实验一：插值

胡子畅　23336091

1. 已知函数在下列各点的值为

x_i	0.2	0.4	0.6	0.8	1.0
$f(x_i)$	0.98	0.92	0.81	0.64	0.38

试用 4 次牛顿插值多项式 $P_4(x)$ 及三次样条函数 $S(x)$（自然边界条件）对数据进行插值. 用图给出$\{(x_i, y_i), x_i=0.2+0.08i, i=0,1,11,10\}$, $P_4(x)$ 及$S(x)$.

一、牛顿插值

1. 差商计算

差商是牛顿插值的核心计算单元，采用递归定义：

零阶差商：$f[x_i] = y_i$

一阶差商：$f[x_i, x_j] = \dfrac{f[x_j] - f[x_i]}{x_j - x_i}$

k阶差商：$f[x_0, \ldots, x_k] = \dfrac{f[x_1, \ldots, x_k] - f[x_0, \ldots, x_{k-1}]}{x_k - x_0}$

2. 插值多项式构造

牛顿插值多项式形式：

$$P_n(x) = f[x_0] + \sum_{k=1}^{n} f[x_0, \ldots, x_k] \cdot \prod_{j=0}^{k-1} (x - x_j)$$

展开形式示例（3个节点）：

$$\begin{aligned}P_2(x) = f[x_0] \\ &+ f[x_0, x_1](x - x_0) \\ &+ f[x_0, x_1, x_2](x - x_0)(x - x_1) \\ &+ \cdots \\ &+ f[x_0, \ldots, x_n]\prod_{j=0}^{n-1}(x - x_j)\end{aligned}$$

3. 差商表生成

差商表的构造过程如下（以4个节点为例）：

x_0	$f[x_0]$	$f[x_0, x_1]$	$f[x_0, x_1, x_2]$	$f[x_0, x_1, x_2, x_3]$
x_1	$f[x_1]$	$f[x_1, x_2]$	$f[x_1, x_2, x_3]$	
x_2	$f[x_2]$	$f[x_2, x_3]$		
x_3	$f[x_3]$			

4. 计算案例

给定节点 (1,1), (2,4), (3,9):

$$
\begin{aligned}
f[x_0] &= 1 \\
f[x_1] &= 4 \\
f[x_2] &= 9 \\
f[x_0, x_1] &= \frac{4-1}{2-1} = 3 \\
f[x_1, x_2] &= \frac{9-4}{3-2} = 5 \\
f[x_0, x_1, x_2] &= \frac{5-3}{3-1} = 1 \\
P_2(x) &= 1 + 3(x-1) + 1(x-1)(x-2) \\
&= x^2
\end{aligned}
$$

