目录

基于 Python 从头实现可视化插值工具

1120201479-胡栩浩

2021 年 12 月 13 日

1 项目背景

在《数值分析》这门课六个单元的学习中，我选择了“插值法”作为这次项目的选题。插值法在如今的各大科学计算领域有着重要的应用，各种优秀的计算软件例如 matlab, spss 等，各种编程语言的第三方库例如 python 的 scipy 库等都对其插值法做了很好的集成与优化。在熟练运用这些库的基础上，能够回过头来详细深究底层实现方法将更有助于我们科学思维的形成，更有助于我们筑牢基础更好地探索科学的领域。

于我个人而言，由于我在做的许多事情包括搭建网站，包括深度学习都与 Python 有着密不可分的关系，于是我选择了 Python 作为这次大作业的编程语言。同时，为了让项目更具有趣味性，我选择了 PyQT5（Python 桌面程序框架）作为主要工具，实现一个可视化的插值小工具。另外因为我近期需要学习 Latex，我将采用 Latex 来编写此次大作业的文本文档，排版会更加清晰美观也更易于老师审阅。

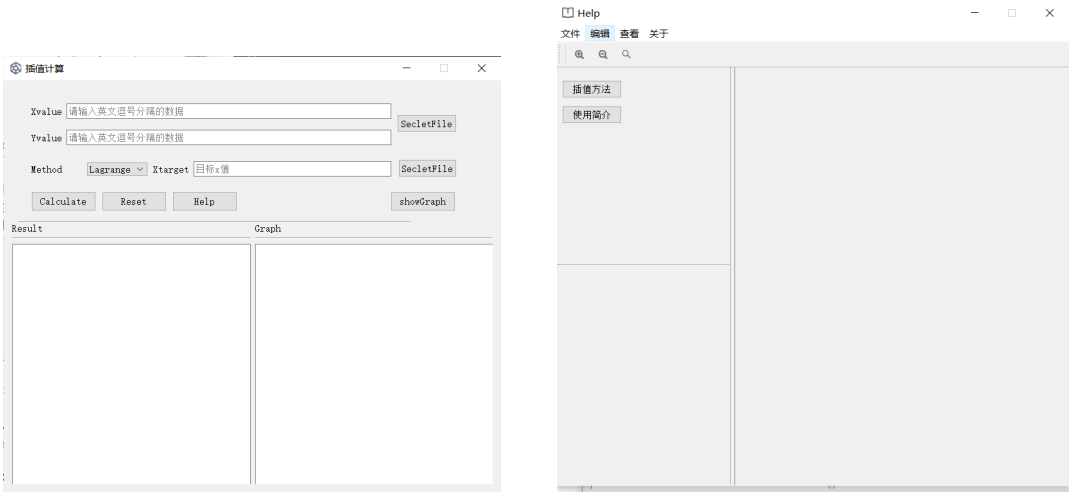
2 项目概览

2.1 主要功能

- 用户输入指定格式数据（手动输入或者选择指定格式文件）
- 用户选择不同的插值方法
- 计算并且显示出计算结果
- 软件自动绘制出可视化图形，并且标注明晰
- 用户使用手册查询与详细说明

2.2 软件可视化部分

数据输入 用户输入以英文字符*’,’* 分隔的数据。也可以点击文件选项按钮选择指定格式的*.csv* 或者*.txt* 文件。需要注意的是文件需要以英文逗号分割，X 数据和 Y 数据应该分别占一列。



(a) 主界面 (b) 帮助界面

图 1: 软件界面

数据计算 本软件提供“牛顿插值法”，“拉格朗日插值法”，“反插值法”，“三次样条插值”这四种方法。通过计算机模拟手动计算的过程来计算指定数据插值。

结果演示 通过 matplotlib 可视化作图来实现插值法的图形可视化，使得结果更加具有一般性和可行性，同时也加强了用户的软件使用体验。

帮助文档 本软件提供了帮助文档，通过点击“Help”按钮可以获得更加详细地使用说明和方法简介，同时也可以联系作者。

2.3 插值方法部分

牛顿插值法 本方法默认采用基本的牛顿插商公式，通过计算机模拟查商表计算，取取插商表第一斜行的数据值进行计算。

拉格朗日插值法 本方法仍然采用计算机模拟手动计算的过程来实现。公式如下：

$$l_i(x) = \frac{(x-x_0)\dots(x-x_(i-1)(x-x_(i+1)\dots(x-x_n)}{(x_i-x_0)\dots(x_i-x_i-1)(x_i-x_i+1)\dots(x_i-x_n)} \tag{1}$$

