数值分析第 3 次上机作业

学号：221840189，姓名：王晨光

§ 1 问题

考虑函数

$$R(x) = \frac{1}{1+x^2},$$

利用下列条件做插值逼近，并与 $R(x)$ 的图像比较.

§ 2 问题一

2.1 问题

用节点

$$x_i = 5cos(\frac{2i+1}{42}\pi), i = 0, 1, ..., 20.$$

画出 20 次 Lagrange 插值多项式的图像;

2.2 算法思路

利用 Lagrange 插值公式:

$$p_n(x) = \sum_{i=0}^{n} f(x_i) l_i(x)$$

其中

$$l_i(x) = \prod_{\substack{j=0, \\ j\neq i}}^{n} \frac{x - x_j}{x_i - x_j}, \quad i = 0, 1, \cdots, n$$

这是 Lagrange 基本多项式.

由此在编程时只需记录下插值基点以及对应的函数值，就可以通过 Python 的 Scipy 库中的 `interpolate.lagrange` 函数构造出 Lagrange 插值多项式.

Algorithm 1 Lagrange 插值多项式

for i from 0 to 20 **do**
    $x_i \leftarrow 5cos(\frac{2i+1}{42}\pi)$
    $y_i \leftarrow \frac{1}{1+x_i^2}$
end for
$x = -5$
for i from 1 to 2000 **do**
    for j from 0 to 20 **do**
        for k from 0 to 20 **do**
            if $j \neq k$ **then**
                $T(j,i) \leftarrow \frac{x-x_k}{x_j-x_k}T(j,i)$
            end if
        end for
    end for
    $x \leftarrow x + 0.005$
end for
$x = -5$
for i from 1 to 2000 **do**
    for j from 0 to 20 **do**
        $f(x) \leftarrow y_j T(j,i) + f(x)$
    end for
    $x \leftarrow x + 0.005$
end for
return f

具体的代码由 Python 实现. 由于 Python 作图是通过大量的散点做出的，故在实际的作图中选取了横坐标间距为 0.005 个单位的点来作图 (共 2000 个)，这样在兼顾时间复杂度的同时，保证了图像具有一定的平滑度.

同样由于 Python 作图是通过大量的散点做出的，故我们不需要给出多项式的具体表达式，只需要计算 Lagrange 多项式在 2000 个作图点的取值即可 (后四个作图的思路同样如此，故再不赘述，并省略处理 2000 个采样点的伪代码过程).

