凸优化课程实验报告

1 实验目标

（1） 理解并在数学层面解决凸优化问题：注水。题干如下：

$$\begin{aligned} & minimize \quad && -\sum_{i=1}^{n} \log\left(\alpha_i + x_i\right) \\ & subject\ to \quad && x \succcurlyeq 0, \qquad I^T x = 1 \end{aligned}$$

（2） 编程实现寻求该优化问题最优解的过程；

（3） 绘制数据变化图像，实现求最优过程可视化。

2 实验过程

2.1 问题分析

这是一个含约束的凸优化问题。对不等式约束 $x \succcurlyeq 0$ 引入 $Lagrange$ 乘子 λ，对等式约束 $I^T x = 1$ 引入乘子 v，我们得到如下 KKT 条件

$$\begin{cases} x^* \succcurlyeq 0 & (1) \\ I^T x^* = 1 & (2) \\ \lambda^* \succcurlyeq 0 & (3) \\ \lambda_i^* x_i^* = 0, \quad i = 1,2,\ldots,n & (4) \\ -\dfrac{1}{\alpha_i + x_i^*} - \lambda_i^* + v^* = 0, \quad i = 1,2,\ldots,n & (5) \end{cases}$$

可以直接求解这些方程得到 x^*，λ^*，以及v^*。注意到 λ^* 在最后一个方程里是一个松弛变量，所以可以消去得

$$\lambda_i^* = -\frac{1}{\alpha_i + x_i^*} + v^*$$

带入(4)得

$$\begin{cases} x^* \succcurlyeq 0 & (1) \\ I^T x^* = 1 & (2) \\ -\dfrac{1}{\alpha_i + x_i^*} + v^* \geq 0 & (3) \\ \left(-\dfrac{1}{\alpha_i + x_i^*} + v^*\right) x_i^* = 0 & (4) \end{cases}$$

1） 当$v^* \geq \frac{1}{\alpha_i}$，若 $x_i^* > 0$，则$-\frac{1}{\alpha_i+x_i^*} + v^* \geq \frac{1}{\alpha_i} - \frac{1}{\alpha_i+x_i^*} > 0$

则 x_i^* 和$-\frac{1}{\alpha_i+x_i^*} + v^*$ 均不为零，无法满足互补松弛条件。

故$x_i^* = 0$。

2） 当$v^* < \frac{1}{\alpha_i}$，即 $\frac{1}{\alpha_i} > v^* \geq \frac{1}{\alpha_i} - \frac{1}{\alpha_i+x_i^*}$，那么$\frac{1}{\alpha_i} > \frac{1}{\alpha_i} - \frac{1}{\alpha_i+x_i^*}$

又因为$\alpha_i + x_i > 0$

故有 $x_i^* > 0$，即 x_i^* 不为零

那么为了满足互补松弛条件：$v^* = \frac{1}{\alpha_i+x_i^*}$

即$x_i^* = \frac{1}{v^*} - \ \alpha_i$

综上可得

$$x_i^* = \begin{cases} 0 & v^* \geq \dfrac{1}{\alpha_i} \\ \dfrac{1}{v^*} - \ \alpha_i & v^* < \dfrac{1}{\alpha_i} \end{cases}$$

即

$$x_i^* = \max\{0, \frac{1}{v^*} - \ \alpha_i\}$$

由等式约束(2)得

$$\sum_{i=0}^{n} x_i^* = \sum_{i=0}^{n} \max\{0, \frac{1}{v^*} - \ \alpha_i\} = 1$$

2.2 可视化：注水

上述解决问题的方法称为注水。在第 i 片区域，将 α_i 视作该区域的水平线，即图 1 中的红

色线段。向该虚拟容器中注水，当水平面高度到达 $\frac{1}{v^*}$，i 区域的水的深度即为其对应的 x_i^* 星。对于本问题的最优解 x_i^* 和对偶问题的解 v^*，必有水的总量为 1。故令水的总量为 1，对应的高度即为 v^*，同时可以获得所有的 x_i^*。

2.3 编程实现

2.3.1 数据准备

利用 **C++**的 **time()**函数随机生成参考范围内的 20 个浮点数作为 α_i 的 20 个分量。
参考范围：(0,1)
精度：1e-10

```
// 设置随机种子
std::srand(static_cast<unsigned int>(std::time(nullptr)));

//随机生成实验所用的20个alpha向量的分量数据
//参考范围（0，1）
//精度为10^(-10)
for (int i = 0; i < D; ++i) {
    alpha[i] = static_cast<double>(rand()) / RAND_MAX;  // 生成（0，1）内的随机浮点数
    alpha[i] = std::round(alpha[i] * 1e11) / 1e11;  // 限制精度到小数点后10位

}
```

