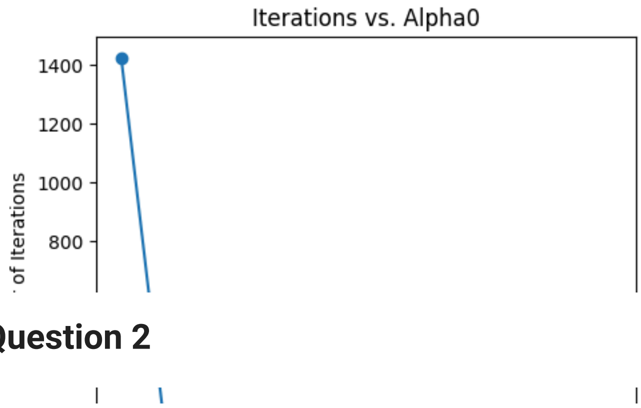
```
import numpy as np
import matplotlib.pyplot as plt
```

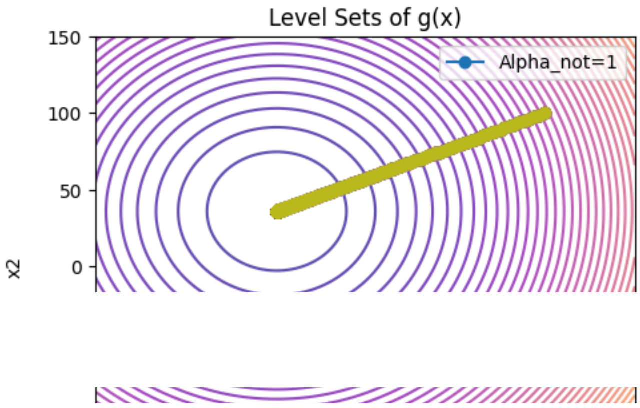
Question 1

```
# Define the function g(x)
def gradient_g(x):
    return np.array([2 * (x[0] + 49), 2 * (x[1] - 36)])

# define its gradient
def g(x):
    return (x[0] + 49)**2 + (x[1] - 36)**2

# Backtracking (inexact) line search algorithm
def backtracking_line_search(x, pk, alpha0, rho, gamma):
    alpha = alpha0
    while g(x + alpha * pk) > g(x) + gamma * alpha * gradient_g(x).dot(pk):
        alpha *= rho
    return alpha

# Gradient descent with backtracking line search
def gradient_descent_backtracking_line_search(x0, tolerance, alpha0, rho, gamma):
    x = x0
    iterations = 0
    trajectory = [x]
    while np.linalg.norm(gradient_g(x)) > tolerance:
        pk = -gradient_g(x)
        alpha = backtracking_line_search(x, pk, alpha0, rho, gamma)
        x = x + alpha * pk
        iterations += 1
        trajectory.append(x)
    return np.array(trajectory), x, g(x), iterations

# Experiment setup
x0 = np.array([100, 100])
tolerance = 1e-10
rho = 0.5
gamma = 0.5
alpha_values = [1, 0.9, 0.75, 0.6, 0.5, 0.4, 0.25, 0.1, 0.01]

# Run experiments and record results
minimizers = []
objective_values = []
iteration_counts = []

for alpha0 in alpha_values:
    trajectory_x, x_min, f_min, iterations = gradient_descent_backtracking_line_search(x0, tolerance, alpha0, rho, gamma)
    minimizers.append(x_min)
    objective_values.append(f_min)
    iteration_counts.append(iterations)

# Print  results for each alpha0
for i, alpha0 in enumerate(alpha_values):
    print(f"Alpha_not = {alpha0}")
    print(f"Minimizer: {minimizers[i]}")
    print(f"Objective Function Value: {objective_values[i]}")
    print(f"Number_of_Iterations: {iteration_counts[i]}")
    print("=" * 40)

# Plotting
plt.figure(figsize=(10, 8))

# Plot number of iterations vs. alpha0
plt.subplot(2, 2, 1)
plt.plot(alpha_values, iteration_counts, marker='o', linestyle='-')
plt.xlabel('Alpha_not')
plt.ylabel('Number of Iterations')
plt.title('Iterations vs. Alpha0')

# Plot level sets of the function g(x)
x_range = np.linspace(-150, 150, 400)
y_range = np.linspace(-150, 150, 400)
X, Y = np.meshgrid(x_range, y_range)
Z = (X + 49)**2 + (Y - 36)**2
plt.subplot(2, 2, 2)
plt.contour(X, Y, Z, levels=50, cmap='plasma', alpha=0.7)
plt.xlabel('x1')
plt.ylabel('x2')
```

```
    plt.title('Level Sets of g(x)')

    # Plot trajectory of optimization for inexact line search and fixed step length
    for i, alpha0 in enumerate(alpha_values):
        plt.subplot(2, 2, 2)
        plt.plot(trajectory_x[:, 0], trajectory_x[:, 1], label=f'Alpha_not={alpha0}', marker='o')
        if i == 0:
            plt.legend()

    plt.tight_layout()
    plt.show()
```

```
Alpha_not = 1
Minimizer: [-49.  36.]
Objective Function Value: 0.0
Number_of_Iterations: 1
========================================
Alpha_not = 0.9
Minimizer: [-49.  36.]
Objective Function Value: 2.629978954295262e-22
Number_of_Iterations: 13
========================================
Alpha_not = 0.75
Minimizer: [-49.  36.]
Objective Function Value: 1.3595230355191855e-21
Number_of_Iterations: 21
========================================
Alpha_not = 0.6
Minimizer: [-49.  36.]
Objective Function Value: 8.948165620682675e-22
Number_of_Iterations: 32
========================================
Alpha_not = 0.5
Minimizer: [-49.  36.]
Objective Function Value: 0.0
Number_of_Iterations: 1
========================================
Alpha_not = 0.4
Minimizer: [-49.  36.]
Objective Function Value: 1.8070003829032932e-21
Number_of_Iterations: 18
========================================
Alpha_not = 0.25
Minimizer: [-49.  36.]
Objective Function Value: 1.3595230355191855e-21
Number_of_Iterations: 42
========================================
Alpha_not = 0.1
Minimizer: [-49.  36.]
Objective Function Value: 1.672152121111579e-21
Number_of_Iterations: 130
========================================
Alpha_not = 0.01
Minimizer: [-49.  36.]
Objective Function Value: 2.4913093745480103e-21
Number_of_Iterations: 1426
========================================
```