2.3 结果分析

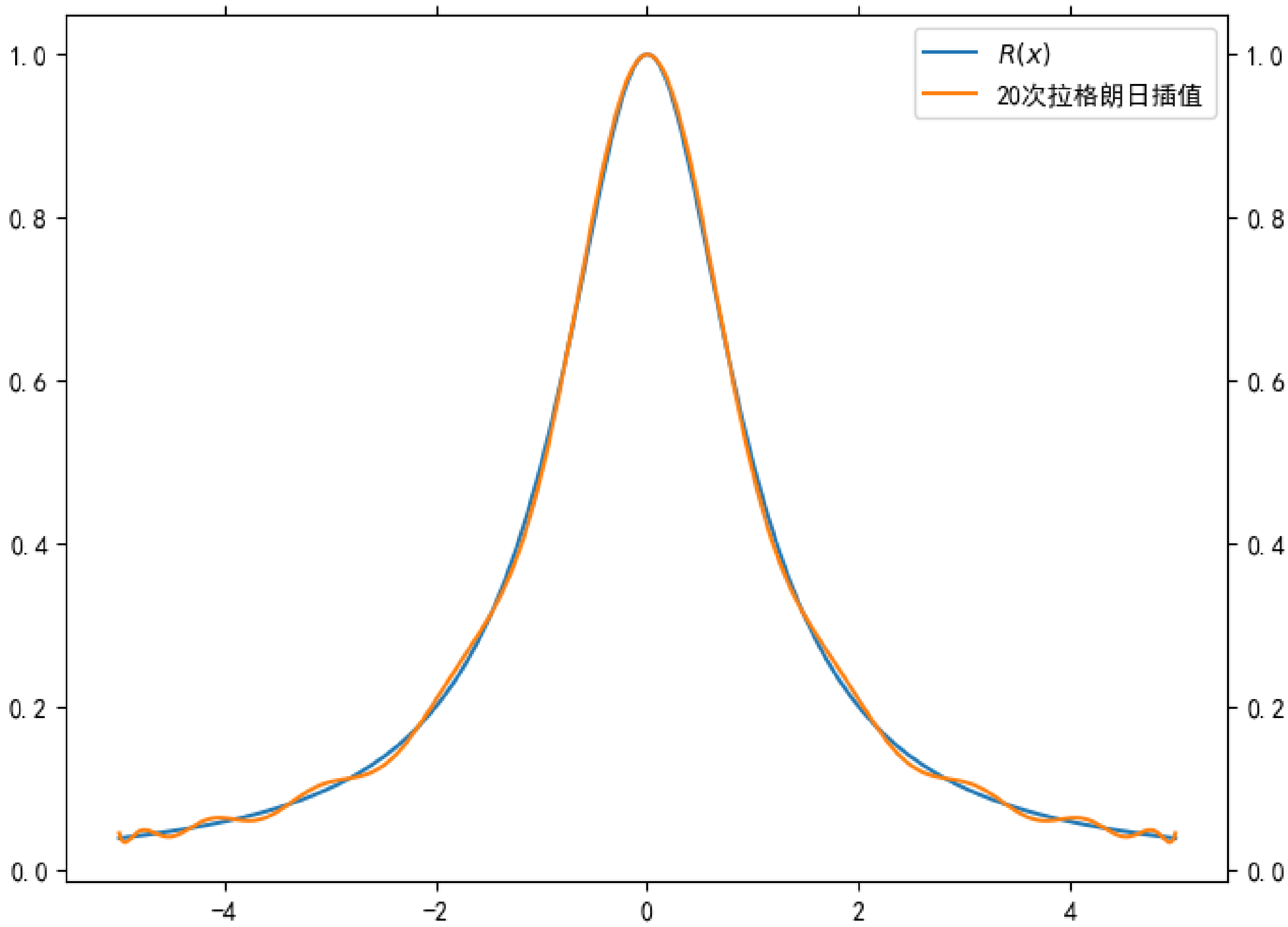

图 1: 问题一图像对比

可以看出，在图像中间部分 Lagrange 插值多项式逼近较好，而在靠近两端的部分出现了一些震荡. 值得一提的是，虽然使用插值多项式次数较高 (20 次)，但由于选择插值基点间距不是均匀的，所以并未出现课本所说的 Runge 现象.

§ 3 问题二

3.1 问题

用等距节点 $x_i = -5 + i, i = 0, 1, ..., 10$. 画出 10 次 Newton 插值多项式的图像;

3.2 算法思路

对于 Newton 插值多项式，首先需要构造出 n 阶均差表，即一个 $n \times n$ 的矩阵，其对角线上的元素即为所需的系数.

由于 Newton 插值多项式与 Lagrange 插值多项式的最终表达式相同，同样可以通过 Python 的 Scipy 库中的 `interpolate.lagrange` 函数构造出 Newton 插值多项式.

Algorithm 2 Newton 插值多项式

for i from 1 to 11 **do**
  $x_i \leftarrow -5 + i - 1$
  $Q_{i+1,1} \leftarrow \dfrac{1}{1+x_i^2}$
end for
for j from 2 to 11 **do**
  for i from j to 11 **do**
    if $j \neq i$ **then**
      $Q_{ij} \leftarrow \dfrac{Q_{i,j-1} - Q_{i-1,j-1}}{x_i - x_{i-j+1}}$
    end if
  end for
end for
$b_{11} \leftarrow Q_{11,11}$
for i from 11 to 2 **do**
  $b_{i-1} \leftarrow Q_{i-1,i-1} + b_i(x - x_{k-1})$
end for
return b_1

