数值分析第 4 章上机

By 211870125 陈睿硕

§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:

Q1 用等距节点 $x_i=-5+i, i=0,1,\cdots,10$. 给出它的 10 次 Newton 插值多项式的图像；

Q2 用节点 $x_i=5\cos\frac{2i+1}{42}\pi, i=0,1,\cdots,20$. 给出它的 20 次 Lagrange 插值多项式的图像；

Q3 用等距节点 $x_i=-5+i, i=0,1,\cdots,10$. 给出它的分段线性插值函数的图像；

Q4 用等距节点 $x_i=-5+i, i=0,1,\cdots,10$. 给出它的三次自然样条插值函数的图像；

Q5 用等距节点 $x_i=-5+i, i=0,1,\cdots,10$. 给出它的分段三次 Hermite 插值函数的图像；

§2 算法思路

I 对Q1的解答

使用 Python 实现，代码见代码 1。如图 1。

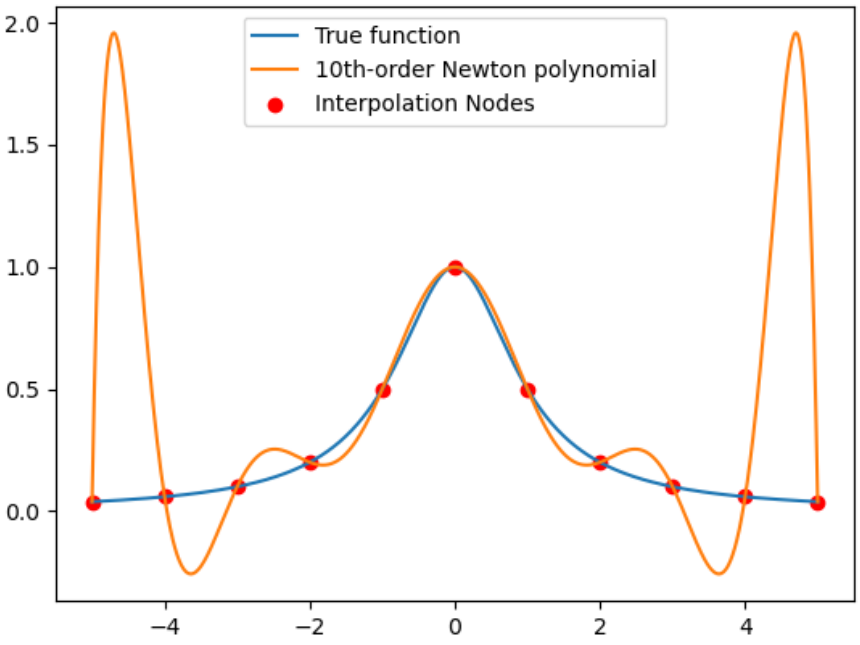

图 1: 10 次 Newton 插值多项式

II 对Q2的解答