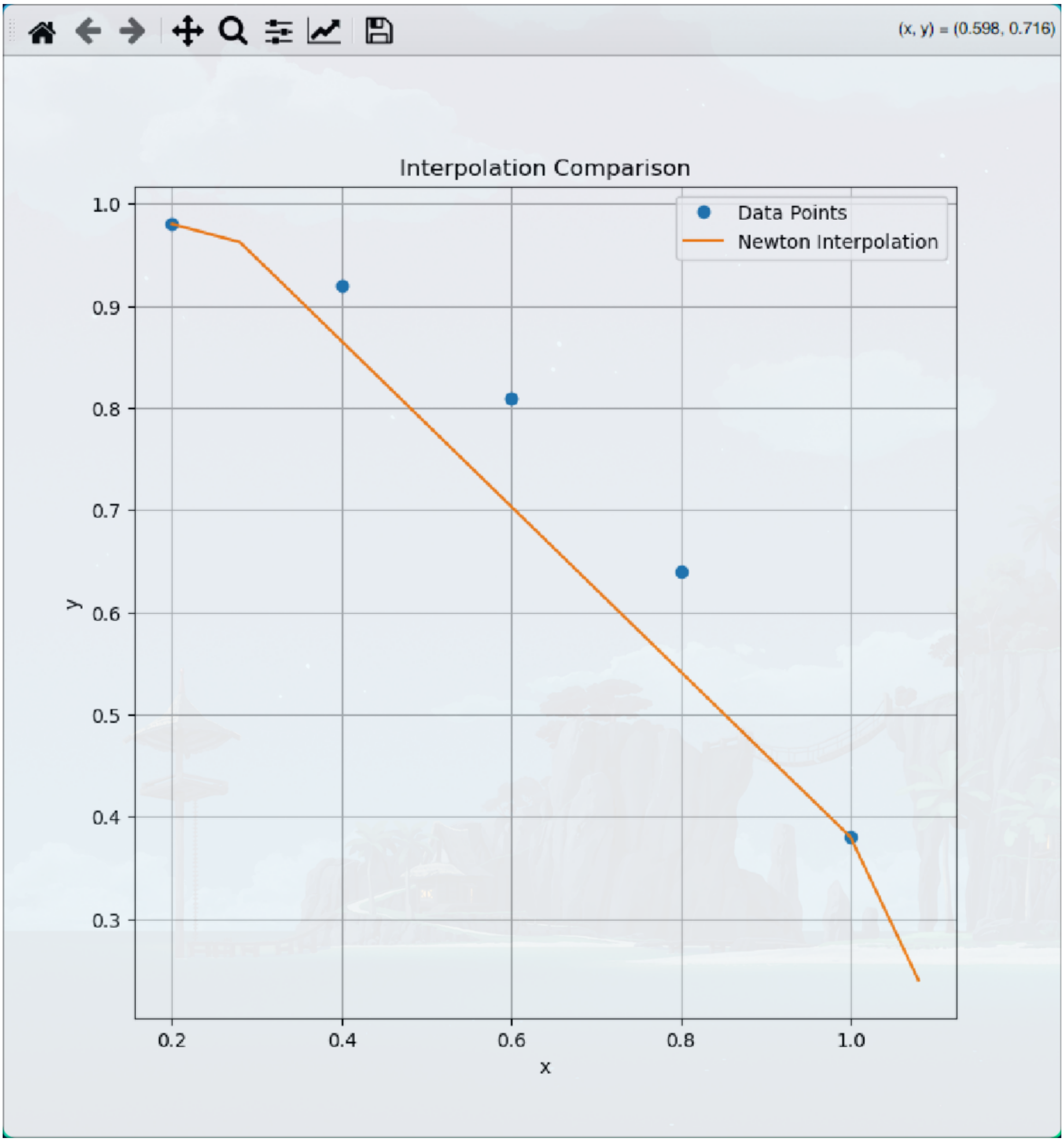
5. 编程实现

```
def newton_interpolation(x, y, xi):
    n = len(y)
    divided_diff = y.copy() # 初始化差商表
    # 枚举阶数
    for j in range(1, n):
        # 计算每阶差商时原地更新（从后往前，防止被覆盖）
        for i in range(n - 1, j - 1, -1):
            divided_diff[i] = (divided_diff[i]
- divided_diff[i - 1]) / (x[i] - x[i - j])

    m = len(xi)
    yi = [0.0] * m
    for k in range(m):
        for i in range(n):
            term = divided_diff[i]
            for j in range(i):
                term *= (xi[k] - x[j])
```

