计算流体力学作业 2

郑恒 2200011086

2025 年 3 月 26 日

1 数据算法原理

1.1 一阶导数 $\frac{\partial u}{\partial x}$ 的差分格式

1.1.1 采用 2 个点的一阶差分格式

- 二点格式：将 u_{i+1} 泰勒展开：

$$u_{i+1} = u_i + \Delta x \frac{\partial u}{\partial x} + \frac{1}{2}\Delta x^2 \frac{\partial^2 u}{\partial x^2} + \mathcal{O}(\Delta x^3)$$

$$\frac{\partial u}{\partial x} = \frac{u_{i+1} - u_i}{\Delta x} + \mathcal{O}(\Delta x)$$

精度：$\mathcal{O}(\Delta x)$。

1.1.2 采用 2 个点的二阶差分格式

- 二点格式：将 u_{i+1} 和 u_{i-1} 泰勒展开：

$$u_{i+1} = u_i + \Delta x \frac{\partial u}{\partial x} + \frac{1}{2}\Delta x^2 \frac{\partial^2 u}{\partial x^2} + \frac{1}{6}\Delta x^3 \frac{\partial^3 u}{\partial x^3} + \mathcal{O}(\Delta x^4)$$

$$u_{i-1} = u_i - \Delta x \frac{\partial u}{\partial x} + \frac{1}{2}\Delta x^2 \frac{\partial^2 u}{\partial x^2} - \frac{1}{6}\Delta x^3 \frac{\partial^3 u}{\partial x^3} + \mathcal{O}(\Delta x^4)$$

两式相减：

$$\frac{\partial u}{\partial x} = \frac{u_{i+1} - u_{i-1}}{2\Delta x} + \mathcal{O}(\Delta x^2)$$

精度：$\mathcal{O}(\Delta x^2)$。

1.2 二阶导数 $\frac{\partial^2 u}{\partial x^2}$ 的差分格式

1.2.1 采用 3 个点的一阶差分格式

- 三点格式：将 u_{i-2} 和 u_{i-1} 泰勒展开到 4 阶：

$$u_{i-2} = u_i - 2\Delta x\frac{\partial u}{\partial x} + 2\Delta x^2\frac{\partial^2 u}{\partial x^2} - \frac{4}{3}\Delta x^3\frac{\partial^3 u}{\partial x^3} + \frac{2}{3}\Delta x^4\frac{\partial^4 u}{\partial x^4} + \mathcal{O}(\Delta x^5)$$

$$u_{i-1} = u_i - \Delta x\frac{\partial u}{\partial x} + \frac{1}{2}\Delta x^2\frac{\partial^2 u}{\partial x^2} - \frac{1}{6}\Delta x^3\frac{\partial^3 u}{\partial x^3} + \frac{1}{24}\Delta x^4\frac{\partial^4 u}{\partial x^4} + \mathcal{O}(\Delta x^5)$$

- 假设一阶导数的差分公式为：

$$\frac{\partial^2 u}{\partial^2 x} = au_{i-2} + bu_{i-1} + cu_i$$

  其中 a, b, c 是待定系数。

- 将 u_{i-2}, u_{i-1} 的泰勒展开式代入差分公式，整理各阶导数的系数，得到以下方程组：

$$a + b + c = 0 \quad \text{（消除常数项）}$$

$$-2a - b = 0 \quad \text{（消除一阶导数项）}$$

$$4a + b = \frac{2}{\Delta x^2} \quad \text{（二阶导数项系数为 1）}$$

- 解方程组，得到：

$$a = \frac{1}{\Delta x^2}, \quad b = \frac{-2}{\Delta x^2}, \quad c = \frac{1}{\Delta x^2}$$

- 将系数代入差分公式，得到：

$$\frac{\partial^2 u}{\partial x^2} = \frac{u_{i-2} - 2u_{i-1} + u_i}{\Delta x^2} + \mathcal{O}(\Delta x)$$