使用 Python 实现，代码见代码 2。如图 2。

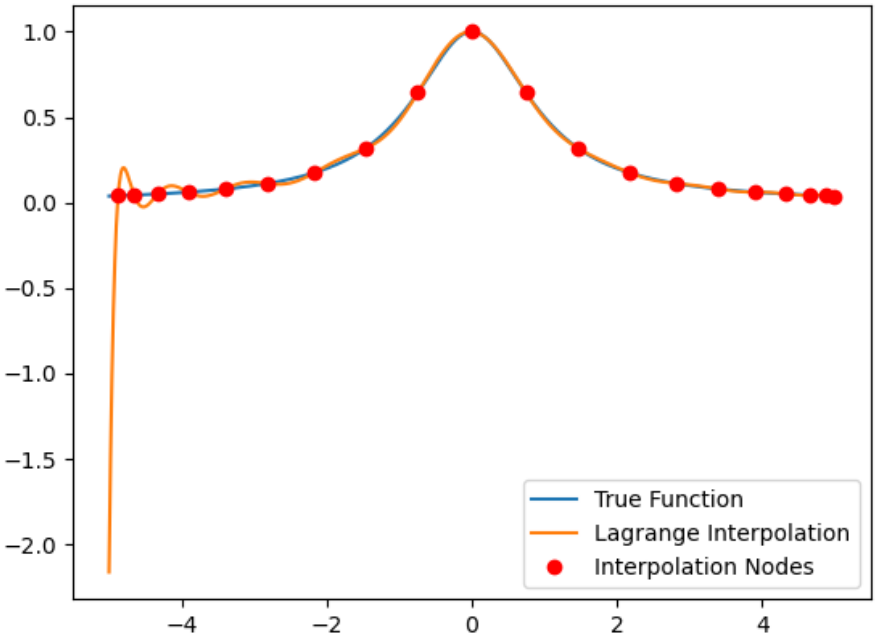

图 2: 20 次 Lagrange 插值多项式

III 对Q3的解答

使用 Python 实现，代码见代码 3。如图 3。

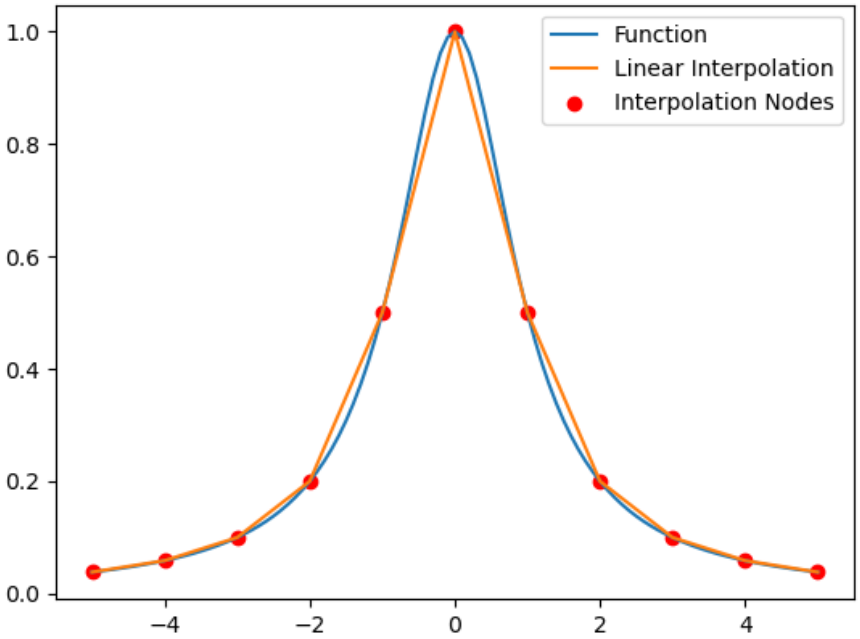

图 3: 分段线性插值函数

IV 对Q4的解答

可以快捷地调用 scipy.interpolate 中的 CubicSpline 库完成，见代码 4。也可以”从零开始“，将三次样条插值函数展开，使用 Python 程序计算其系数，见代码 5。如图 4。

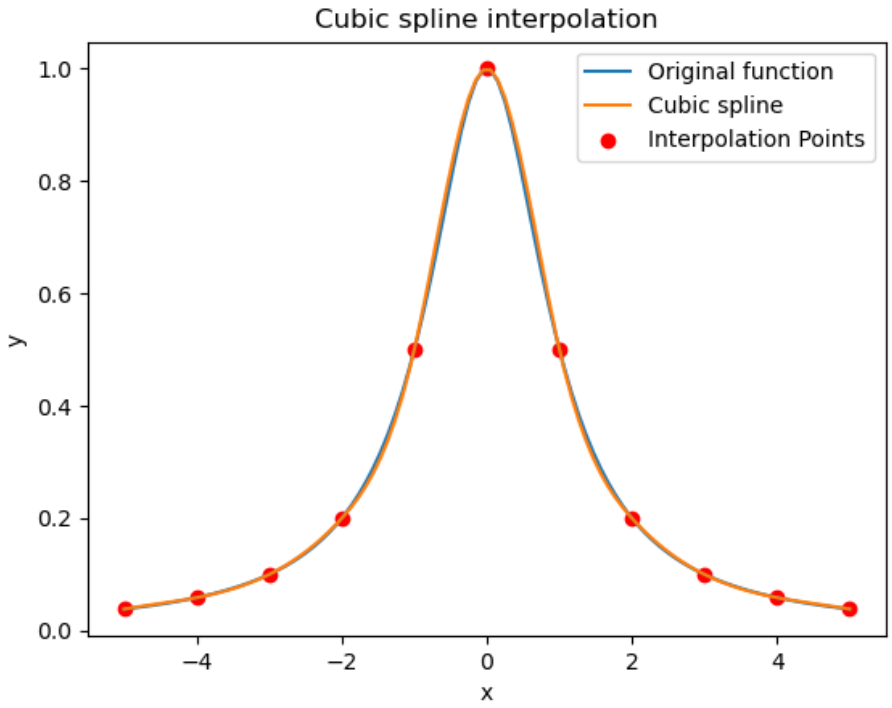

图 4: 三次自然样条插值函数

V 对Q5的解答

调用 scipy.interpolate 中的 PchipInterpolator 库完成，见代码 6。也可以”从零开始“。由于此处是等距插值，故可以手动解出 $m_i, i = 1, 2, \cdots, n+1$（书上记号），写出系数表达式，从而直接求出每个小区间的三次 Hermite 插值多项式。见代码 7。如图 5。

