# Name:- Harsh Arora
# Roll NO:-AE-1218
# NUMERICAL OPTIMIZATION

File Edit Format Run Options Window Help

```python
import numpy as np

# Define the objective function
def f(x):
    return x**2 + 5 * x + 6  # Example function: f(x) = x^2 + 5x + 6

# Define the derivative of the objective function
def f_prime(x):
    return 2 * x + 5  # Derivative of f(x)

# Line search method to find the optimal solution
def line_search_method(x_start, direction, step_size, epsilon=1e-5, max_iterations=1000):
    x = x_start
    iteration = 0

    while iteration < max_iterations:
        gradient = f_prime(x)
        new_x = x + step_size * direction

        # If the change is negligible or the gradient is close to zero, stop
        if np.abs(new_x - x) < epsilon or np.abs(gradient) < epsilon:
            break

        x = new_x
        iteration += 1

    return x, f(x)

# Set initial values
x_start = 0  # Initial value of x
search_direction = -1  # Direction of search (-1 for minimizing the function)
step = 0.1  # Step size for the line search

# Perform line search
optimal_solution, minimum_value = line_search_method(x_start, search_direction, step)

print("Optimal Solution (x):", optimal_solution)
print("Minimum Value of f(x):", minimum_value)
```

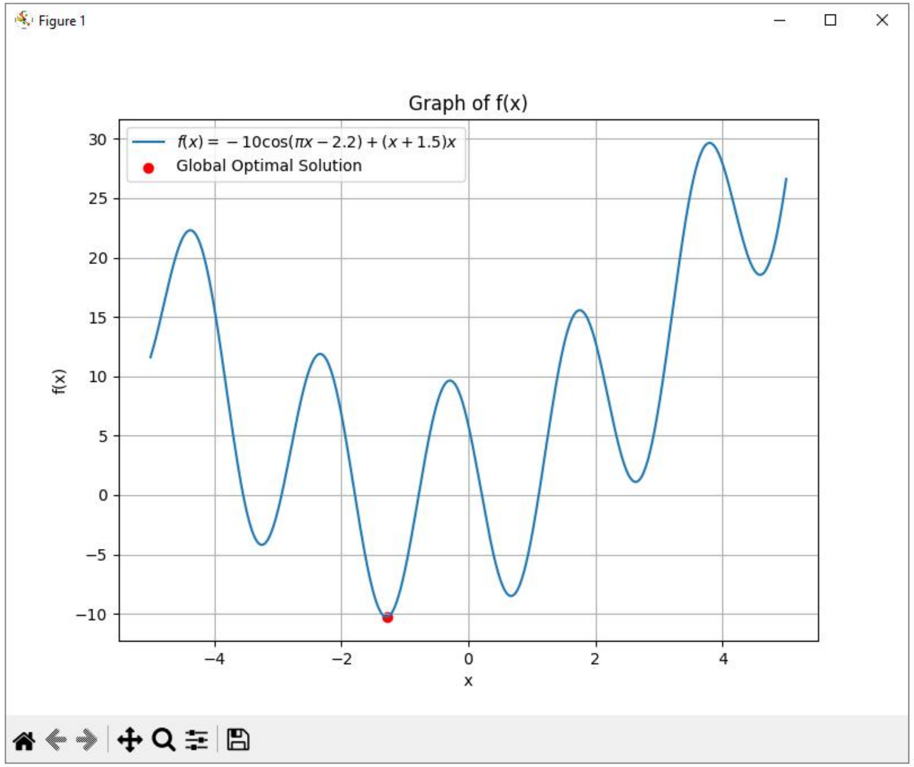
Ln: 39 Col: 0

```
= RESTART: C:/Users/Mehul/AppData/Local/Programs/Python/Python311/College work Mehul NO/Line search method.py
Optimal Solution (x): -2.500000000000001
Minimum Value of f(x): -0.2499999999999991
>>>
```

