Muhammad Waleed
20b-115-se
SE-B
Artificial Intelligence
Lab#08
Sir Nasir Ud Deen

## Exercise 1:

Run the code provided in the website and use it for the optimization of function provided in this manual and the ones used in previous lab.

### Code

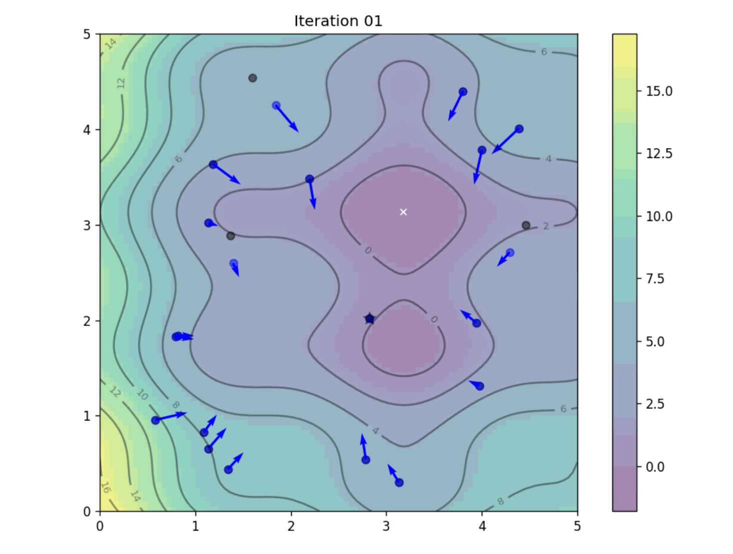
```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation

def f(x,y):
    "Objective function"
    return (x-3.14)**2 + (y-2.72)**2 + np.sin(3*x+1.41) + np.sin(4*y-1.73)

# Compute and plot the function in 3D within [0,5]x[0,5]
x, y = np.array(np.meshgrid(np.linspace(0,5,100), np.linspace(0,5,100)))
z = f(x, y)

# Find the global minimum
x_min = x.ravel()[z.argmin()]
y_min = y.ravel()[z.argmin()]

# Hyper-parameter of the algorithm
c1 = c2 = 0.1
w = 0.8

# Create particles
n_particles = 20
np.random.seed(100)
X = np.random.rand(2, n_particles) * 5
V = np.random.randn(2, n_particles) * 0.1

# Initialize data
pbest = X
pbest_obj = f(X[0], X[1])
gbest = pbest[:, pbest_obj.argmin()]
gbest_obj = pbest_obj.min()

def update():
    "Function to do one iteration of particle swarm optimization"
    global V, X, pbest, pbest_obj, gbest, gbest_obj
    # Update params
    r1, r2 = np.random.rand(2)
    V = w * V + c1*r1*(pbest - X) + c2*r2*(gbest.reshape(-1,1)-X)
    X = X + V
    obj = f(X[0], X[1])
```

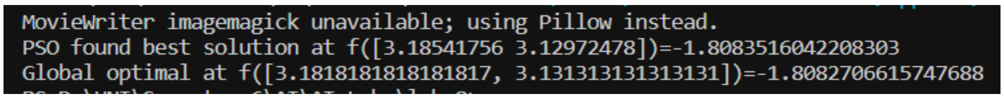
Muhammad Waleed
20b-115-se
SE-B
Artificial Intelligence
Lab#08
Sir Nasir Ud Deen

```python
    pbest[:, (pbest_obj >= obj)] = X[:, (pbest_obj >= obj)]
    pbest_obj = np.array([pbest_obj, obj]).min(axis=0)
    gbest = pbest[:, pbest_obj.argmin()]
    gbest_obj = pbest_obj.min()

# Set up base figure: The contour map
fig, ax = plt.subplots(figsize=(8,6))
fig.set_tight_layout(True)
img = ax.imshow(z, extent=[0, 5, 0, 5], origin='lower', cmap='viridis',
alpha=0.5)
fig.colorbar(img, ax=ax)
ax.plot([x_min], [y_min], marker='x', markersize=5, color="white")
contours = ax.contour(x, y, z, 10, colors='black', alpha=0.4)
ax.clabel(contours, inline=True, fontsize=8, fmt="%.0f")
pbest_plot = ax.scatter(pbest[0], pbest[1], marker='o', color='black',
alpha=0.5)
p_plot = ax.scatter(X[0], X[1], marker='o', color='blue', alpha=0.5)
p_arrow = ax.quiver(X[0], X[1], V[0], V[1], color='blue', width=0.005,
angles='xy', scale_units='xy', scale=1)
gbest_plot = plt.scatter([gbest[0]], [gbest[1]], marker='*', s=100,
color='black', alpha=0.4)
ax.set_xlim([0,5])
ax.set_ylim([0,5])

def animate(i):
    "Steps of PSO: algorithm update and show in plot"
    title = 'Iteration {:02d}'.format(i)
    # Update params
    update()
    # Set picture
    ax.set_title(title)
    pbest_plot.set_offsets(pbest.T)
    p_plot.set_offsets(X.T)
    p_arrow.set_offsets(X.T)
    p_arrow.set_UVC(V[0], V[1])
    gbest_plot.set_offsets(gbest.reshape(1,-1))
    return ax, pbest_plot, p_plot, p_arrow, gbest_plot

anim = FuncAnimation(fig, animate, frames=list(range(1,50)), interval=500,
blit=False, repeat=True)
anim.save("task1.gif", dpi=120, writer="imagemagick")

print("PSO found best solution at f({})={}".format(gbest, gbest_obj))
print("Global optimal at f({})={}".format([x_min,y_min], f(x_min,y_min)))
```

