数学建模1-魏生辉2023211075

第一题

题目概述：

> 💡 利用Lagrange 插值方法，使插值多项式经过点(xo,yo)=(0,1),(x1,y)=(2,3),(x2,y_2)=(3,0)，(x3,y3)=(5,18)，计算x=2/3处的函数值。进一步，在计算机上利用Lagrange插值的函数命令画出函数图像并计算指定点函数值

解法1：手动计算法

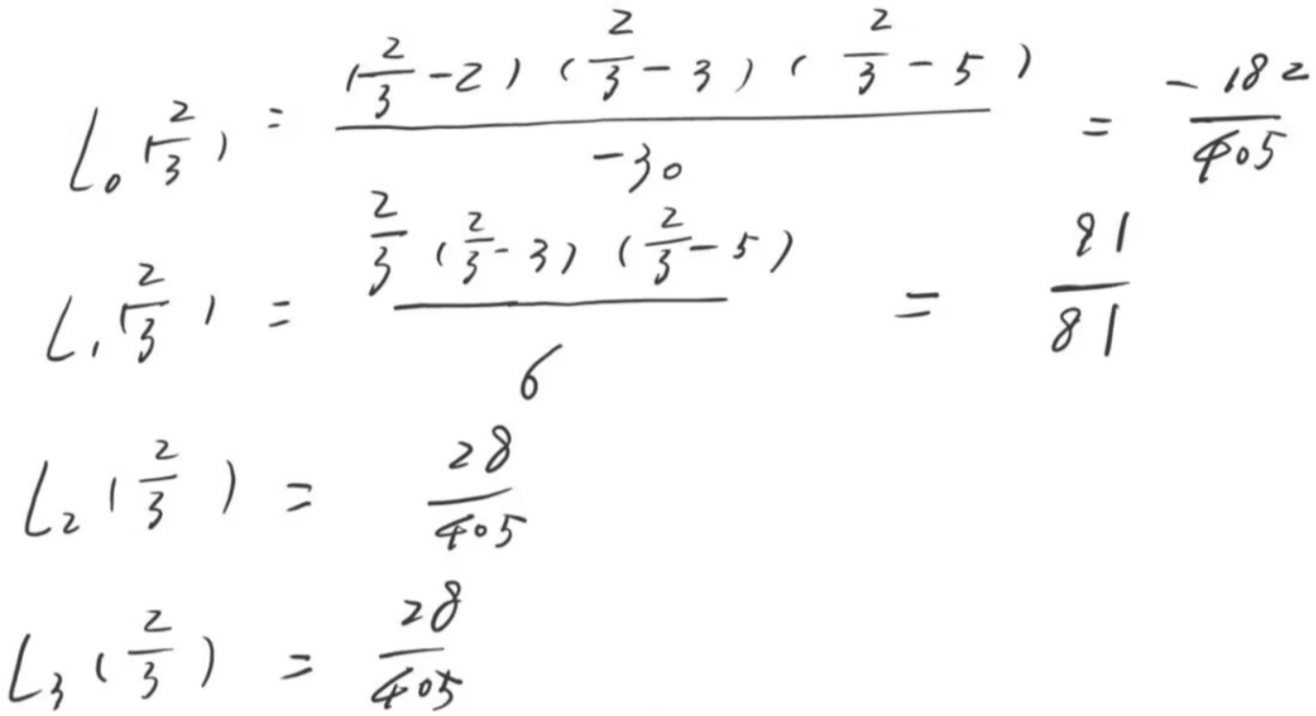
1.我们有四个已知点：

- (x0,y0)=(0,1)
- (x1,y1)=(2,3)
- (x2,y2)=(3,0)
- (x3,y3)=(5,18)

2分别计算几个基函数：

1. **基函数 L0(x)L_0(x)L0(x)**：
2. **基函数 L1(x)L_1(x)L1(x)：**
3. **基函数 L2(x)L_2(x)L2(x)：**
4. **基函数 L3(x)L_3(x)L3(x)：**
5. **构造插值多项式**

```
P(x)=y0 L0(x)+y1 L1(x)+y2 L2(x)+y3 L3(x)
```

