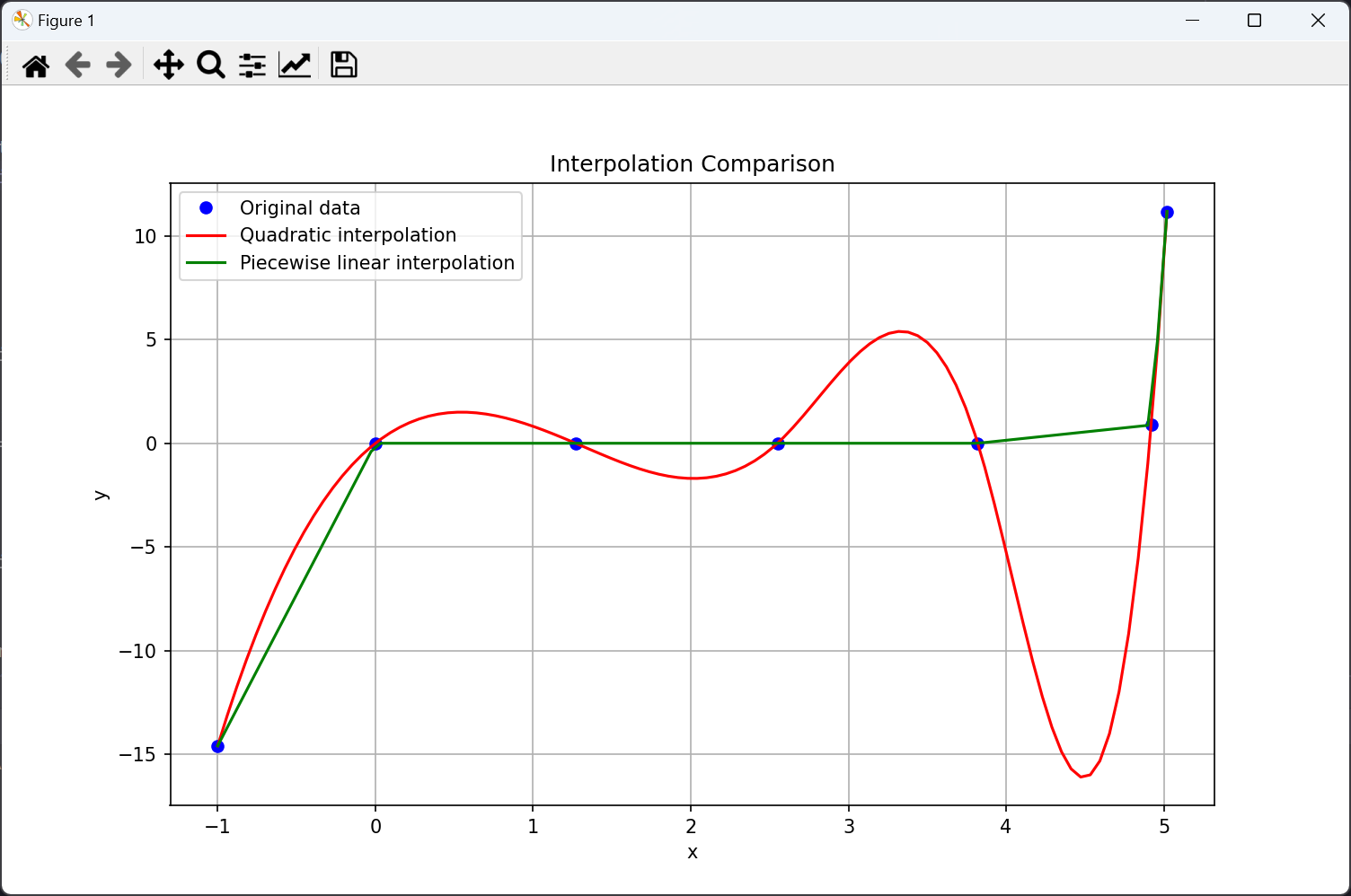
计算物理第六次作业

白博臣 2022141220036

## Problem 1：

1. import numpy as np
2. from scipy.interpolate import interp1d
3. import matplotlib.pyplot as plt
4. *# 生成示例数据并进行排序*
5. data = [(-1.00, -14.58), (0.00, 0.00), (1.27, 0.00), (2.55, 0.00), (3.82, 0.00), (4.92, 0.88), (5.02,11.17)]
6. *# 对数据按照 x 值排序*
7. data.sort(key=lambda d: d[0])
8. x, y = zip(\*data)
9. *# 连续多项式插值*
10. f\_poly = interp1d(x, y, kind='cubic')
11. *# 低阶多项式分段插值*
12. f\_piecewise = interp1d(x, y, kind='linear')
13. *# 生成更密集的x值，用于绘制插值结果的曲线*
14. x\_dense = np.linspace(min(x), max(x), 100)
15. *# 计算插值结果*
16. y\_poly = f\_poly(x\_dense)
17. y\_piecewise = f\_piecewise(x\_dense)
18. *# 绘制原始数据和插值结果*
19. plt.figure(figsize=(10, 6))
20. plt.plot(x, y, 'bo', label='Original data')
21. plt.plot(x\_dense, y\_poly, 'r-', label='Quadratic interpolation')
22. plt.plot(x\_dense, y\_piecewise, 'g-', label='Piecewise linear interpolation')
23. plt.legend()
24. plt.xlabel('x')
25. plt.ylabel('y')
26. plt.title('Interpolation Comparison')
27. plt.grid(True)
28. plt.show()