3.3 结果分析

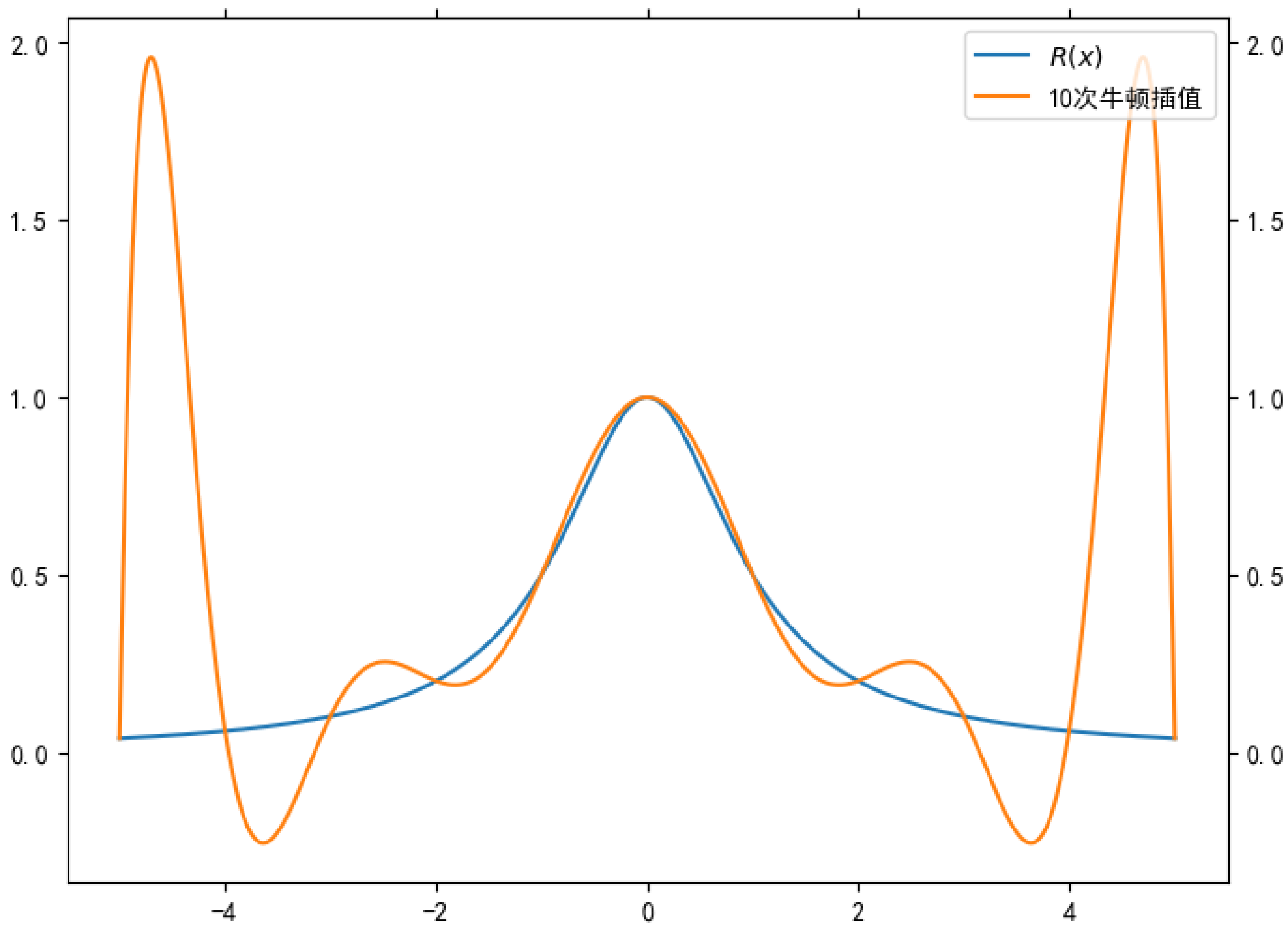

图 2: 问题二图像对比

可以看出，在中间部分 Newton 插值多项式的逼近效果较好，但在两端出现了很大的震荡，原因在于插值点的选取是等距的，而多项式的次数较高 (10 次)，所以会出现 Runge 现象，即在两端有较大的误差. 与图一 20 次 Lagrange 插值多项式 (不等距基点) 的图像形成了对比.

§4 问题三

4.1 问题

用等距节点 $x_i = -5 + i, i = 0, 1, ..., 10$. 画出分段线性插值函数的图像;

4.2 算法思路

对于分段线性插值，只需记录插值点的横坐标与纵坐标，再逐点用直线段连接即可.

可以通过 Python 的 Scipy 库中的 `interpolate.interp1d` 函数构造出分段线性插值多项式.

4.3 结果分析

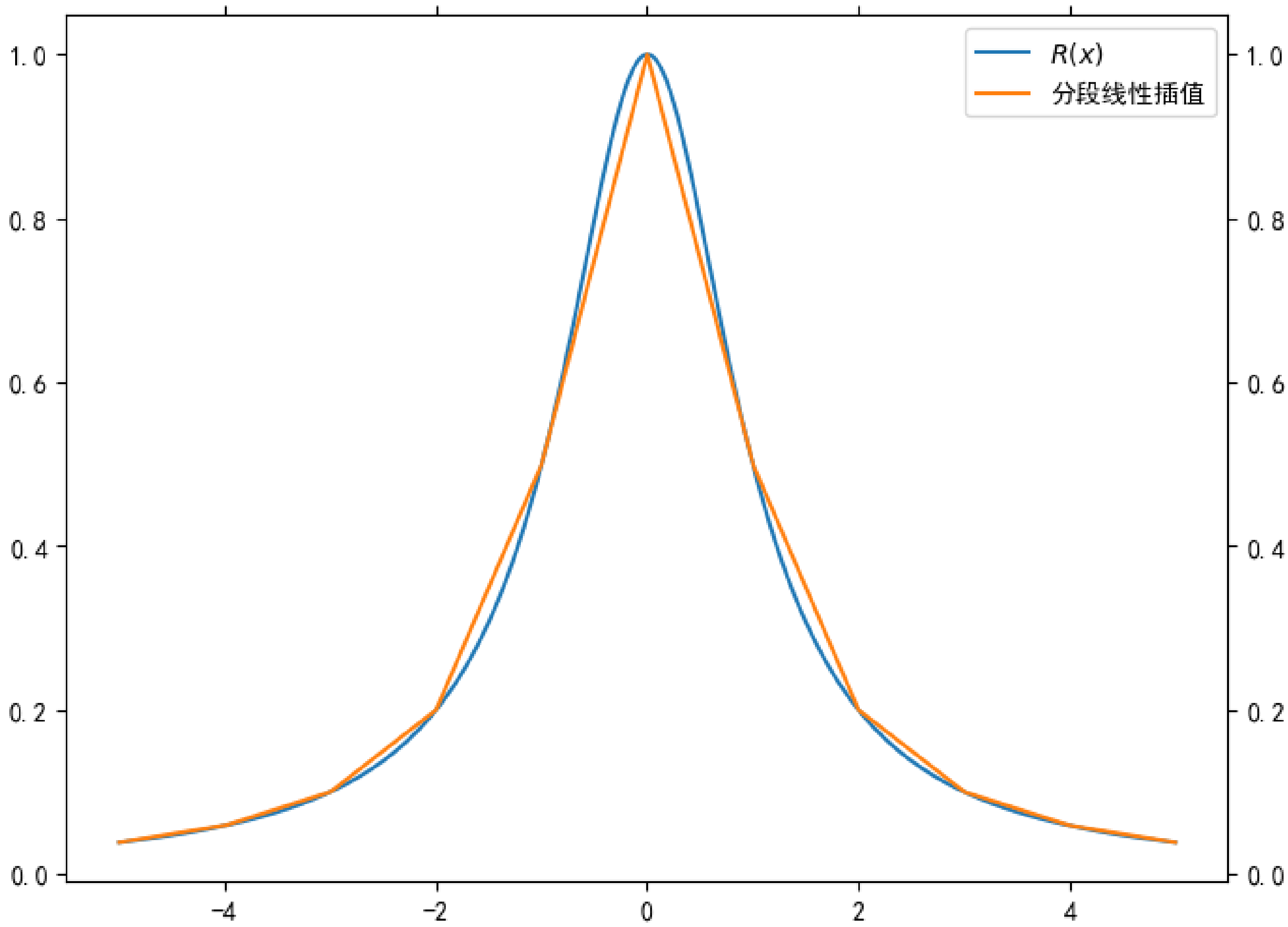

图 3: 问题三图像对比

可以看出，在原函数导数的绝对值较小的部分，线性插值的效果较好，而在原函数导数绝对值较大的部分，线性插值的误差较大.