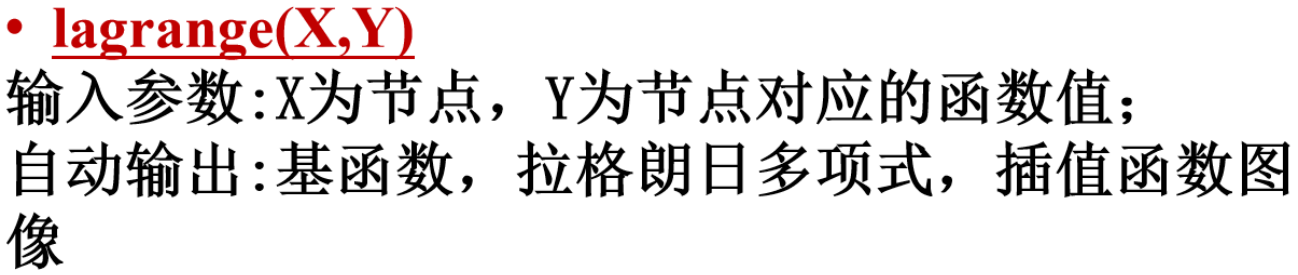


6. 代入

   计算得5.064

解法2：在计算 机上利用Lagrange插值的函数命令画出函数图像并计算指定点函数值

根据ppt指导

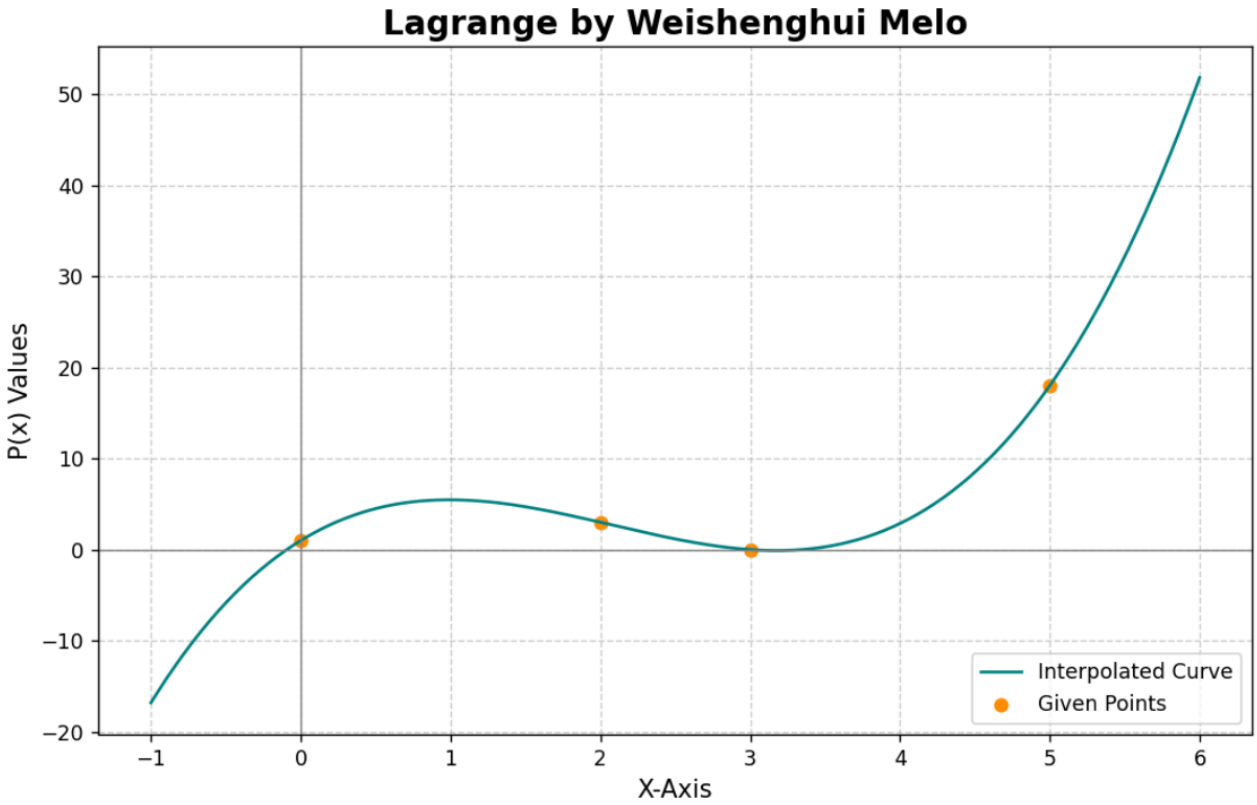


这里笔者想试试py就写了个函数 **lagrange(x_values, y_values, x):**

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

# 定义Lagrange插值函数
def lagrange(x_values, y_values, x):
    n = len(x_values)  # 点的数量
    result = 0.0  # 插值多项式的初始值

    for i in range(n):
```

```
        term = y_values[i]  # 每一项初始为对应的y值
        for j in range(n):
            if i != j:
                term *= (x - x_values[j]) / (x_values[i] - x_values[j])  # 计算Lagrange基函数
        result += term  # 将每一项累加到结果中
    return result

# 数据
xvalues = [0, 2, 3, 5]
yvalues = [1, 3, 0, 18]

# 计算 x = 2/3 处的函数值
x = 2 / 3
result1 = lagrange(xvalues, yvalues, x)
print(f"The value at x = {x} is {result1:.4f}")


def plotlagrange(xvalues, yvalues, x_min, x_max, title):
    x_plot = np.linspace(x_min, x_max, 500)
    y_plot = [lagrange_interpolation(xvalues, yvalues, x) for x in x_plot]  # 计算每个x点的插值多项式值

    plt.figure(figsize=(8, 6))
    plt.plot(x_plot, y_plot, label='Lagrange Polynomial', color='blue')
    plt.scatter(xvalues, yvalues, color='red', label='Given Points')
    plt.title(title)
    plt.xlabel('x')
    plt.ylabel('P(x)')
    plt.legend()
    plt.grid(True)
    plt.axhline(0, color='black',linewidth=0.5)  # 添加x轴
    plt.axvline(0, color='black',linewidth=0.5)  # 添加y轴
    plt.show(block=True)# 这里很重要！！！！！！