Ln: 52 Col: 0

File Edit Format Run Options Window Help

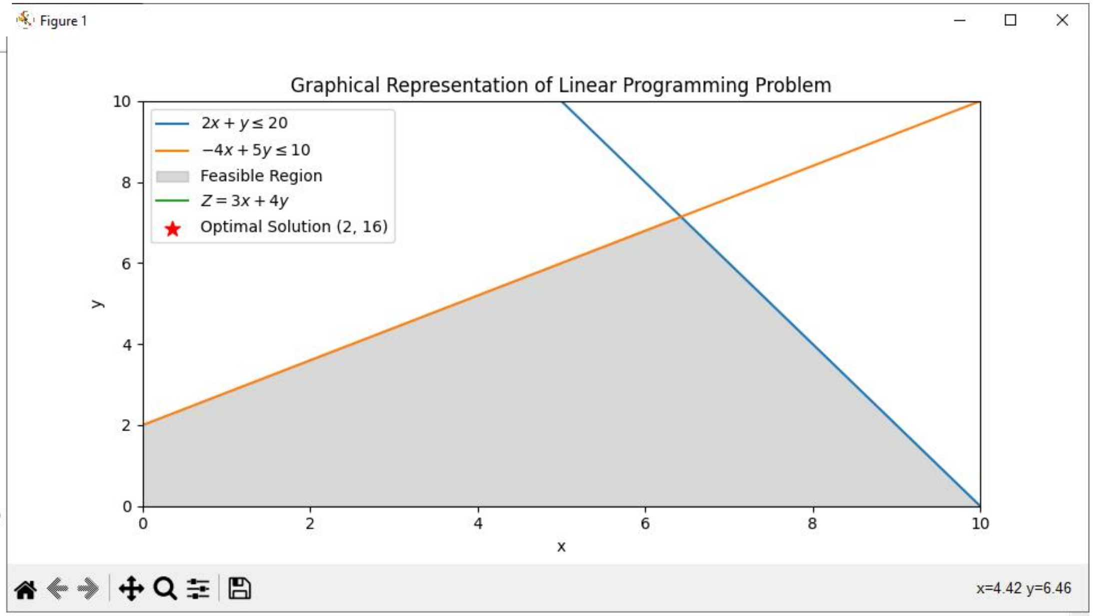
```python
import numpy as np
import matplotlib.pyplot as plt

# Define the function
def f(x):
    return -10 * np.cos(np.pi * x - 2.2) + (x + 1.5) * x

# Generate x values
x_values = np.linspace(-5, 5, 1000)

# Calculate corresponding y values (function values)
y_values = f(x_values)

# Find the x value that corresponds to the minimum y value (global minimum)
optimal_x = x_values[np.argmin(y_values)]
optimal_y = np.min(y_values)

# Plot the function
plt.figure(figsize=(8, 6))
plt.plot(x_values, y_values, label=r'$f(x)=-10\cos(\pi x - 2.2)+(x+1.5)x$')
plt.scatter(optimal_x, optimal_y, color='red', label='Global Optimal Solution')

plt.title('Graph of f(x)')
plt.xlabel('x')
plt.ylabel('f(x)')
plt.legend()
plt.grid(True)
plt.show()

print("Global Optimal Solution (x):", optimal_x)
print("Minimum Value of f(x):", optimal_y)
```




Ln: 23 Col: 26

File Edit Format Run Options Window Help

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

# Define the objective function coefficients (Z = cx + dy)
c = 3
d = 4

# Define the constraints (in the form ax + by <= c)
constraint1 = {'a': 2, 'b': 1, 'c': 20}  # 2x + y <= 20
constraint2 = {'a': -4, 'b': 5, 'c': 10}  # -4x + 5y <= 10

# Calculate the feasible region
x = np.linspace(0, 10, 400)  # Range of x values

# Constraint 1: 2x + y <= 20
y1 = (constraint1['c'] - constraint1['a']*x) / constraint1['b']

# Constraint 2: -4x + 5y <= 10
y2 = (constraint2['c'] - constraint2['a']*x) / constraint2['b']

# Plotting the constraints and feasible region
plt.figure(figsize=(8, 6))

plt.plot(x, y1, label=r'$2x + y \leq 20$')
plt.plot(x, y2, label=r'$-4x + 5y \leq 10$')
plt.fill_between(x, 0, np.minimum(y1, y2), where=(y1 > 0) & (y2 > 0), color='gray', alpha=0.3, label='Feasible Region')
plt.xlim((0, 10))
plt.ylim((0, 10))
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.title('Graphical Representation of Linear Programming Problem')