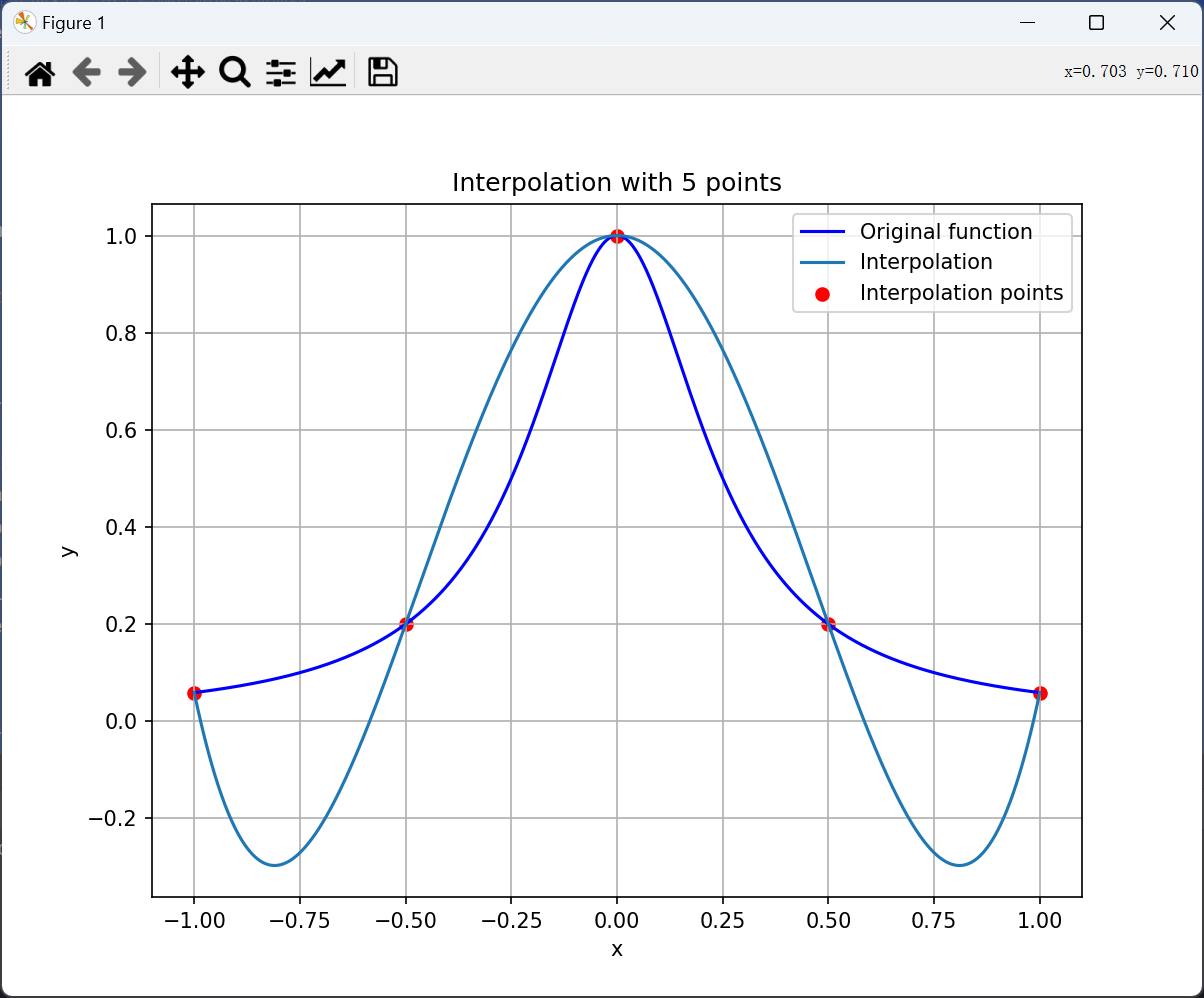
观察到原本应该平稳（值为0）的点产生了震荡现象；反而采用平滑分段低阶多项式拟合的效果更好一些。说明了局部拟合并非阶数越高越好。

