```
import random
# Objective function: f(x) = x^2
def fitness_function(x):
    return x**2

# Particle class to represent each particle in the swarm
class Particle:
    def __init__(self, min_x, max_x):
        self.position = random.uniform(min_x, max_x)  # Current position
        self.velocity = random.uniform(-1, 1)          # Current velocity
        self.best_position = self.position               # Best position found by the particle
        self.best_fitness = fitness_function(self.position)  # Best fitness value

    def update_velocity(self, global_best_position, inertia_weight, cognitive_coefficient, social_coefficient):
        r1, r2 = random.random(), random.random()
        cognitive_velocity = cognitive_coefficient * r1 * (self.best_position - self.position)
        social_velocity = social_coefficient * r2 * (global_best_position - self.position)
        self.velocity = (inertia_weight * self.velocity) + cognitive_velocity + social_velocity

    def update_position(self, min_x, max_x):
        self.position += self.velocity
        # Ensure the position is within bounds
        self.position = max(min_x, min(self.position, max_x))
        # Update the best position and fitness if needed
        fitness = fitness_function(self.position)
        if fitness < self.best_fitness:  # We want to minimize
            self.best_fitness = fitness
            self.best_position = self.position

# PSO algorithm
def particle_swarm_optimization(pop_size, min_x, max_x, generations, inertia_weight, cognitive_coefficient, social_coefficient):
    # Initialize particles
    swarm = [Particle(min_x, max_x) for _ in range(pop_size)]

    # Global best position initialized to None
    global_best_position = swarm[0].best_position
    global_best_fitness = swarm[0].best_fitness

    for generation in range(generations):
        for particle in swarm:
            # Update global best position
            if particle.best_fitness < global_best_fitness:
                global_best_fitness = particle.best_fitness
                global_best_position = particle.best_position

            # Update particle velocity and position
            particle.update_velocity(global_best_position, inertia_weight, cognitive_coefficient, social_coefficient)
            particle.update_position(min_x, max_x)

        # Print the best fitness in the current generation
        print(f"Generation {generation + 1}: Best solution = {global_best_position}, Fitness = {global_best_fitness}")

    return global_best_position

# Parameters
population_size = 30
min_value = -10
max_value = 10
num_generations = 50
inertia_weight = 0.5
cognitive_coefficient = 1.5
social_coefficient = 1.5

# Run Particle Swarm Optimization
best_solution = particle_swarm_optimization(population_size, min_value, max_value, num_generations, inertia_weight, cognitive_coefficient, s
print(f"Best solution found: {best_solution}, Fitness: {fitness_function(best_solution)}")
```

```
Generation 1: Best solution = 0.3586780593084349, Fitness = 0.12864995022926512
Generation 2: Best solution = 0.16333265099349337, Fitness = 0.02667755488056231
Generation 3: Best solution = 0.06719959663271477, Fitness = 0.00451578578759957
Generation 4: Best solution = 0.012496564840293836, Fitness = 0.0001561641328076681
Generation 5: Best solution = -0.009518377276859225, Fitness = 9.059950598463003e-05
Generation 6: Best solution = -0.009518377276859225, Fitness = 9.059950598463003e-05
```

```
Generation 7: Best solution = -0.009518377276859225, Fitness = 9.059950598463003e-05
Generation 8: Best solution = -0.008619022904601458, Fitness = 7.428755583004456e-05