- 精度：$\mathcal{O}(\Delta x)$。

1.2.2 采用 3 个点的二阶差分格式

- 三点格式：将 u_{i+1},u_{i-1} 泰勒展开到 4 阶：

$$u_{i+1} = u_i + \Delta x\frac{\partial u}{\partial x} + \frac{1}{2}\Delta x^2\frac{\partial^2 u}{\partial x^2} + \frac{1}{6}\Delta x^3\frac{\partial^3 u}{\partial x^3} + \frac{1}{24}\Delta x^4\frac{\partial^4 u}{\partial x^4} + \mathcal{O}(\Delta x^5)$$

$$u_{i-1} = u_i - \Delta x\frac{\partial u}{\partial x} + \frac{1}{2}\Delta x^2\frac{\partial^2 u}{\partial x^2} - \frac{1}{6}\Delta x^3\frac{\partial^3 u}{\partial x^3} + \frac{1}{24}\Delta x^4\frac{\partial^4 u}{\partial x^4} + \mathcal{O}(\Delta x^5)$$

- 假设二阶导数的差分公式为：

$$\frac{\partial^2 u}{\partial x^2} = au_{i-1} + bu_i + cu_{i+1}$$

  其中 a, b, c 是待定系数。

- 将 u_{i+1}, u_{i-1} 的泰勒展开式代入差分公式，整理各阶导数的系数，得到以下方程组：

$$a + b + c = 0 \quad \text{（消除常数项）}$$

$$-a + c = 0 \quad \text{（消除一阶导数项）}$$

$$a + c = \frac{2}{\Delta x^2} \quad \text{（二阶导数项系数为 1）}$$

$$-a + c = 0 \quad \text{（消除三阶导数项）}$$

- 解方程组，得到：

$$a = \frac{1}{\Delta x^2}, \quad b = \frac{-2}{\Delta x^2}, \quad c = \frac{1}{\Delta x^2}$$

- 将系数代入差分公式，得到：

$$\frac{\partial^2 u}{\partial x^2} = \frac{u_{i-1} - 2u_i + u_{i+1}}{\Delta x^2} + \mathcal{O}(\Delta x^2)$$

- 精度：$\mathcal{O}(\Delta x^2)$。

后续代码中以 h 代替 Δx。

2 代码生成和调试

以下是用于验证格式精度的 Python 代码：

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

# ======================
# 格式函数
# ======================
def first_order_forward(u, dx):
    """一阶前向差分（数组长度自动减1）
    公式：(u_{i+1} - u_{i}) / dx
    """
    return (u[1:] - u[:-1]) / dx  # 直接返回N-1长度的数组



def first_order_centered(u, dx):
    """一阶导数中心差分（数组长度自动减2）
    公式：(u_{i+1} - u_{i-1} )/2dx
    """
    return (u[2:] - u[:-2]) / (2 * dx)  # 直接返回N-2长度的数组


def second_order_back(u, dx):
    """
    二阶导数：一阶精度差分（3个点）
    公式：(u_{i} - 2u_{i-1} + u_{i-2})/dx²
    """

    return (u[2:] - 2 * u[1:-1] + u[:-2]) / ( dx ** 2)  # 直接返回N-2长度的数组
```

```
29
30
31  def second_order_centered(u, dx):
32      """
33      二阶导数：二阶中心差分（3点模板）
34      公式：(u_{i+1} - 2u_i + u_{i-1})/dx²
35      """
36      return (u[2:] - 2 * u[1:-1] + u[:-2]) / (dx ** 2)  # 直
            接返回N-2长度的数组
37  # ======================
38  # 误差分析模块
39  # ======================
40
41  def compute_errors(f, df_exact, d2f_exact, x_range=(0.0, 2
        * np.pi), num_points=100, dtype=np.float64):
42      """
43      计算不同网格间距下的误差
44      """
45      steps = np.array([16, 32, 64, 128, 256, 512])  # 网格点
            数
46      h_list = (x_range[1] - x_range[0]) / (steps - 1)  # Δx
            值列表
47
48      errors = {
49          'first_order_forward': [],
50          'first_order_centered': [],
51          'second_order_back': [],
52          'second_order_centered': []
53      }
54
55      for n in steps:
56          x = np.linspace(x_range[0], x_range[1], n, dtype=
                dtype)
```