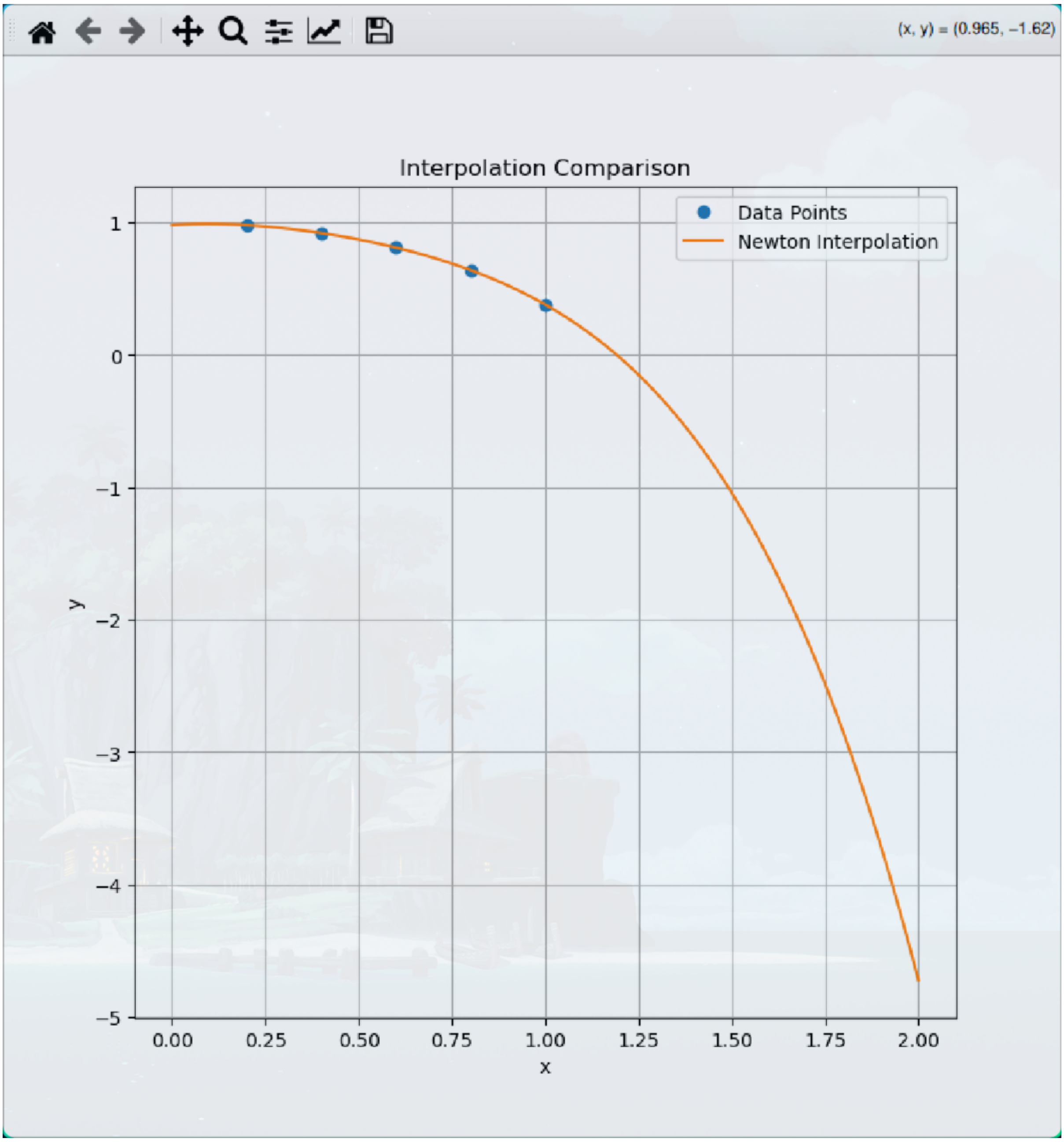
```
            yi[k] += term

    return yi
```

6. 图像生成

根据题意，插值点为4个时：



插值点为100个时：



二、三次样条插值

1. 基本定义

三次样条插值构造分段三次多项式 $S(x)$，满足：

1. 插值条件：$S(x_i) = y_i$
2. 连续性条件：$\begin{cases} S_i(x_{i+1}) = S_{i+1}(x_{i+1}) \\ S_i'(x_{i+1}) = S_{i+1}'(x_{i+1}) \\ S_i''(x_{i+1}) = S_{i+1}''(x_{i+1}) \end{cases}$

2. 分段多项式形式

每个区间 $[x_i, x_{i+1}]$ 上的表达式：

$$S_i(x) = a_i + b_i(x - x_i) + c_i(x - x_i)^2 + d_i(x - x_i)^3$$

3. 系数计算步骤

(1) 计算步长

$$h_i = x_{i+1} - x_i \quad (i = 0, 1, \dots, n-1)$$

(2) 构造三对角方程组

$$\begin{bmatrix} 2 & \lambda_1 & & \\ \mu_2 & 2 & \lambda_2 & \\ & \ddots & \ddots & \ddots \\ & & \mu_{n-1} & 2 \end{bmatrix} \begin{bmatrix} M_1 \\ M_2 \\ \vdots \\ M_{n-1} \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_{n-1} \end{bmatrix}$$

其中：

$$\begin{aligned} \mu_i &= \frac{h_{i-1}}{h_{i-1} + h_i} \\ \lambda_i &= \frac{h_i}{h_{i-1} + h_i} \\ d_i &= 6 \cdot \frac{f[x_{i+1}, x_i] - f[x_i, x_{i-1}]}{h_{i-1} + h_i} \end{aligned}$$

(3) 边界条件处理

- **自然样条：**

$$M_0 = M_n = 0$$

- **固定边界：**

$$S'(x_0) = f'(x_0),\ S'(x_n) = f'(x_n)$$

(4) 求解系数

$$\begin{aligned}
a_i &= y_i \\
b_i &= \frac{y_{i+1} - y_i}{h_i} - \frac{h_i}{6}(M_{i+1} + 2M_i) \\
c_i &= \frac{M_i}{2} \\
d_i &= \frac{M_{i+1} - M_i}{6h_i}
\end{aligned}$$