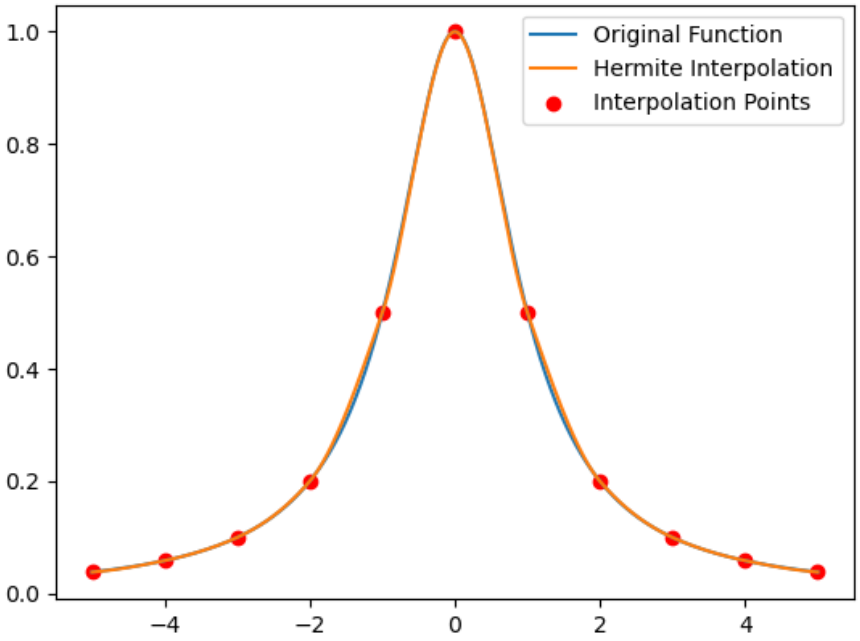

图 5: 分段三次 Hermite 插值函数

§3 结果分析

I 对Q1的分析

观察图 1可知，等距插值多项式发生了严重的 Runge 现象。

II 对Q2的分析

观察图 2可知，此时的插值多项式 Runge 现象并不严重，这主要是因为采用了 Chebyshev 插值法，导致误差的上界被严格控制，故函数逼近较为准确。

III 对Q3的分析

观察图 3可知，线性插值函数无 Runge 现象，但插值函数在插值点处不平滑。

IV 对Q4、Q5的分析

观察图 4、图 5可知，分段三次 Hermite 插值函数和分段三次样条差值函数都不会发生 Runge 现象，这主要是因为小区间拟合减少了插值函数的可波动性（即减少了与原函数之间的差值上界）。

§4 附录：程序代码

代码 1: Fourth Week 1A.py

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

def f(x):
```

```
    return 1 / (1 + x ** 2)

x = np.linspace(-5, 5, 11)
y = f(x)
n = len(x)
c = y.copy()
for j in range(1, n):
    c[j:n] = (c[j:n] - c[j - 1]) / (x[j:n] - x[j - 1])

def poly(xval):
    p = c[-1] * np.ones_like(xval)
    for j in range(n - 2, -1, -1):
        p = (xval - x[j]) * p + c[j]
    return p

xval = np.linspace(-5, 5, 1000)
plt.plot(xval, f(xval), label='True function')
plt.plot(xval, poly(xval), label='10th-order Newton polynomial')
plt.scatter(x, y, color="red", label='Interpolation Nodes')
plt.legend()
plt.show()
```

代码 2: Fourth Week 1B.py

```
import numpy as np
import matplotlib.pyplot as plt
def f(x):
    return 1 / (1 + x ** 2)
n = 20
x_nodes = 5 * np.cos((2 * np.arange(n) + 1) * np.pi / 42)
def Lk(x, k, x_nodes):
    l = np.ones_like(x)
    for i in range(len(x_nodes)):
        if i == k:
            continue
        l *= (x - x_nodes[i]) / (x_nodes[k] - x_nodes[i])
    return l
def lagrange_interp(x, x_nodes, y_nodes):
    interp = 0
    for k in range(len(x_nodes)):
        interp += y_nodes[k] * Lk(x, k, x_nodes)
    return interp
```

```
x = np.linspace(-5, 5, 1000)
y_true = f(x)
y_interp = lagrange_interp(x, x_nodes, f(x_nodes))

plt.plot(x, y_true, label='True Function')
plt.plot(x, y_interp, label='Lagrange Interpolation')
plt.plot(x_nodes, f(x_nodes), 'ro', label='Interpolation Nodes')
plt.legend()
plt.show()
```

代码 3: Fourth Week 1C.py

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

def f(x):
    return 1/(1+x**2)

x = np.linspace(-5, 5, 11)
y = f(x)
from scipy import interpolate
lin_interp = interpolate.interp1d(x, y)
xx = np.linspace(-5, 5, 101)
yy = lin_interp(xx)

plt.plot(xx, f(xx), label='Function')
plt.plot(xx, yy, label='Linear Interpolation')
plt.scatter(x, y, color="red", label='Interpolation Nodes')
plt.legend()
plt.show()
```

