Problem – 1:

Code:

```
import numpy as np
import math
def f(x, y):
    return math.sin(x + y) + (x - y)**2 - 1.5 * x + 2.5 * y + 1
#derivative of f(x,y) w.r.t x and y
def gradient_f(x, y):
    df_dx = math.cos(x + y) + 2 * (x - y) - 1.5
    df_dy = math.cos(x + y) - 2 * (x - y) + 2.5
    return np.array([df_dx, df_dy])
def gd_optimize(a):
    learning_rate = 1.0
    threshold = 1e-20
    x, y = a[0], a[1]
    prev_obj_value = f(x, y)
    while True:
        grad = gradient_f(x, y)
        x -= learning_rate * grad[0]
        y -= learning_rate * grad[1]
        obj_value = f(x, y)
        print("Objective function value:", obj_value)
        if abs(obj_value - prev_obj_value) < threshold:
            print(f"Converged to minimum at x = {x}, y = {y}")
            break
        if obj_value > prev_obj_value:
            learning_rate /= 2.0
        else:
            learning_rate *= 1.1
        prev_obj_value = obj_value

print("-----------------------------Problem 1-----------------------------
-----")
print("Gradient descent algorithm Results:")
print("\nOptimizing for initial point [-0.2, -1.0]:")
gd_optimize(np.array([-0.2, -1.0]))

print("----------------------------------------------------------------")
print("Optimizing for initial point [-0.5, -1.5]:")
gd_optimize(np.array([-0.5, -1.5]))
```

Output:

```
-----------------------------Problem 1-----------------------------------
Gradient descent algorithm Results:

Optimizing for initial point [-0.2, -1.0]:
Objective function value: -1.3175387318156826
Objective function value: -1.503265873161276
Objective function value: -1.3933929562543743
Objective function value: -1.9076321773193428
Objective function value: -1.912900015321147
Objective function value: -1.9131807504289906
Objective function value: -1.9132152450977031
Objective function value: -1.91322073144749
Objective function value: -1.913221746385557
Objective function value: -1.9132218772859027
Objective function value: -1.9132215459436885
Objective function value: -1.9132229477178324
Objective function value: -1.913222954748297
Objective function value: -1.913222954960514
Objective function value: -1.9132229549773028
Objective function value: -1.9132229549798279
Objective function value: -1.913222954980399
Objective function value: -1.9132229549805215
Objective function value: -1.913222954980426
Objective function value: -1.913222954981035
Objective function value: -1.9132229549810367
Objective function value: -1.9132229549810367
Converged to minimum at x = -0.5471975518820887, y = -1.547197550524929
--------------------------------------------------------------
Optimizing for initial point [-0.5, -1.5]:
Objective function value: -1.9109295805761808
Objective function value: -1.9114681674883558
Objective function value: -1.9110297007042236
Objective function value: -1.9132215281704674
Objective function value: -1.9132229214706045
Objective function value: -1.913222952576786
Objective function value: -1.9132229546063524
Objective function value: -1.9132229548741102
Objective function value: -1.9132229549304762
Objective function value: -1.9132229549439543
Objective function value: -1.9132229549407707
Objective function value: -1.9132229549810185
Objective function value: -1.9132229549810362
Objective function value: -1.9132229549810358
Objective function value: -1.9132229549810367
Objective function value: -1.9132229549810362
Objective function value: -1.9132229549810367
Objective function value: -1.9132229549810367
Converged to minimum at x = -0.5471975510477202, y = -1.5471975510478024
```

Problem – 2:

Code:

```
def hessian_f(x, y):
    d2f_dx2 = -math.sin(x + y) + 2
    d2f_dxdy = -math.sin(x + y) - 2
    d2f_dydx = -math.sin(x + y) - 2
    d2f_dy2 = -math.sin(x + y) + 2
    return np.array([[d2f_dx2, d2f_dxdy], [d2f_dydx, d2f_dy2]])
def nm_optimize(a):
    prev_value = f(a[0], a[1])
    threshold = 1e-20
    while True:
        grad = gradient_f(a[0], a[1])
        hessian = hessian_f(a[0], a[1])
        hessian_inv = np.linalg.inv(hessian)
        new_a = a - np.dot(hessian_inv, grad)
        new_value = f(new_a[0], new_a[1])
        print("Objective function value:",new_value)
        if abs(new_value - prev_value) < threshold:
            print(f"Converged to minimum at x = {new_a[0]}, y = 
{new_a[1]}")
            break
        prev_value = new_value
        a = new_a

print("------------------------------Problem 2------------------------------
-----")
print("Newton's Method Optimization Results:")
print("\nOptimizing for initial point [-0.2, -1.0]:")
nm_optimize(np.array([-0.2, -1.0]))
print("------------------------------------------------------------------")
print("Optimizing for initial point [-0.5, -1.5]:")
nm_optimize(np.array([-0.5, -1.5]))
```

Output:

```
------------------------------Problem 2-----------------------------------
Newton's Method Optimization Results:

Optimizing for initial point [-0.2, -1.0]:
Objective function value: -1.9128135207487111
Objective function value: -1.9132229186591214
```

```
Objective function value: -1.9132229549810362
Objective function value: -1.9132229549810362
Converged to minimum at x = -0.5471975511965976, y = -1.5471975511965976
---------------------------------------------------------------
Optimizing for initial point [-0.5, -1.5]:
Objective function value: -1.9132209008539096
Objective function value: -1.913222954980231
Objective function value: -1.9132229549810362
Objective function value: -1.9132229549810367
Objective function value: -1.9132229549810367
Converged to minimum at x = -0.5471975511965976, y = -1.5471975511965979
```