4. 计算案例

给定节点 $(0,1), (1,2), (2,1), (3,0)$：

(1) 计算步长

$$h = [1, 1, 1]$$

(2) 构造方程组（自然边界）

$$\begin{bmatrix} 4 & 1 & 0 \\ 1 & 4 & 1 \\ 0 & 1 & 4 \end{bmatrix} \begin{bmatrix} M_1 \\ M_2 \\ M_3 \end{bmatrix} = \begin{bmatrix} -6 \\ -12 \\ -6 \end{bmatrix}$$

(3) 求解得到

$$[M_1, M_2, M_3] = [-1.2857, -2.5714, -1.2857]$$

(4) 第一个区间系数

$$\begin{aligned} a_0 &= 1 \\ b_0 &= 1.1429 \\ c_0 &= 0 \\ d_0 &= -0.2143 \end{aligned}$$

5. 编程实现

原始版本：

```
def calc(x, y, h, M, i, xx):
    a = y[i]
    b = (y[i + 1] - y[i]) / h[i] - h[i] * (M[i + 1] + 2 * M[i]) / 6
    c = M[i] / 2
    d = (M[i + 1] - M[i]) / (6 * h[i])
    dx = xx - x[i]
    return a + b * dx + c * dx**2 + d * dx**3

def spline_interpolation(x, y, xi):
    n = len(x)
    # 区间长度
    h = np.diff(x)
    # 初始化矩阵
    A = np.zeros((n, n))
    b = np.zeros(n)
    # 填充矩阵
```

```
    for i in range(1, n - 1):
        A[i, i - 1] = h[i - 1]
        A[i, i] = 2 * (h[i - 1] + h[i])
        A[i, i + 1] = h[i]
        b[i] = 6 * ((y[i + 1] - y[i]) / h[i] -
(y[i] - y[i - 1]) / h[i - 1])
    # 自然边界
    A[0, 0] = 2
    A[-1, -1] = 2
    # 求解线性方程组
    M = np.linalg.solve(A, b)

    m = len(xi)
    yi = [0.0] * m
    for k in range(m):
        for i in range(n - 1):
            if x[i] <= xi[k] <= x[i + 1]:
                yi[k] = calc(x, y, h, M, i,
xi[k])
                break

    return yi
```

此时编程实现的三次样条插值可以满足题意，根据其插值点画图，但在我后续测试时发现，其不能正确处理在区间之外的点，使得在边界外的图像呈现断崖。后续改进版本，即可正确处理异常插值点，使得插值图像光滑。