Generation 9: Best solution = -0.008025064862352008, Fitness = 6.440166604495685e-05
Generation 10: Best solution = -0.007728085841227282, Fitness = 5.972331076937759e-05
Generation 11: Best solution = -0.0036537397385366654, Fitness = 1.334981407696198e-05
Generation 12: Best solution = 0.0011022594371599692, Fitness = 1.214975866808212e-06
Generation 13: Best solution = -0.0002217740484501433, Fitness = 4.918372856596651e-08
Generation 14: Best solution = -0.0002217740484501433, Fitness = 4.918372856596651e-08
Generation 15: Best solution = -0.00013306715484908294, Fitness = 1.7706867699629817e-08
Generation 16: Best solution = -0.00013306715484908294, Fitness = 1.7706867699629817e-08
Generation 17: Best solution = 0.00012161101722563735, Fitness = 1.4789239510654264e-08
Generation 18: Best solution = 0.00012161101722563735, Fitness = 1.4789239510654264e-08
Generation 19: Best solution = 0.00012161101722563735, Fitness = 1.4789239510654264e-08
Generation 20: Best solution = 0.00012161101722563735, Fitness = 1.4789239510654264e-08
Generation 21: Best solution = -0.00010837020366403897, Fitness = 1.1744101042185285e-08
Generation 22: Best solution = 4.121410786483287e-05, Fitness = 1.6986026870940788e-09
Generation 23: Best solution = -3.957616322184884e-05, Fitness = 1.5662726953624208e-09
Generation 24: Best solution = -3.957616322184884e-05, Fitness = 1.5662726953624208e-09
Generation 25: Best solution = 1.0321560969101478e-05, Fitness = 1.0653462083887904e-10
Generation 26: Best solution = -1.008642467092459e-05, Fitness = 1.0173596264223623e-10
Generation 27: Best solution = -1.008642467092459e-05, Fitness = 1.0173596264223623e-10
Generation 28: Best solution = -3.732973988727613e-06, Fitness = 1.3935094800516944e-11
Generation 29: Best solution = -3.732973988727613e-06, Fitness = 1.3935094800516944e-11
Generation 30: Best solution = -3.397754721001625e-06, Fitness = 1.1544737144088829e-11
Generation 31: Best solution = -3.397754721001625e-06, Fitness = 1.1544737144088829e-11
Generation 32: Best solution = -3.397754721001625e-06, Fitness = 1.1544737144088829e-11
Generation 33: Best solution = -1.1742510218670594e-06, Fitness = 1.378865462355833e-12
Generation 34: Best solution = -1.1742510218670594e-06, Fitness = 1.378865462355833e-12
Generation 35: Best solution = -1.1742510218670594e-06, Fitness = 1.378865462355833e-12
Generation 36: Best solution = -1.1742510218670594e-06, Fitness = 1.378865462355833e-12
Generation 37: Best solution = -1.1742510218670594e-06, Fitness = 1.378865462355833e-12
Generation 38: Best solution = 6.457067211882121e-07, Fitness = 4.1693716978763156e-13
Generation 39: Best solution = 6.457067211882121e-07, Fitness = 4.1693716978763156e-13
Generation 40: Best solution = 6.457067211882121e-07, Fitness = 4.1693716978763156e-13
Generation 41: Best solution = -6.241868878925394e-07, Fitness = 3.896092710169736e-13
Generation 42: Best solution = -6.241868878925394e-07, Fitness = 3.896092710169736e-13
Generation 43: Best solution = -6.241868878925394e-07, Fitness = 3.896092710169736e-13
Generation 44: Best solution = -8.229021545498525e-08, Fitness = 6.7716795596278924e-15
Generation 45: Best solution = 7.768669357181376e-08, Fitness = 6.035222358120889e-15
Generation 46: Best solution = 7.768669357181376e-08, Fitness = 6.035222358120889e-15
Generation 47: Best solution = 7.768669357181376e-08, Fitness = 6.035222358120889e-15
Generation 48: Best solution = 7.768669357181376e-08, Fitness = 6.035222358120889e-15
Generation 49: Best solution = 7.768669357181376e-08, Fitness = 6.035222358120889e-15
Generation 50: Best solution = 6.217085905201875e-08, Fitness = 3.865215715265982e-15
Best solution found: 6.217085905201875e-08, Fitness: 3.865215715265982e-15