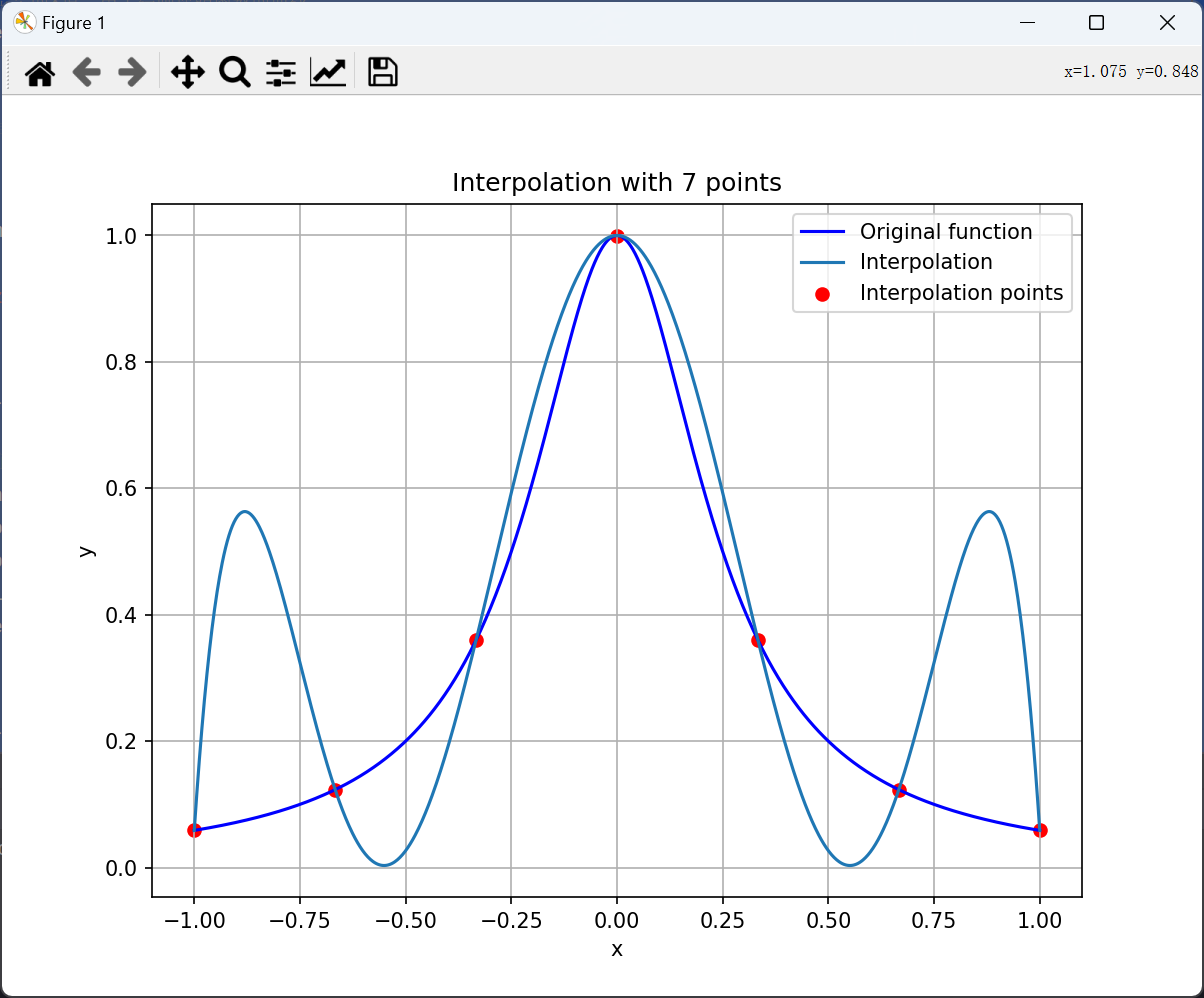
## Problem 2：

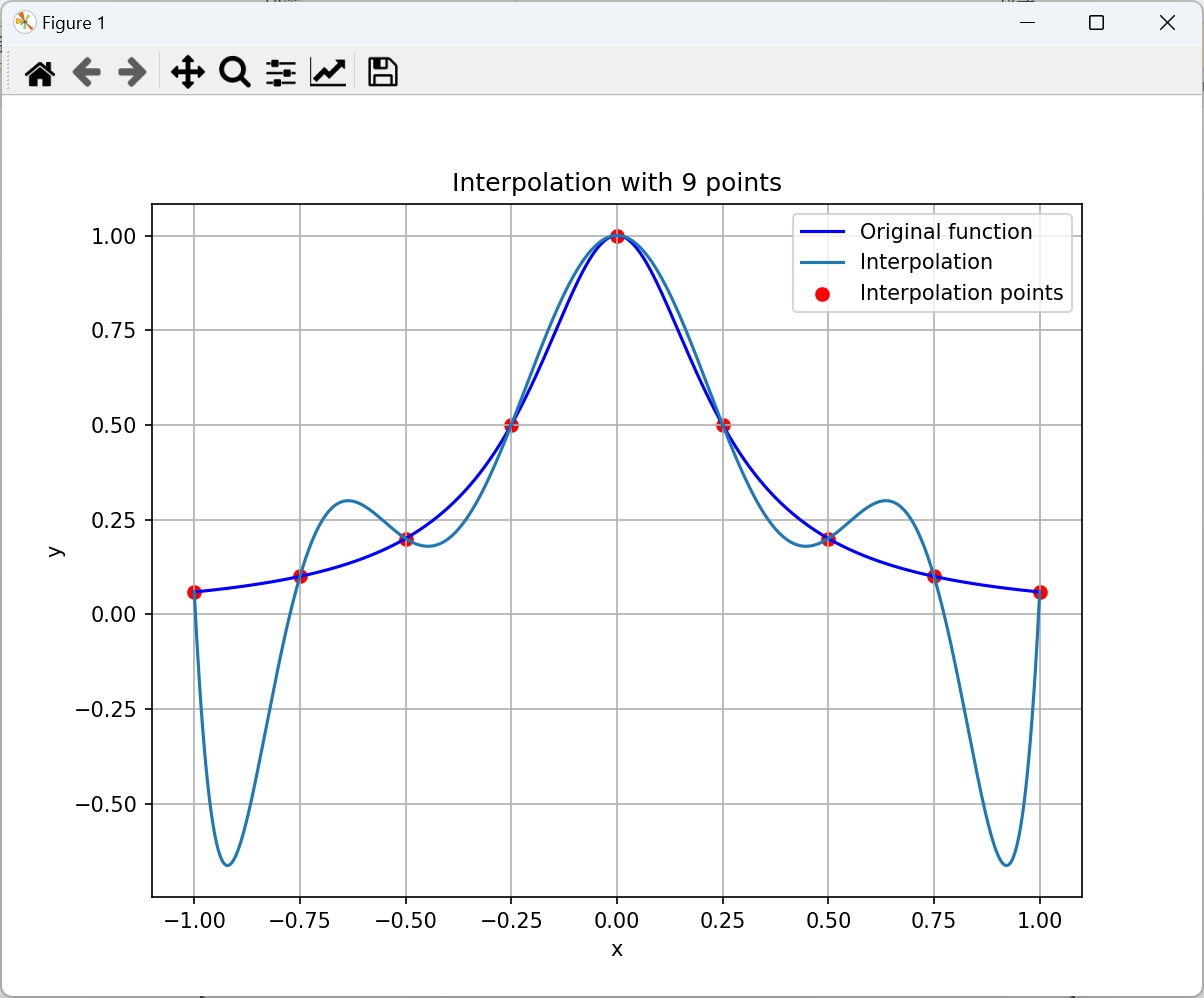
由于不能按照题目要求5个点进行5次多项式拟合，故改为5个点进行4次多项式拟合，后面的所有点均按照降一阶处理。

