西安电子科技大学

最优化理论与方法 课程实验报告

报告名称 上机报告 1-4

人工智能 学院 23200XX 班

姓名 X X X 学号 2300920XXXX

备 注 python

上机日期 2025 年 月

成 绩

指导教师评语: 指导教师: 年 月 日
实验报告内容基本要求及参考格式 一、题目描述 二、问题描述 三、方法描述 四、过程代码 五、程序执行结果与分析

一、 题目描述

1.问题一

使用 MATLAB/Python 通过图解法求解线性规划问题。

2.问题二

使用黄金分割法（一维搜索）求解函数 $f(x)=x^3-4x-1$的极小值。

3.问题三

用最速下降法优化 Rosenbrock 函数 $f(x)=(x_1^2-x_2)^2+(1-x_1)^2$

4.问题四

使用 DFP 拟牛顿法优化 Rosenbrock 函数（与实验三相同函数）。

二、 问题描述

1.问题一

该问题是资源分配、生产计划等线性约束下的最优解问题。该目标函数和约束均为线性，可行域为凸多边形。该过程中需手动绘制约束条件并确定可行域边界。目标函数平移时需动态调整斜率以找到最优交点。我们可以通过 Python 绘制约束线，填充可行域，平移目标函数等值线直至与可行域顶点相切。

2.问题二

该问题是一维函数优化，该过程中所涉及的单峰函数 $f(x)=x^3-4x-1$，区间 $[0,3]$ 内存在极小值。我们在计算时需保证初始区间包含极小值点，同时收敛精度要求高（0.001）。我们可以通过迭代缩小区间，利用黄金比例分割点比较函数值，逐步逼近极小值。

3.问题三

该问题为非线性函数优化，Rosenbrock 函数存在“香蕉形”谷底，收敛缓慢。该过程中可能会出现梯度方向振荡导致收敛慢。学习率选择不当易发散。我们应该沿负梯度方向搜索，通过线性搜索确定步长。

4. 问题四

该问题是高维非线性优化，拟牛顿法通过近似 Hessian 矩阵加速收敛。该过程中需要注意 DFP 公式需维护正定矩阵，一维搜索精度影响收敛性。我们可以利用 DFP 更新逆 Hessian 矩阵，结合线性搜索确定步长。

三、 方法描述

1.问题一

我们可以定义目标函数系数 c 和约束矩阵 A、b，通过调用 linprog 计算最优解，绘制约束线、可行域及目标函数等值线。代码中使用 linprog 求解线性规划，避免手动计算顶点。此外，通过 fill-between 可视化可行域，等值线反映目标函数变化趋势。

2.问题二

我们可以利用黄金分割函数：实现区间收缩逻辑，返回极小值点和迭代次数，通过展示函数曲线及极小值点。在算法中每次迭代保留更优分割点，区间长度按黄金比例收缩。终止条件为区间长度小于精度 tol。

3.问题三

我们对函数进行定义，以此来计算 Rosenbrock 函数值及梯度，配合最速下降法来迭代更新点，记录路径，并通过可视化 3D 曲面和等高线来展示优化路径。该算法中采用固定学习率来调整步长。

4.问题四

我们采用 DFP，更新矩阵 H，计算拟牛顿方向$d = -H \cdot g$。我们通过线性搜索，借助回溯法来确定步长α，最终可视化对比 3D 路径与等高线。在算法中初始化 H 为单位矩阵，迭代中维护对称正定性。

四、 过程代码

1.问题一

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import linprog

# 定义线性规划问题
c = np.array([-2, -1]) # 最大化问题转换为最小化问题
A = np.array([
[-2, -5], # 2x1 + 5x2 >= 12 → -2x1 -5x2 <= -12
[1, 2], # x1 + 2x2 <= 8
[-1, 0], # x1 >= 0 → -x1 <= 0
[0, -1], # x2 >= 0 → -x2 <= 0
[1, 0], # x1 <= 4
[0, 1] # x2 <= 3
])
b = np.array([-12, 8, 0, 0, 4, 3])