Screenshots:

```python
import numpy as np
import math
def f(x, y):
    return math.sin(x + y) + (x - y)**2 - 1.5 * x + 2.5 * y + 1
#derivative of f(x,y) w.r.t x and y
def gradient_f(x, y):
    df_dx = math.cos(x + y) + 2 * (x - y) - 1.5
    df_dy = math.cos(x + y) - 2 * (x - y) + 2.5
    return np.array([df_dx, df_dy])
def gd_optimize(a):
    learning_rate = 1.0
    threshold = 1e-20
    x, y = a[0], a[1]
    prev_obj_value = f(x, y)
    while True:
        grad = gradient_f(x, y)
        x -= learning_rate * grad[0]
        y -= learning_rate * grad[1]
        obj_value = f(x, y)
        print("Objective function value:", obj_value)
        if abs(obj_value - prev_obj_value) < threshold:
            print(f"Converged to minimum at x = {x}, y = {y}")
            break
        if obj_value > prev_obj_value:
            learning_rate /= 2.0
        else:
            learning_rate *= 1.1
        prev_obj_value = obj_value

print("-----------------------------Problem 1----------------------------------")
print("Gradient descent algorithm Results:")
print("\nOptimizing for initial point [-0.2, -1.0]:")
gd_optimize(np.array([-0.2, -1.0]))

print("---------------------------------------------------------------")
print("Optimizing for initial point [-0.5, -1.5]:")
gd_optimize(np.array([-0.5, -1.5]))
```

```
-----------------------------Problem 1----------------------------------
Gradient descent algorithm Results:

Optimizing for initial point [-0.2, -1.0]:
Objective function value: -1.3175387318156826
Objective function value: -1.503265873161276
Objective function value: -1.3933929562543743
Objective function value: -1.9076321773193428
Objective function value: -1.912900015321147
Objective function value: -1.9131807504289906
Objective function value: -1.9132152450977031
Objective function value: -1.91322073144749
Objective function value: -1.913221746385557
Objective function value: -1.9132218772859027
Objective function value: -1.9132215459436885
Objective function value: -1.9132229477178324
Objective function value: -1.913222954748297
Objective function value: -1.913222954960514
Objective function value: -1.9132229549773028
Objective function value: -1.9132229549798279
Objective function value: -1.913222954980399
Objective function value: -1.9132229549805215
Objective function value: -1.913222954980426
Objective function value: -1.913222954981035
Objective function value: -1.9132229549810367
Objective function value: -1.9132229549810367
Converged to minimum at x = -0.5471975518820887, y = -1.547197550524929
----------------------------------------------------------------
Optimizing for initial point [-0.5, -1.5]:
Objective function value: -1.9109295805761808
Objective function value: -1.9114681674883558
Objective function value: -1.9110297007042236
Objective function value: -1.9132215281704674
Objective function value: -1.9132229214706045
Objective function value: -1.913222952576786
Objective function value: -1.9132229546063524
Objective function value: -1.9132229548741102
Objective function value: -1.9132229549304762
Objective function value: -1.9132229549439543
Objective function value: -1.9132229549407707
```

```
Objective function value: -1.9132229549810358
Objective function value: -1.9132229549810367
Objective function value: -1.9132229549810362
Objective function value: -1.9132229549810367
Objective function value: -1.9132229549810367
Converged to minimum at x = -0.5471975510477202, y = -1.5471975510478024
```

```python
def hessian_f(x, y):
    d2f_dx2 = -math.sin(x + y) + 2
    d2f_dxdy = -math.sin(x + y) - 2
    d2f_dydx = -math.sin(x + y) - 2
    d2f_dy2 = -math.sin(x + y) + 2
    return np.array([[d2f_dx2, d2f_dxdy], [d2f_dydx, d2f_dy2]])
def nm_optimize(a):
    prev_value = f(a[0], a[1])
    threshold = 1e-20
    while True:
        grad = gradient_f(a[0], a[1])
        hessian = hessian_f(a[0], a[1])
        hessian_inv = np.linalg.inv(hessian)
        new_a = a - np.dot(hessian_inv, grad)
        new_value = f(new_a[0], new_a[1])
        print("Objective function value:",new_value)
        if abs(new_value - prev_value) < threshold:
            print(f"Converged to minimum at x = {new_a[0]}, y = {new_a[1]}")
            break
        prev_value = new_value
        a = new_a

print("----------------------------Problem 2--------------------------------")
print("Newton's Method Optimization Results:")
print("\nOptimizing for initial point [-0.2, -1.0]:")
nm_optimize(np.array([-0.2, -1.0]))
print("-------------------------------------------------------------")
print("Optimizing for initial point [-0.5, -1.5]:")
nm_optimize(np.array([-0.5, -1.5]))
```

```python
[2] nm_optimize(np.array([-0.5, -1.5]))
```

```
----------------------------Problem 2--------------------------------
Newton's Method Optimization Results:

Optimizing for initial point [-0.2, -1.0]:
Objective function value: -1.9128135207487111
Objective function value: -1.9132229186591214
Objective function value: -1.9132229549810362
Objective function value: -1.9132229549810362
Converged to minimum at x = -0.5471975511965976, y = -1.5471975511965976
-------------------------------------------------------------
Optimizing for initial point [-0.5, -1.5]:
Objective function value: -1.9132209008539096
Objective function value: -1.913222954980231
Objective function value: -1.9132229549810362
Objective function value: -1.9132229549810367
Objective function value: -1.9132229549810367
Converged to minimum at x = -0.5471975511965976, y = -1.5471975511965979
```