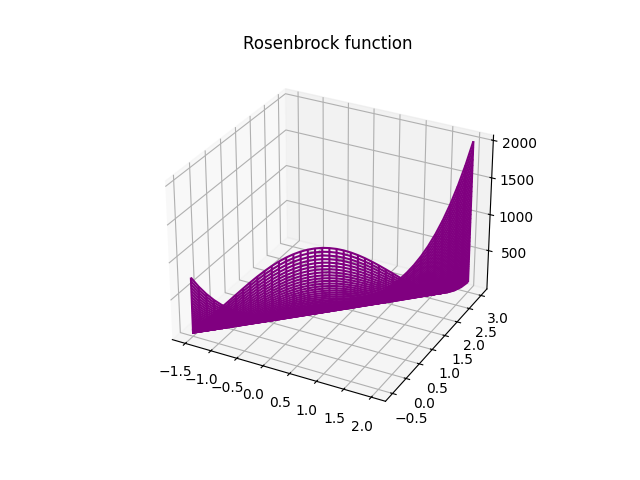
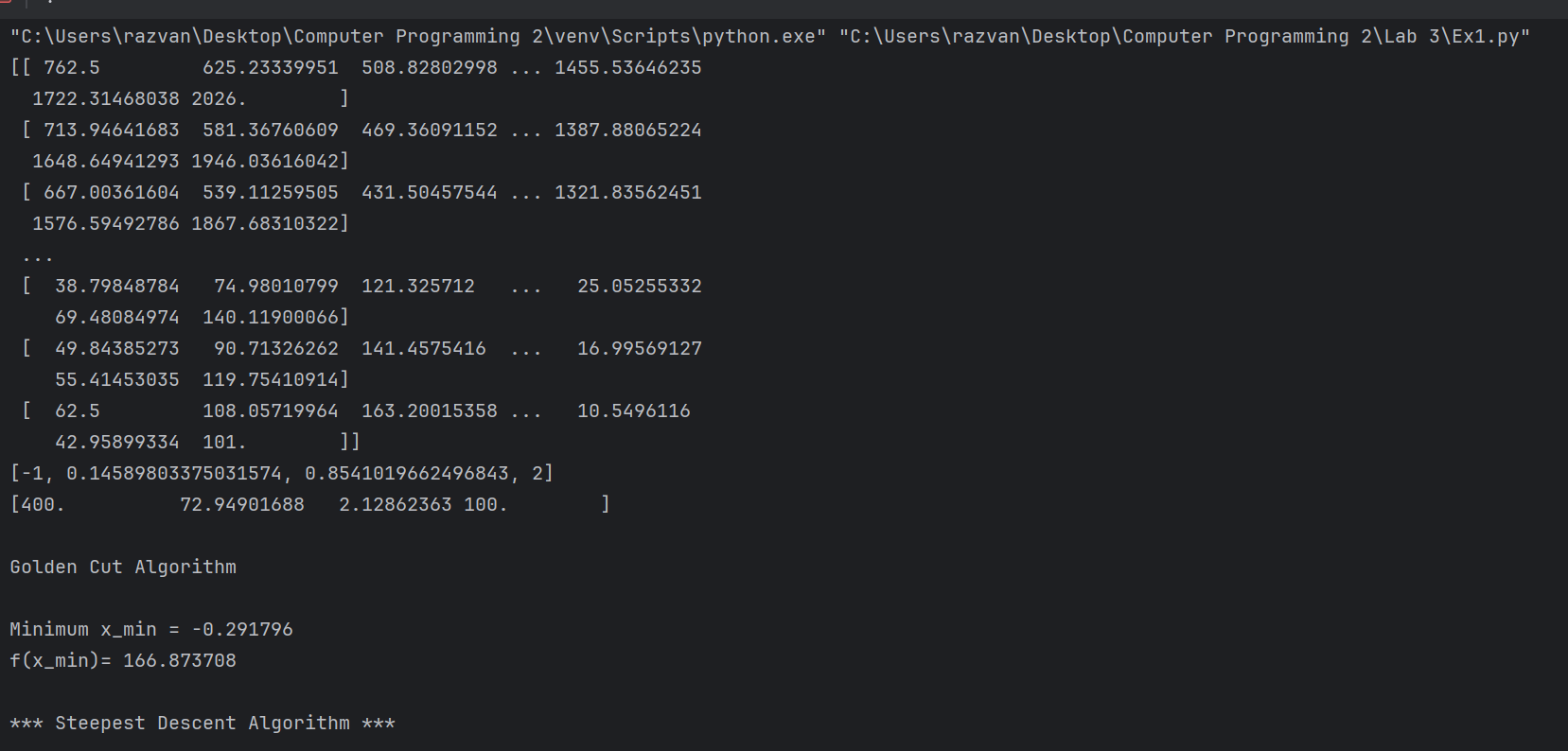
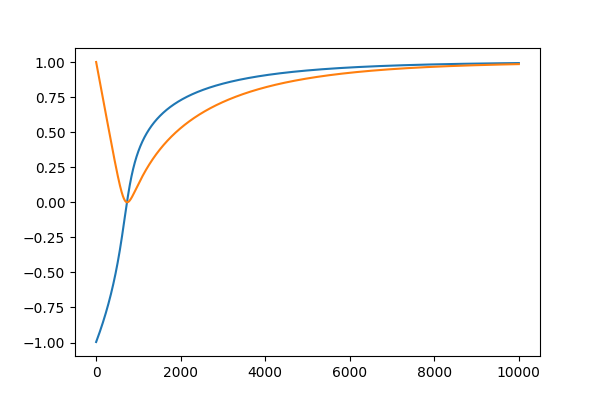
Ex 1