```

```
#Application: Given a set of stocks determine the optimal portfolio mix to maximize return and minimize risk
import random
import numpy as np

# Fitness function: Calculate Sharpe Ratio (maximize return, minimize risk)
def fitness_function(weights, returns, cov_matrix, risk_free_rate=0):
    # Portfolio return
    portfolio_return = np.dot(weights, returns)

    # Portfolio variance (risk)
    portfolio_variance = np.dot(weights.T, np.dot(cov_matrix, weights))
    portfolio_risk = np.sqrt(portfolio_variance)

    # Sharpe ratio (maximize return, minimize risk)
    sharpe_ratio = (portfolio_return - risk_free_rate) / portfolio_risk
    return sharpe_ratio

# Particle class to represent each particle in the swarm (portfolio weights)
class Particle:
    def __init__(self, num_assets, min_weight, max_weight):
        self.position = np.random.uniform(min_weight, max_weight, num_assets)  # Portfolio weights
        self.velocity = np.random.uniform(-0.1, 0.1, num_assets)  # Portfolio velocity
        self.best_position = self.position  # Best position found by the particle
        self.best_fitness = -float('inf')  # Best fitness value (initially very low)

    def update_velocity(self, global_best_position, inertia_weight, cognitive_coefficient, social_coefficient):
        r1, r2 = random.random(), random.random()
        cognitive_velocity = cognitive_coefficient * r1 * (self.best_position - self.position)
        social_velocity = social_coefficient * r2 * (global_best_position - self.position)
        self.velocity = (inertia_weight * self.velocity) + cognitive_velocity + social_velocity

    def update_position(self, min_weight, max_weight):
        self.position += self.velocity
        # Ensure the weights are between min_weight and max_weight, and the sum of weights is 1
        self.position = np.clip(self.position, min_weight, max_weight)
```

```
            self.position = np.clip(self.position, min_weight, max_weight)
            self.position /= np.sum(self.position)  # Normalize weights to sum to 1

            # Update the best position and fitness if needed
            fitness = fitness_function(self.position, returns, cov_matrix)
            if fitness > self.best_fitness:  # Maximize fitness
                self.best_fitness = fitness
                self.best_position = self.position

    # PSO algorithm to find the optimal portfolio mix
    def particle_swarm_optimization(pop_size, num_assets, returns, cov_matrix, min_weight, max_weight, generations, inertia_weight, cognitive_coe
        # Initialize particles
        swarm = [Particle(num_assets, min_weight, max_weight) for _ in range(pop_size)]

        # Global best position initialized to None
        global_best_position = swarm[0].best_position
        global_best_fitness = swarm[0].best_fitness

        for generation in range(generations):
            for particle in swarm:
                # Update global best position
                if particle.best_fitness > global_best_fitness:
                    global_best_fitness = particle.best_fitness
                    global_best_position = particle.best_position

                # Update particle velocity and position
                particle.update_velocity(global_best_position, inertia_weight, cognitive_coefficient, social_coefficient)
                particle.update_position(min_weight, max_weight)

            # Print the best fitness in the current generation
            print(f"Generation {generation + 1}: Best solution (weights) = {global_best_position}, Fitness = {global_best_fitness}")

        return global_best_position