# 画出问题1的插值多项式图像
plotlagrange(xvalues, yvalues, -1, 6, "Lagrange by Weishenghui Melo")
```

结果图如下： py输出结果和手动计算完全一致/图示结果大概为5.06

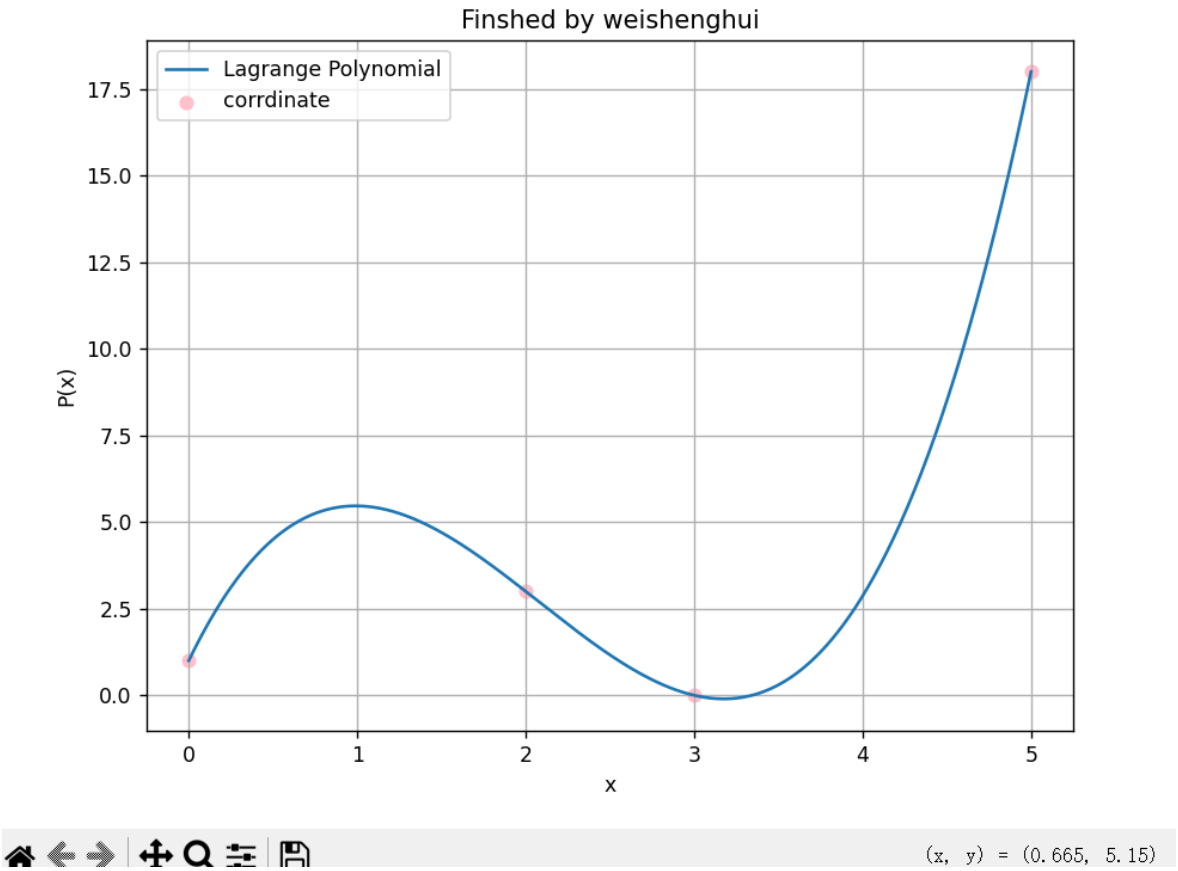


几点说明+小小心得：

1.最开始怎么都运行不了程序 **发现是缺少matplotlib包，**下载好运行就可以了；

2.运行成功后发现居然一直加载不出我想要的图片，查了csdn，**发现plt.show()改成plt.show(block=True)就正常了；**

3.改了一下画图范围 **把-1到6改成了0-5，发现精度会出现一些问题，误差大概0.1-0.2，图示如下**



第二题

题目概述：

利用双线性差值方法，构造插值多项式经过点x0=(1,2),y0 =5, X1=(0,1)，y1= 2, X2=(1,0)，y2= -1 X4=(0,0)，y3= 1,计算x=(1/3，1/3)处的函数值。进一步，在计算机上利用griddata插值的函数计算指定点函数值

解法1：手动计算法

1. 查阅一下计算公式

首先在 x 方向进行线性插值，得到

