```python
import numpy as np

# Objective function
def objective_function(x):
    return x ** 2  # Example: f(x) = x^2

# Grey Wolf Optimizer (GWO)
def grey_wolf_optimizer(num_wolves, num_iterations, search_space):
    # Initialize positions of wolves randomly within the search space
    wolves = np.random.uniform(search_space[0], search_space[1], num_wolves)
    fitness = np.array([objective_function(x) for x in wolves])

    # Identify initial alpha, beta, and delta
    sorted_indices = np.argsort(fitness)
    alpha = wolves[sorted_indices[0]]  # Best wolf (alpha)
    beta = wolves[sorted_indices[1]]   # Second-best wolf (beta)
    delta = wolves[sorted_indices[2]]  # Third-best wolf (delta)

    # Main loop
    for iteration in range(num_iterations):
        # Coefficients for position update
        a = 2 - 2 * (iteration / num_iterations)  # Decreases linearly from 2 to 0

        for i in range(num_wolves):
            # Calculate distances and update positions based on alpha, beta, and delta
            r1, r2 = np.random.rand(), np.random.rand()
            A1 = 2 * a * r1 - a
            C1 = 2 * r2
            D_alpha = abs(C1 * alpha - wolves[i])
            X1 = alpha - A1 * D_alpha

            r1, r2 = np.random.rand(), np.random.rand()
            A2 = 2 * a * r1 - a
            C2 = 2 * r2
            D_beta = abs(C2 * beta - wolves[i])
            X2 = beta - A2 * D_beta

            r1, r2 = np.random.rand(), np.random.rand()
            A3 = 2 * a * r1 - a
            C3 = 2 * r2
            D_delta = abs(C3 * delta - wolves[i])
            X3 = delta - A3 * D_delta

            # Update wolf position
            wolves[i] = (X1 + X2 + X3) / 3

        # Re-evaluate fitness and reassign alpha, beta, delta
        fitness = np.array([objective_function(x) for x in wolves])
        sorted_indices = np.argsort(fitness)
        alpha = wolves[sorted_indices[0]]
        beta = wolves[sorted_indices[1]]
        delta = wolves[sorted_indices[2]]

        # Print progress
        print(f"Iteration {iteration + 1}: Best fitness = {fitness[sorted_indices[0]]}")

    # Return the best solution
    return alpha, objective_function(alpha)

# Parameters
num_wolves = 5
num_iterations = 10
search_space = [-10, 10]

# Run GWO
best_position, best_fitness = grey_wolf_optimizer(num_wolves, num_iterations, search_space)
print(f"\nBest Position: {best_position}")
print(f"Best Fitness: {best_fitness}")
```

```
Iteration 1: Best fitness = 0.3522631805385104
Iteration 2: Best fitness = 0.11982429039380918
Iteration 3: Best fitness = 0.06375050381345825
Iteration 4: Best fitness = 0.033344468209391985
Iteration 5: Best fitness = 0.02132925039887001
Iteration 6: Best fitness = 4.942618715279982e-05
Iteration 7: Best fitness = 0.0005565119245818511
Iteration 8: Best fitness = 0.00028527780034656714
Iteration 9: Best fitness = 0.00032784721439228033
Iteration 10: Best fitness = 0.00041767311689656553

Best Position: -0.020437052549146257
```

```
Best Fitness: 0.00041767311689656553
```

Start coding or generate with AI.

```python
import numpy as np

# Objective function: Combines risk and return with diversification penalties
def portfolio_fitness(weights, returns, covariance_matrix):
    # Normalize weights to ensure they sum to 1
    weights = weights / np.sum(weights)

    # Calculate portfolio return
    portfolio_return = np.dot(weights, returns)

    # Calculate portfolio risk
    portfolio_risk = np.sqrt(np.dot(weights.T, np.dot(covariance_matrix, weights)))

    # Penalty for lack of diversification (encourage uniform distribution of weights)
    diversification_penalty = np.sum((weights - 1/len(weights))**2)

    # Fitness: Maximize return, minimize risk and diversification penalty
    return -portfolio_return + portfolio_risk + diversification_penalty

# Grey Wolf Optimizer for Portfolio Optimization
def grey_wolf_optimizer(num_wolves, num_iterations, returns, covariance_matrix):
    num_assets = len(returns)

    # Initialize positions of wolves randomly within [0, 1] for asset weights
    wolves = np.random.rand(num_wolves, num_assets)

    # Evaluate fitness of each wolf
    fitness = np.array([portfolio_fitness(wolf, returns, covariance_matrix) for wolf in wolves])

    # Identify initial alpha, beta, and delta
    sorted_indices = np.argsort(fitness)
    alpha = wolves[sorted_indices[0]]  # Best wolf (alpha)
    beta = wolves[sorted_indices[1]]  # Second-best wolf (beta)
    delta = wolves[sorted_indices[2]]  # Third-best wolf (delta)

    # Main loop
    for iteration in range(num_iterations):
        # Update the coefficient "a" (linearly decreasing)
        a = 2 - 2 * (iteration / num_iterations)

        for i in range(num_wolves):
            # Update positions based on alpha, beta, and delta