2.3.2 函数设计

考虑定义两个函数分别求解最优解和最优值。

(1)最优解（包括拉格朗日乘子 v 的解）:**Solution()**函数

输入 **alpha[]**、**x[]**、**height**，利用 **max()**函数计算 **x[i]**，并通过累加求得当前 **height** 下的水容量。

```
//通过计算水深总和获得原问题最优解x和对应乘子v的倒数(即水的高度height)
//通过二分法来求解
void Solution(double* alpha, double* x, double& height)
{

    //对于每一个v计算出一个总容量，初始值设为0
    double curVolume = 0.0;

    //初始范围设置为(0,1)
    double left = 0.0;
    double right = 1.0;
```

```
//计算在当下的v的标准下的x分量以及curVolume
for (int i = 0; i < D; i++)
{
    x[i] = max(0.0, height-alpha[i]);

    //累加x[i]即为总容量
    curVolume += x[i];

}
```

利用二分法计算 **height** 值。将累加值与 **1** 进行比较，若偏大，则 **height** 选取过大，在 **(height,right)**上选取中点作为新的 **height** 值；若偏小则相反。

```
//若累加值大于1，则所选height过高
if (curVolume > 1.0)
{
    //将此次循环的height作为上临界
    right = height;

}

//若累加值小于1，则所选height过低
if (curVolume < 1.0)
{
    //将此次循环的height作为下临界
    left = height;
}
```

```
//取区间中点作为新的height
height = (left + right) / 2.0;
```

当总容量与 1 的误差在所设定的范围之内（本代码设置的是低于 1e-12），跳出循环。

```
//x[i]累加值与目标值1的误差小于10的-12次方，视为相等
if (fabs(curVolume - 1.0) < 1e-12) break;
```

(2)最优值计算函数:**BestValue()**

输入 **alpha[]**和 **x[]**进行计算累加即可。

```
//计算最优值
double BestValue(double* alpha, double* x)
{
    double sum = 0;
    for (int i = 0; i < D; i++)

        sum += -log(alpha[i] + x[i]) / log(exp(1));

    return sum;
}
```

3 实验结果

生成的 alpha[]数组:

```
alpha的20个分量:
alpha[0] = 0.7936948759
alpha[1] = 0.2746971038
alpha[2] = 0.4771568957
alpha[3] = 0.1591540269
alpha[4] = 0.0669881283
alpha[5] = 0.5167394025
alpha[6] = 0.0437635426
alpha[7] = 0.1458784753
alpha[8] = 0.3537400433
alpha[9] = 0.9955442976
alpha[10] = 0.3498336741
alpha[11] = 0.7209082308
alpha[12] = 0.9869380779
alpha[13] = 0.5076143681
alpha[14] = 0.1197546312
alpha[15] = 0.8697164831
alpha[16] = 0.0339060640
alpha[17] = 0.2790307321
alpha[18] = 0.8904995880
alpha[19] = 0.8354441969
```

计算得到的 x[]数组(最优解):

```
x*的20个分量:
x[0] = 0.0000000000
x[1] = 0.0000000000
x[2] = 0.0000000000
x[3] = 0.1024201178
x[4] = 0.1945860164
x[5] = 0.0000000000
x[6] = 0.2178106021
x[7] = 0.1156956694
x[8] = 0.0000000000
x[9] = 0.0000000000
x[10] = 0.0000000000
x[11] = 0.0000000000
x[12] = 0.0000000000
x[13] = 0.0000000000
x[14] = 0.1418195135
x[15] = 0.0000000000
x[16] = 0.2276680807
x[17] = 0.0000000000
x[18] = 0.0000000000
x[19] = 0.0000000000
```

该最优解 x[]数组的和(水的总容量):

```
循环结束时容量为: 1.0000000000
```

乘子 1/v*和最优值:

