```
import random
import numpy as np

def objective_function(position):
    """The function to be minimized."""
    x, y = position
    return x**2 + y**2

def pso(objective_function, dimensions, iterations, population_size, w=0.7, c1=1.4, c2=1.4):
    """
    Particle Swarm Optimization algorithm.

    Args:
        objective_function: The function to be minimized.
        dimensions: The number of dimensions of the search space.
        iterations: The number of iterations to run the algorithm.
        population_size: The number of particles in the swarm.
        w: Inertia weight.
        c1: Cognitive parameter.
        c2: Social parameter.

    Returns:
        A tuple containing the best solution found and its corresponding objective function value.
    """
    particles = []
    for _ in range(population_size):
        position = np.random.uniform(-10, 10, dimensions)
        velocity = np.random.uniform(-1, 1, dimensions)
        particles.append({
            'position': position,
            'velocity': velocity,
            'best_position': position.copy(),
            'best_value': objective_function(position)
        })

    global_best_position = particles[0]['best_position'].copy()
    global_best_value = particles[0]['best_value']

    for _ in range(iterations):
        for particle in particles:

            r1 = random.random()
            r2 = random.random()
            particle['velocity'] = (w * particle['velocity'] +
                                    c1 * r1 * (particle['best_position'] - particle['position']) +
                                    c2 * r2 * (global_best_position - particle['position']))

            particle['position'] = particle['position'] + particle['velocity']

            particle['position'] = np.clip(particle['position'], -10, 10)

            value = objective_function(particle['position'])

            if value < particle['best_value']:
                particle['best_value'] = value
                particle['best_position'] = particle['position'].copy()

            if value < global_best_value:
                global_best_value = value
                global_best_position = particle['position'].copy()

    return global_best_position, global_best_value
```

```
dimensions = 2
iterations = 100
population_size = 50

best_position, best_value = pso(objective_function, dimensions, iterations, population_size)
print(f"Best position found: {best_position}")
print(f"Best value found: {best_value}")
```

```
Best position found: [ 4.04789703e-08 -2.23363404e-08]
Best value found: 2.137459138638845e-15
```