Muhammad Waleed
20b-115-se
SE-B
Artificial Intelligence
Lab#08
Sir Nasir Ud Deen

## Output





```
MovieWriter imagemagick unavailable; using Pillow instead.
PSO found best solution at f([3.18541756 3.12972478])=-1.8083516042208303
Global optimal at f([3.1818181818181817, 3.131313131313131])=-1.8082706615747688
```

## Exercise 1 Lab 8 Functions

### Code

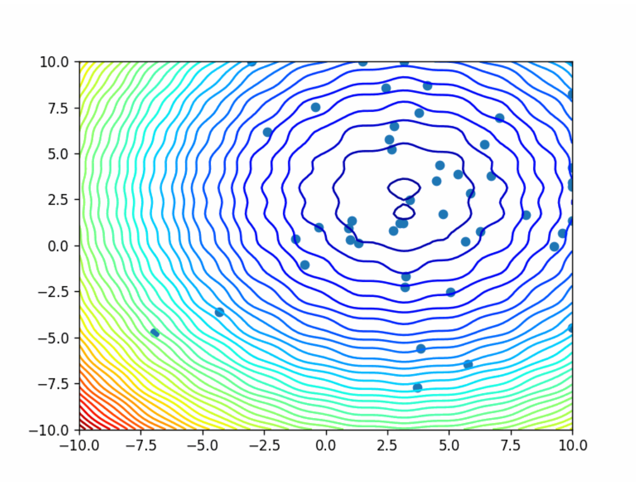
```
# Exercise 1
# Lab 8 Functions
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation

def f(x, y):
    return (x - 3.14) ** 2 + (y - 2.72) ** 2 + np.sin(3 * x + 1.41) + np.sin(4
* y - 1.73)

def pso_animation():
    search_min = -10
    search_max = 10
    num_dims = 2
    swarm_size = 50
    max_iter = 100
    c1 = 2.0
    c2 = 2.0
    w = 0.7

    swarm_pos = np.random.uniform(search_min, search_max, (swarm_size,
num_dims))
```

Muhammad Waleed
20b-115-se
SE-B
Artificial Intelligence
Lab#08
Sir Nasir Ud Deen

```python
    swarm_vel = np.zeros((swarm_size, num_dims))
    swarm_best_pos = np.copy(swarm_pos)
    swarm_best_fitness = np.full((swarm_size,), np.inf)
    global_best_pos = np.zeros((num_dims,))
    global_best_fitness = np.inf

    fig, ax = plt.subplots()
    x = np.linspace(search_min, search_max, 100)
    y = np.linspace(search_min, search_max, 100)
    X, Y = np.meshgrid(x, y)
    Z = f(X, Y)
    ax.contour(X, Y, Z, levels=50, cmap='jet')
    ax.set_xlim([search_min, search_max])
    ax.set_ylim([search_min, search_max])
    sc = ax.scatter([], [])

    def animate(i):
        nonlocal swarm_pos, swarm_vel, swarm_best_pos, swarm_best_fitness,
global_best_pos, global_best_fitness
        fitness = f(swarm_pos[:, 0], swarm_pos[:, 1])
        better_fitness_mask = fitness < swarm_best_fitness
        swarm_best_pos[better_fitness_mask] = swarm_pos[better_fitness_mask]
        swarm_best_fitness[better_fitness_mask] = fitness[better_fitness_mask]
        best_particle_idx = np.argmin(swarm_best_fitness)
        if swarm_best_fitness[best_particle_idx] < global_best_fitness:
            global_best_pos = np.copy(swarm_best_pos[best_particle_idx])
            global_best_fitness = swarm_best_fitness[best_particle_idx]
        r1 = np.random.uniform(size=(swarm_size, num_dims))
        r2 = np.random.uniform(size=(swarm_size, num_dims))
        swarm_vel = w * swarm_vel + c1 * r1 * (swarm_best_pos - swarm_pos) +
c2 * r2 * (global_best_pos - swarm_pos)
        swarm_pos += swarm_vel
        swarm_pos = np.clip(swarm_pos, search_min, search_max)

        sc.set_offsets(swarm_pos)
        return sc,

    ani = animation.FuncAnimation(fig, animate, frames=max_iter, interval=50,
blit=True)
    ani.save("Task1_2.gif", dpi=120, writer="imagemagick")
    plt.show()

pso_animation()
```