```
57          dx = x[1] - x[0]
58          u = f(x)
59
60          # 计算数值导数
61          du_forward = first_order_forward(u, dx)
62          du_centered = first_order_centered(u, dx)
63          d2u_back = second_order_back(u, dx)
64          d2u_fourth = second_order_centered(u, dx)
65
66          # 计算最大误差
67          errors['first_order_forward'].append(np.abs(
                du_forward[:] - df_exact(x)[:-1]).max())
68          errors['first_order_centered'].append(np.abs(
                du_centered[:] - df_exact(x)[1:-1]).max())
69          errors['second_order_back'].append(np.abs(d2u_back
                [:] - d2f_exact(x)[2:]).max())
70          errors['second_order_centered'].append(np.abs(
                d2u_fourth[:] - d2f_exact(x)[1:-1]).max())
71
72      return h_list, errors
73
74
75  def plot_errors(h_list, errors, title):
76      """绘制误差随步长的变化曲线"""
77      plt.figure(figsize=(10, 6))
78      markers = {'first_order_forward': 'o', '
            first_order_centered': 's',
79                 'second_order_back': 'D', '
                       second_order_centered': '^'}
80      for key in errors:
81          plt.loglog(h_list, errors[key], f'{markers[key]}-',
                label=key)
82      plt.loglog(h_list, h_list, 'k--', label='O(h)')
```

```
83      plt.loglog(h_list, h_list ** 2, 'k:', label='O(h²)')
84      # 设置坐标轴
85      ax = plt.gca()
86
87      # 增加x轴刻度密度
88      from matplotlib.ticker import LogLocator,
            LogFormatterSciNotation
89      ax.xaxis.set_major_locator(LogLocator(base=10, numticks
            =15))  # 主刻度
90      ax.xaxis.set_minor_locator(LogLocator(base=10, subs=np.
            arange(2, 10) * 0.1))  # 次刻度
91      ax.xaxis.set_major_formatter(LogFormatterSciNotation())
              # 科学计数法格式
92
93      # 强制显示所有步长标签
94      ax.set_xticks(h_list)
95      ax.set_xticklabels([f"{h:.2e}" for h in h_list],
            rotation=45, ha='right', fontsize=9)
96
97      # 标签和标题
98      plt.xlabel(' Δx', fontsize=12)
99      plt.ylabel('maximum error', fontsize=12)
100     plt.title(title, fontsize=14)
101
102     # 网格和图例
103     plt.grid(True, which='both', linestyle='--', alpha=0.5)
104     plt.legend(fontsize=10)
105     plt.tight_layout()
106     plt.show()
107
108 # ======================
109 # 测试函数与主程序
110 # ======================
```

```
111
112 if __name__ == "__main__":
113     # 测试函数1: 三次多项式
114     f_poly = lambda x: x ** 3
115     df_poly = lambda x: 3 * x ** 2
116     d2f_poly = lambda x: 6 * x
117
118     # 测试函数2: 正弦函数
119     k = 2
120     f_sine = lambda x: np.sin(k * x)
121     df_sine = lambda x: k * np.cos(k * x)
122     d2f_sine = lambda x: -k ** 2 * np.sin(k * x)
123
124     # 双精度分析舍入误差和截断误差
125     h_list_double_rounding, errors_double_rounding =
            compute_errors(f_poly, df_poly, d2f_poly, dtype=np.
            float64)
126     plot_errors(h_list_double_rounding,
            errors_double_rounding, "double-precision error
            analysis (f(x)=x³)")
127
128     h_list_double_truncation, errors_double_truncation =
            compute_errors(f_sine, df_sine, d2f_sine, dtype=np.
            float64)
129     plot_errors(h_list_double_truncation,
            errors_double_truncation, "double-precision error
            analysis (f(x)=sin(2x))")
130     # 单精度分析舍入误差和截断误差
131     h_list_single_rounding, errors_single_rounding =
            compute_errors(f_poly, df_poly, d2f_poly, dtype=np.
            float32)
132     plot_errors(h_list_single_rounding,
            errors_single_rounding, "single-precision error
```