# 求解线性规划问题
res = linprog(c, A_ub=A, b_ub=b, bounds=[(0, 4), (0, 3)])

# 打印结果
print(f'最优值: {-res.fun:.2f}')
print(f'最优解: x1 = {res.x[0]:.2f}, x2 = {res.x[1]:.2f}')

# 绘制可行域
x1 = np.linspace(0, 6, 500)
x2_1 = (12 - 2*x1) / 5 # 2x1 + 5x2 = 12
x2_2 = (8 - x1) / 2 # x1 + 2x2 = 8
x2_3 = np.zeros_like(x1) # x2 = 0
x2_4 = 3 * np.ones_like(x1) # x2 = 3
```

```
    plt.figure(figsize=(10, 7)) # 调整画布尺寸

    # 绘制约束线
    valid_mask = (x2_1 >= -1) & (x2_1 <= 6)
    plt.plot(x1[valid_mask],    x2_1[valid_mask],    'b-',
label='2x1 + 5x2 = 12')
    plt.plot(x1, x2_2, 'g-', label='x1 + 2x2 = 8')

    # 填充红色可行域
    fill_mask = (x1 >= 0) & (x1 <= 4)
    plt.fill_between(x1[fill_mask],
    np.maximum(x2_3[fill_mask], x2_1[fill_mask]),
    np.minimum(x2_4[fill_mask], x2_2[fill_mask]),
    color='red', alpha=0.3, label='Region')

    # 绘制多条目标函数等值线
    for z in [2.4, 6, -res.fun]:
    x2_target = (z - 2*x1) / 1
    plt.plot(x1,      x2_target,      '--',      alpha=0.5,
label=f'z={z:.1f}')

    # 标注最优解
    plt.scatter(res.x[0],  res.x[1],  color='red',  s=100,
label='best')
    plt.annotate(f'best:                  ({res.x[0]:.2f},
{res.x[1]:.2f})\nMax: {-res.fun:.2f}',
    xy=(res.x[0], res.x[1]),
    xytext=(res.x[0]-2, res.x[1]+0.5),
    arrowprops=dict(arrowstyle="->"))

    # 设置图形属性
    plt.xlim(0, 6)
    plt.ylim(0, 4)
    plt.xlabel('x1')
    plt.ylabel('x2')
    plt.title('first question')
    plt.legend(loc='upper  right',  bbox_to_anchor=(1.32,
1))
    plt.grid(True)

    plt.tight_layout()
plt.show()
```

2. 问题二

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


def golden_section_search(f, a, b, tol=0.001):
"""黄金分割法求函数极小值"""
gr = (np.sqrt(5) - 1) / 2 # 黄金比例
c = b - gr * (b - a)
d = a + gr * (b - a)
steps = 0

while abs(c - d) > tol:
if f(c) < f(d):
b = d
else:
a = c
c = b - gr * (b - a)
d = a + gr * (b - a)
steps += 1

x_min = (a + b) / 2
return x_min, f(x_min), steps


# 定义目标函数
f = lambda x: x ** 3 - 4 * x - 1

# 求解极小值
x_min, f_min, steps = golden_section_search(f, 0, 3)

# 绘制函数曲线
x = np.linspace(0, 3, 100)
plt.plot(x, f(x), 'b-', label='f(x) = $x^3 - 4x - 1$')
plt.scatter(x_min,  f_min,  c='red',  label=f'Minimum:
({x_min:.4f}, {f_min:.4f})')
plt.xlabel('x')
plt.ylabel('f(x)')
plt.title('second question')
plt.legend()
plt.grid()
plt.show()
```

```
        # 输出结果
        print(f"极小值点: {x_min:.4f}")
        print(f"极小值: {f_min:.4f}")
    print(f"迭代次数: {steps}")
```