Muhammad Waleed
20b-115-se
SE-B
Artificial Intelligence
Lab#08
Sir Nasir Ud Deen

### Output

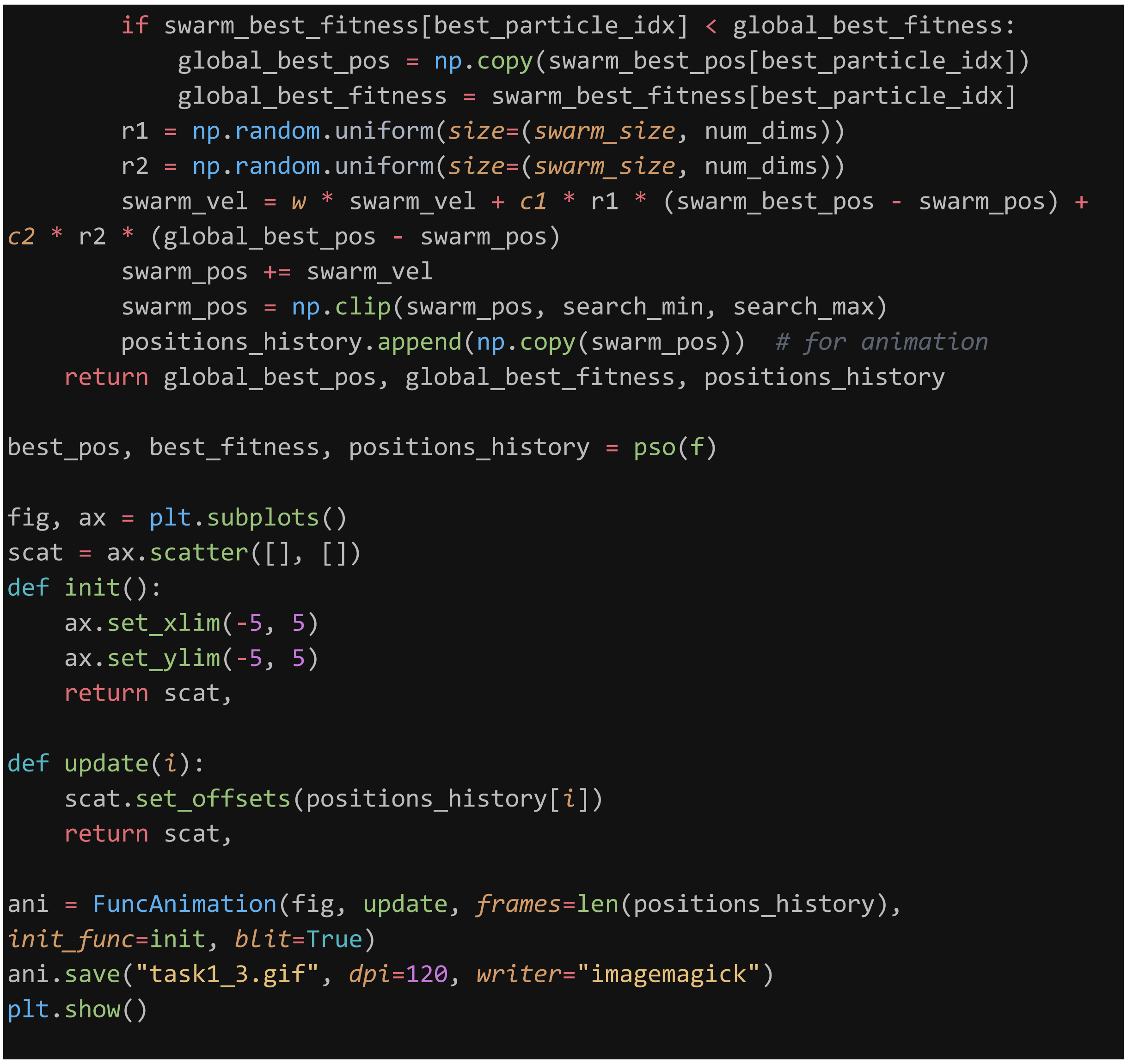


## Exercise 1 Lab 7 Functions

### Code

```
# Exercise 1
# Lab 7 Functions
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation

def f(x, y):
    return (1 - x) ** 2 + 100 * (y - x ** 2) ** 2

def pso(fitness_func, swarm_size=50, max_iter=100, c1=2.0, c2=2.0, w=0.7):
    search_min = -5
    search_max = 5
    num_dims = 2
    swarm_pos = np.random.uniform(search_min, search_max, (swarm_size,
num_dims))
    swarm_vel = np.zeros((swarm_size, num_dims))
    swarm_best_pos = np.copy(swarm_pos)
    swarm_best_fitness = np.full((swarm_size,), np.inf)
    global_best_pos = np.zeros((num_dims,))
    global_best_fitness = np.inf
    positions_history = []  # for animation
    for i in range(max_iter):
        fitness = fitness_func(swarm_pos[:, 0], swarm_pos[:, 1])
        better_fitness_mask = fitness < swarm_best_fitness
        swarm_best_pos[better_fitness_mask] = swarm_pos[better_fitness_mask]
        swarm_best_fitness[better_fitness_mask] = fitness[better_fitness_mask]
        best_particle_idx = np.argmin(swarm_best_fitness)
```