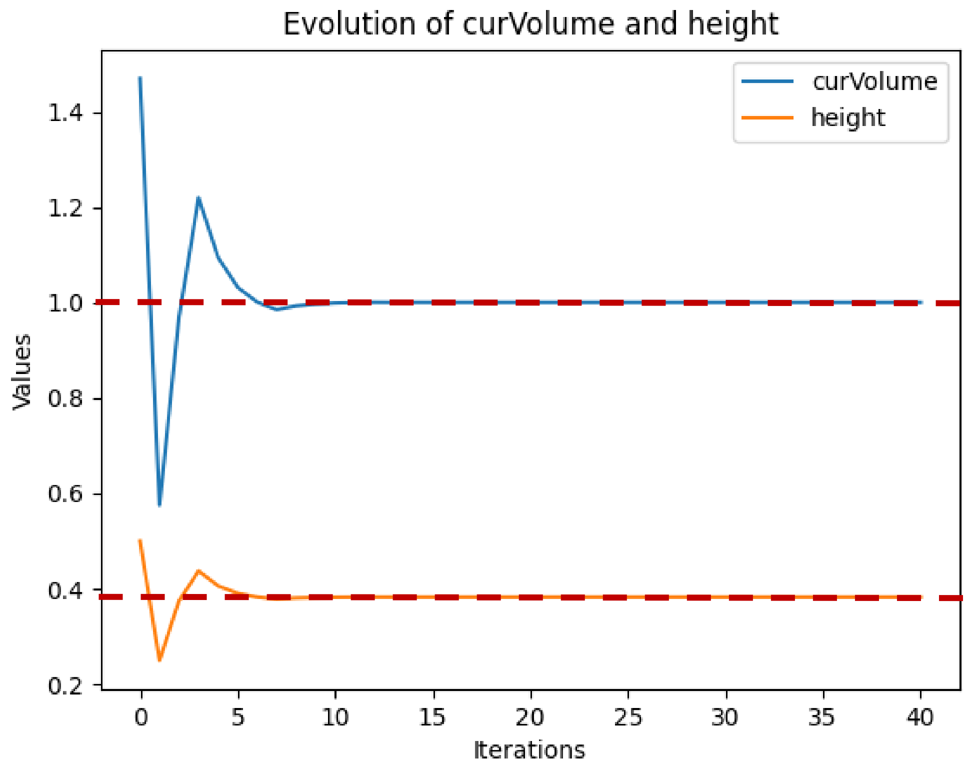
```
1/v*的值:0.2615741447

p*的值:15.7936603664
```

4 绘图与实验结果分析

4.1 二分法求解逼近过程图示

基于 python 进行了在 Solution 函数中 curVolume(水的总容量)和 height 的变化过程。

```python
    # 绘制 cur_volume 和 height 的变化过程
    plt.plot(cur_volume_history, label='curVolume')
    plt.plot(height_history, label='height')
    plt.xlabel('Iterations')
    plt.ylabel('Values')
    plt.legend()
    plt.title('Evolution of curVolume and height')
    plt.show()
```

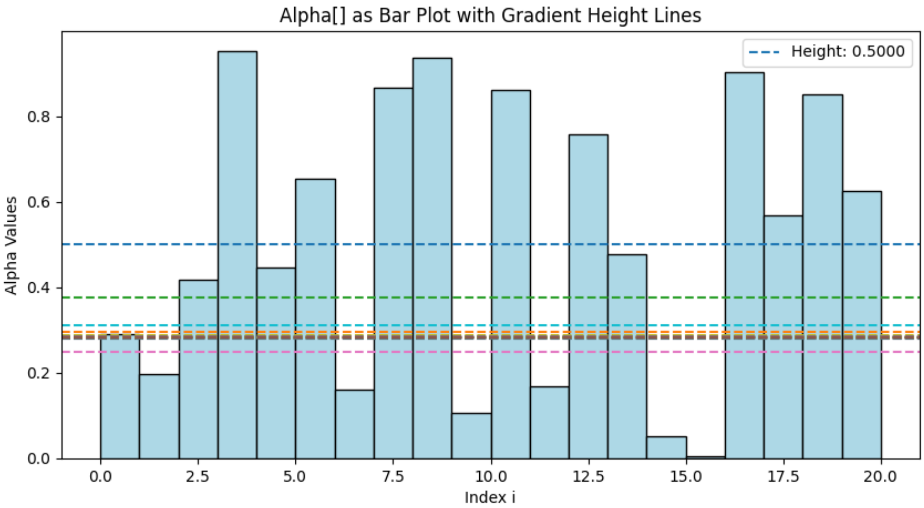


如图所示，随着循环次数的增加，curVolume 在几次条约后逐渐逼近 1.0。Height 的走向与 curVolume 也一致。这是由于 curVolume 的值(也就是 x[]的累加和)与水的高度 height 是呈线性增函数关系的。