$$L_n(x) = \sum_{i=0}^{n} y_i l_i(x) \tag{2}$$

三次样条插值 本方法融入了分段插值的思想，给定 n 个数据点，可以获得 $n-1$ 段插值区间，在每一个插值区间内构造三次样条函数 ax^3+bx^2+cx+d。构造方法具体如下，首先建立节点函数值相等的 n 个方程，然后建立首位端点的两个方程，再建立节点处一阶导数相等的 n 个方程，再是二阶导函数相等的 n 个方程，再是二阶导函数相等的，最后建设端点处的二阶导数为 0 构建 2 个方程。共计 $2n+4$ 个方程。

x_i	$f(x_i)$	一阶差商	二阶差商	三阶差商	…
x_0	$f(x_0)$				
x_1	$f(x_1)$	$f[x_0,x_1]$			
x_2	$f(x_2)$	$f[x_1,x_2]$	$f[x_0,x_1,x_2]$		
x_3	$f(x_3)$	$f[x_2,x_3]$	$f[x_1,x_2,x_3]$	$f[x_0,x_1,x_2,x_3,]$	
…	…	…	…	…	…
x_n	$f(x_n)$	$f[x_{n-1},x_n]$	$f[x_{n-2},x_{n-1},x_n]$	$f[x_{n-3},\ldots,x_n]$	…

图 2: 插商表示意图

反插值法 这里我们采取简单的调换 x 和 y 值的方法快速实现，默认采用的是牛顿法。

3 项目实现重点与细节

3.1 界面实现

布局 本项目布局采用 QTDesigner 工具实现拖拽布局，利用 PYUIC 实现.ui 文件转化为.py 文件，从而实现窗口布局。如下图所示：

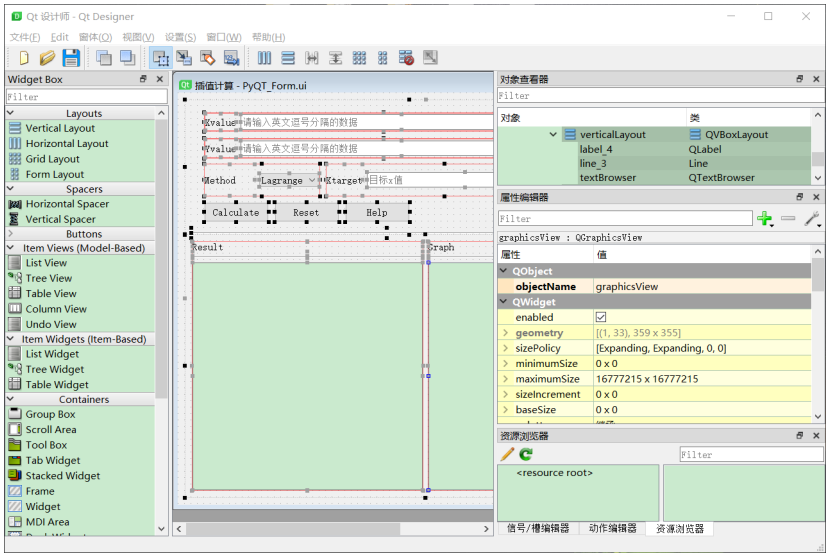


图 3: QTDesiger

输入格式限制 利用 Python 正则表达式来简单实现对于用户输入数据格式的限制 (英文逗号分割的实数数据)，符合格式则可以输入成功并且显示，反之则输入失败，且不会显示在输入框中，正则表达式如下：

```
reg = QRegExp('^(((-?\d+)(\.\d+)?),)+((-?\d+)(\.\d+))$')
single_reg = QRegExp('^(-?\d+)(\.\d+)?$')
```