# Plot the objective function Z = cx + dy for some values of x and corresponding y in the feasible region
Z = c*x + d*y1  # Using y1 as it represents the upper bound of feasible y values
plt.plot(x, Z, label=r'$Z = 3x + 4y$')
optimal_x = 2
optimal_y = 16
plt.scatter(optimal_x, optimal_y, color='red', marker='*', s=100, label='Optimal Solution (2, 16)')

plt.legend()
plt.show()
```




Ln: 43 Col: 12

File Edit Format Run Options Window Help

```python
from scipy.optimize import fsolve
import numpy as np

# Define the derivative function
def derivative(x):
    return 10 * np.pi * np.sin(np.pi * x - 2.2) + 2 * x + 1.5

# Use fsolve to find the roots (where the derivative is zero)
critical_points = fsolve(derivative, [-2, 2])  # Initial guesses for roots

# Evaluate the function at the critical points
values_at_critical_points = -10 * np.cos(np.pi * critical_points - 2.2) + (critical_points + 1.5) * critical_points

# Find the minimum value among the critical points
global_min_index = np.argmin(values_at_critical_points)
global_optimal_solution = critical_points[global_min_index]
min_function_value = values_at_critical_points[global_min_index]

print("Critical Points:", critical_points)
print("Function Values at Critical Points:", values_at_critical_points)
print("Global Optimal Solution (Minimum):", global_optimal_solution)
print("Minimum Function Value:", min_function_value)
```

```
Critical Points: [-2.3318272   1.75118297]
Function Values at Critical Points: [11.88884336 15.565831  ]
Global Optimal Solution (Minimum): -2.3318271970896833
Minimum Function Value: 11.888843364338118
>>>
```

Ln: 9 Col: 0

Ln: 23 Col: 0

File Edit Format Run Options Window Help

```python
import sympy as sp

# Define the variables
x1, x2 = sp.symbols('x1 x2')

# Define the function
f = 100 * (x2 - x1**2)**2 + (1 - x1)**2

# Compute the gradient
gradient = [sp.diff(f, var) for var in (x1, x2)]

# Compute the Hessian matrix
hessian = sp.hessian(f, (x1, x2))

# Print the gradient and Hessian matrix
print("Gradient of f(x):", gradient)
print("\nHessian of f(x):")
print(hessian)
```

```
Gradient of f(x): [-400*x1*(-x1**2 + x2) + 2*x1 - 2, -200*x1**2 + 200*x2]

Hessian of f(x):
Matrix([[1200*x1**2 - 400*x2 + 2, -400*x1], [-400*x1, 200]])
>>>
= RESTART: C:/Users/Mehul/AppData/Local/Programs/Python/Python311/College Work M
ehul NO/WAP Hessian of the fuction.py
Gradient of f(x): [-400*x1*(-x1**2 + x2) + 2*x1 - 2, -200*x1**2 + 200*x2]

Hessian of f(x):
Matrix([[1200*x1**2 - 400*x2 + 2, -400*x1], [-400*x1, 200]])
>>>
```

Ln: 17 Col: 0

File Edit Format Run Options Window Help

```python
from scipy.optimize import minimize

# Define the objective function to minimize
def objective_function(variables):
    x, y = variables
    return x**2 + y**2

# Define the inequality constraints
def inequality_constraints(variables):
    x, y = variables
    return [
        1 - 2*x - y,  # 2x + y >= 1 becomes 2x + y - 1 >= 0
        2 - x - 3*y   # x + 3y >= 2 becomes x + 3y - 2 >= 0
    ]