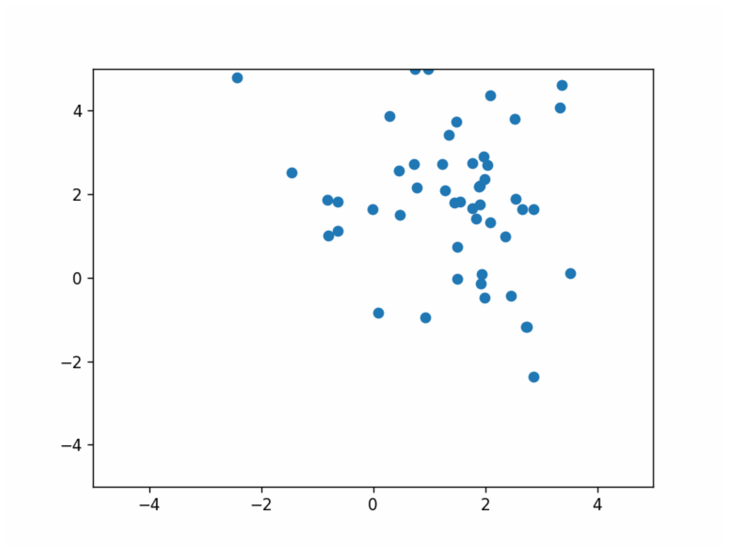
Muhammad Waleed
20b-115-se
SE-B
Artificial Intelligence
Lab#08
Sir Nasir Ud Deen



```python
        if swarm_best_fitness[best_particle_idx] < global_best_fitness:
            global_best_pos = np.copy(swarm_best_pos[best_particle_idx])
            global_best_fitness = swarm_best_fitness[best_particle_idx]
        r1 = np.random.uniform(size=(swarm_size, num_dims))
        r2 = np.random.uniform(size=(swarm_size, num_dims))
        swarm_vel = w * swarm_vel + c1 * r1 * (swarm_best_pos - swarm_pos) +
c2 * r2 * (global_best_pos - swarm_pos)
        swarm_pos += swarm_vel
        swarm_pos = np.clip(swarm_pos, search_min, search_max)
        positions_history.append(np.copy(swarm_pos))  # for animation
    return global_best_pos, global_best_fitness, positions_history

best_pos, best_fitness, positions_history = pso(f)

fig, ax = plt.subplots()
scat = ax.scatter([], [])
def init():
    ax.set_xlim(-5, 5)
    ax.set_ylim(-5, 5)
    return scat,

def update(i):
    scat.set_offsets(positions_history[i])
    return scat,

ani = FuncAnimation(fig, update, frames=len(positions_history),
init_func=init, blit=True)
ani.save("task1_3.gif", dpi=120, writer="imagemagick")
plt.show()
```

Muhammad Waleed
20b-115-se
SE-B
Artificial Intelligence
Lab#08
Sir Nasir Ud Deen

## Output



## Exercise 2

Use matplotlib to plot the objective functions and creating an animation showing the convergence process. Like the one shown in the site
Swarm Intelligence: Coding and Visualising Particle Swarm Optimisation in Python | by Ken Moriwaki | Towards Data Science

## Code from Website

```
import random
import numpy as np
from matplotlib import pyplot as plt
from matplotlib import animation


def fitness_function(x1, x2):
    f1 = x1 + 2 * -x2 + 3
    f2 = 2 * x1 + x2 - 8
    z = f1 ** 2 + f2 ** 2
    return z


def update_velocity(particle, velocity, pbest, gbest, w_min=0.5, max=1.0,
c=0.1):
    num_particle = len(particle)
```

Muhammad Waleed
20b-115-se
SE-B
Artificial Intelligence
Lab#08
Sir Nasir Ud Deen

```
    new_velocity = np.array([0.0 for i in range(num_particle)])
    r1 = random.uniform(0, max)
    r2 = random.uniform(0, max)
    w = random.uniform(w_min, max)
    c1 = c
    c2 = c
    for i in range(num_particle):
        new_velocity[i] = w * velocity[i] + c1 * r1 * (pbest[i] - particle[i])
+ c2 * r2 * (gbest[i] - particle[i])
    return new_velocity


def update_position(particle, velocity):
    new_particle = particle + velocity
    return new_particle


def pso_2d(population, dimension, position_min, position_max, generation,
fitness_criterion):
    particles = [[random.uniform(position_min, position_max) for j in
range(dimension)] for i in range(population)]
    pbest_position = particles
    pbest_fitness = [fitness_function(p[0], p[1]) for p in particles]
    gbest_index = np.argmin(pbest_fitness)
    gbest_position = pbest_position[gbest_index]
    velocity = [[0.0 for j in range(dimension)] for i in range(population)]
    for t in range(generation):
        if np.average(pbest_fitness) <= fitness_criterion:
            break
        else:
            for n in range(population):
                velocity[n] = update_velocity(particles[n], velocity[n],
pbest_position[n], gbest_position)
                particles[n] = update_position(particles[n], velocity[n])
        pbest_fitness = [fitness_function(p[0], p[1]) for p in particles]
        pbest_position = [particles[i] if pbest_fitness[i] <
fitness_function(pbest_position[i][0], pbest_position[i][1]) else
pbest_position[i] for i in range(population)]
        gbest_index = np.argmin(pbest_fitness)
        gbest_position = pbest_position[gbest_index]
    print('Global Best Position: ', gbest_position)
    print('Best Fitness Value: ', min(pbest_fitness))
    print('Average Particle Best Fitness Value: ', np.average(pbest_fitness))
    print('Number of Generation: ', t)
    return particles, pbest_position, pbest_fitness
```