4.2 注水图像绘制

基于 python 进行了 alpha[]柱状图和每次循环 height 水平线的绘制。并通过代码设置随着循环次数的增加，height 水平线颜色逐渐加深。

```
# 绘制 alpha[] 的柱状图
plt.figure(figsize=(10, 5))
plt.bar(range(len(alpha)), alpha, width=1, color='lightblue', align='edge', edgecolor='black')

# 添加 height 的水平线和渐变色
num_iterations = len(height_values)
for i, h in enumerate(height_values):
    # 计算渐变色，随着循环次数增加颜色变深
    color_val = i / (num_iterations - 1)  # 0 到 1 的渐变色值
    color = mcolors.to_rgba(f"C{int(color_val * 255)}")[:3]  # 按照渐变色值生成颜色
    plt.axhline(y=h, linestyle='--', color=color, label=f'Height: {h:.4f}' if i == 0 else None)
```



5 结论

利用凸优化相关知识，调用 KKT 条件，分析并完成了注水问题。

在精度为小数点后十位，维度为 20 的条件下，通过编程求得了 alpha[]数组与 x[]数组，找到了最优解，计算求得了最优值。

基于 python 绘制了程序求解过程的可视化图例。

6 源码

6.1 求解优化问题

(C++)

```
#include <iostream>
#include <iomanip>
#include <cstdlib>
#include <ctime>
#include<cmath>
#define D 20 //宏定义该问题中的x和alpha向量的维度

using namespace std;


//通过计算水深总和获得原问题最优解x和对应乘子v的倒数(即水的高度height)
//通过二分法来求解
void Solution(double* alpha, double* x, double& height)
{
```

```
//对于每一个v计算出一个总容量，初始值设为0
double curVolume = 0.0;

//初始范围设置为(0,1)
double left = 0.0;
double right = 1.0;


do {
    //每次循环开始前将curVolume置零
    curVolume = 0.0;

    //取区间中点作为新的height
    height = (left + right) / 2.0;

    //计算在当下的v的标准下的x分量以及curVolume
    for (int i = 0; i < D; i++)
    {
        x[i] = max(0.0, height-alpha[i]);

        //累加x[i]即为总容量
        curVolume += x[i];

    }

    //x[i]累加值与目标值1的误差小于10的-12次方，视为相等
    if (fabs(curVolume - 1.0) < 1e-12) break;
    else
    {
        //若累加值大于1，则所选height过高
        if (curVolume > 1.0)
        {
            //将此次循环的height作为上临界
            right = height;

        }

        //若累加值小于1，则所选height过低
        if (curVolume < 1.0)
        {
            //将此次循环的height作为下临界
            left = height;
        }
    }
```

```
    } while (1);

    //规定输出精确度到小数点后10位
    cout << fixed << setprecision(10);
    cout << "循环结束时容量为：" << curVolume << endl << endl;
}

//计算最优值
double BestValue(double* alpha, double* x)
{
    double sum = 0;
    for (int i = 0; i < D; i++)

        sum += -log(alpha[i] + x[i]) / log(exp(1));

    return sum;
}



int main() {
    double alpha[D] = { 0 };
    double x[D] = { 0 };
    double height = 0;

    // 设置随机种子
    std::srand(static_cast<unsigned int>(std::time(nullptr)));

    //随机生成实验所用的20个alpha向量的分量数据
    //参考范围（0，1）
    //精度为10^(-10)
    for (int i = 0; i < D; ++i) {
        alpha[i] = static_cast<double>(rand()) / RAND_MAX;  // 生成（0，1）内
的随机浮点数
        alpha[i] = std::round(alpha[i] * 1e11) / 1e11;  // 限制精度到小数点后
10位

    }

    //求最优解
    Solution(alpha, x, height);

    cout << fixed << setprecision(10);
    //输出alpha向量的20个分量
```

```
    cout << "alpha的20个分量:" << endl;
    for (int i = 0; i < D; ++i) {
        cout << "alpha[" << i << "] = " << alpha[i] << endl;
    }
    cout << endl;

    //输出最优解的20个分量
    cout << "x*的20个分量:" << endl;
    for (int i = 0; i < D; ++i) {
        cout << "x[" << i << "] = " << double(x[i]) << endl;
    }

    //输出对应乘子
    cout << endl<< "1/v*的值:" << height << endl;

    //输出原问题的最优值
    cout << endl << "p*的值:" << BestValue(alpha, x) << endl;

    return 0;
}
```