$$f(R_1) \approx \frac{x_2 - x}{x_2 - x_1} f(Q_{11}) + \frac{x - x_1}{x_2 - x_1} f(Q_{21}) \quad \text{Where} \quad R_1 = (x, y_1),$$

$$f(R_2) \approx \frac{x_2 - x}{x_2 - x_1} f(Q_{12}) + \frac{x - x_1}{x_2 - x_1} f(Q_{22}) \quad \text{Where} \quad R_2 = (x, y_2).$$

然后在 y 方向进行线性插值，得到

$$f(P) \approx \frac{y_2 - y}{y_2 - y_1} f(R_1) + \frac{y - y_1}{y_2 - y_1} f(R_2).$$

这样就得到所要的结果 $f(x, y)$，

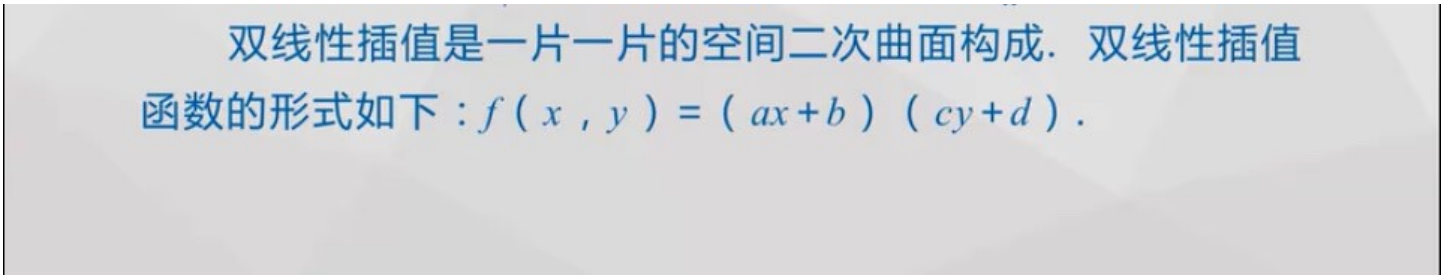
$$f(x,y) \approx \frac{f(Q_{11})}{(x_2 - x_1)(y_2 - y_1)}(x_2 - x)(y_2 - y) + \frac{f(Q_{21})}{(x_2 - x_1)(y_2 - y_1)}(x - x_1)(y_2 - y)$$