    # Example usage: optimizing a portfolio of 4 stocks

    # Expected returns for the stocks (e.g., annual returns as decimals)
    returns = np.array([0.10, 0.12, 0.15, 0.08])

    # Covariance matrix of the asset returns (assumed here for 4 stocks)
    cov_matrix = np.array([[0.0004, 0.0002, 0.0001, 0.0003],
                           [0.0002, 0.0005, 0.0002, 0.0004],
                           [0.0001, 0.0002, 0.0006, 0.0002],
                           [0.0003, 0.0004, 0.0002, 0.0007]])

    # Parameters for PSO
    population_size = 30
    num_assets = len(returns)
    min_weight = 0.05  # Minimum weight per asset
    max_weight = 0.40  # Maximum weight per asset
    num_generations = 60
    inertia_weight = 0.5
    cognitive_coefficient = 1.5
    social_coefficient = 1.5

    # Run Particle Swarm Optimization for portfolio optimization
    best_portfolio = particle_swarm_optimization(population_size, num_assets, returns, cov_matrix, min_weight, max_weight, num_generations, inert

    print(f"Best portfolio weights: {best_portfolio}")
    print(f"Expected return: {np.dot(best_portfolio, returns)}")
    print(f"Portfolio risk (standard deviation): {np.sqrt(np.dot(best_portfolio.T, np.dot(cov_matrix, best_portfolio)))}")
```

```
Generation 6: Best solution (weights) = [0.33993396 0.20684163 0.4242227  0.02864128], Fitness = 7.337407904546191
Generation 7: Best solution (weights) = [0.33993396 0.20684163 0.4242227  0.02864128], Fitness = 7.337407904546191
Generation 8: Best solution (weights) = [0.33993396 0.20684163 0.4242227  0.02864128], Fitness = 7.337407904546191
Generation 9: Best solution (weights) = [0.3216712  0.21249834 0.45736064 0.01648268], Fitness = 7.337489338114627
Generation 10: Best solution (weights) = [0.32292594 0.21507887 0.43230695 0.03369467], Fitness = 7.337564462375809
Generation 11: Best solution (weights) = [0.32488684 0.21727135 0.41779278 0.04205225], Fitness = 7.337575624279727
Generation 12: Best solution (weights) = [0.32488684 0.21727135 0.41779278 0.04205225], Fitness = 7.337575624279727
Generation 13: Best solution (weights) = [0.31785622 0.22238094 0.43179577 0.03232614], Fitness = 7.337585324522822
Generation 14: Best solution (weights) = [0.31785622 0.22238094 0.43179577 0.03232614], Fitness = 7.337585324522822
Generation 15: Best solution (weights) = [ 0.32562283  0.21111019  0.47098461 -0.0004757 ], Fitness = 7.337585357246438
Generation 16: Best solution (weights) = [ 0.32562283  0.21111019  0.47098461 -0.0004757 ], Fitness = 7.337585357246438
Generation 17: Best solution (weights) = [ 0.32562283  0.21111019  0.47098461 -0.0004757 ], Fitness = 7.337585357246438
```

```
Generation 23: Best solution (weights) = [ 0.32562283  0.21111019  0.47098461 -0.0004757 ], Fitness = 7.337585357246438
Generation 24: Best solution (weights) = [ 0.32562283  0.21111019  0.47098461 -0.0004757 ], Fitness = 7.337585357246438
Generation 25: Best solution (weights) = [ 0.32562283  0.21111019  0.47098461 -0.0004757 ], Fitness = 7.337585357246438
Generation 26: Best solution (weights) = [ 0.32562283  0.21111019  0.47098461 -0.0004757 ], Fitness = 7.337585357246438
Generation 27: Best solution (weights) = [ 0.32562283  0.21111019  0.47098461 -0.0004757 ], Fitness = 7.337585357246438
Generation 28: Best solution (weights) = [ 0.32562283  0.21111019  0.47098461 -0.0004757 ], Fitness = 7.337585357246438
Generation 29: Best solution (weights) = [ 0.32562283  0.21111019  0.47098461 -0.0004757 ], Fitness = 7.337585357246438
Generation 30: Best solution (weights) = [ 0.32562283  0.21111019  0.47098461 -0.0004757 ], Fitness = 7.337585357246438
Generation 31: Best solution (weights) = [ 0.32562283  0.21111019  0.47098461 -0.0004757 ], Fitness = 7.337585357246438