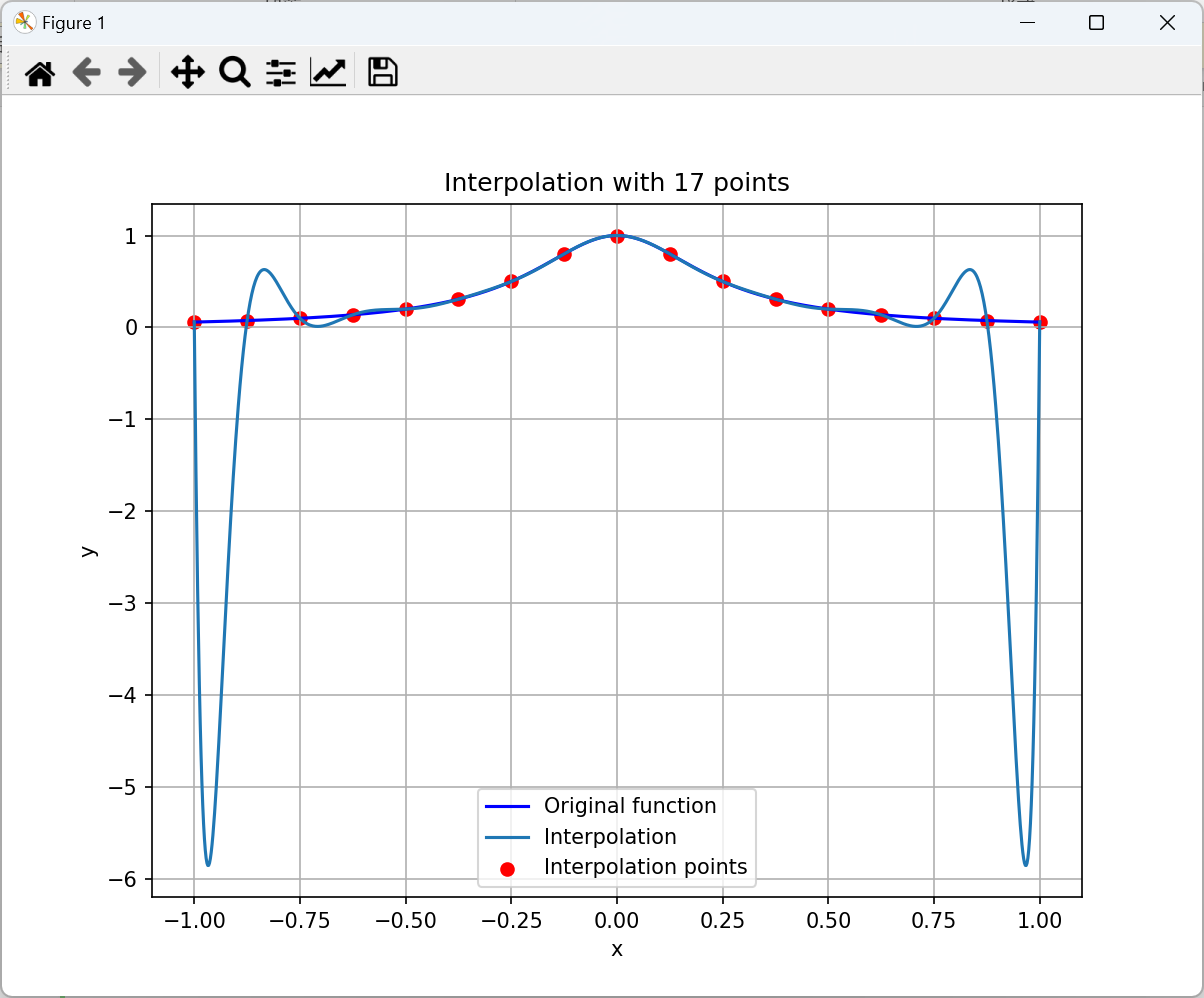
代码：

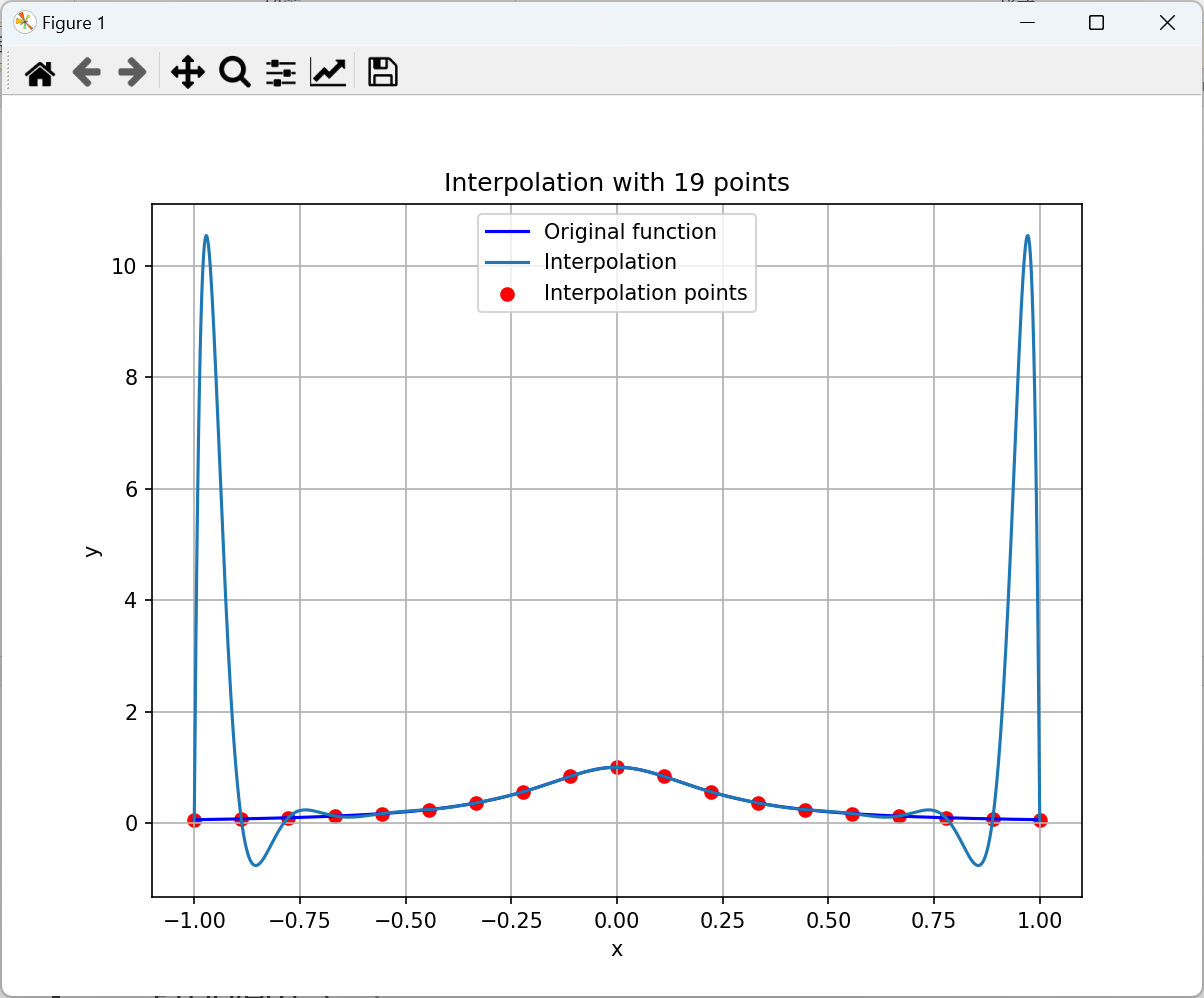
1. import numpy as np
2. import matplotlib.pyplot as plt
3. *# 定义函数*
4. def f(x):
5. return 1 / (1 + 16 \* x \*\* 2)
6. *# 在[-1, 1]范围内生成更密集的x值，用于绘制原始函数的曲线*
7. x\_dense = np.linspace(-1, 1, 1000)
8. y\_dense = f(x\_dense)
9. *# 在[-1, 1]范围内生成不同数量点的情况*
10. num\_points\_list = [5, 7, 9, 17, 19, 21]
11. for num\_points in num\_points\_list:
12. *# 创建新的图*
13. plt.figure(figsize=(8, 6))
14. *# 绘制原始函数*
15. plt.plot(x\_dense, y\_dense, 'b-', label='Original function')
16. *# 生成划分点并计算插值结果*
17. x\_points = np.linspace(-1, 1, num\_points)
18. y\_points = f(x\_points)
19. coefficients = np.polyfit(x\_points, y\_points, num\_points - 1)
20. f\_interp = np.poly1d(coefficients)
21. y\_interp = f\_interp(x\_dense)
22. *# 绘制插值结果*
23. plt.plot(x\_dense, y\_interp, label='Interpolation')
24. *# 添加划分点*
25. plt.scatter(x\_points, y\_points, color='red', label='Interpolation points')
26. *# 添加标题、标签和图例*
27. plt.title(f'Interpolation with {num\_points} points')
28. plt.xlabel('x')
29. plt.ylabel('y')
30. plt.legend()
31. *# 显示网格*
32. plt.grid(True)
33. *# 显示图*
34. plt.show()