$$+ \frac{f(Q_{12})}{(x_2 - x_1)(y_2 - y_1)}(x_2 - x)(y - y_1) + \frac{f(Q_{22})}{(x_2 - x_1)(y_2 - y_1)}(x - x_1)(y - y_1).$$

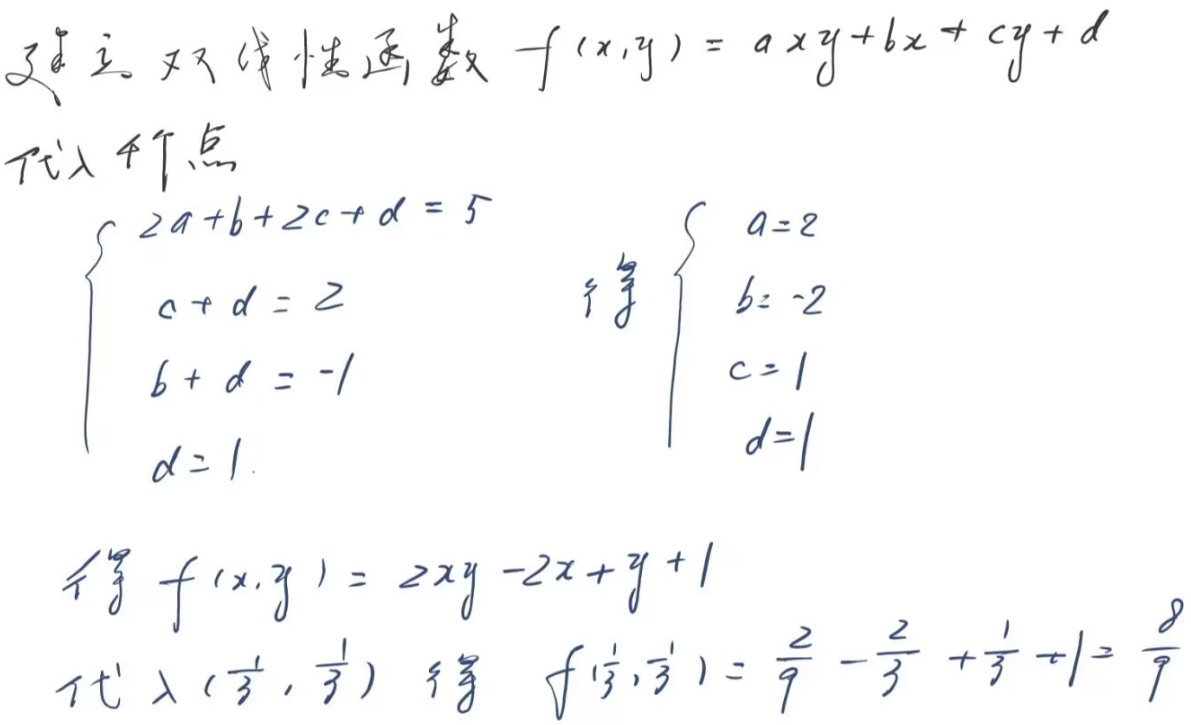
如果选择一个坐标系统使得 f 的四个已知点坐标分别为 (0, 0)、(0, 1)、(1, 0) 和 (1, 1)，那么插值公式就可以化简为

$$f(x,y) \approx f(0,0)\,(1-x)(1-y) + f(1,0)\,x(1-y) + f(0,1)\,(1-x)y + f(1,1)xy.$$

9.23日改： 发现这四个点无法构成矩形，不能使用这个方法，**qq询问老师得到以下计算方式**



2. **代入计算**



解法2：在计算机上利用griddata插值的函数计算指定点函数值

1.查阅csdn上的资料，利用py

通过头文件：`from scipy.interpolate import griddata` 直接调用griddata函数

同时自己手动写一个函数 `bilinear_interpolation` 进行验证：

2.