```
        analysis (f(x)=x³)")
133
134     h_list_single_truncation, errors_single_truncation =
            compute_errors(f_sine, df_sine, d2f_sine, dtype=np.
            float32)
135     plot_errors(h_list_single_truncation,
            errors_single_truncation, "single-precision error
            analysis (f(x)=sin(2x))")
```

2.1 代码结构

- **数值微分模块** (4 种差分格式)
- **误差分析模块**:
  - `compute_errors()`: 误差计算
    误差计算采用最大绝对误差准则：

$$E_{\max} = \max\left(|\mathbf{u}_{\text{数值解}} - \mathbf{u}_{\text{精确解}}|\right)$$

  - `plot_errors()`: 可视化绘图
- **测试模块** (两类测试函数，分双精度和单精度绘制一共 4 幅误差图)

3 数值结果讨论以及物理解释

3.1 收敛阶验证

由代码实现的误差分析结果图如下：

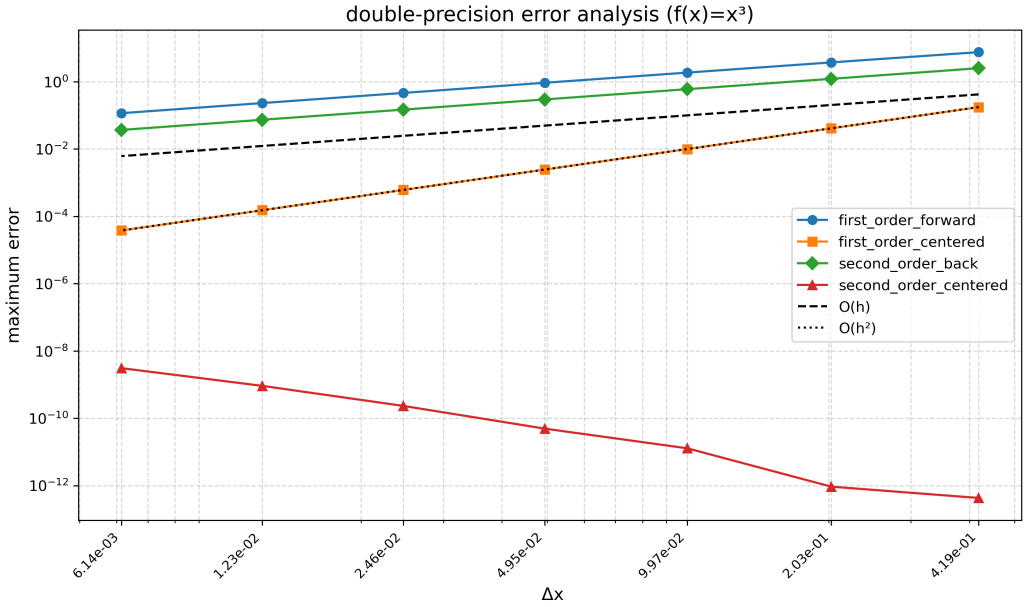