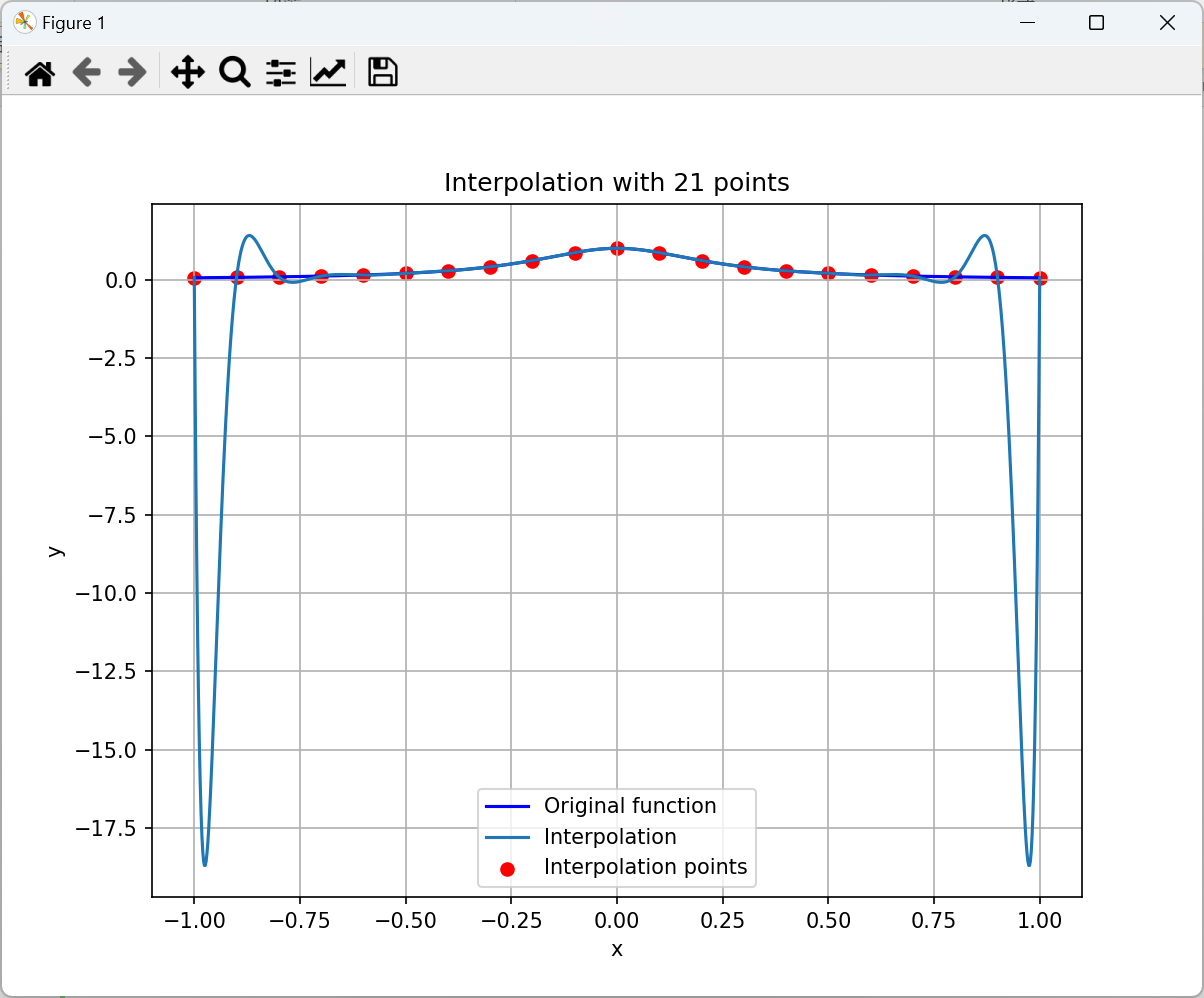










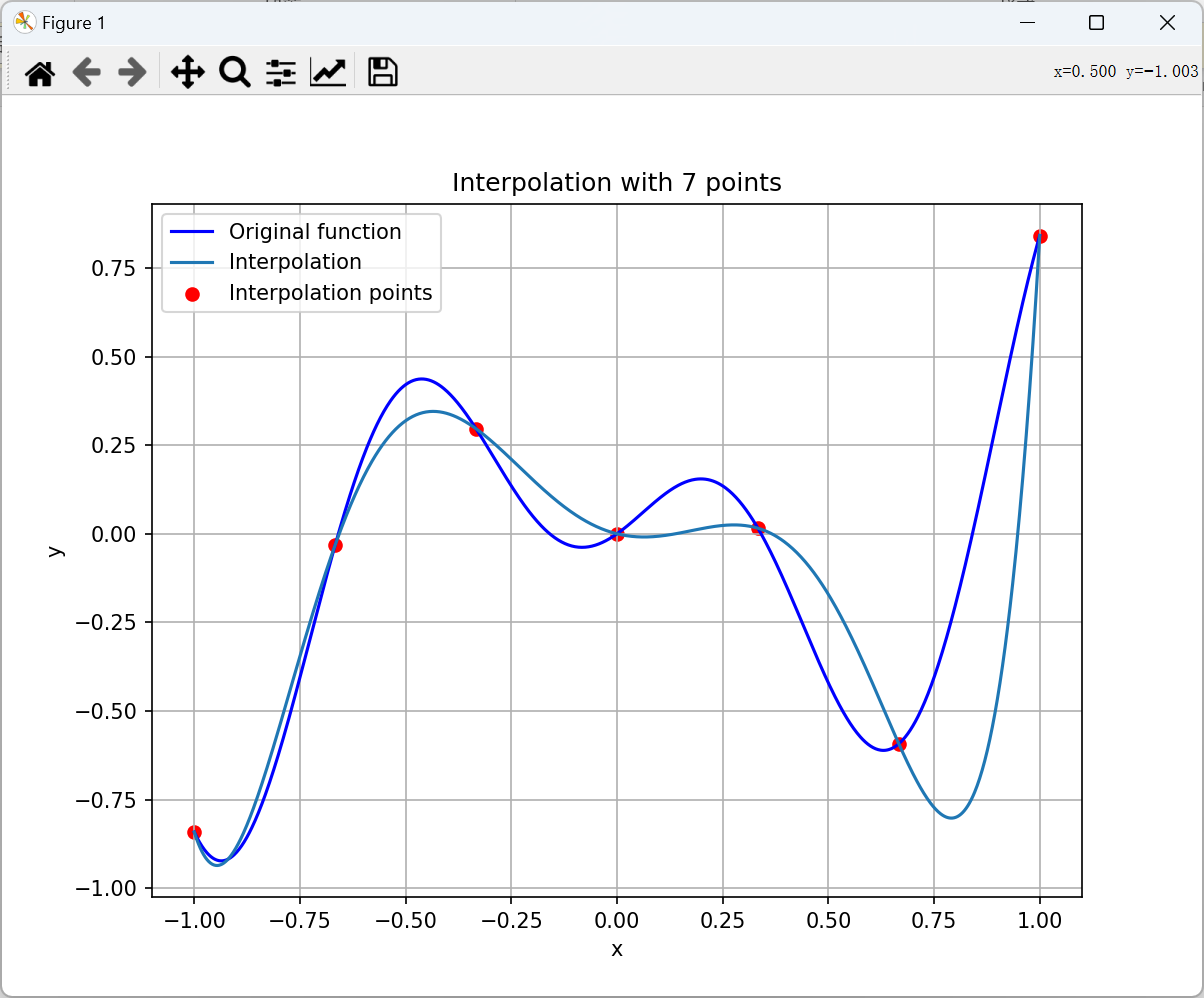