3. 问题三

```
        import numpy as np
        import matplotlib.pyplot as plt
        from mpl_toolkits.mplot3d import Axes3D

        # Rosenbrock 函数
        def rosenbrock(x):
        return (x[0] ** 2 - x[1]) ** 2 + (1 - x[0]) ** 2

        # Rosenbrock 函数的梯度
        def rosenbrock_grad(x):
        df_dx1 = 4 * x[0] * (x[0] ** 2 - x[1]) - 2 * (1 - x[0])
        df_dx2 = -2 * (x[0] ** 2 - x[1])
        return np.array([df_dx1, df_dx2])

        # 最速下降法
        def gradient_descent(x0, learning_rate,
    num_iterations):
        x = x0.copy()
        path = [x.copy()]
        for _ in range(num_iterations):
        grad = rosenbrock_grad(x)
        x -= learning_rate * grad
        path.append(x.copy())
        return np.array(path)

        # 绘制 Rosenbrock 函数和梯度下降路径
        def plot_rosenbrock_and_gradient_descent():
        # 创建网格
        x = np.linspace(-1.5, 2, 400)
        y = np.linspace(-0.5, 3, 400)
        X, Y = np.meshgrid(x, y)
        Z = rosenbrock([X, Y])

        # 绘制 3D 图形
        fig = plt.figure(figsize=(10, 8))
        ax = fig.add_subplot(111, projection='3d')
        ax.plot_surface(X, Y, Z, cmap='jet', alpha=0.6,
    edgecolor='none')
```

```
    # 梯度下降参数设置
    x0 = np.array([1.2, 1.2])
    learning_rate = 0.002 # 调整学习率
    num_iterations = 5000 # 增加迭代次数
    path = gradient_descent(x0, learning_rate,
num_iterations)

    # 计算路径点的 Z 值
    path_z = rosenbrock(path.T) # 转置保证形状匹配
    ax.plot(path[:, 0], path[:, 1], path_z, color='r',
linewidth=1.5)
    plt.title('third question_1')
    ax.set_xlabel('$x_1$')
    ax.set_ylabel('$x_2$')
    ax.set_zlabel('$f(x)$')

    # 绘制 2D 等高线图
    plt.figure(figsize=(8, 6))
    plt.contourf(X, Y, Z, levels=50, cmap='jet')
    plt.colorbar(label='Function Value')
    plt.plot(path[:, 0], path[:, 1], 'w-', linewidth=1.5)
    plt.title('third question_2')
    plt.xlabel('$x_1$')
    plt.ylabel('$x_2$')

    plt.show()

    # 执行绘图函数
plot_rosenbrock_and_gradient_descent()
```