§ 5 问题四

5.1 问题

用等距节点 $x_i = -5 + i, i = 0, 1, ..., 10$. 画出三次自然样条插值函数的图像;

5.2 算法思路

记 $m_i = S''(x_i)$, $h_i = x_{i+1} - x_i$，则每一个小区间上的函数 $S_i(x)$ 由以下公式给出：

$$S_i(x) = \frac{1}{h_i}[\frac{m_i}{6}(x_{i+1} - x)^3 + \frac{m_{i+1}}{6}(x - x_i)^3] + f(x_i) + f[x_i, x_{i+1}](x - x_i) - \frac{{h_i}^2}{6}[(m_{i+1} - m_i)\frac{x - x_i}{h_i} + m_i] \tag{1}$$

于是可先计算出二阶均差表，再计算 m_i，只需解下述线性方程组：

$$\begin{pmatrix} 2h_1 & h_1 & & & & \\ h_1 & 2(h_1+h_2) & h_2 & & & \\ & h_2 & 2(h_2+h_3) & h_3 & & \\ & & \ddots & \ddots & \ddots & \\ & & & h_{n-1} & 2(h_{n-1}+h_n) & h_n \\ & & & & h_n & 2h_n \end{pmatrix} \begin{pmatrix} m_1 \\ m_2 \\ \vdots \\ \vdots \\ m_n \\ m_{n+1} \end{pmatrix} = 6 \begin{pmatrix} d_1 \\ d_2 \\ \vdots \\ \vdots \\ d_n \\ d_{n+1} \end{pmatrix}$$

其中

$$\begin{aligned} d_1 &= f[x_1, x_2] \\ d_i &= f[x_i, x_{i+1}] - f[x_{i-1}, x_i], \quad i = 2, \cdots, n \\ d_{n+1} &= -f[x_n, x_{n+1}] \end{aligned}$$

(1) 式可改写为

$$S_i(x) = \sum_{k=1}^{4} A_{k,i}(x - x_i)^{k-1},\ x \in [x_i, x_{i+1}] \tag{2}$$

其中

$$\begin{aligned} A_{1,i} &= f(x_i), \\ A_{2,i} &= f[x_i, x_{i+1}] - \frac{h_i}{6}(m_{i+1} + 2m_2), \\ A_{3,i} &= \frac{m_i}{2}, \\ A_{4,i} &= \frac{m_{i+1} - m_i}{6h_i} \end{aligned}$$

已知 $f[x_i, x_{i+1}], m_i, d_i, h_i$，即可按照 (2) 式用三重循环计算出 $S_i(x)$ 在若干所需作图点的值.

可以通过 Python 的 Scipy 库中的 `interpolate.CubicSpline` 函数构造出自然三次样条插值多项式.