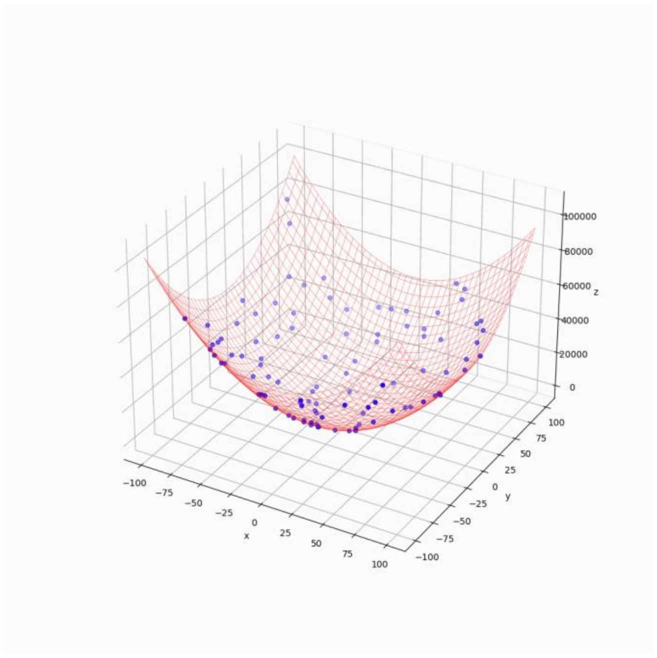
Muhammad Waleed
20b-115-se
SE-B
Artificial Intelligence
Lab#08
Sir Nasir Ud Deen

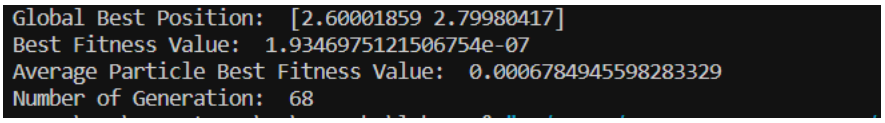
```
population = 100
dimension = 2
position_min = -100.0
position_max = 100.0
generation = 400
fitness_criterion = 10e-4

particles = [[random.uniform(position_min, position_max) for j in
range(dimension)] for i in range(population)]
pbest_position = particles
pbest_fitness = [fitness_function(p[0],p[1]) for p in particles]
gbest_index = np.argmin(pbest_fitness)
gbest_position = pbest_position[gbest_index]
velocity = [[0.0 for j in range(dimension)] for i in range(population)]

fig = plt.figure(figsize=(10, 10))
ax = fig.add_subplot(111, projection='3d')
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('z')
x = np.linspace(position_min, position_max, 80)
y = np.linspace(position_min, position_max, 80)
X, Y = np.meshgrid(x, y)
Z= fitness_function(X,Y)
ax.plot_wireframe(X, Y, Z, color='r', linewidth=0.2)
images = []

for t in range(generation):
    if np.average(pbest_fitness) <= fitness_criterion:
        break
    else:
        for n in range(population):
            velocity[n] = update_velocity(particles[n], velocity[n],
pbest_position[n], gbest_position)
            particles[n] = update_position(particles[n], velocity[n])
        pbest_fitness = [fitness_function(p[0],p[1]) for p in particles]
        for i in range(population):
            if pbest_fitness[i] < fitness_function(pbest_position[i][0],
pbest_position[i][1]):
                pbest_position[i] = particles[i]
        gbest_index = np.argmin(pbest_fitness)
        gbest_position = pbest_position[gbest_index]

    image = ax.scatter3D([particles[n][0] for n in range(population)],
```

Muhammad Waleed
20b-115-se
SE-B
Artificial Intelligence
Lab#08
Sir Nasir Ud Deen

```
                        [particles[n][1] for n in range(population)],
                        [fitness_function(particles[n][0],particles[n][1]) for
n in range(population)], c='b')
    images.append([image])

animated_image = animation.ArtistAnimation(fig, images)
animated_image.save('./task2.gif', writer='pillow')

print('Global Best Position: ', gbest_position)
print('Best Fitness Value: ', min(pbest_fitness))
print('Average Particle Best Fitness Value: ', np.average(pbest_fitness))
print('Number of Generation: ', t+1)
```

## Output





```
Global Best Position:  [2.60001859 2.79980417]
Best Fitness Value:  1.9346975121506754e-07
Average Particle Best Fitness Value:  0.0006784945598283329
Number of Generation:  68
```

## Exercise 2 Using matplotlib to plot the objective functions and creating an animation showing the convergence process.