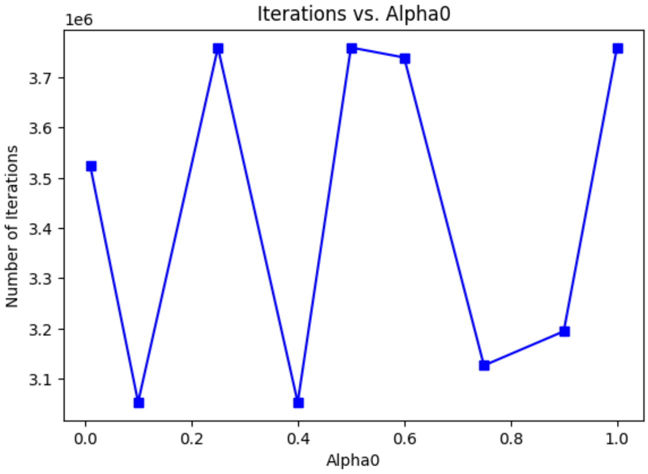
Question 2

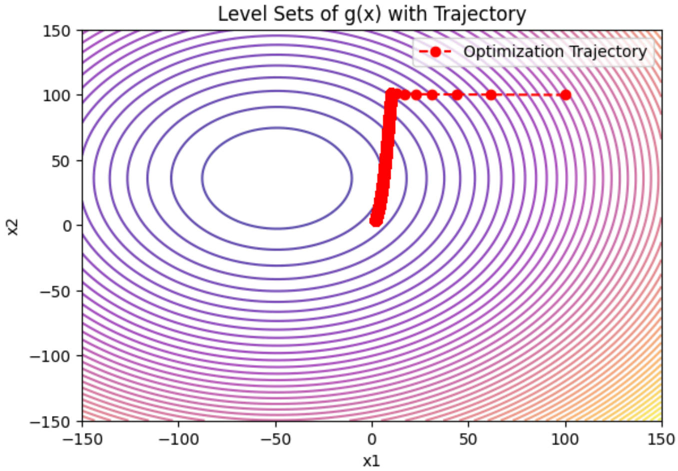
```
def gradient_g(x):
        df_dx1 = -1024 * x[0] * (x[1] - x[0]**2) - 2 * (2 - x[0])
        df_dx2 = 512 * (x[1] - x[0]**2)
        return np.array([df_dx1, df_dx2])

def g(x):
    return (256 * (x[1] - x[0]**2)**2 + (2 - x[0])**2)

def gradient_g(x):
        df_dx1 = -1024 * x[0] * (x[1] - x[0]**2) - 2 * (2 - x[0])
        df_dx2 = 512 * (x[1] - x[0]**2)
        return np.array([df_dx1, df_dx2])

# Algorithm for  line search algorithm Backtracking (inexact)
def backtracking_line_search(x, pk, alpha0, rho, gamma):
    alpha = alpha0
    while g(x + alpha * pk) > g(x) + gamma * alpha * gradient_g(x).dot(pk):
        alpha *= rho
    return alpha

# Function for Gradient descent with backtracking line search
def gradient_descent_backtracking_line_search(x0, tolerance, alpha0, rho, gamma):
    x = x0
    iterations = 0
    trajectory = [x]
    while np.linalg.norm(gradient_g(x)) > tolerance:
        pk = -gradient_g(x)
        alpha = backtracking_line_search(x, pk, alpha0, rho, gamma)
        x = x + alpha * pk
        iterations += 1
        trajectory.append(x)
    return np.array(trajectory), x, g(x), iterations


x0 = np.array([100, 100])
tolerance = 1e-10
rho = 0.5
gamma = 0.5
alpha_values = [1, 0.9, 0.75, 0.6, 0.5, 0.4, 0.25, 0.1, 0.01]

# Append the result in the list which will be useful in plotting the graph
minimizers = []
objective_values = []
iteration_counts = []

for alpha0 in alpha_values:
    trajectory_x, x_min, f_min, iterations = gradient_descent_backtracking_line_search(x0, tolerance, alpha0, rho, gamma)
    minimizers.append(x_min)
    objective_values.append(f_min)
    iteration_counts.append(iterations)

# Print the result
for i, alpha0 in enumerate(alpha_values):
    print(f"Alpha_not = {alpha0}")
    print(f"Minimizer: {minimizers[i]}")
    print(f"Objective_Function_Value: {objective_values[i]}")
    print(f"Number_of_Iterations: {iteration_counts[i]}")
    print("=" * 40)
```

```
<ipython-input-6-2e3d643c7ddb>:7: RuntimeWarning: overflow encountered in long_scalars
  return (256 * (x[1] - x[0]**2)**2 + (2 - x[0])**2)
Alpha_not = 1
Minimizer: [2. 4.]
Objective_Function_Value: 4.177663477180963e-20
Number_of_Iterations: 3758704
========================================
Alpha_not = 0.9
Minimizer: [2. 4.]
Objective_Function_Value: 4.229895036956251e-20
Number_of_Iterations: 3194229
========================================
Alpha_not = 0.75
Minimizer: [2. 4.]
Objective_Function_Value: 3.9603113583475924e-20
Number_of_Iterations: 3126206