图 1: 双精度误差分析：$f(x) = x^3$

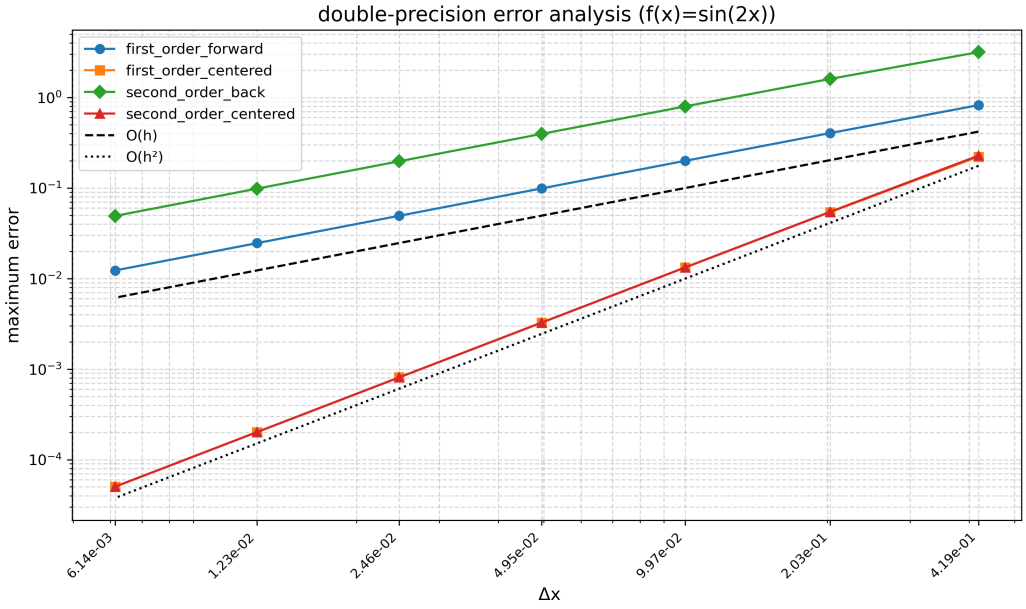

图 2: 双精度误差分析: $f(x) = \sin(2x)$

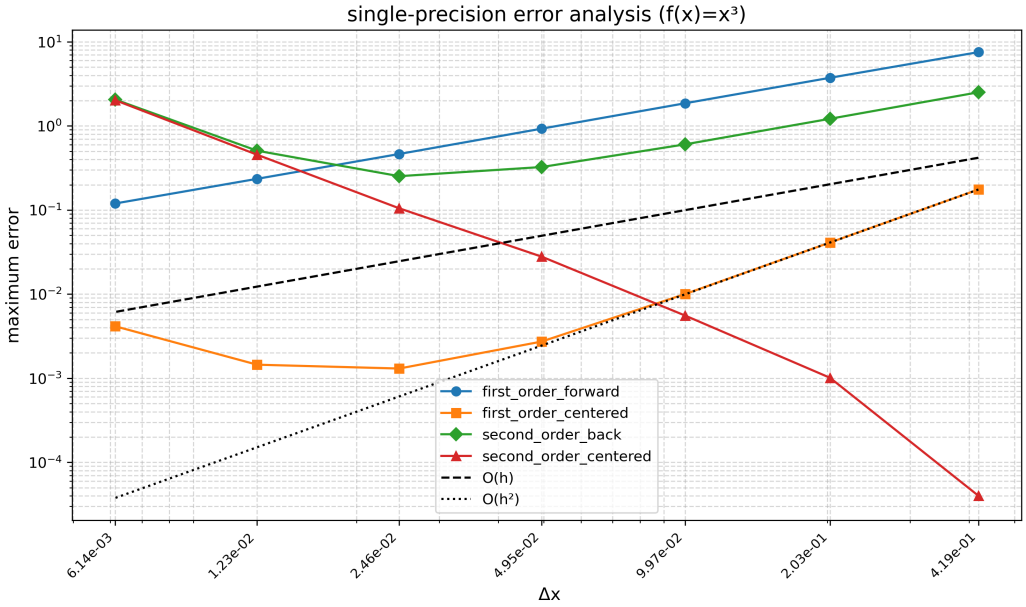

图 3: 单精度误差分析：$f(x) = x^3$

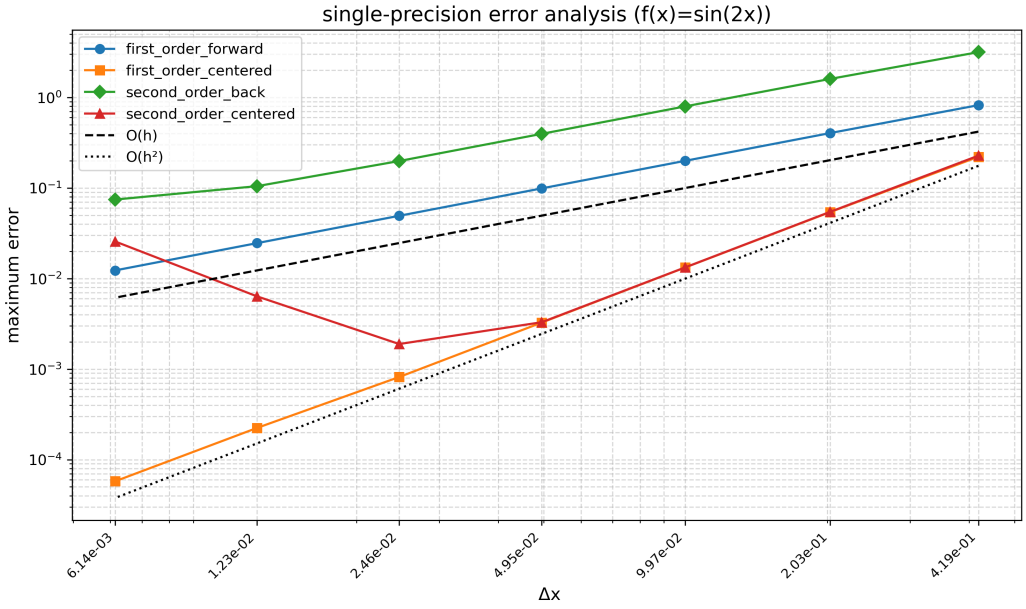

图 4: 单精度误差分析: $f(x) = \sin(2x)$

由图 1 到图 4 可以看出，双精度和单精度的误差分析结果基本一致。对一阶导数，向前差分的误差收敛阶为 $O(\Delta x)$，中心差分的误差收敛阶为 $O(\Delta x^2)$。对二阶导数，向前差分的误差收敛阶为 $O(\Delta x)$，中心差分的误差收敛阶为 $O(\Delta x^2)$。对二阶导数，向后差分的误差收敛阶为 $O(\Delta x)$，中心差分的误差收敛阶为 $O(\Delta x^2)$。

3.2 误差特性分析

由图 1 可以发现，对 $f(x) = x^3$，一阶导数的向前差分和中心差分的误差随着 Δx 的减小而减小，但是向前差分的误差始终大于中心差分的误差。这是因为此时二者的主导误差项为截断误差，向前差分的截断误差为 $O(\Delta x)$，而中心差分的截断误差为 $O(\Delta x^2)$。但是对二阶导数中心差分格式，此时没有截断误差，只有舍入误差，所以误差随着 Δx 的减小而减小。