### Code

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
```

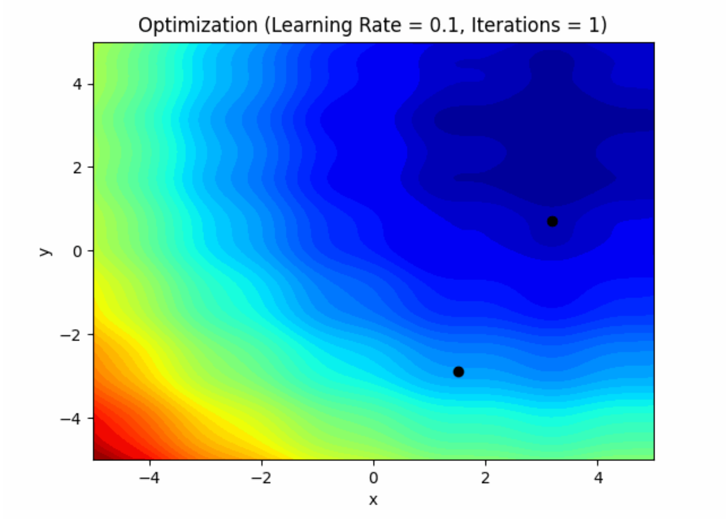
Muhammad Waleed
20b-115-se
SE-B
Artificial Intelligence
Lab#08
Sir Nasir Ud Deen

```python
def objective(x, y):
    return (x - 3.14) ** 2 + (y - 2.72) ** 2 + np.sin(3 * x + 1.41) + np.sin(4
* y - 1.73)


x_range = np.linspace(-5, 5, 100)
y_range = np.linspace(-5, 5, 100)

X, Y = np.meshgrid(x_range, y_range)

Z = objective(X, Y)


fig, ax = plt.subplots()
ax.contourf(X, Y, Z, levels=50, cmap='jet')
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_title('Objective Function')


def optimize(learning_rate, num_iterations):

    x = np.random.uniform(-5, 5)
    y = np.random.uniform(-5, 5)

    history = []

    for i in range(num_iterations):
        gradient_x = 2 * (x - 3.14) + 3 * np.cos(3 * x + 1.41)
        gradient_y = 2 * (y - 2.72) + 4 * np.cos(4 * y - 1.73)

        x -= learning_rate * gradient_x
        y -= learning_rate * gradient_y

        history.append((x, y))

    return (x, y), history


def animate(frame):
    learning_rate = 0.1
    num_iterations = frame + 1
    (x, y), history = optimize(learning_rate, num_iterations)
```

Muhammad Waleed
20b-115-se
SE-B
Artificial Intelligence
Lab#08
Sir Nasir Ud Deen

```
    path = np.array(history)
    ax.plot(path[:, 0], path[:, 1], color='black', alpha=0.1)
    ax.plot(x, y, 'o', color='black')
    ax.set_title(f'Optimization (Learning Rate = {learning_rate}, Iterations =
{num_iterations})')
ani = FuncAnimation(fig, animate, frames=100, interval=100)
ani.save('./task2_1.gif', writer='pillow')
plt.show()
```

## Output



## Exercise 3

Use your code to find the optimal solutions of all the optimization functions in the class lecture of week 4.

## Exercise 3 from Week 4 Lecture

### Trajectory (S-metaheuristics) Algorithm 1

### Code

```
# Trajectory (S-metaheuristics)

import random
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation

def objective_function(x):
    return x ** 2

def generate_candidates(current_solution):
```

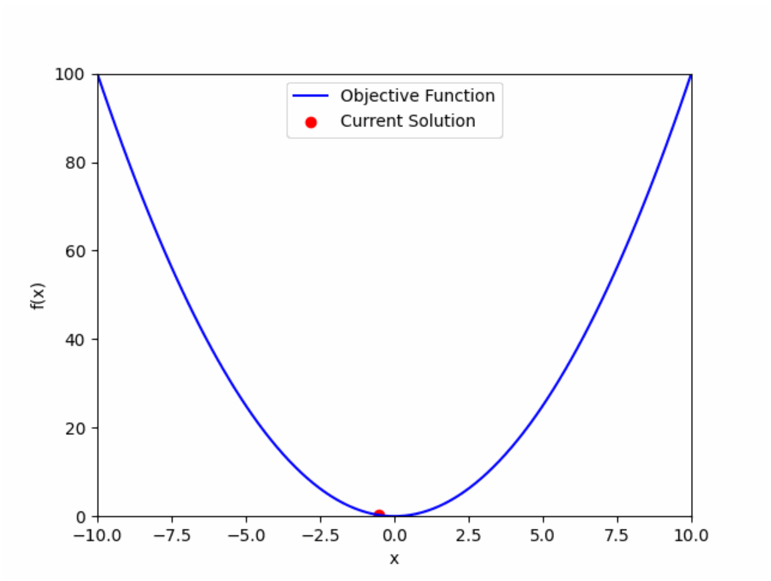
Muhammad Waleed
20b-115-se
SE-B
Artificial Intelligence
Lab#08
Sir Nasir Ud Deen

```python
    candidates = []
    for _ in range(10):
        candidate = random.uniform(-10, 10)
        candidates.append(candidate)
    return candidates

def select_candidate(candidates):
    selected_candidate = min(candidates, key=objective_function)
    return selected_candidate

def s_metaheuristic(initial_solution, stopping_criteria):
    current_solution = initial_solution
    iteration = 0

    while True:
        candidates = generate_candidates(current_solution)
        selected_candidate = select_candidate(candidates)
        current_solution = selected_candidate
        iteration += 1
        stopping_criteria = True
        if stopping_criteria:
            break

    best_solution = current_solution
    return best_solution

def update_plot(solution):
    plt.cla()

    x = np.linspace(-10, 10, 100)
    y = objective_function(x)
    plt.plot(x, y, 'b-', label='Objective Function')

    plt.scatter(solution, objective_function(solution), c='r', label='Current
Solution')

    plt.xlim(-10, 10)
    plt.ylim(0, 100)

    plt.xlabel('x')
    plt.ylabel('f(x)')
    plt.legend()

initial_solution = 0
```

Muhammad Waleed
20b-115-se
SE-B
Artificial Intelligence
Lab#08
Sir Nasir Ud Deen