========================================
Alpha_not = 0.6
Minimizer: [2. 4.]
Objective_Function_Value: 4.01668276314331e-20
Number_of_Iterations: 3739599
========================================
Alpha_not = 0.5
Minimizer: [2. 4.]
```

```
Objective_Function_Value: 4.177754236121147e-20
Number_of_Iterations: 3759497
========================================
Alpha_not = 0.4
Minimizer: [2. 4.]
Objective_Function_Value: 4.0946009834969444e-20
Number_of_Iterations: 3052650
========================================
Alpha_not = 0.25
Minimizer: [2. 4.]
Objective_Function_Value: 4.1780083663995865e-20
Number_of_Iterations: 3759270
========================================
Alpha_not = 0.1
Minimizer: [2. 4.]
Objective_Function_Value: 4.0946009834969444e-20
Number_of_Iterations: 3053106
========================================
Alpha_not = 0.01
Minimizer: [2. 4.]
Objective_Function_Value: 3.8352382564641123e-20
Number_of_Iterations: 3523709
========================================
```

```
# Plotting
plt.figure(figsize=(12, 8))

# Plot number of iterations vs. alpha0
plt.subplot(2, 2, 1)
plt.plot(alpha_values, iteration_counts, marker='s', linestyle='-', color='blue')
plt.xlabel('Alpha0')
plt.ylabel('Number of Iterations')
plt.title('Iterations vs. Alpha0')

# Plot level sets of the function g(x) and trajectory
plt.subplot(2, 2, 2)
plt.contour(X, Y, Z, levels=50, cmap='plasma', alpha=0.7)
plt.plot(trajectory_x[:, 0], trajectory_x[:, 1], label='Optimization Trajectory', marker='o', linestyle='--', color='red')
plt.xlabel('x1')
plt.ylabel('x2')
plt.title('Level Sets of g(x) with Trajectory')
plt.legend()

plt.tight_layout()
plt.show()
```





Question 3

After performing the above experiment it can be found out with two line search approaches (i.e., exact line search and the backtracking line search) is as follows:

The backtracking line search need the number of iteration is 3523709.

Backtracking line search needs much more iterations compared to the exact line search approach for the given function f(x). This may be the case beacause the step size is dynamic in nature allowing it to adjust to the local characteristics of the objective function.

Backtracking, the inexact line search method, is good with all kinds of problems because it doesn't need an exact step size; it figures out the right size as it goes. This is especially helpful when dealing with complicated problems that change a lot.

The inexact line search approach, specifically backtracking, demonstrates advantages in terms of adaptability and convergence speed, making it a preferred choice in many optimization scenarios.