5.3 结果分析

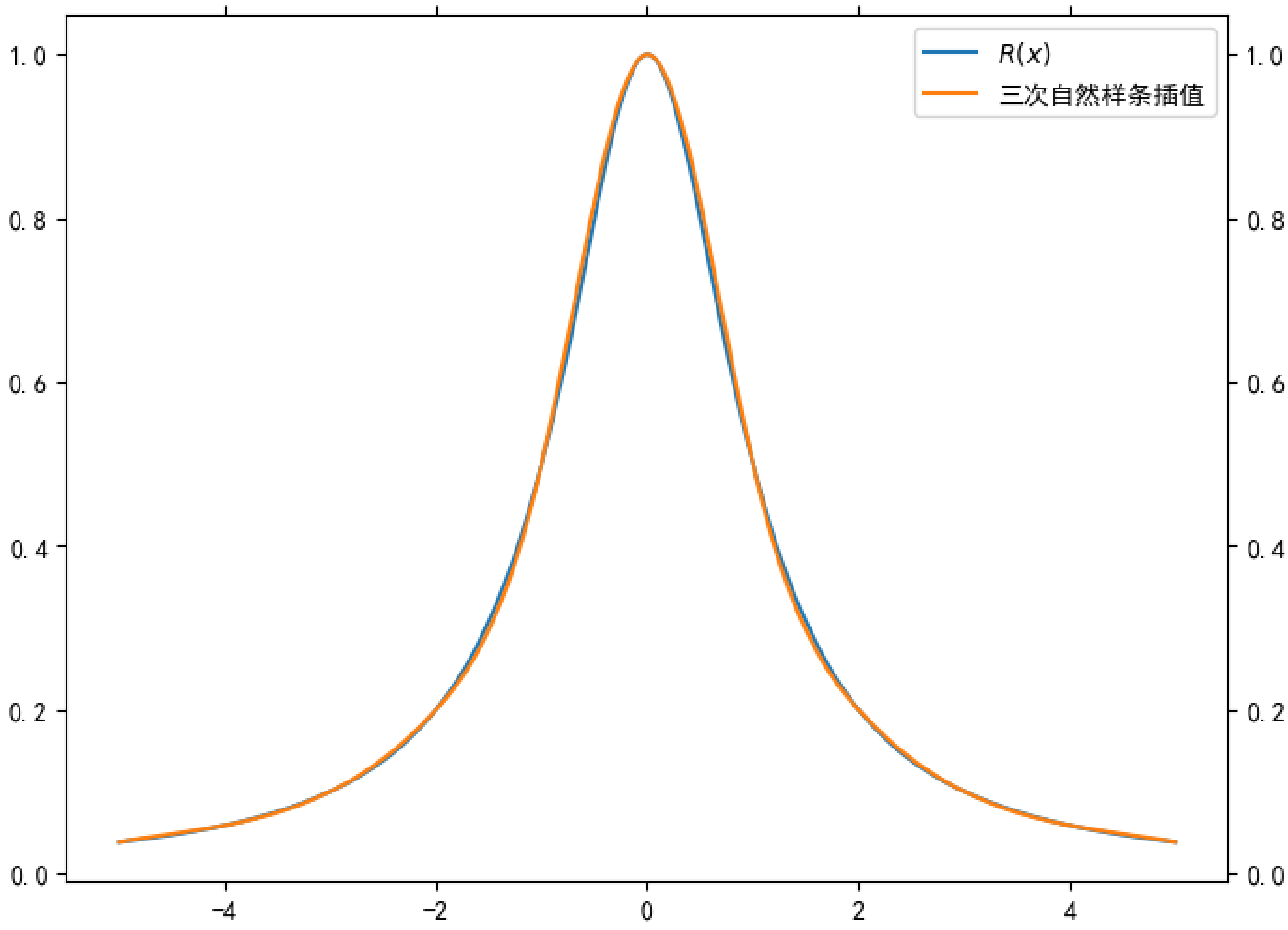

图 4: 问题四图像对比

可以看出，自然三次样条插值的逼近效果在整个区间上都很好.

§6 问题五

6.1 问题

用等距节点 $x_i = -5 + i, i = 0, 1, ..., 10$. 画出分段三次 Hemite 插值（相邻两点的函数值和一阶导数值）函数的图像.

6.2 算法思路

根据相邻两点的函数值和一阶导数值，可以在每个区间上得到三阶 Hermite 插值多项式 $H_3(x)$：

$$H_3(x) = \sum_{i=0}^{1}[1 - 2(x - x_i)l_i{}'(x_i)]l_i{}^2(x) + \sum_{i=0}^{1} f'(x_i)(x - x_i)l_i{}^2(x)$$

其中

$$l_i(x) = \prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{x - x_i}{x_i - x_j}, \quad i = 0, 1$$

$$l_i'(x) = \frac{1}{x_i - x_j}, \quad i = 0, 1,\ j \neq i$$

利用上述公式，可以类似地用两重循环求得 Hermite 多项式在若干所需作图点的值.

可以通过 Python 的 Scipy 库中的 `interpolate.PchipInterpolator` 函数构造出分段三次 Hermite 插值多项式.