```
stopping_criteria = False

fig, ax = plt.subplots()

def animate(frame):
    best_solution = s_metaheuristic(initial_solution, stopping_criteria)
    update_plot(best_solution)
    if stopping_criteria:
        anim.event_source.stop()
anim = FuncAnimation(fig, animate, frames=range(1), repeat=False)
anim.save('./task3.gif', writer='pillow')
plt.show()
```

## Output



## A Population-based metaheuristics (P-metaheuristics) Algorithm 2

### Code

```
#  a population-based metaheuristic (P-metaheuristic) algorithm
import random
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation

def objective_function(x):
    return x ** 2

def generate_population(population_size):
    population = []
    for _ in range(population_size):
```

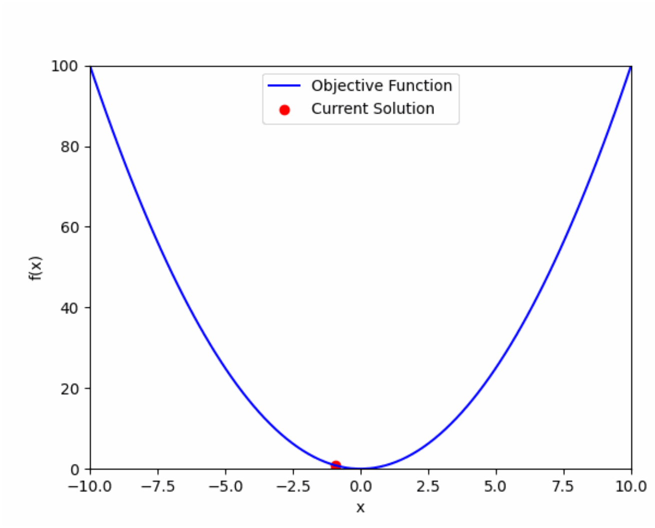
Muhammad Waleed
20b-115-se
SE-B
Artificial Intelligence
Lab#08
Sir Nasir Ud Deen

```
        solution = random.uniform(-10, 10)  # Example: Generate random
solutions within range [-10, 10]
        population.append(solution)
    return population

def replace_population(population):
    # Generate a new population based on the current population
    new_population = population.copy()
    return new_population

def preselect_population(population):
    new_population = population.copy()
    return new_population

def p_metaheuristic(population_size, stopping_criteria):
    population = generate_population(population_size)
    iteration = 0

    while True:
        population = replace_population(population)
        population = preselect_population(population)
        iteration += 1

        stopping_criteria = True
        if stopping_criteria:
            break

    best_solution = min(population, key=objective_function)
    return best_solution

def update_plot(solution):
    plt.cla()
    x = np.linspace(-10, 10, 100)
    y = objective_function(x)
    plt.plot(x, y, 'b-', label='Objective Function')

    plt.scatter(solution, objective_function(solution), c='r', label='Current
Solution')
    plt.xlim(-10, 10)
    plt.ylim(0, 100)
    plt.xlabel('x')
    plt.ylabel('f(x)')
    plt.legend()
```

Muhammad Waleed
20b-115-se
SE-B
Artificial Intelligence
Lab#08
Sir Nasir Ud Deen

```
population_size = 10
stopping_criteria = False
fig, ax = plt.subplots()

def animate(frame):
    best_solution = p_metaheuristic(population_size, stopping_criteria)
    update_plot(best_solution)
    if stopping_criteria:
        anim.event_source.stop()
anim = FuncAnimation(fig, animate, frames=range(1), repeat=False)
anim.save('./task3_1.gif', writer='pillow')
plt.show()
```

## Output



## A Single-based metaheuristics (S-metaheuristics) Algorithm 3

### Code

```
# a single-based metaheuristic (S-metaheuristic)

import random
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation

def objective_function(x):
    return x ** 2

def generate_candidates(current_solutions):
    candidates = []
    for solution in current_solutions:
```

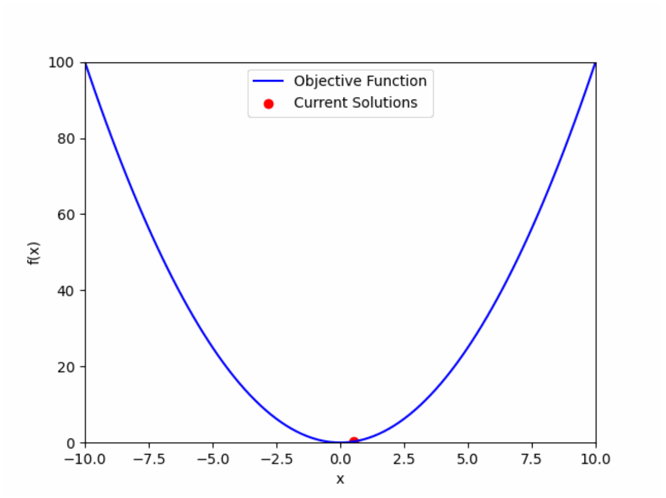
Muhammad Waleed
20b-115-se
SE-B
Artificial Intelligence
Lab#08
Sir Nasir Ud Deen