Generation 32: Best solution (weights) = [ 0.32562283  0.21111019  0.47098461 -0.0004757 ], Fitness = 7.337585357246438
Generation 33: Best solution (weights) = [ 0.32562283  0.21111019  0.47098461 -0.0004757 ], Fitness = 7.337585357246438
Generation 34: Best solution (weights) = [ 0.32562283  0.21111019  0.47098461 -0.0004757 ], Fitness = 7.337585357246438
Generation 35: Best solution (weights) = [ 0.32562283  0.21111019  0.47098461 -0.0004757 ], Fitness = 7.337585357246438
Generation 36: Best solution (weights) = [ 0.32562283  0.21111019  0.47098461 -0.0004757 ], Fitness = 7.337585357246438
Generation 37: Best solution (weights) = [ 0.32562283  0.21111019  0.47098461 -0.0004757 ], Fitness = 7.337585357246438
Generation 38: Best solution (weights) = [ 0.32562283  0.21111019  0.47098461 -0.0004757 ], Fitness = 7.337585357246438
Generation 39: Best solution (weights) = [ 0.32562283  0.21111019  0.47098461 -0.0004757 ], Fitness = 7.337585357246438
Generation 40: Best solution (weights) = [ 0.32562283  0.21111019  0.47098461 -0.0004757 ], Fitness = 7.337585357246438
Generation 41: Best solution (weights) = [ 0.32562283  0.21111019  0.47098461 -0.0004757 ], Fitness = 7.337585357246438
Generation 42: Best solution (weights) = [ 0.32562283  0.21111019  0.47098461 -0.0004757 ], Fitness = 7.337585357246438
Generation 43: Best solution (weights) = [ 0.32562283  0.21111019  0.47098461 -0.0004757 ], Fitness = 7.337585357246438
Generation 44: Best solution (weights) = [ 0.32562283  0.21111019  0.47098461 -0.0004757 ], Fitness = 7.337585357246438
Generation 45: Best solution (weights) = [ 0.32562283  0.21111019  0.47098461 -0.0004757 ], Fitness = 7.337585357246438
Generation 46: Best solution (weights) = [ 0.32562283  0.21111019  0.47098461 -0.0004757 ], Fitness = 7.337585357246438
Generation 47: Best solution (weights) = [ 0.32562283  0.21111019  0.47098461 -0.0004757 ], Fitness = 7.337585357246438
Generation 48: Best solution (weights) = [ 0.32562283  0.21111019  0.47098461 -0.0004757 ], Fitness = 7.337585357246438
Generation 49: Best solution (weights) = [ 0.32562283  0.21111019  0.47098461 -0.0004757 ], Fitness = 7.337585357246438
Generation 50: Best solution (weights) = [ 0.32562283  0.21111019  0.47098461 -0.0004757 ], Fitness = 7.337585357246438
Generation 51: Best solution (weights) = [ 0.32562283  0.21111019  0.47098461 -0.0004757 ], Fitness = 7.337585357246438
Generation 52: Best solution (weights) = [ 0.32562283  0.21111019  0.47098461 -0.0004757 ], Fitness = 7.337585357246438
Generation 53: Best solution (weights) = [ 0.32562283  0.21111019  0.47098461 -0.0004757 ], Fitness = 7.337585357246438
Generation 54: Best solution (weights) = [ 0.32562283  0.21111019  0.47098461 -0.0004757 ], Fitness = 7.337585357246438
Generation 55: Best solution (weights) = [ 0.32562283  0.21111019  0.47098461 -0.0004757 ], Fitness = 7.337585357246438
Generation 56: Best solution (weights) = [ 0.32562283  0.21111019  0.47098461 -0.0004757 ], Fitness = 7.337585357246438
Generation 57: Best solution (weights) = [ 0.32562283  0.21111019  0.47098461 -0.0004757 ], Fitness = 7.337585357246438
Generation 58: Best solution (weights) = [ 0.32562283  0.21111019  0.47098461 -0.0004757 ], Fitness = 7.337585357246438
Generation 59: Best solution (weights) = [ 0.32562283  0.21111019  0.47098461 -0.0004757 ], Fitness = 7.337585357246438
Generation 60: Best solution (weights) = [ 0.32562283  0.21111019  0.47098461 -0.0004757 ], Fitness = 7.337585357246438
Best portfolio weights: [ 0.32562283  0.21111019  0.47098461 -0.0004757 ]
Expected return: 0.12850514095294682
Portfolio risk (standard deviation): 0.017189254076930227
```