6.3 结果分析

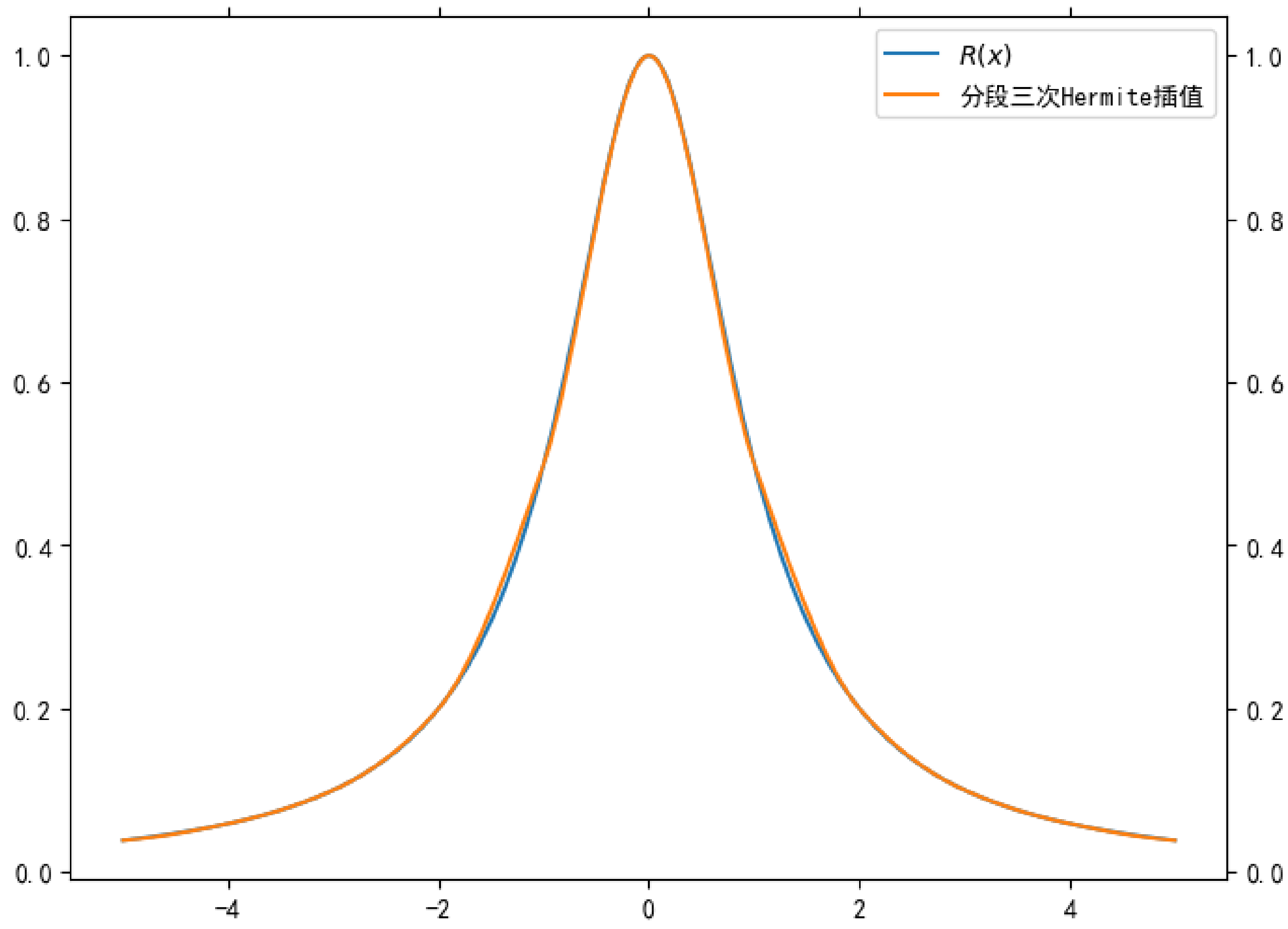

图 5: 问题五图像对比

可以看出，Hermite 插值的逼近效果在整个区间上都很好, 几乎和原函数重合.