改进版本：

```
def calc(x, y, h, M, i, xx):
```

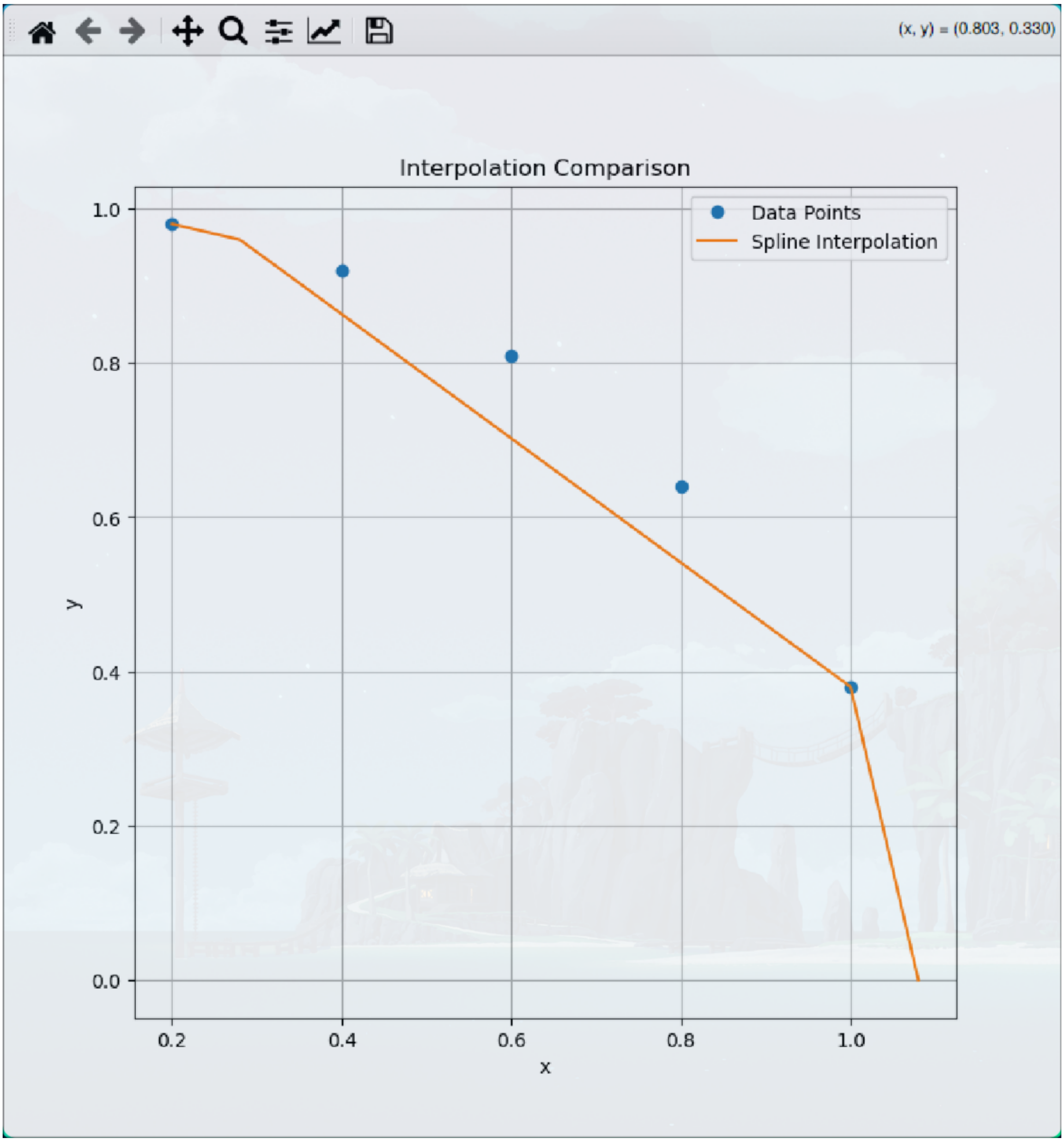
```
    a = y[i]
    b = (y[i + 1] - y[i]) / h[i] - h[i] * (M[i
+ 1] + 2 * M[i]) / 6
    c = M[i] / 2
    d = (M[i + 1] - M[i]) / (6 * h[i])
    dx = xx - x[i]
    return a + b * dx + c * dx**2 + d * dx**3

def spline_interpolation(x, y, xi):
    n = len(x)
    # 区间长度
    h = np.diff(x)
    # 初始化矩阵
    A = np.zeros((n, n))
    b = np.zeros(n)
    # 填充矩阵
    for i in range(1, n - 1):
        A[i, i - 1] = h[i - 1]
        A[i, i] = 2 * (h[i - 1] + h[i])
        A[i, i + 1] = h[i]
        b[i] = 6 * ((y[i + 1] - y[i]) / h[i] -
(y[i] - y[i - 1]) / h[i - 1])
    # 自然边界
    A[0, 0] = 2
    A[-1, -1] = 2
    # 求解线性方程组
    M = np.linalg.solve(A, b)

    yi = []
    # 计算插值点区间的坐标i
    indices = np.searchsorted(x, xi) - 1
    # 将插值点限制在有效区间内，处理异常插值点
```

```python
    indices = np.clip(indices, 0, n-2)
    for k, i in enumerate(indices):
        yi.append(calc(x, y, h, M, i, xi[k]))

    return yi
```

