1  0  1  0
 0 -1  1  0  0  0
 1  1 -1 -1 -1  0
-------------------------
Move took 0.88s, nodes visited: 1456
-------------------------
 0  0  1  0 -1  1
 0  0  0  1 -1  0
 1  0  0 -1  0  0
 1  0 -1  0  1  0
 1 -1  1  0  0  0
-1  1 -1 -1 -1  0
-------------------------
1 wins, 0 ties, 0 losses
```

3.1 Questions

3.1.1 Question 1:

In the Minimax algorithm, increasing the search depth leads to better decision-making, which can improve the odds of winning. However, this comes at the cost of exponentially increased computation time and more nodes being evaluated. While alpha-beta pruning can reduce the number of nodes, deeper depths still significantly impact performance.

Depth	Odds of Winning	Time Complexity	Nodes Seen
Low	Poor	Fast	Few
Medium	Decent	Moderate	Moderate
High	Strong	Slow	Many

3.1.2 Question 2:

Yes, it's possible to order child nodes to maximize pruning in alpha-beta Minimax.

If we evaluate the best moves first (high to low for MAX, low to high for MIN), the algorithm can prune more branches earlier, reducing the number of nodes explored.

We can use heuristics or results from shallower searches to guess which moves are better, and evaluate those first.

This improves efficiency without affecting the correctness of the result.

For example sorting children in descending order for MAX and ascending order for MIN.

3.1.3 Question 3:

The branching factor is the average number of possible moves a player can make from a given game state. In the context of search algorithms like Minimax, it determines how many child nodes are generated from each node.

- **Early Game**:

High branching factor.

Many pieces/moves are available.

More options lead to more children per node.

- **Mid Game**:

Still relatively high, depending on the game complexity.

Tactical possibilities increase, so decision trees can be wide.

- **Late Game**:

Branching factor decreases.

Fewer pieces/moves remain.

Fewer legal actions, so fewer child nodes are generated.

3.1.4 Question 4:

Alpha-beta pruning works by cutting off parts of the game tree that don't need to be evaluated because they won't affect the final decision.

It speeds up the search by ignoring useless branches, but it never skips over a move that could be the best one. So the result is just as accurate.

3.1.5 Question 5:

Minimax assumes that the opponent always plays optimally, trying to minimize your score. But if the opponent acts randomly, this assumption doesn't hold.

It picks moves based on the worst-case scenario and it can lead to over-defensive or suboptimal decisions in practice.

Expectimax replaces the **min** step with an **expectation** step. Instead of assuming the opponent minimizes your score, it calculates the average outcome based on the probability of each move.

Feature	Minimax	Expectimax
Opponent Model	Assumes perfect, optimal play	Assumes random or probabilistic play
Decision Style	Worst-case (defensive)	Average-case (realistic)
Use Case	Competitive, perfect-play games	Games with randomness or casual play
Behavior	Conservative, safe moves	Exploitative, balanced decisions

- **Minimax**: Good for where opponent is smart.
- **Expectimax**: Better for card games, dice games, or AI playing against casual players.