§7 结论

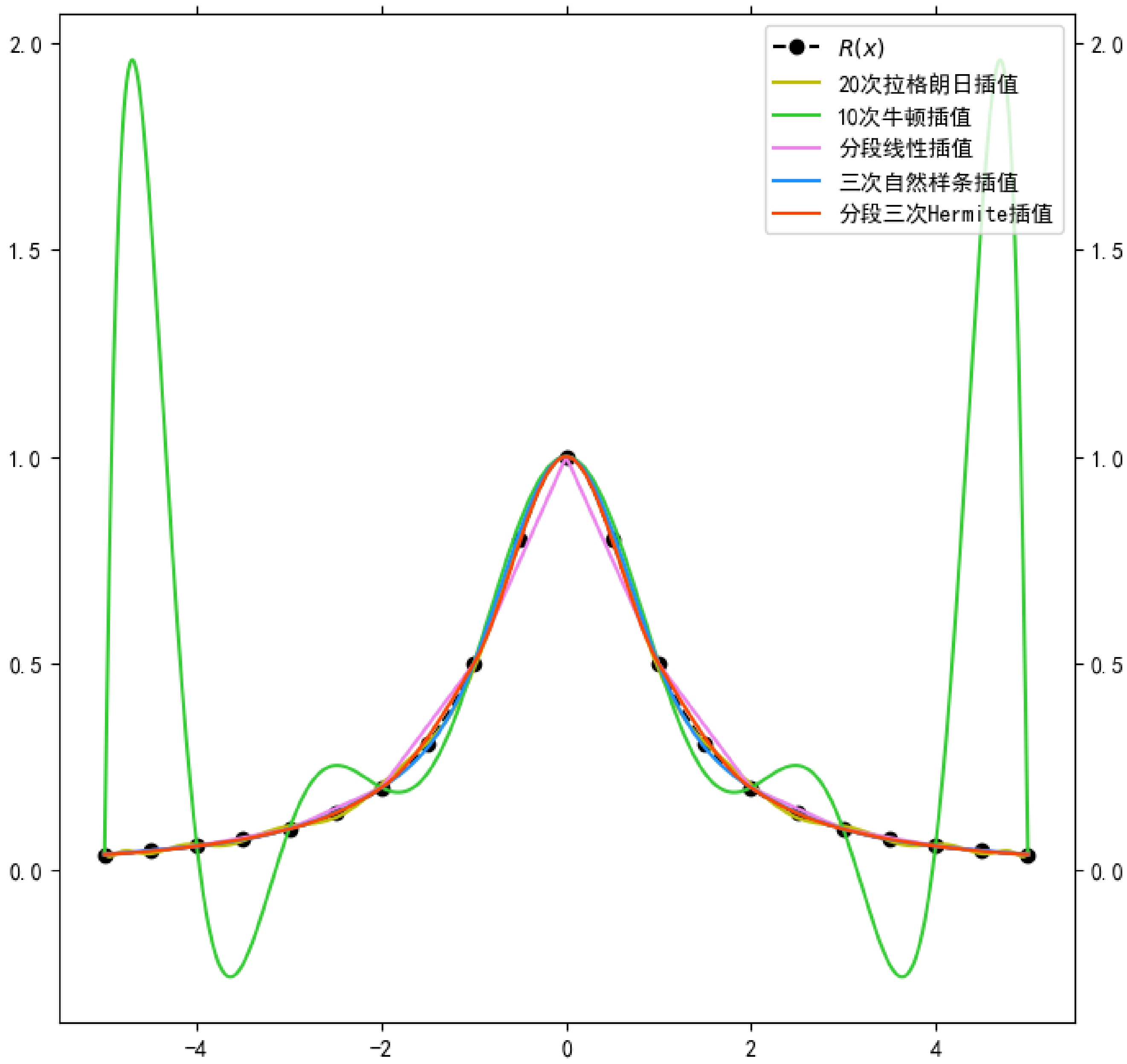

图 6: 全部插值函数图像对比

对比上述五种插值方法的效果，可以得出以下结论：

1. 在进行 Lagrange 插值和 Newton 插值时，对于次数较高的情形要避免等距基点的情况，否则在端点附近会出现 Runge 现象，误差较大.
2. 分段线性插值在计算时很方便，但若得到更好的逼近效果，则需要增加插值点的个数.
3. 三次样条插值和分段 Hermite 插值的效果要明显优于 Newton 插值和 Lagrange 插值.

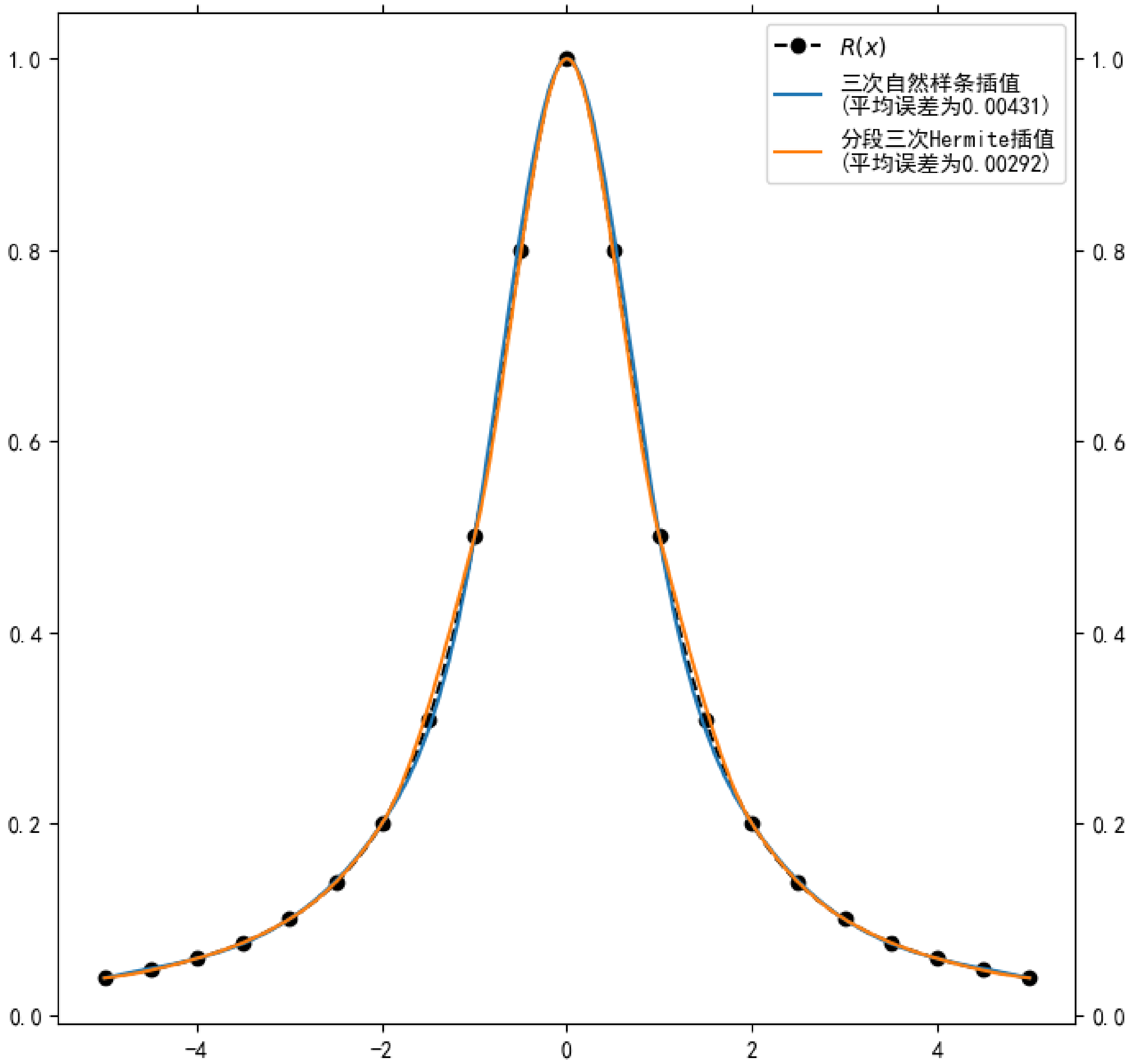

图 7: 三次样条插值与分段 Hermite 插值图像对比

4. 三次样条插值和分段 Hermite 插值各有优劣：一方面，三次样条插值保证了插值函数在这个区间上是二阶连续可微的，而分段 Hermite 插值不能保证在相邻小区间公共点处二阶可导；另一方面，在本例中，分段 Hermite 插值的逼近效果要略优于三次样条插值.