同时支持通过选择指定格式的文件 (.csv/.txt) 来输入数据并且显示，如下图所示：

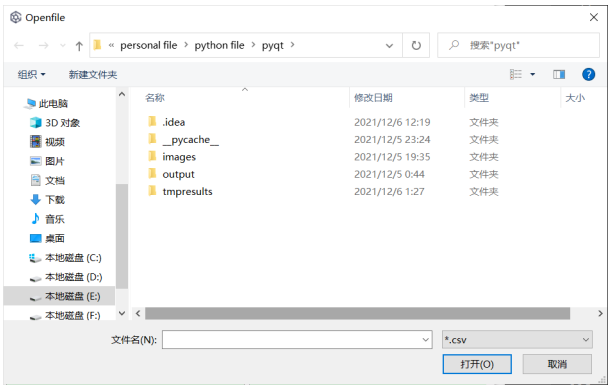


图 4: 打开文件

Calculate 的逻辑 这是一个重要的部分，用户点击计算按钮之后，程序会执行以下步骤，首先是判断是否有输入，然后是根据方法调用不同的 api，随后再将结果显示在 text-browser 上，最后执行 show-in-graph 的绘图操作。操作结果如下图所示：

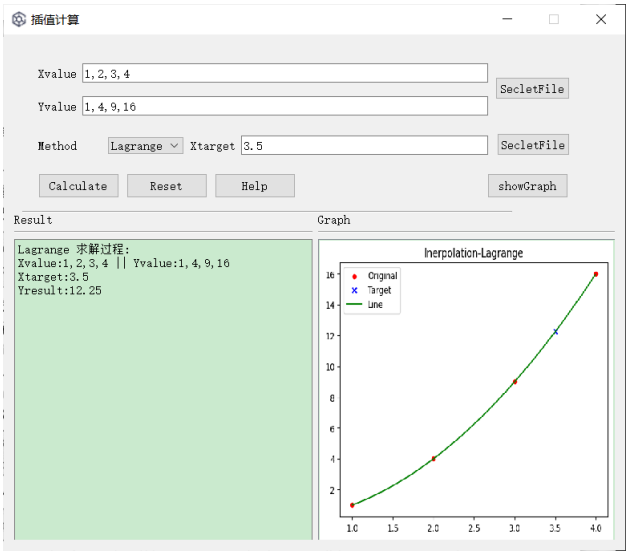


图 5: 测试

显示 结果的文本显示，调用了 self.textbrowser 的 api，结果的图形展示是调用了 self.graphicsview 的 api 将 matplotlib 的图集合到了软件里面。

3.2 插值实现

3.2.1 牛顿插值法

计算机模拟手动求解插商表的关键核心代码是一条双重循环语句，如下：

```
            """从一阶差商开始一直到k阶差商"""
            for i in range(1, k + 1):
            """这是上一阶的插商，是计算这一阶插商的必要条件"""
            last_quotient = diff_quotient[i - 1]
            tmp_quotient = []  # 初始化这一阶插商
            """遍历上一层差商的数值"""
            for j in range(0, len(last_quotient) - 1):
                a = last_quotient[j]
                b = last_quotient[j + 1]
                """这里的索引需要仔细推敲"""
                tmp_quotient.append((b - a) / (x_list[j + i] - x_list[j]))
            diff_quotient.append(tmp_quotient)
            """我们只需要这一阶插商的第一个数值 所以将他放入参数列表中"""
            parameters.append(tmp_quotient[0])
```

3.2.2 拉格朗日插值法

该方法的实现主要利用了 numpy 库的 broadcast 广播运算机制，利用 concatenate 函数方便地切片组合，其核心代码如下：

```
            upper = mul(x_target - np.concatenate((x_list[:i], x_list[i + 1:])))
            downer = mul(x0 - np.concatenate((x_list[:i], x_list[i + 1:])))
            result += upper / downer * y0
```

3.2.3 三次样条插值

由于需要分段求解三次插值函数，并且每一段插值函数都存在许多的参数，实现过程较为复杂，具体处理步骤如下：

```
            def calcEquationPara()
                """输入x_list，返回方程的系数，
```

```
            返回形式是一个系数矩阵，按照降幂的方式排列
        """
    def solutionOfEquation()
        """输入x_list,y_list,返回每一段区间的三次函数的参数，降幂排列
            获得方程解，然后计算机求解一个n元高次方程，
            调用np.liag.solve(a, b)来解方程
        """
        """核心代码"""
        while i < size_of_interval:  # 设置方程的右值
            result[(i-1)*2] = y_list[i]
            result[(i-1)*2+1] = y_list[i]
            i += 1
        result[(size_of_interval-1)*2] = y_list[0]
        result[(size_of_interval-1)*2+1] = y_list[-1]
        a = np.array(calcEquationPara(x_list))  # 获取方程左侧的系数矩阵
        b = np.array(result)
        return np.linalg.solve(a, b)  # 解方程
    def calculate()
        """输入parameters和x值，返回插值计算的结果"""
```