            r1, r2 = np.random.rand(), np.random.rand()
            A1 = 2 * a * r1 - a
            C1 = 2 * r2
            D_alpha = abs(C1 * alpha - wolves[i])
            X1 = alpha - A1 * D_alpha

            r1, r2 = np.random.rand(), np.random.rand()
            A2 = 2 * a * r1 - a
            C2 = 2 * r2
            D_beta = abs(C2 * beta - wolves[i])
            X2 = beta - A2 * D_beta

            r1, r2 = np.random.rand(), np.random.rand()
            A3 = 2 * a * r1 - a
            C3 = 2 * r2
            D_delta = abs(C3 * delta - wolves[i])
            X3 = delta - A3 * D_delta

            # Update wolf position
            wolves[i] = (X1 + X2 + X3) / 3

        # Re-evaluate fitness
        fitness = np.array([portfolio_fitness(wolf, returns, covariance_matrix) for wolf in wolves])

        # Update alpha, beta, and delta
        sorted_indices = np.argsort(fitness)
        alpha = wolves[sorted_indices[0]]
        beta = wolves[sorted_indices[1]]
        delta = wolves[sorted_indices[2]]

        # Print progress
        print(f"Iteration {iteration + 1}: Best fitness = {fitness[sorted_indices[0]]}")

    # Return the best solution (alpha)
    return alpha / np.sum(alpha)  # Normalize weights to sum to 1
```

```
# Example usage
if __name__ == "__main__":
    # Expected returns and covariance matrix (example data)
    expected_returns = np.array([0.12, 0.18, 0.15, 0.10])  # Example returns for 4 assets
    covariance_matrix = np.array([
        [0.1, 0.02, 0.04, 0.01],
        [0.02, 0.08, 0.03, 0.02],
        [0.04, 0.03, 0.09, 0.02],
        [0.01, 0.02, 0.02, 0.07]
    ])

    # Parameters
    num_wolves = 10
    num_iterations = 50

    # Run GWO for portfolio optimization
    optimal_weights = grey_wolf_optimizer(num_wolves, num_iterations, expected_returns, covariance_matrix)

    # Output results
    print("\nOptimal Portfolio Weights:")
    print(optimal_weights)
    print("\nPortfolio Return:", np.dot(optimal_weights, expected_returns))
    portfolio_risk = np.sqrt(np.dot(optimal_weights.T, np.dot(covariance_matrix, optimal_weights)))
    print("Portfolio Risk:", portfolio_risk)
```

```
Iteration 1: Best fitness = 0.06080664398844936
Iteration 2: Best fitness = 0.06046174890762165
Iteration 3: Best fitness = 0.060403371785169746
Iteration 4: Best fitness = 0.06057080014306048
Iteration 5: Best fitness = 0.06055240693049007
Iteration 6: Best fitness = 0.06033141435061663
Iteration 7: Best fitness = 0.06040610343899056
Iteration 8: Best fitness = 0.060364334176937665
Iteration 9: Best fitness = 0.060359144750372476
Iteration 10: Best fitness = 0.06035571994692859
Iteration 11: Best fitness = 0.06034793130872334
Iteration 12: Best fitness = 0.06035119407382191
Iteration 13: Best fitness = 0.0603485499055236
Iteration 14: Best fitness = 0.06034987787847512
Iteration 15: Best fitness = 0.0603422462818675
Iteration 16: Best fitness = 0.06034653550904897
Iteration 17: Best fitness = 0.06034609378052573
Iteration 18: Best fitness = 0.060346271818934524
Iteration 19: Best fitness = 0.06034625703236586
Iteration 20: Best fitness = 0.06034587874210832
Iteration 21: Best fitness = 0.06034600235156937
Iteration 22: Best fitness = 0.060344501002452804
Iteration 23: Best fitness = 0.06034576907772764
Iteration 24: Best fitness = 0.060345716540848485
Iteration 25: Best fitness = 0.06034568023038788
Iteration 26: Best fitness = 0.06034568438274399
Iteration 27: Best fitness = 0.060345685843000756
Iteration 28: Best fitness = 0.060345684778832995
Iteration 29: Best fitness = 0.060345683716534045
Iteration 30: Best fitness = 0.06034568374277274
Iteration 31: Best fitness = 0.06034568346037377
Iteration 32: Best fitness = 0.060345683271467186
Iteration 33: Best fitness = 0.060345683501958194
Iteration 34: Best fitness = 0.06034568346503435
Iteration 35: Best fitness = 0.0603456834707761
Iteration 36: Best fitness = 0.060345683466756145
Iteration 37: Best fitness = 0.06034568346838635
Iteration 38: Best fitness = 0.06034568346857804
Iteration 39: Best fitness = 0.06034568346862877
Iteration 40: Best fitness = 0.06034568346864242
Iteration 41: Best fitness = 0.06034568346864313
Iteration 42: Best fitness = 0.060345683468643135
Iteration 43: Best fitness = 0.06034568346864318
Iteration 44: Best fitness = 0.06034568346864315
Iteration 45: Best fitness = 0.06034568346864315
Iteration 46: Best fitness = 0.06034568346864315
Iteration 47: Best fitness = 0.06034568346864315
Iteration 48: Best fitness = 0.06034568346864315
Iteration 49: Best fitness = 0.06034568346864315
Iteration 50: Best fitness = 0.06034568346864315

Optimal Portfolio Weights:
[0.24273111 0.27748636 0.20760227 0.27218026]

Portfolio Return: 0.1374336443491166
Portfolio Risk: 0.19468145985596363
```