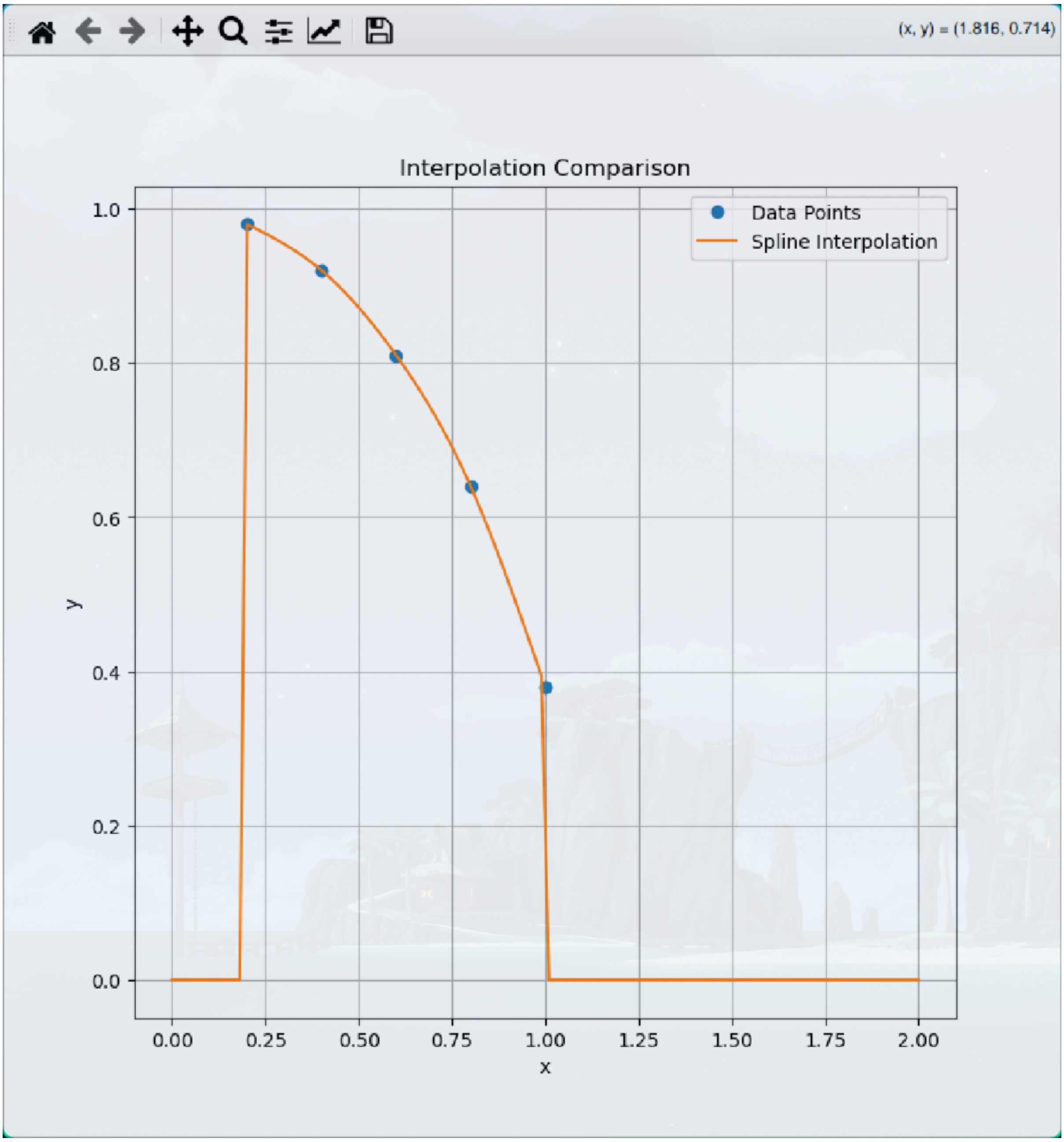
6. 图像生成

原始版本：

根据题意，插值点为4个时：

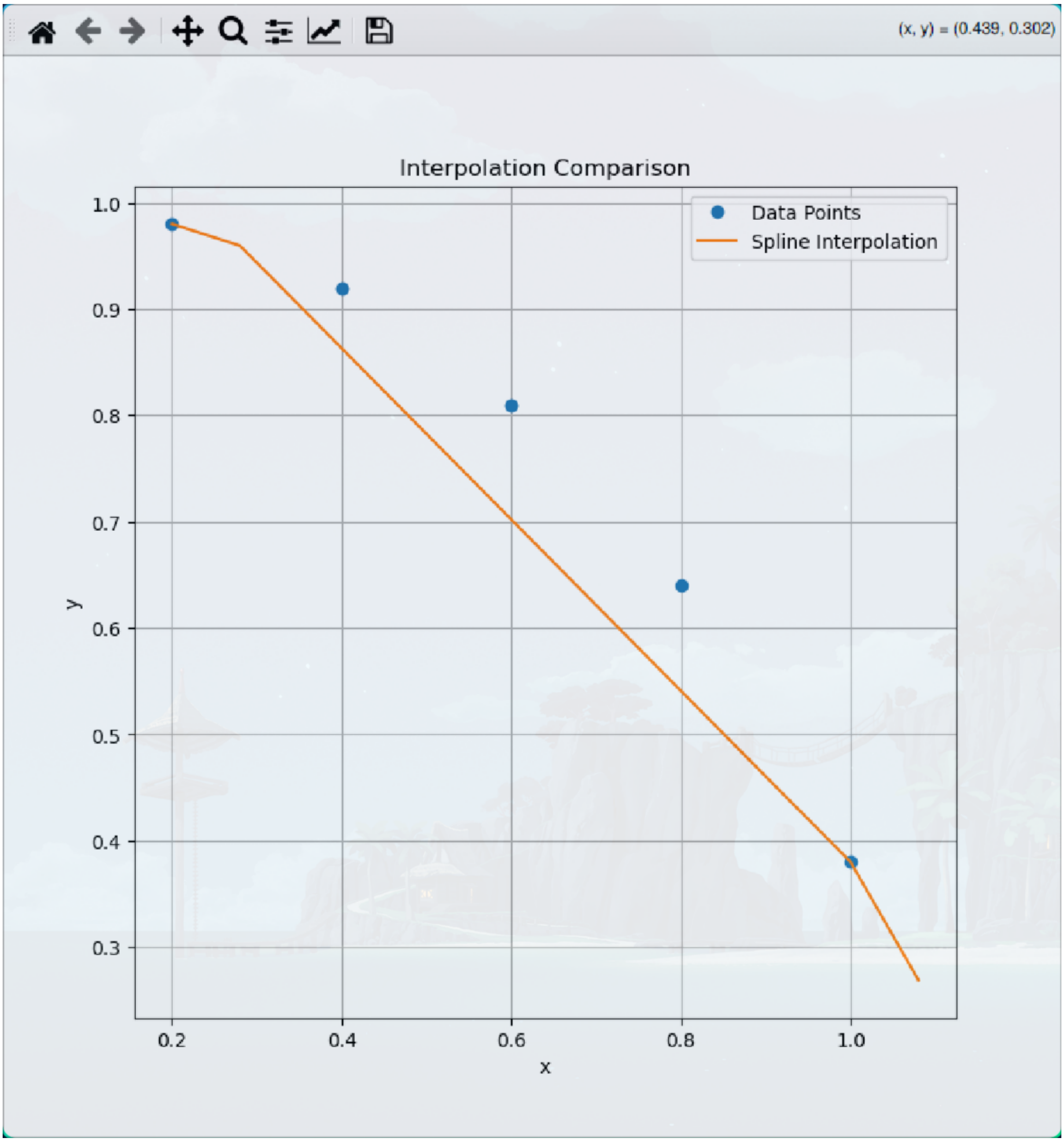


插值点为100个时：



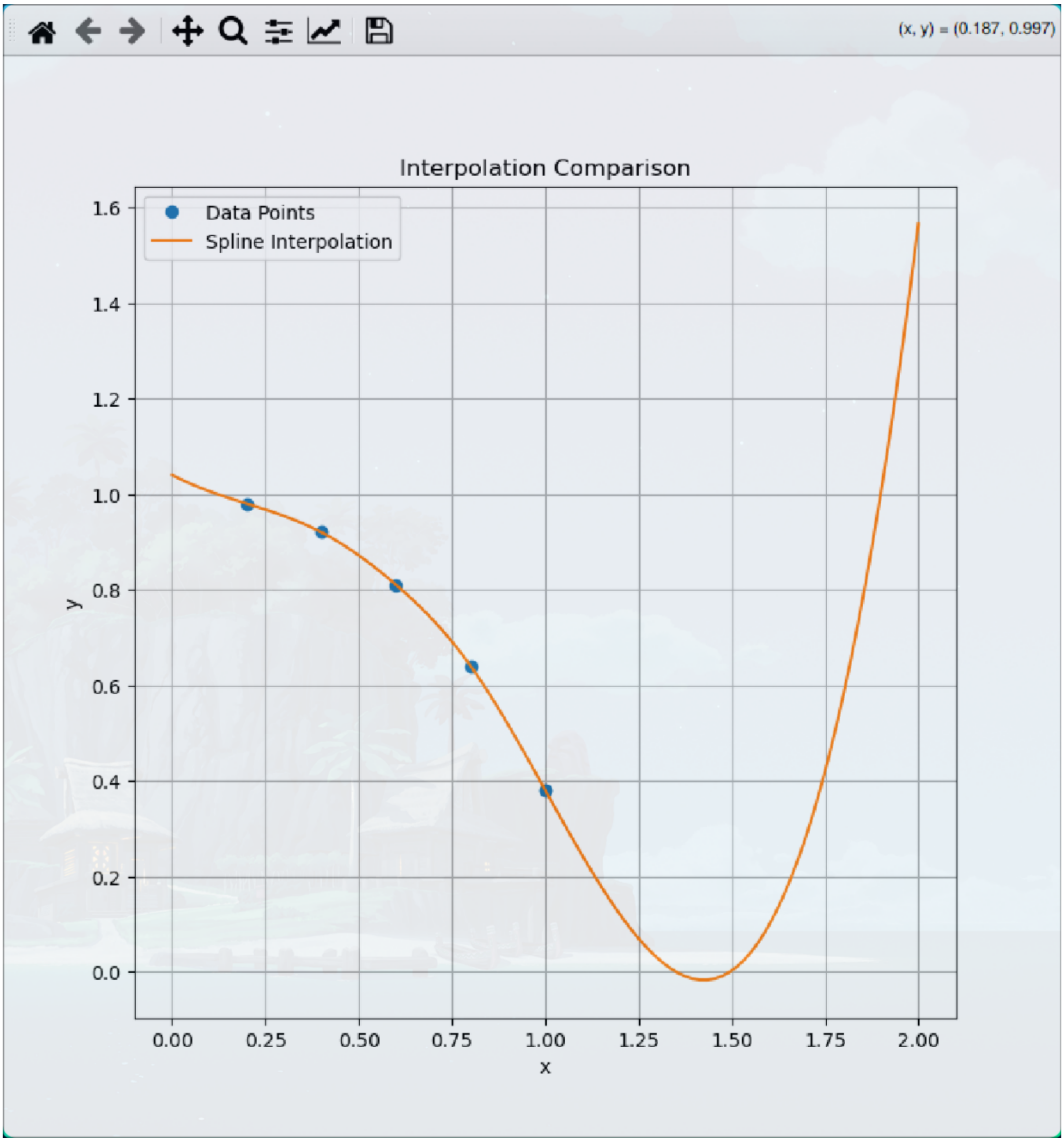
改进版本：

根据题意，插值点为4个时：



(x, y) = (0.439, 0.302)
Interpolation Comparison
Data Points
Spline Interpolation
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
y
0.2
0.4
0.6
0.8
1.0
x

插值点为100个时：



三、牛顿插值与三次样条插值对比分析

1. 数学特性对比

特性	牛顿插值	三次样条插值
插值形式	全局单多项式	分段三次多项式