3.3 绘图实现与集成

3.3.1 matplotlib 绘图

通过 matplotlib（Python 语言优秀可视化第三方库）来对原始数据进行离散展示，特别标出目标点，并且绘制出一条完整的插值曲线

```
        """核心代码"""
        x = np.array(x, dtype=float)
        x = np.append(x, target_x)
        # 找到最大最小值求取x_range
        min_x = x[np.argmin(x)]
        max_x = x[np.argmax(x)]
        x = np.delete(x, len(x)-1)
        y = np.array(y, dtype=float)
        target_x = float(target_x)
        result = float(result)
```

```
                plt.figure()
                # 原始数据点
                plt.scatter(x, y, c='r', marker='o', s=20, label='Original')
                # 目标点
                plt.scatter(target_x, result,c='b', marker='x',
                    s=30, label='Target')
                x_line = np.arange(min_x, max_x, (max_x - min_x) / 1000)
                y_line = np.zeros(len(x_line))
                # 函数拟合
                plt.plot(x_line, y_line, c='g', label='Line')
```

3.3.2 集成到软件

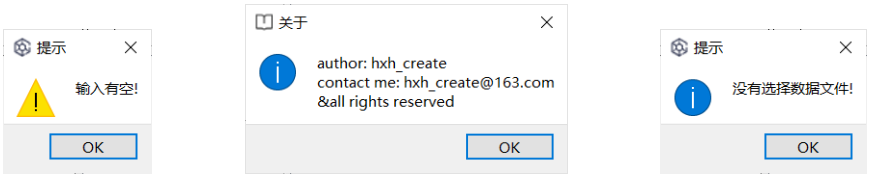
通过 PyQT5 的内置 graphicsview 模块来将已经生成好的 matplotlib 的.png 文件格式显示到软件界面中，并且自动适应大小，软件体验更加完整。

```
                # 获取文件路径
                last_path = os.path.join(os.getcwd(), "tmpresults/" +
                    str(self.cal_times) + ".png")
                img = QPixmap()
                img.load(last_path)
                # 设置显示区域
                self.graphicsView.scene = QGraphicsScene()
                item = QGraphicsPixmapItem(img)
                # 添加显示元素，这里就是图片
                self.graphicsView.scene.addItem(item)
                self.graphicsView.setScene(self.graphicsView.scene)
                #图片自适应view栏的大小
                self.graphicsView.fitInView(QGraphicsPixmapItem(img))
```

4 项目特点

优秀的用户体验 软件具有良好的用户反馈，包括成功提示，错误提示，未选择提示，输入检验等效果如下图所示：

支持帮助文档 软件提供内嵌式的帮助说明文档可以更加有效地帮助用户使用该工具，了解该工具的实现原理等



(a) 输入反馈　　(b) 关于作者　　(c) 文件反馈

图 6: 用户体验

资源占用　所有产生的数据文件，图片文件都将存储在 tmpresults 文件夹里面，当点击软件关闭的同时，会清楚该文件夹的所有内容，减少资源的占用，提升该工具的运行程序，避免了大量数据的堆积。

计算的鲁棒性　本工具在做插值计算的时候特别针对程序进行了鲁棒性的优化。

- 第一点是用户的 xlist 和 ylist 的输入不需要按照一定的顺序，程序会自动进行一个排序之后再进行插值处理。
- 是用户的 target 目标值并不需要处于 xlist 或者 ylist 的范围，仍然能够完成运算。因为我们插值实现的过程中是采用了原始的插商表或者插值公式进行计算的。（三次样条插值由于其实现的局限性，目标值必须出现在范围内，否则无法进行计算）
- 第三点在于数据支持多种类型。整数支持达到 32 位，小数的支持达到小数点后 17 位，几乎覆盖了日常的所有计算需求。

5　体会与总结

1. 学习了 Python 的 GUI 编程的基本知识，对软件开发流程有了基本的熟悉与了解。
2. 通过大量资料的查询也对插值法的各大语言和库的优秀实现方法做了一个了解。
3. 从理论层面的数值分析课程中跳出来，结合编程知识对已经掌握的知识进行了编程的实现，对于插值法有了更为深入的理解。