```python
import numpy as np
from scipy.interpolate import griddata
import matplotlib.pyplot as plt

# 已知的四个插值点和对应的函数值
points = np.array([
    [1, 2],  # (x0, y0)
    [0, 1],  # (x1, y1)
    [1, 0],  # (x2, y2)
    [0, 0]  # (x3, y3)
])
values = np.array([5, 2, -1, 1])  # 对应的函数值

# 目标插值点
target_point = (1 / 3, 1 / 3)



# 使用scipy中的griddata进行插值计算
grid_result = griddata(points, values, target_point, method='linear')#!!!!!!!!!!!!  method有三种 计算出的结果不一样
print(f"griddata计算的插值结果: {grid_result:.4f}")
```

几点说明+小小心得：

1.本想自己写一个griddata 后来发现scipy中的griddata可以直接插值计算；

2.9.24补充：griddata（points, values, target_point, method='cubic'）的最后一个参数很有说法

`griddata` 的 `method` 参数可以更改，常用的插值方法包括：

1. **'linear'**：线性插值（默认方法），适用于不规则点分布。
2. **'nearest'**：最近邻插值，选择最近的已知点的值。
3. **'cubic'**：三次插值，提供更平滑的结果，但需要更多的点，并且要求数据是二维的。

三种 `method` 插值结果输出如下

```
D:\code\python\venv\Scripts\python.exe "D:/我的app/PyCharm Community Edition 2

import sys; print('Python %s on %s' % (sys.version, sys.platform))
sys.path.extend(['D:\\code\\python'])

Python Console
... from scipy.interpolate import griddata
... import matplotlib.pyplot as plt
...
... # 已知的四个插值点和对应的函数值
... points = np.array([
...     [1, 2],  # (x0, y0)
...     [0, 1],  # (x1, y1)
...     [1, 0],  # (x2, y2)
...     [0, 0]  # (x3, y3)
... ])
... values = np.array([5, 2, -1, 1])  # 对应的函数值
...
... # 目标插值点
... target_point = (1 / 3, 1 / 3)
...
...
...
...
... # 使用scipy中的griddata进行插值计算
... grid_result = griddata(points, values, target_point, method='linear')
... print(f"griddata计算的插值结果: {grid_result:.4f}")
...
...
griddata计算的插值结果: 0.6667

>>>
```

```
... from scipy.interpolate import griddata
... import matplotlib.pyplot as plt
...
... # 已知的四个插值点和对应的函数值
... points = np.array([
...     [1, 2],  # (x0, y0)
...     [0, 1],  # (x1, y1)
...     [1, 0],  # (x2, y2)
...     [0, 0]  # (x3, y3)
... ])
... values = np.array([5, 2, -1, 1])  # 对应的函数值
...
... # 目标插值点
... target_point = (1 / 3, 1 / 3)
...
...
...
...
... # 使用scipy中的griddata进行插值计算
... grid_result = griddata(points, values, target_point, method='nearest')
... print(f"griddata计算的插值结果: {grid_result:.4f}")
...
...
griddata计算的插值结果: 1.0000

>>>
```

```
>>> import numpy as np
... from scipy.interpolate import griddata
... import matplotlib.pyplot as plt
...
... # 已知的四个插值点和对应的函数值
... points = np.array([
...     [1, 2],  # (x0, y0)
...     [0, 1],  # (x1, y1)
...     [1, 0],  # (x2, y2)
...     [0, 0]  # (x3, y3)
... ])
... values = np.array([5, 2, -1, 1])  # 对应的函数值
...
... # 目标插值点
... target_point = (1 / 3, 1 / 3)
...
...
...
...
... # 使用scipy中的griddata进行插值计算
... grid_result = griddata(points, values, target_point, method='cubic')
... print(f"griddata计算的插值结果: {grid_result:.4f}")
...
...
...
griddata计算的插值结果: 0.7321
```

🎉 输出结果：**0.6667，0.7321，1.000**，取平均值得到0.7999 很接近我们手动计算的结果。