МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»

Лабораторна робота № 8

З дисципліни

“Математичні методи дослідження операцій”

Виконав:

Студент групи КН-314

Ляшеник Остап

Прийняв

Шиманський Володимир Михайлович

Львів - *2023*

Постановка завдання

4. $f(X) = 4x_1^2 - 4x_1x_2 + 5x_2^2 - x_1 - x_2.$

```
import numpy as np
from scipy.optimize import minimize
from sympy import diff, symbols, lambdify
from sympy.abc import x, y

# Constants
l = -8
r = 3
Eps = 1e-8
X0 = np.array([l, r])


# Fibonacci sequence generator
def fibonacci(n):
    a, b = 0, 1
    for _ in range(n):
        a, b = b, a + b
    return a


# Determine the number of iterations needed for the Fibonacci
method
def get_iteration(n):
    i = 0
    while fibonacci(i) < n:
        i += 1
    return i


# Fibonacci search method for optimization along a line
def fibonacci_method(f, a, b, n):
    for k in range(n - 1):
        x1 = a + (fibonacci(n - k - 2) / fibonacci(n - k)) *
(b - a)
        x2 = a + (fibonacci(n - k - 1) / fibonacci(n - k)) *
(b - a)
        if f(x1) < f(x2):
            b = x2
        else:
            a = x1
    return (a + b) / 2
```

```
# The function to minimize
def f(vars):
    return  4 * vars[0] ** 2 - 4 * vars[0] * vars[1] + 5 *
vars[1] ** 2 - vars[0] - vars[1]


# Gradient descent method
def gradient_method(X0, f, Eps):
    x_sym, y_sym = symbols('x_sym y_sym', real=True)
    F = 4 * x_sym ** 2 - 4 * x_sym * y_sym + 5 * y_sym ** 2 -
x_sym - y_sym
    F_dx = diff(F, x_sym)  # Derivative with respect to x
    F_dy = diff(F, y_sym)  # Derivative with respect to y
    Fx_0 = lambdify((x_sym, y_sym), F_dx)  # Convert
expression to function
    Fy_0 = lambdify((x_sym, y_sym), F_dy)

    step = 0
    while True:
        grad = np.array([Fx_0(X0[0], X0[1]), Fy_0(X0[0],
X0[1])])
        func_beta = lambdify((x_sym, y_sym), f([x_sym, y_sym]))

        # Find optimal beta using Fibonacci method
        def func_to_optimize(beta):
            return func_beta(X0[0] - beta * grad[0], X0[1] -
beta * grad[1])

        a, b = -1000, 1000  # Search bounds for beta
        n = get_iteration((b - a) / Eps) + 1
        beta = fibonacci_method(func_to_optimize, a, b, n)

        # Update the position
        X0 = X0 - beta * grad
        if np.linalg.norm(grad) <= Eps:
            break
        step += 1

    return X0


# Find the minimum
min_X = gradient_method(X0, f, Eps)
min_func = f(min_X)
print("Minimum found by gradient method:", min_func)

# Use scipy.optimize.minimize for comparison
res = minimize(f, X0)
```

```python
print("Minimum found by scipy.optimize.minimize:", res.fun)
```

```
Minimum found by gradient method: -0.203125

Minimum found by scipy.optimize.minimize: -0.20312499999893752


Process finished with exit code 0
```