6　附录

6.1　软件程序源码及可执行文件

点击跳转 gitee
点击跳转 github

6.2　测试用例

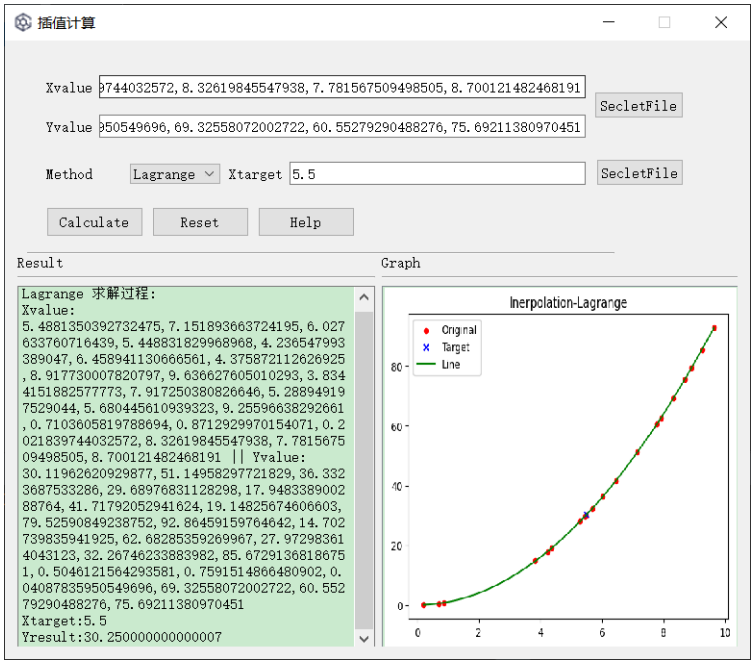

插值计算
Xvalue
9744032572, 8.32619845547938, 7.781567509498505, 8.700121482468191
SecletFile
Yvalue
950549696, 69.32558072002722, 60.55279290488276, 75.69211380970451
Method
Lagrange
Xtarget
5.5
SecletFile
Calculate
Reset
Help
Result
Graph
Lagrange 求解过程:
Xvalue:
5.4881350392732475, 7.151893663724195, 6.027633760716439, 5.448831829968968, 4.236547993389047, 6.458941130666561, 4.375872112626925, 8.917730007820797, 9.636627605010293, 3.8344151882577773, 7.917250380826646, 5.288949197529044, 5.680445610939323, 9.25596638292661, 0.7103605819788694, 0.8712929970154071, 0.20218397440325 72, 8.32619845547938, 7.781567509498505, 8.700121482468191 || Yvalue:
30.11962620929877, 51.14958297721829, 36.3323687533286, 29.68976831128298, 17.948338900288764, 41.71792052941624, 19.14825674606603, 79.52590849238752, 92.86459159764642, 14.702739835941925, 62.68285359269967, 27.972983614043123, 32.26746233883982, 85.67291368186751, 0.5046121564293581, 0.7591514866480902, 0.040878359505496 96, 69.32558072002722, 60.55279290488276, 75.69211380970451
Xtarget:5.5
Yresult:30.250000000000007
Inerpolation-Lagrange
Original
Target
Line