# Define initial guess for variables
initial_guess = [0.5, 0.5]  # Initial guess for x and y

# Set up bounds for x and y (non-negative)
bounds = [(0, None), (0, None)]  # x and y should be non-negative

# Define constraints using dictionary format
constraints = {'type': 'ineq', 'fun': inequality_constraints}

# Use minimize function to solve the optimization problem
result = minimize(objective_function, initial_guess, bounds=bounds, constraints=constraints)

# Print the optimal solution and minimum value of the objective function
print("Optimal Solution (x, y):", result.x)
print("Minimum Value of f(x, y):", result.fun)
```

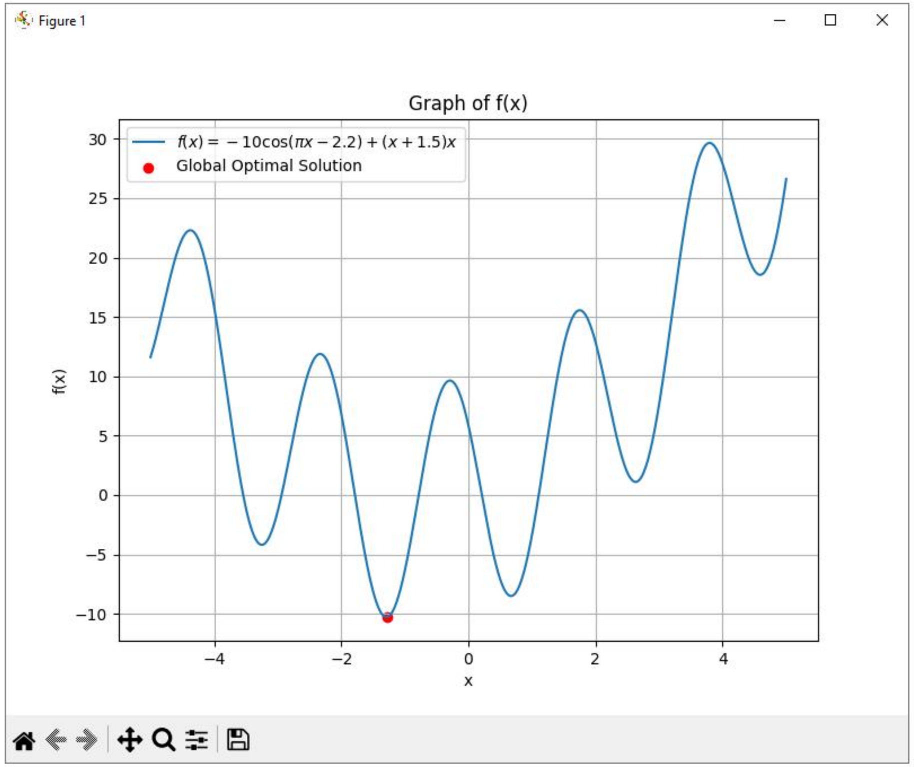
Ln: 31 Col: 0

```
Optimal Solution (x, y): [1.11022302e-16 1.11022302e-16]
Minimum Value of f(x, y): 2.465190328815662e-32
>>>
```

Ln: 60 Col: 0

File Edit Format Run Options Window Help

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

# Define the function
def f(x):
    return -10 * np.cos(np.pi * x - 2.2) + (x + 1.5) * x

# Generate x values
x_values = np.linspace(-5, 5, 1000)

# Calculate corresponding y values (function values)
y_values = f(x_values)

# Find the x value that corresponds to the minimum y value (global minimum)
optimal_x = x_values[np.argmin(y_values)]
optimal_y = np.min(y_values)

# Plot the function
plt.figure(figsize=(8, 6))
plt.plot(x_values, y_values, label=r'$f(x)=-10\cos(\pi x - 2.2)+(x+1.5)x$')
plt.scatter(optimal_x, optimal_y, color='red', label='Global Optimal Solution')

plt.title('Graph of f(x)')
plt.xlabel('x')
plt.ylabel('f(x)')
plt.legend()
plt.grid(True)
plt.show()

print("Global Optimal Solution (x):", optimal_x)
print("Minimum Value of f(x):", optimal_y)
```




Ln: 23 Col: 26