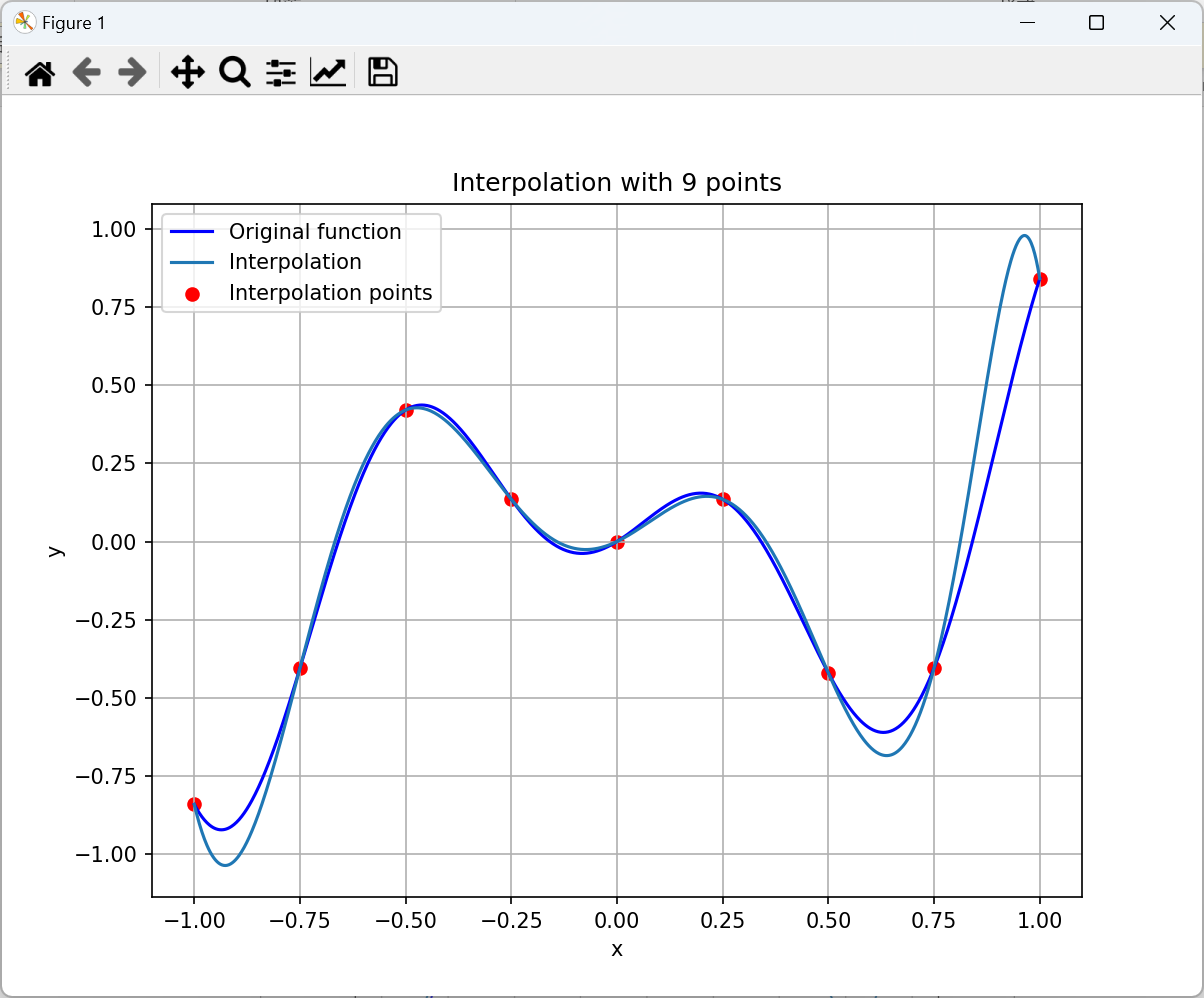
观察图像我们发现图像的两端均出现了震荡，而且拟合阶数越高震荡效果越明显，说明此时该局部拟合不应选取过高的阶数进行拟合。