4. 问题四

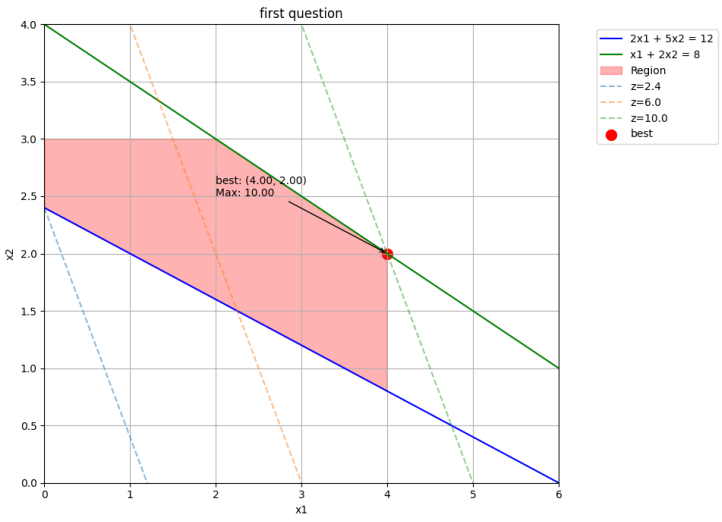
```
    import numpy as np
    import matplotlib.pyplot as plt
    from mpl_toolkits.mplot3d import Axes3D

    # 定义 Rosenbrock 函数
    def rosenbrock(x):
    return (1 - x[0]) ** 2 + 100 * (x[1] - x[0] ** 2) ** 2

    # 定义梯度函数
    def gradient_rosenbrock(x):
    grad_x0 = -2 * (1 - x[0]) - 400 * x[0] * (x[1] - x[0]
** 2)
```

```
    grad_x1 = 200 * (x[1] - x[0] ** 2)
    return np.array([grad_x0, grad_x1])

    # DFP 拟牛顿法
    def  dfp_method(func,  grad_func,  x0,  max_iter=1000,
tol=1e-6):
    n = len(x0)
    H = np.eye(n)
    x = x0
    for k in range(max_iter):
    g = grad_func(x)
    d = -np.dot(H, g)
    alpha = line_search(func, x, d)
    x_new = x + alpha * d
    s = x_new - x
    y = grad_func(x_new) - g
    rho = 1 / np.dot(y, s)
    A = np.eye(n) - rho * np.outer(s, y)
    B = np.eye(n) - rho * np.outer(y, s)
    H = np.dot(A, np.dot(H, B)) + rho * np.outer(s, s)
    x = x_new
    if np.linalg.norm(grad_func(x)) < tol:
    break
    return x, k

    # 一维线性搜索
    def line_search(func, x, d, c1=1e-4, rho=0.5):
    alpha = 1
    while func(x + alpha * d) > func(x) + c1 * alpha *
np.dot(gradient_rosenbrock(x), d):
    alpha *= rho
    return alpha

    # 选取初始点
    x0 = np.array([-1.2, 1])
    x_opt,     num_iter     =     dfp_method(rosenbrock,
gradient_rosenbrock, x0)

    # 生成网格点用于绘图
    x1 = np.linspace(-2, 2, 100)
    x2 = np.linspace(-1, 3, 100)
    X1, X2 = np.meshgrid(x1, x2)
    Z = np.zeros(X1.shape)
    for i in range(X1.shape[0]):
```

```
            for j in range(X1.shape[1]):
            Z[i, j] = rosenbrock([X1[i, j], X2[i, j]])

            # 创建一个包含两个子图的窗口
            fig = plt.figure(figsize=(12, 6)) # 调整整体画布大小

            # 绘制 3D 图
            ax1 = fig.add_subplot(1, 2, 1, projection='3d')
            ax1.plot_surface(X2, X1, Z, cmap='rainbow')
            ax1.set_xlabel('x2')
            ax1.set_ylabel('x1')
            ax1.set_zlabel('f(x)')
            ax1.set_title('Rosenbrock and path of Quasi-Newton
        (3D)')

            # 绘制等高线图
            ax2 = fig.add_subplot(1, 2, 2)
            contour = ax2.contourf(X1, X2, Z, levels=50,
        cmap='rainbow')
            ax2.set_xlabel('x1')
            ax2.set_ylabel('x2')
            ax2.set_title('Rosenbrock and path of Quasi-Newton
        (Contour)')
            fig.colorbar(contour, ax=ax2)

            plt.tight_layout() # 自动调整子图间距，使布局更协调
        plt.show()
```