光滑性	C^∞ 连续	C^2 连续（二阶导数连续）
局部性	修改一个节点影响全局	只影响相邻区间
边界条件	无需特殊处理	需指定自然/固定边界条件
龙格现象	高阶时可能出现	完全避免

2. 计算复杂度对比

$$
\text{牛顿插值：}\begin{cases} \text{差商表构造} \quad \mathcal{O}(n^2) \\ \text{单点求值} \quad \mathcal{O}(n) \end{cases}
$$

$$
\text{三次样条：}\begin{cases} \text{矩阵构造} \quad \mathcal{O}(n) \\ \text{方程组求解} \quad \mathcal{O}(n)\ (\text{追赶法}) \\ \text{单点求值} \quad \mathcal{O}(1)\ (\text{区间定位后}) \end{cases}
$$

3. 数值稳定性对比

场景	牛顿插值	三次样条插值
高次插值	容易数值不稳定	稳定性良好
非均匀节点分布	差商计算可能溢出	分段处理更稳健
外推预测	误差急剧增大	边界外延相对平滑

4. 实际应用对比

(1) 牛顿插值适用场景

- 节点数量较少（通常 < 10）
- 需要快速实现简单插值

(2) 三次样条适用场景

- 节点数量较多（≥10）
- 要求曲线光滑的工程应用

5. 典型误差对比

测试函数 $f(x) = \sin(x)$ 在 $[0, \pi]$ 的插值误差：

节点数	牛顿最大误差	样条最大误差
5	0.08	0.05
10	0.25	0.003
20	1.47（发散）	0.0001

6. 选择建议

1. **优先使用三次样条**当：
    - 需要保证曲线光滑性
    - 节点间距不均匀
    - 数据量较大
2. **考虑牛顿插值**当：
    - 实现简单性优先于精度

    - 节点分布均匀且数量少
    - 不需要外推预测

> 注：本实验中的改进版样条代码已解决边界外插问题，推荐在实际应用中采用