import matplotlib.pyplot as plt  
import numpy as np  
import math  
  
  
def rb(x1, x2):  
 return 100 \* (x2 - x1 \*\* 2) \*\* 2 + (1 - x1) \*\* 2  
  
  
n = 40  
x1 = np.linspace(start=-1.5, stop=2, num=n)  
x2 = np.linspace(start=-0.5, stop=3, num=n)  
X1, X2 = np.meshgrid(x1, x2)  
F = rb(X1, X2)  
print(F)  
  
  
# GOLDEN CUT  
tau = 2 / (1 + math.sqrt(5))  
  
x = [0, 1 - tau, tau, 1]  
y = np.zeros(4)  
  
xmin = -1  
xmax = 2  
  
for i in range(4):  
 x[i] = x[i] \* (xmax - xmin) + xmin  
 y[i] = rb(1, x[i])  
  
print(x)  
print(y)  
  
while (x[2] - x[1]) >= 0.01:  
 if y[2] > y[1]:  
 x[2:4] = x[1:3]  
 x[1] = (1 - tau) \* (x[3] - x[0]) + x[0]  
 y[2:4] = y[1:3]  
 y[1] = rb(1, x[1])  
 else:  
 x[1:3] = x[0:2]  
 x[2] = tau \* (x[3] - x[0]) + x[0]  
 y[1:3] = x[0:2]  
 y[2] = rb(1, x[2])  
#  
fig = plt.figure()  
wf = fig.add\_subplot(111, projection="3d")  
wf.plot\_wireframe(x1, x2, F, rstride=1, cstride=1,color="purple")  
wf.set\_title("Rosenbrock function")  
  
print("\nGolden Cut Algorithm\n")  
print("Minimum x\_min = %f" % x[1])  
print("f(x\_min)= %f" % y[1])  
  
M = 10000  
x = [-1, +1]  
  
alpha = 1e-3  
h = 1e-3  
  
g = np.zeros(2)  
out = np.zeros((M, 2))  
  
for n in range(M):  
 g[0] = (rb(x[0] + h / 2, x[1]) - rb(x[0] - h / 2, x[1])) / h  
 g[1] = (rb(x[0], x[1] + h / 2) - rb(x[0], x[1] - h / 2)) / h  
 x = x - alpha \* g  
 out[n][:] = x  
  
print("\n\*\*\* Steepest Descent Algorithm \*\*\*\n")  
plt.figure(3, figsize=(6, 4))  
plt.plot(np.linspace(0, M, M), out)  
  
plt.show()

Result 



Ex2

import matplotlib.pyplot as plt  
import numpy as np  
import math  
  
  
def mt(x, y):  
 return 0.26 \* (x \*\* 2 + y \*\* 2) - 0.48 \* x \* y  
  
  
n = 40  
x1 = np.linspace(start=-20, stop=-20, num=n)  
x2 = np.linspace(start=-20, stop=-20, num=n)  
X1, X2 = np.meshgrid(x1, x2)  
F = mt(X1, X2)  
  
  
# GOLDEN CUT  
tau = 2 / (1 + math.sqrt(5))  
  
x = [0, 1 - tau, tau, 1]  
y = np.zeros(4)  
  
xmin = -1  
xmax = 2  
  
for i in range(4):  
 x[i] = x[i] \* (xmax - xmin) + xmin  
 y[i] = mt(0, x[i])  
  
  
  
while (x[2] - x[1]) >= 0.01:  
 if y[2] > y[1]:  
 x[2:4] = x[1:3]  
 x[1] = (1 - tau) \* (x[3] - x[0]) + x[0]  
 y[2:4] = y[1:3]  
 y[1] = mt(0, x[1])  
 else:  
 x[1:3] = x[0:2]  
 x[2] = tau \* (x[3] - x[0]) + x[0]  
 y[1:3] = x[0:2]  
 y[2] = mt(0, x[2])  
  
print("\nGolden Cut Algorithm\n")  
print("Minimum x\_min = %f" % x[1])  
print("f(x\_min)= %f" % y[1])  
  