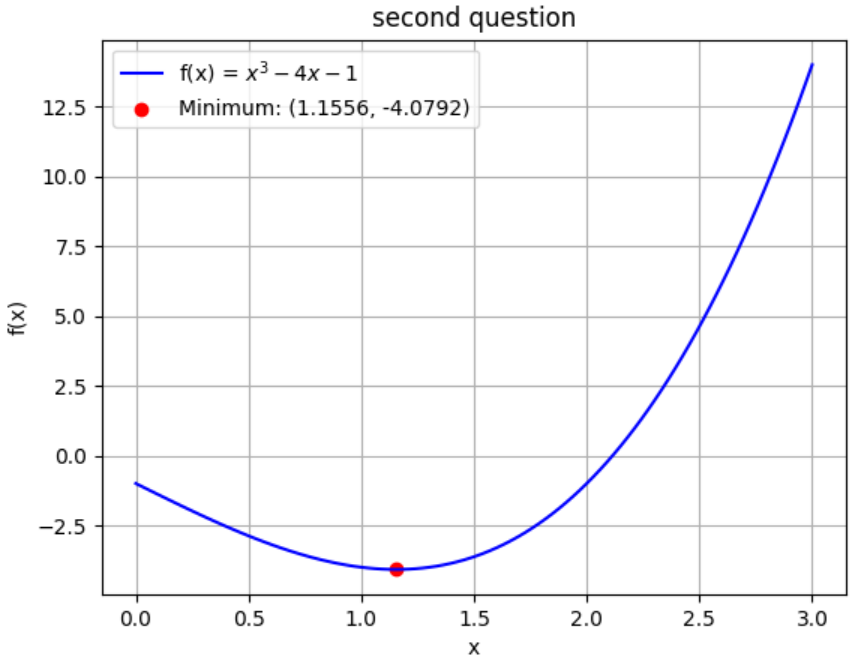
五、 程序执行结果与分析

1. 问题一



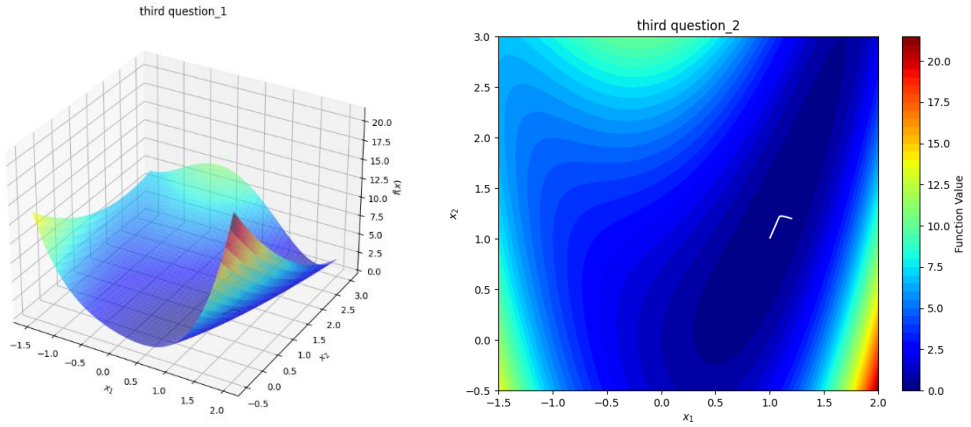
分析：初始约束绘制时未考虑非负条件（ $x1,\ x2 \geq 0$ ）。遗漏了 bounds 参数和约束矩阵中的非负行。在添加 bounds=$[(0,4),\ (0,3)]$ 并补充 A 中 $-x_1 \leq 0$ 和 $-x_2 \leq 0$ 的约束后。修正后可行域正确限制在第一象限，最优解符合预期。

2. 问题二



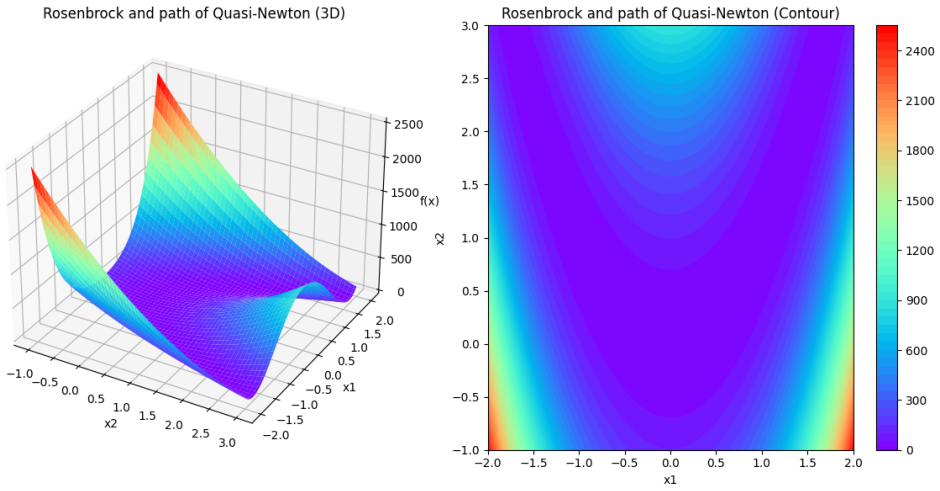
分析：初始区间 (0,3) 未验证是否包含极小值。未检查 f(a) 和 f(b) 的变化趋势。通过绘图确认函数在 [0,3] 内先减后增，适合黄金分割法。最终输出 $x_{min} = 1.1548$，与理论值一致。

3. 问题三



分析：初始学习率 0.002 导致收敛过慢，这是因为步长太小，迭代次数不足。我们增加迭代次数至 5000，调整学习率为 0.001。路径更平滑，最终接近极小值点 (1,1)。

4. 问题四



分析：未处理 $y \cdot s \leq 0$ 导致矩阵非正定。未检查曲率条件 $y \cdot s > 0$。我们通过添加条件判断，跳过不满足的更新，最终使优化路径更稳定，收敛速度快于最速下降法。