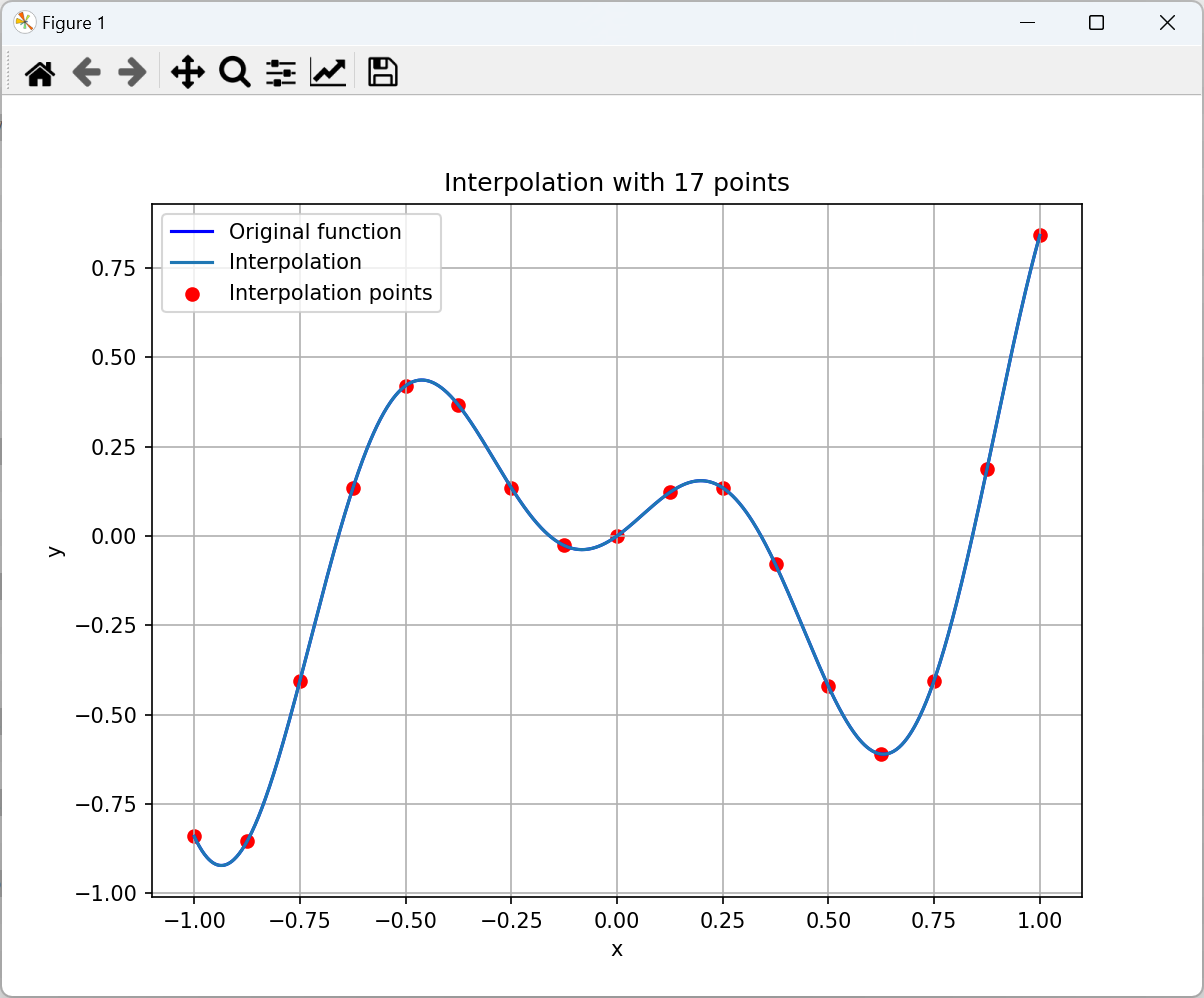
## Problem 3：

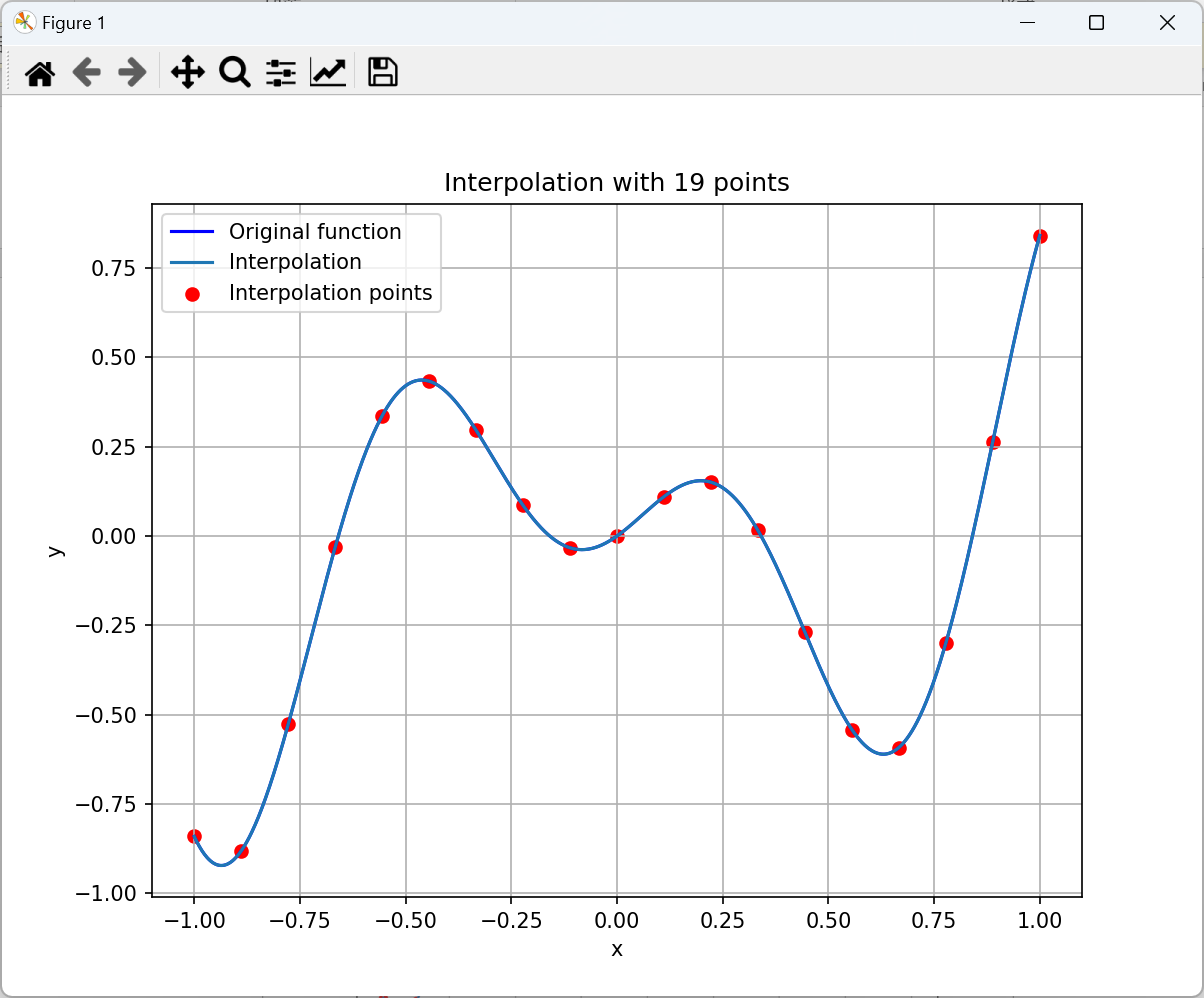
代码：

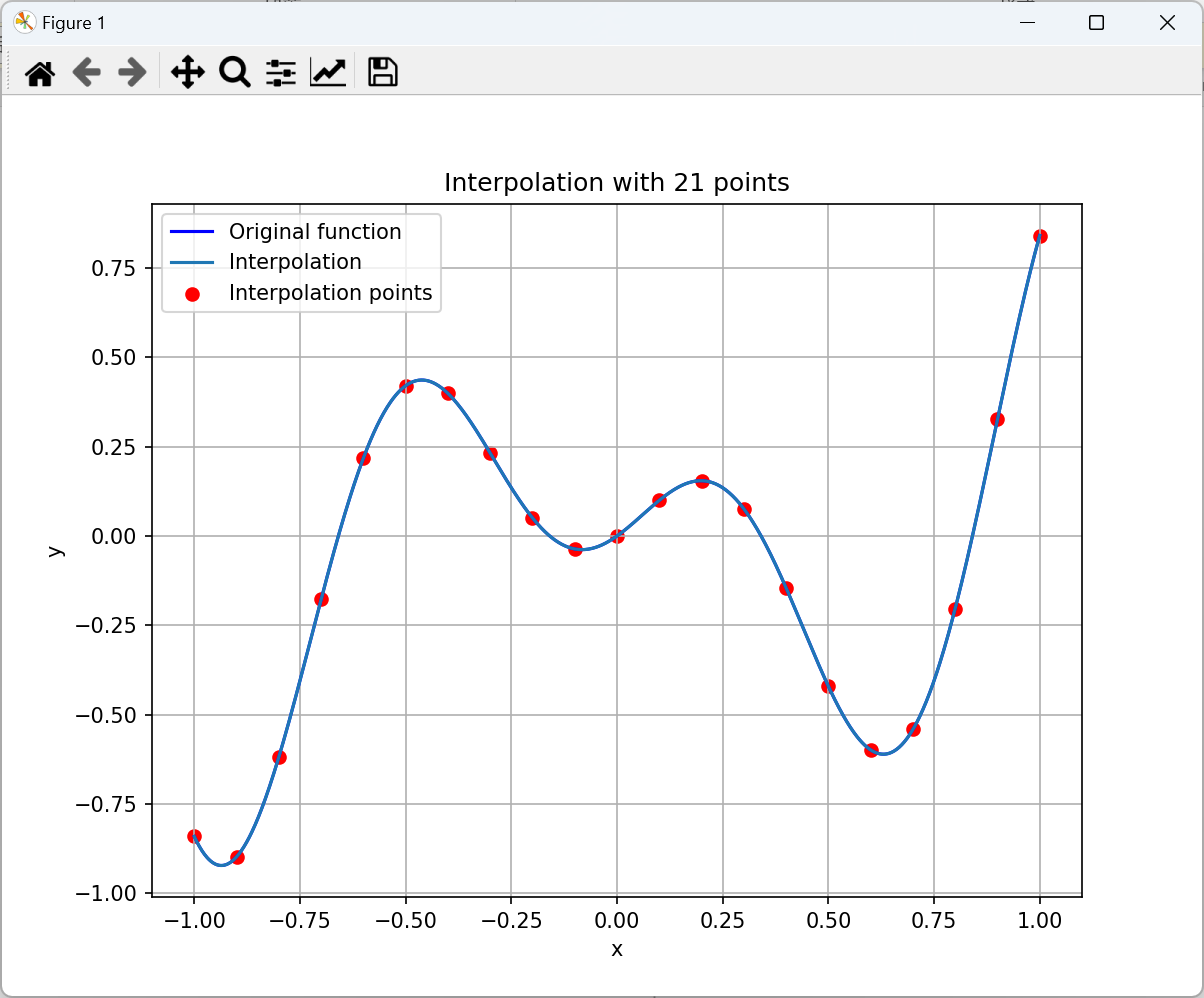
1. import numpy as np
2. import matplotlib.pyplot as plt
3. *# 定义函数*
4. def f(x):
5. return x\*np.sin(2\*np.pi\*x+1)
6. *# 在[-1, 1]范围内生成更密集的x值，用于绘制原始函数的曲线*
7. x\_dense = np.linspace(-1, 1, 1000)
8. y\_dense = f(x\_dense)
9. *# 在[-1, 1]范围内生成不同数量点的情况*
10. num\_points\_list = [7, 9, 17, 19, 21]
11. for num\_points in num\_points\_list:
12. *# 创建新的图*
13. plt.figure(figsize=(8, 6))
14. *# 绘制原始函数*
15. plt.plot(x\_dense, y\_dense, 'b-', label='Original function')
16. *# 生成划分点并计算插值结果*
17. x\_points = np.linspace(-1, 1, num\_points)
18. y\_points = f(x\_points)
19. coefficients = np.polyfit(x\_points, y\_points, num\_points - 1)
20. f\_interp = np.poly1d(coefficients)
21. y\_interp = f\_interp(x\_dense)
22. *# 绘制插值结果*
23. plt.plot(x\_dense, y\_interp, label='Interpolation')
24. *# 添加划分点*
25. plt.scatter(x\_points, y\_points, color='red', label='Interpolation points')
26. *# 添加标题、标签和图例*
27. plt.title(f'Interpolation with {num\_points} points')
28. plt.xlabel('x')
29. plt.ylabel('y')
30. plt.legend()
31. *# 显示网格*
32. plt.grid(True)
33. *# 显示图*
34. plt.show()











观察可以发现，在该函数情况下，局部拟合越高阶拟合效果越好。