§8 附录：程序代码

注：所有题目共用一个.ipynb 源文件.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.interpolate import lagrange, interp1d,
CubicSpline, PchipInterpolator
#display Chinese in figures
plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False
%matplotlib inline
```

```
# 定义 R(x) 函数
def R(x):
    return 1 / (1 + x**2)

# 定义节点
n_lagrange = 21
n_newton = 11
x_lagrange = [5 * np.cos((2 * i + 1) / (42) * np.pi) for i in range(n_lagrange)]
x_newton = [-5 + i for i in range(n_newton)]
```

```
# 定义 Lagrange 插值多项式
def lagrange_interpolation(x, x_nodes, y_nodes):
    poly = lagrange(x_nodes, y_nodes)
    return poly(x)
# 定义 Newton 插值多项式
def newton_interpolation(x, x_nodes, y_nodes):
    poly = lagrange(x_nodes, y_nodes)
    return poly(x)
```

```
# 计算插值多项式的值
x_values = np.linspace(-5, 5, 2001)
lagrange_values = lagrange_interpolation(x_values, x_lagrange,
R(np.array(x_lagrange)))
newton_values = newton_interpolation(x_values, x_newton, R(np.array(x_newton)))
```

```
# 20次Lagrange插值多项式的图像
plt.figure(figsize=(10,6))
plt.tick_params(axis='both', which='both', bottom=True, top=True, left=True,
right=True,labelright=True)
plt.plot(x_values, R(x_values), label='$R(x)$', color='blue')
plt.plot(x_values, lagrange_values, label='20次拉格朗日插值', color='red')
plt.legend(loc='upper right')
```

```
# 10次Newton插值多项式的图像
plt.figure(figsize=(10,6))
```

```
plt.tick_params(axis='both', which='both', bottom=True, top=True, left=True,
right=True,labelright=True)
plt.plot(x_values, R(x_values), label='$R(x)$', color='blue')
plt.plot(x_values, newton_values, label='10次牛顿插值', color='red')
plt.legend(loc='upper right')
```

```
# 分段线性插值函数的图像
plt.figure(figsize=(10,6))
plt.tick_params(axis='both', which='both', bottom=True, top=True, left=True,
right=True,labelright=True)
linear_interp = interp1d(x_newton, R(np.array(x_newton)), kind='linear',
fill_value="extrapolate")
plt.plot(x_values, R(x_values), label='$R(x)$', color='blue')
plt.plot(x_values, linear_interp(x_values), label='分段线性插值', color='red')
plt.legend(loc='upper right')
```

```
# 三次自然样条插值函数的图像
plt.figure(figsize=(10,6))
plt.tick_params(axis='both', which='both', bottom=True, top=True, left=True,
right=True,labelright=True)
cubic_spline = CubicSpline(x_newton, R(np.array(x_newton)))
plt.plot(x_values, R(x_values), label='$R(x)$', color='blue')
plt.plot(x_values, cubic_spline(x_values), label='三次自然样条插值', color='red')
plt.legend(loc='upper right')
```

```
# 分段三次Hermite插值函数的图像
plt.figure(figsize=(10,6))
plt.tick_params(axis='both', which='both', bottom=True, top=True, left=True,
right=True,labelright=True)
pchip_interp = PchipInterpolator(x_newton, R(np.array(x_newton)))
plt.plot(x_values, R(x_values), label='$R(x)$', color='blue')
plt.plot(x_values, pchip_interp(x_values), label='分段三次Hermite插值', color='red
plt.legend(loc='upper right')
```

```
import seaborn as sns
sns.set_theme(style="darkgrid")
# display Chinese in sns plot
plt.rcParams['font.sans-serif'] = ['SimHei']
# plot all curves in one gragh
plt.figure(figsize=(20,12))
plt.tick_params(axis='both', which='both', bottom=True, top=True, left=True,
right=True,labelright=True)
plt.plot(x_values, R(x_values), label='$R(x)$', linestyle='--',color='black',
marker="o", markeredgecolor='black', markerfacecolor='black', markevery=100)
plt.plot(x_values, lagrange_values, label='20次拉格朗日插值', color='y')
plt.plot(x_values, newton_values, label='10次牛顿插值', color='limegreen')
```

```python
plt.plot(x_values, linear_interp(x_values), label='分段线性插值',
color='violet')
plt.plot(x_values, cubic_spline(x_values), label='三次自然样条插值',
color='dodgerblue')
plt.plot(x_values, pchip_interp(x_values), label='分段三次Hermite插值',
color='orangered')
plt.legend(loc='upper right')
```

```python
plt.figure(figsize=(8,8))
plt.tick_params(axis='both', which='both', bottom=True, top=True, left=True,
right=True,labelright=True)
plt.plot(x_values, R(x_values), label='$R(x)$', linestyle='--',color='black',
marker="o", markeredgecolor='black', markerfacecolor='black', markevery=100)
plt.plot(x_values, cubic_spline(x_values), label='三次自然样条插值\n
(平均误差为0.00431)')
plt.plot(x_values, pchip_interp(x_values), label='分段三次Hermite插值\n
(平均误差为0.00292)')
plt.legend(loc='upper right')
r1=(np.sum((np.abs(R(x_values)-cubic_spline(x_values)))))/2000
r2=(np.sum((np.abs(R(x_values)-pchip_interp(x_values)))))/2000
print(f'三次样条插值的平均误差为{r1}')
print(f'分段Hermite插值的平均误差为{r2}')
```