图 7: x*x

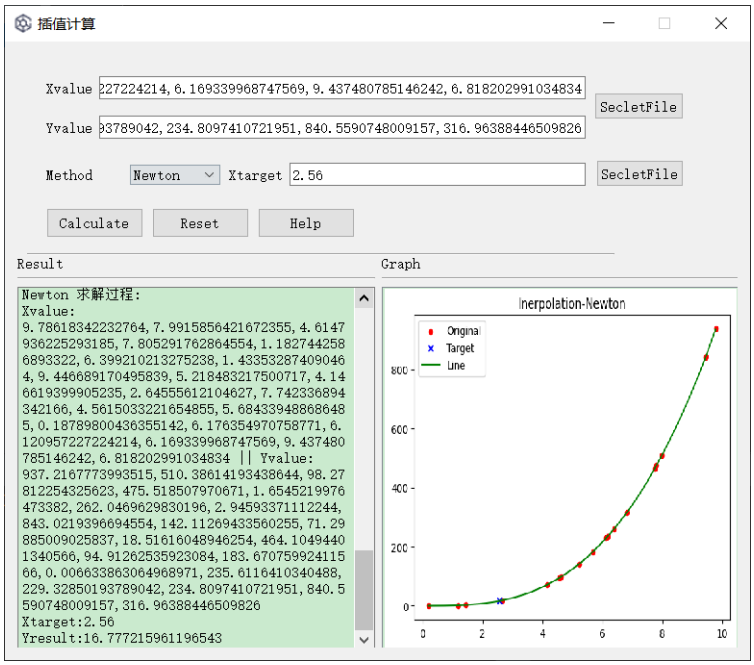

插值计算
Xvalue
227224214, 6.169339968747569, 9.437480785146242, 6.818202991034834
SecletFile
Yvalue
93789042, 234.8097410721951, 840.5590748009157, 316.96388446509826
Method
Newton
Xtarget
2.56
SecletFile
Calculate
Reset
Help
Result
Graph
Newton 求解过程:
Xvalue:
9.78618342232764, 7.9915856421672355, 4.6147936225293185, 7.805291762864554, 1.1827442586893322, 6.399210213275238, 1.43353287409046 4, 9.446689170495839, 5.218483217500717, 4.146619399905235, 2.64555612104627, 7.742336894342166, 4.5615033221654855, 5.68433948868648 5, 0.18789800436355142, 6.176354970758771, 6.120957227224214, 6.169339968747569, 9.437480 785146242, 6.818202991034834 || Yvalue:
937.2167773993515, 510.38614193438644, 98.27812254325623, 475.518507970671, 1.6545219976473382, 262.0469629830196, 2.94593371112244, 843.0219396694554, 142.11269433560255, 71.29885009025837, 18.51616048946254, 464.10494401340566, 94.91262535923084, 183.670759924115 66, 0.006633863064968971, 235.6116410340488, 229.32850193789042, 234.8097410721951, 840.5590748009157, 316.96388446509826
Xtarget:2.56
Yresult:16.777215961196543
Inerpolation-Newton
Original
Target
Line