M = 10000  
x = [-1, +1]  
  
alpha = 1e-3  
h = 1e-3  
  
g = np.zeros(2)  
out = np.zeros((M, 2))  
  
for n in range(M):  
 g[0] = (mt(x[0] + h / 2, x[1]) - mt(x[0] - h / 2, x[1])) / h  
 g[1] = (mt(x[0], x[1] + h / 2) - mt(x[0], x[1] - h / 2)) / h  
 x = x - alpha \* g  
 out[n][:] = x  
  
print("\n\*\*\* Steepest Descent Algorithm \*\*\*\n")  
plt.figure( figsize=(6, 4))  
plt.plot(np.linspace(0, M, M), out)  
plt.title("Steepest Descent")  
  
plt.show()

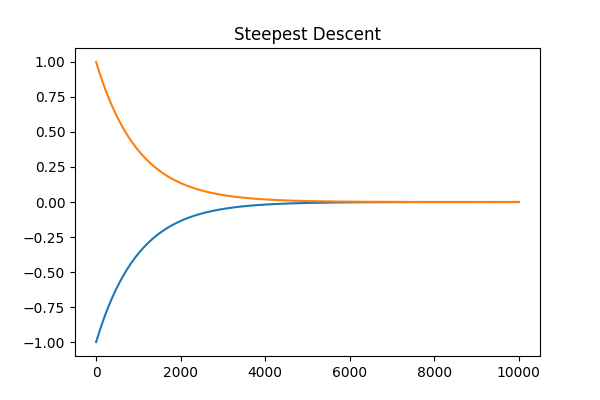
Result

Golden Cut Algorithm

Minimum x\_min = -0.562306

f(x\_min)= 0.082209

\*\*\* Steepest Descent Algorithm \*\*\*



Ex 3

import numpy as np  
import matplotlib.pyplot as plt  
  
  
# Define the Three-Hump Camel Function  
def three\_hump\_camel(x, y):  
 return 2 \* x \*\* 2 - 1.05 \* x \*\* 4 + (x \*\* 6) / 6 + x \* y + y \*\* 2  
  
  
# Calculate the Gradient and Hessian Matrix  
def gradient(x, y):  
 df\_dx = 4 \* x \*\* 3 - 4.2 \* x \*\* 3 + x \*\* 5 + y  
 df\_dy = x + 2 \* y  
 return np.array([df\_dx, df\_dy])  
  
  
def hessian(x, y):  
 d2f\_dx2 = 12 \* x \*\* 2 - 12.6 \* x \*\* 2 + 5 \* x \*\* 4  
 d2f\_dy2 = 2  
 d2f\_dx\_dy = 1  
 return np.array([[d2f\_dx2, d2f\_dx\_dy], [d2f\_dx\_dy, d2f\_dy2]])  
  
  
# Newton's Method with Convergence Tracking  
def newtons\_method(initial\_guess, tolerance=1e-6, max\_iterations=100000):  
 x, y = initial\_guess  
 x\_history, y\_history = [x], [y]  
  
 for iteration in range(max\_iterations):  
 grad = gradient(x, y)  
 hess = hessian(x, y)  
 delta = np.linalg.solve(hess, -grad)  
 x += delta[0]  
 y += delta[1]  
  
 x\_history.append(x)  
 y\_history.append(y)  
  
 # Check convergence  
 if np.linalg.norm(delta) < tolerance:  
 break  
  
 return x, y, x\_history, y\_history  
  
  
# Different initial guess  
initial\_guess = [1.5, -1.5]  
  
# Find the minimum and track convergence  
result\_x, result\_y, x\_history, y\_history = newtons\_method(initial\_guess)  
  
# Create a grid of x and y values for surface plotting  
x\_range = np.linspace(-2, 2, 400)  
y\_range = np.linspace(-2, 2, 400)  
X, Y = np.meshgrid(x\_range, y\_range)  
Z = three\_hump\_camel(X, Y)  
  
# Plot the convergence in 3D  
fig = plt.figure(figsize=(10, 6))  
ax = fig.add\_subplot(111, projection='3d')  
ax.plot\_surface(X, Y, Z, cmap='viridis', alpha=0.8)  
ax.plot(x\_history, y\_history, [three\_hump\_camel(x, y) for x, y in zip(x\_history, y\_history)], marker='o', color='red')  
ax.set\_xlabel('x')  
ax.set\_ylabel('y')  
ax.set\_zlabel('f(x, y)')  
ax.set\_title('Convergence of Newton\'s Method for Three-Hump Camel Function')  
plt.show()  
  
print("Minimum point (x, y):", (result\_x, result\_y))  
print("Minimum value of the Three-Hump Camel Function:", three\_hump\_camel(result\_x, result\_y))

Result:

Minimum point (x, y): (0.9022986439390706, -0.4511493219695353)

Minimum value of the Three-Hump Camel Function: 0.8187195005191514

