Prac-2

import numpy as np

def objective\_function(x):

 return 100 \* (x[1] - x[0]\*\*2)\*\*2 + (1 - x[0])\*\*2

def gradient(x):

 df\_dx0 = -400 \* x[0] \* (x[1] - x[0]\*\*2) - 2 \* (1 - x[0])

 df\_dx1 = 200 \* (x[1] - x[0]\*\*2)

 return np.array([df\_dx0, df\_dx1])

def line\_search(x, direction, alpha=0.2, beta=0.8):

 t = 1.0

 while objective\_function(x + t \* direction) > objective\_function(x) + alpha \* t \* np.dot(gradient(x), direction):

  t \*= beta

 return t

def gradient\_descent\_line\_search(initial\_x, max\_iterations=1000,tolerance=1e-6):

 x = initial\_x

 for iteration in range(max\_iterations):

  grad = gradient(x)

  if np.linalg.norm(grad) < tolerance:

    print("Converged after", iteration, "iterations.")

    break

  direction = -grad # Search direction is the negative gradient

  step\_size = line\_search(x, direction)

  x = x + step\_size \* direction

  return x

# Initial guess

initial\_guess = np.array([0.0, 0.0])

# Perform gradient descent with line search

optimal\_solution = gradient\_descent\_line\_search(initial\_guess)

# Display the results

print("Optimal solution:", optimal\_solution)

print("Optimal function value:", objective\_function(optimal\_solution))

OUTPUT



import math

# Objective function

def objective(x):

  return x\*\*2 - 4\*x + 4 # Example function, replace with your own# Golden Ratio for Line Search

def golden\_ratio\_line\_search(a, b, tol=1e-5):

  golden\_ratio = (1 + math.sqrt(5)) / 2

  while abs(b - a) > tol:

    x1 = b - (b - a) / golden\_ratio

    x2 = a + (b - a) / golden\_ratio

    if objective(x1) < objective(x2):

      b = x2

    else:

      a = x1

  return (a + b) / 2

# Optimization using Golden Ratio Line Search

def optimize\_golden\_ratio(initial\_guess, tol=1e-5):

  x = initial\_guess

  while True:

    gradient = 2 \* x - 4 # Replace with the derivative of your objective function

    if abs(gradient) < tol:

      break

    # Perform line search

    alpha = golden\_ratio\_line\_search(0, 1)

    # Update x using the line search result

    x -= alpha \* gradient

  return x

# Example usage

initial\_guess = 0.0

optimal\_solution = optimize\_golden\_ratio(initial\_guess)

print("Optimal solution using Golden Ratio:", optimal\_solution)



# Objective function

def objective(x):

  return x\*\*2 - 4\*x + 4 # Example function, replace with your own# Fibonacci sequence for Line Search

def fibonacci\_line\_search(a, b, n, tol=1e-5):

  fib\_sequence = [1, 1]

  for \_ in range(2, n + 1):

    fib\_sequence.append(fib\_sequence[-1] + fib\_sequence[-2])

  for k in range(n, 1, -1):

    ratio = fib\_sequence[k - 1] / fib\_sequence[k]

    x1 = a + ratio \* (b - a)

    x2 = b - ratio \* (b - a)

    if objective(x1) < objective(x2):

      b = x2

    else:

      a = x1

  return (a + b) / 2

# Optimization using Fibonacci Line Search

def optimize\_fibonacci(initial\_guess, n=10, tol=1e-5):

  x = initial\_guess

  while True:

    gradient = 2 \* x - 4 # Replace with the derivative of your objective function

    if abs(gradient) < tol:

      break

    # Perform line search

    alpha = fibonacci\_line\_search(0, 1, n)

    # Update x using the line search result

    x -= alpha \* gradient

  return x

# Example usage

initial\_guess = 0.0

optimal\_solution = optimize\_fibonacci(initial\_guess)

print("Optimal solution using Fibonacci Sequence:", optimal\_solution)