```python
        for _ in range(10):  # Generate 10 candidate solutions for each current solution
            candidate = random.uniform(-10, 10)  # Example: Generate random candidates within range [-10, 10]
            candidates.append(candidate)
    return candidates

def select_solution(candidates):
    selected_solution = min(candidates, key=objective_function)
    return selected_solution

def s_metaheuristic(initial_solutions, stopping_criteria):
    current_solutions = initial_solutions.copy()
    iteration = 0

    while True:
        candidates = generate_candidates(current_solutions)
        selected_solution = select_solution(candidates)
        current_solutions = [selected_solution] * len(current_solutions)
        iteration += 1

        stopping_criteria = True
        if stopping_criteria:
            break

    best_solution = min(current_solutions, key=objective_function)
    return best_solution

def update_plot(solution):
    plt.cla()
    x = np.linspace(-10, 10, 100)
    y = objective_function(x)
    plt.plot(x, y, 'b-', label='Objective Function')
    plt.scatter(solution, objective_function(solution), c='r', label='Current Solutions')

    plt.xlim(-10, 10)
    plt.ylim(0, 100)

    plt.xlabel('x')
    plt.ylabel('f(x)')
    plt.legend()

initial_solutions = [0, 0]
```

Muhammad Waleed
20b-115-se
SE-B
Artificial Intelligence
Lab#08
Sir Nasir Ud Deen

```python
stopping_criteria = False

fig, ax = plt.subplots()

def animate(frame):
    best_solution = s_metaheuristic(initial_solutions, stopping_criteria)
    update_plot(best_solution)
    if stopping_criteria:
        anim.event_source.stop()

anim = FuncAnimation(fig, animate, frames=range(1), repeat=False)
anim.save('./task3_2.gif', writer='pillow')
plt.show()
```

### Output



## Exercise 4

Find the average convergence time for each function.

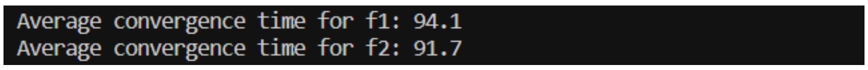
### Code

```python
# Exercise 4
import numpy as np

def f1(x, y):
    return (x - 1) ** 2 + y ** 2

def f2(x, y):
    return (1 - x) ** 2 + 100 * (y - x ** 2) ** 2

def pso(fitness_func, swarm_size=50, max_iter=100, c1=2.0, c2=2.0, w=0.7):
    search_min = -5
    search_max = 5
    num_dims = 2
```

Muhammad Waleed
20b-115-se
SE-B
Artificial Intelligence
Lab#08
Sir Nasir Ud Deen

```
    swarm_pos = np.random.uniform(search_min, search_max, (swarm_size,
num_dims))
    swarm_vel = np.zeros((swarm_size, num_dims))
    swarm_best_pos = np.copy(swarm_pos)
    swarm_best_fitness = np.full((swarm_size,), np.inf)
    global_best_pos = np.zeros((num_dims,))
    global_best_fitness = np.inf
    convergence_time = 0
    for i in range(max_iter):
        fitness = fitness_func(swarm_pos[:, 0], swarm_pos[:, 1])
        better_fitness_mask = fitness < swarm_best_fitness
        swarm_best_pos[better_fitness_mask] = swarm_pos[better_fitness_mask]
        swarm_best_fitness[better_fitness_mask] = fitness[better_fitness_mask]
        best_particle_idx = np.argmin(swarm_best_fitness)
        if swarm_best_fitness[best_particle_idx] < global_best_fitness:
            global_best_pos = np.copy(swarm_best_pos[best_particle_idx])
            global_best_fitness = swarm_best_fitness[best_particle_idx]
            convergence_time = i
        r1 = np.random.uniform(size=(swarm_size, num_dims))
        r2 = np.random.uniform(size=(swarm_size, num_dims))
        swarm_vel = w * swarm_vel + c1 * r1 * (swarm_best_pos - swarm_pos) +
c2 * r2 * (global_best_pos - swarm_pos)
        swarm_pos += swarm_vel
        swarm_pos = np.clip(swarm_pos, search_min, search_max)
    return global_best_pos, global_best_fitness, convergence_time

# Run PSO multiple times to get the average convergence time
num_runs = 10
f1_convergence_times = []
f2_convergence_times = []
for i in range(num_runs):
    _, _, convergence_time = pso(f1)
    f1_convergence_times.append(convergence_time)
    _, _, convergence_time = pso(f2)
    f2_convergence_times.append(convergence_time)

# Print the average convergence time for each function
print("Average convergence time for f1:", np.mean(f1_convergence_times))
print("Average convergence time for f2:", np.mean(f2_convergence_times))
```

## Output



```
Average convergence time for f1: 94.1
Average convergence time for f2: 91.7
```