6.2 绘图

(python)

逼近图:

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

def solution(alpha):
    D = len(alpha)
    x = np.zeros(D)
    height = 0.0

    # 对于每一个 v 计算出一个总容量, 初始值设为 0
    cur_volume = 0.0

    # 初始范围设置为 (0, 1)
    left = 0.0
    right = 1.0

    # 存储每次循环的 height
    height_values = []

    while True:
        # 每次循环开始前将 cur_volume 置零
```

```
        cur_volume = 0.0

        # 取区间中点作为新的 height
        height = (left + right) / 2.0

        # 计算在当前的 v 的标准下的 x 分量以及 cur_volume
        for i in range(D):
            x[i] = max(0.0, height - alpha[i])

            # 累加 x[i] 即为总容量
            cur_volume += x[i]

        # 存储每次循环的 height
        height_values.append(height)

        # x[i] 累加值与目标值 1 的误差小于 1e-12 次方，视为相等
        if np.abs(cur_volume - 1.0) < 1e-12:
            break
        else:
            # 若累加值大于 1，则所选 height 过高
            if cur_volume > 1.0:
                # 将此次循环的 height 作为上临界
                right = height
            # 若累加值小于 1，则所选 height 过低
            if cur_volume < 1.0:
                # 将此次循环的 height 作为下临界
                left = height

    # 绘制 alpha[] 的柱状图
    plt.figure(figsize=(10, 5))
    plt.bar(range(len(alpha)), alpha, width=1, color='lightblue',
align='edge', edgecolor='black')

    # 添加 height 的水平线和标签
    for i, h in enumerate(height_values):
        plt.axhline(y=h, linestyle='--', color='red',
label=f'Height: {h:.4f}' if i == 0 else None)

    plt.xlabel('Index i')
    plt.ylabel('Alpha Values')
    plt.title('Alpha[] as Bar Plot with Height Lines')
    plt.legend()
    plt.show()
```

```python
# 测试函数
alpha_test = np.random.rand(20)  # 随机生成 alpha 数组
solution(alpha_test)
```

柱状图:

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

def solution(alpha):
    D = len(alpha)
    x = np.zeros(D)
    height = 0.0

    # 对于每一个 v 计算出一个总容量，初始值设为 0
    cur_volume = 0.0

    # 初始范围设置为 (0, 1)
    left = 0.0
    right = 1.0

    # 存储每次循环的 height
    height_values = []

    while True:
        # 每次循环开始前将 cur_volume 置零
        cur_volume = 0.0

        # 取区间中点作为新的 height
        height = (left + right) / 2.0

        # 计算在当前的 v 的标准下的 x 分量以及 cur_volume
        for i in range(D):
            x[i] = max(0.0, height - alpha[i])

            # 累加 x[i] 即为总容量
            cur_volume += x[i]

        # 存储每次循环的 height
        height_values.append(height)

        # x[i] 累加值与目标值 1 的误差小于 1e-12 次方，视为相等
        if np.abs(cur_volume - 1.0) < 1e-12:
            break
```

```
        else:
            # 若累加值大于 1，则所选 height 过高
            if cur_volume > 1.0:
                # 将此次循环的 height 作为上临界
                right = height
            # 若累加值小于 1，则所选 height 过低
            if cur_volume < 1.0:
                # 将此次循环的 height 作为下临界
                left = height

    # 绘制 alpha[] 的柱状图
    plt.figure(figsize=(10, 5))
    plt.bar(range(len(alpha)), alpha, width=1, color='lightblue',
align='edge', edgecolor='black')

    # 添加 height 的水平线和渐变色
    num_iterations = len(height_values)
    for i, h in enumerate(height_values):
        # 计算渐变色，随着循环次数增加颜色变深
        color_val = i / (num_iterations - 1)  # 0 到 1 的渐变色值
        color = mcolors.to_rgba(f"C{int(color_val * 255)}")[:3]  # 按
照渐变色值生成颜色
        plt.axhline(y=h, linestyle='--', color=color,
label=f'Height: {h:.4f}' if i == 0 else None)

    plt.xlabel('Index i')
    plt.ylabel('Alpha Values')
    plt.title('Alpha[] as Bar Plot with Gradient Height Lines')
    plt.legend()
    plt.show()

# 测试函数
alpha_test = np.random.rand(20)  # 随机生成 alpha 数组
solution(alpha_test)
```