对比图 1 和图 2，对 $f(x) = \sin(2x)$，一阶导数的两种格式和二阶导数的一阶格式的误差仍以截断误差为主，行为没有太大变化。但是对二阶导数中心差分格式，此时相比于图 1，误差的主要来源已经从舍入误差变成了截断误差。

对比图 1 和图 3，对 $f(x) = x^3$，发现一阶导数的向前差分格式误差没有太大影响，而中心差分格式的误差在 Δx 很小的时候偏离了 $O(\Delta x^2)$ 的直线，说明在单精度的情况下 Δx 所带来的舍入误差比其截断误差大，舍入误差占主导，Δx 越小舍入误差越大。同理对二阶导数的一阶格式，误差也偏离了 $O(\Delta x)$ 的直线，说明此时舍入误差占主导。对二阶导数的中心差分格式，没有截断误差只有舍入误差，相比于图 1 可以看出单精度的舍入误差比双精度的舍入误差大许多。

对比图 2 和图 4，对 $f(x) = \sin(2x)$，发现单双精度对一阶导数的两种格式和二阶导数的一阶格式的误差没有太大影响，而中心差分格式的误差在 Δx 很小的时候偏离了 $O(\Delta x^2)$ 的直线，说明在单精度的情况下 Δx 所带来的舍入误差比其截断误差大，舍入误差占主导，Δx 越小舍入误差越大。

4 AI 工具使用说明表

AI 名称	**生成代码功能**	**使用内容**
Copilot	latex 格式框架	165-188 行图片插入
Deepseek	python 绘图调整	87-106 行误差图绘制的具体参数调整
Deepseek	gitignore 文件忽略	全由 ai 添加