图 8: x*x*x

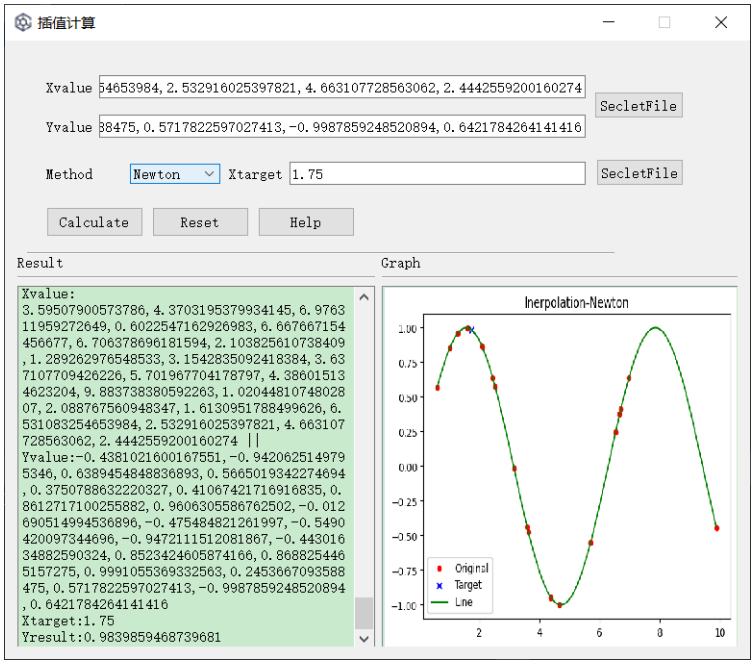


图 9: sinx

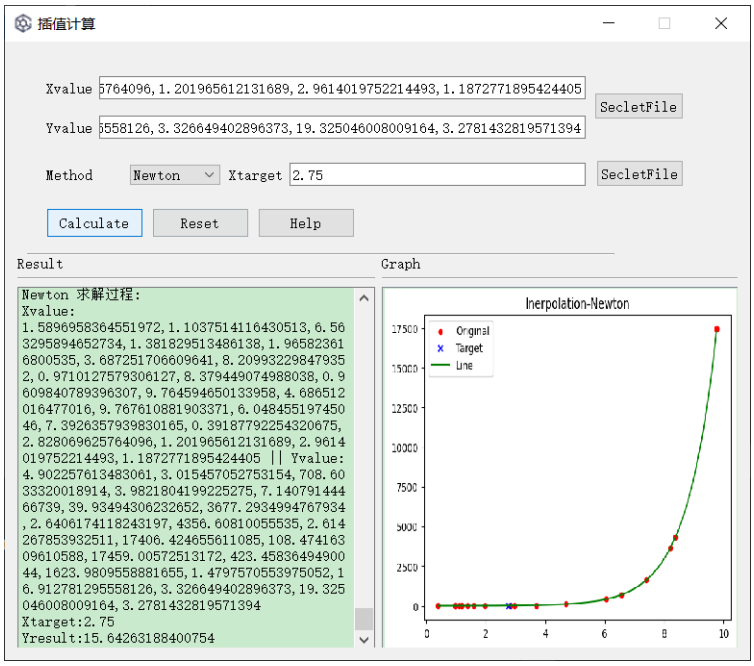


图 10: ex

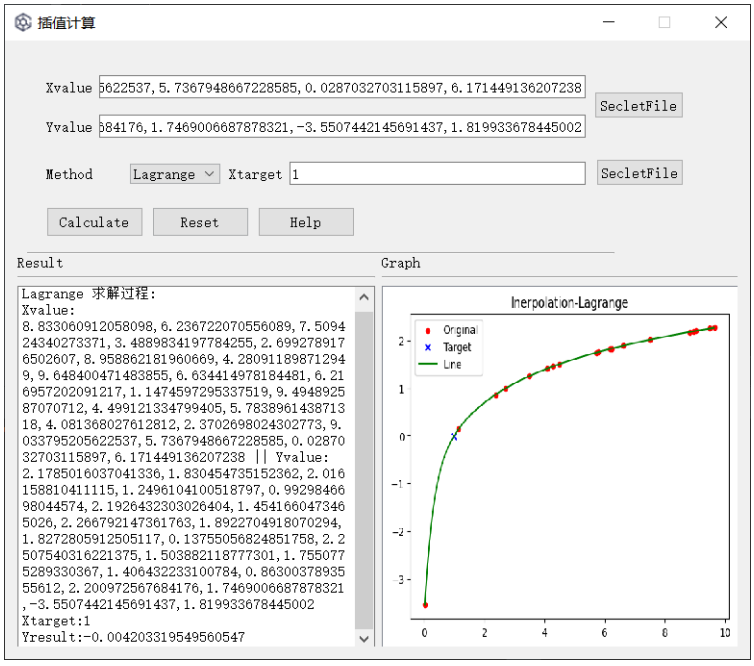

插值计算
Xvalue 5622537, 5.7367948667228585, 0.0287032703115897, 6.171449136207238
SecletFile
Yvalue 584176, 1.7469006687878321, -3.5507442145691437, 1.819933678445002
Method
Lagrange
Xtarget 1
SecletFile
Calculate
Reset
Help
Result
Graph
Lagrange 求解过程:
Xvalue:
8.833060912058098, 6.236722070556089, 7.5094
2434027337 1, 3.4889834197784255, 2.699278917
6502607, 8.958862181960669, 4.28091189871294
9, 9.648400471483855, 6.634414978184481, 6.21
6957202091217, 1.1474597295337519, 9.4948925
87070712, 4.499121334799405, 5.7838961438713
18, 4.081368027612812, 2.3702698024302773, 9.
0337952056225 37, 5.7367948667228585, 0.02870
32703115897, 6.171449136207238 || Yvalue:
2.1785016037041336, 1.830454735152362, 2.016
158810411115, 1.2496104100518797, 0.99298466
98044574, 2.1926432303026404, 1.454166047346
5026, 2.26679214736 1763, 1.8922704918070294,
1.8272805912505117, 0.13755056824851758, 2.2
507540316221375, 1.503882118777301, 1.755077
5289330367, 1.406432233100784, 0.86300378935
55612, 2.200972567684176, 1.7469006687878321
, -3.5507442145691437, 1.819933678445002
Xtarget:1
Yresult:-0.004203319549560547
Interpolation-Lagrange
Orginal
Target
Line

图 11: logx