代码 4: Fourth Week 1D 1.py

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

def f(x):
    return 1 / (1 + x**2)

x = np.linspace(-5, 5, 11)
y = f(x)
cs = CubicSpline(x, y)
```

```
x_vals = np.linspace(-5, 5, 100)
plt.plot(x_vals, f(x_vals), label='Original function')
plt.plot(x_vals, cs(x_vals), label='Cubic spline')
plt.scatter(x, y, c='r', label='Interpolation Points')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Cubic spline interpolation')
plt.legend()
plt.show()
```

代码 5: Fourth Week 1D 2.py

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

def natural_cubic_spline(x, y):
    n = len(x)
    a = y.copy()
    b = np.zeros(n)
    d = np.zeros(n)
    h = np.diff(x)
    alpha = np.zeros(n-1)
    for i in range(1, n-1):
        alpha[i] = 3/h[i]*(a[i+1]-a[i])-3/h[i-1]*(a[i]-a[i-1])
    c = np.zeros(n)
    l = np.zeros(n)
    mu = np.zeros(n-1)
    z = np.zeros(n)
    l[0] = 1
    mu[0] = 0
    z[0] = 0
    for i in range(1, n-1):
        l[i] = 2*(x[i+1]-x[i-1])-h[i-1]*mu[i-1]
        mu[i] = h[i]/l[i]
        z[i] = (alpha[i]-h[i-1]*z[i-1])/l[i]
    l[n-1] = 1
    z[n-1] = 0
    for j in range(n-2, -1, -1):
        c[j] = z[j]-mu[j]*c[j+1]
        b[j] = (a[j+1]-a[j])/h[j]-h[j]*(c[j+1]+2*c[j])/3
        d[j] = (c[j+1]-c[j])/(3*h[j])
```

```
    return a, b, c, d

def eval_cubic_spline(x_eval, x, a, b, c, d):
    idx = np.digitize(x_eval, x) - 1
    idx = np.clip(idx, 0, len(x)-2)
    h = x_eval - x[idx]
    y_eval = a[idx] + b[idx]*h + c[idx]*h**2 + d[idx]*h**3
    return y_eval

R = lambda x: 1/(1+x**2)

x = np.linspace(-5, 5, 11)
y = R(x)

a, b, c, d = natural_cubic_spline(x, y)

x_eval = np.linspace(-5, 5, 501)
y_eval = eval_cubic_spline(x_eval, x, a, b, c, d)

fig, ax = plt.subplots(figsize=(8, 6))
ax.plot(x_eval, R(x_eval), label='Original function')
ax.plot(x_eval, y_eval, label='Cubic Spline')
ax.scatter(x, y, c='r', label='Interpolation Nodes')
ax.legend(loc='upper left')
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_title('Cubic Spline Interpolation')
plt.show()
```

代码 6: Fourth Week 1E 1.py

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

def f(x):
    return 1 / (1 + x**2)

x = np.linspace(-5, 5, 11)
y = f(x)
interp_func = PchipInterpolator(x, y)
xx = np.linspace(-5, 5, 1000)
```

```
yy = f(xx)

plt.plot(xx, yy, label='Original Function')
plt.plot(xx, interp_func(xx), label='Hermite Interpolation')
plt.scatter(x, y, c='r', label='Interpolation Points')
plt.legend()
plt.show()
```

代码 7: Fourth Week 1E 2.py

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

R = lambda x: 1/(1+x**2)

x = np.linspace(-5, 5, 11)
y = R(x)

R_deriv = lambda x: -2*x/(1+x**2)**2
y_deriv = R_deriv(x)

h = np.diff(x)
a = y[:-1]
b = y_deriv[:-1]
c = (-3*h*y[:-1] - 2*h*y_deriv[:-1] + 3*h*y[1:] - h*y_deriv[1:])/(h**2)
d = (2*h*y[:-1] + h*y_deriv[:-1] - 2*h*y[1:] + h*y_deriv[1:])/(h**3)

x_eval = np.linspace(-5, 5, 501)
y_eval = np.zeros_like(x_eval)
for i in range(len(x)-1):
    idx = np.logical_and(x_eval >= x[i], x_eval <= x[i+1])
    h_i = x_eval[idx] - x[i]
    y_eval[idx] = a[i] + b[i]*h_i + c[i]*h_i**2 + d[i]*h_i**3

fig, ax = plt.subplots(figsize=(8, 6))
ax.plot(x_eval, R(x_eval), label='Original function')
ax.plot(x_eval, y_eval, label='Hermite Interpolation')
ax.scatter(x, y, c='r',label='Interpolation Nodes')
ax.legend(loc='upper left')
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_title('Hermite Interpolation')
```

```
plt.show()
```