计算方法实验：线性方程组求解与Attention机制优化

一、实验背景

1. 线性方程组求解

线性方程组是科学与工程计算的基础，在众多领域有广泛应用。高效精确地求解大型稀疏线性方程组 $Ax = b$ 至关重要。经典方法包括直接法（如高斯消去法）和迭代法（如高斯-赛德尔法），它们在不同矩阵类型上表现不同。

2. Attention机制与FlashAttention优化

Attention机制（特别是自注意力）是深度学习（如Transformer模型）的核心，用于动态衡量序列中各部分的重要性。标准Attention计算查询（Query）与键（Key）的相似度，导致 $O(N^2)$ 的时间和空间复杂度（N 为序列长度），在 N 较大时计算和显存开销巨大。

为解决此问题，FlashAttention应运而生。它通过精巧的算法（如分块计算和在线Softmax）及硬件优化，显著降低了Attention的显存占用和计算时间，且几乎不损失精度。本实验将重点实现FlashAttention的这两个核心思想。

本实验旨在通过实现这些方法的核心逻辑，加深对计算方法优化思想的理解。

二、实验目标

1. 掌握高斯消去法和高斯-赛德尔迭代法求解线性方程组的基本原理和实现方法，理解它们在不同条件下的性能特点和适用性。
2. 在掌握标准Attention计算原理的基础上，深入理解FlashAttention如何通过分块计算和在线Softmax技术有效降低显存消耗和潜在的计算瓶颈。
3. （可选）有兴趣和能力的同学，可以通过阅读FlashAttention V1/V2论文，进一步理解其在CUDA层面的精细优化（如kernel fusion, shared memory使用，并行化策略等）是如何带来数量级的性能提升的。

三、实验环境

以下实验环境为确保代码正确运行所需的依赖与配置：

1. Python 3.9+
   推荐使用 Python 3.9 或更高版本。
2. PyTorch 2.0+
   建议安装 PyTorch 2.0 及以上版本。
3. CUDA Toolkit

CUDA 版本需与 PyTorch 和 FlashAttention 库兼容，推荐使用 CUDA 11.x 或 12.x。

4. FlashAttention 官方库
   推荐通过 PyPI 安装 `flash-attn` 包。需注意安装该库可能需要特定的编译环境及 GPU 架构（如 NVIDIA Ampere 系列）支持。若安装遇到困难，可将相关对比作为选做项或进行理论分析。
5. NumPy
6. SciPy

示例代码已在以下环境中经过测试：

- Python: 3.10.14
- PyTorch: 2.1.2
- CUDA: 11.8
- flash-attn: 2.7.3

代码仓库： `numerical_analysis_2025`

可以使用以下命令clone代码仓库：

```
git clone https://git.nju.edu.cn/mq-yuan/numerical_analysis_2025.git --branch
project_2
```

四、实验任务

任务1：求解线性方程组

1. 理论部分

- 高斯消去法 (Gaussian Elimination)：

一种求解线性方程组的直接方法。通过初等行变换将增广矩阵 $[A|b]$ 变换为上三角或行阶梯形矩阵，然后通过回代求解 x。稠密矩阵复杂度为 $O(N^3)$。

- 高斯-赛德尔迭代法 (Gauss-Seidel Iteration)：

一种求解线性方程组的迭代方法。将 A 分解为 $A = D - L - U$，迭代公式为 $x^{(k+1)} = (D - L)^{-1} \cdot (Ux^{(k)} + b)$。更新 x_i 时立即使用本轮已计算的新值。严格对角占优是其收敛的充分条件。

2. 实验内容

(1) 手动实现与对比：

- 在 `solve_le.py` 中，实现高斯消去法和高斯-赛德尔迭代法。

- 使用SciPy库求解函数（如 `scipy.linalg.solve`）作为基准。
- 生成对角占优的稀疏矩阵 `A` 和向量 `b`。
- 比较三种方法求解的精度（与真解的相对误差或相对残差）和计算时间。

(2) 参数敏感性分析：

- 测试不同矩阵大小 `N`（如 `100, 500, 2000`）和稀疏度 `density`（如 `0.01, 0.05, 0.1`）下，各方法在求解速度和精度上的表现。
- 使用表格和图表展示结果，分析参数对性能的影响。

(3) 非对角占优矩阵分析：

- 生成一个非严格对角占优的矩阵（保证有解）。
- 运行高斯-赛德尔迭代法，观察其收敛情况。
- 与对角占优情况对比，分析矩阵性质对迭代法收敛性的影响。

任务2：FlashAttention核心模块的Python实现与分析

1. 理论部分

- 标准Attention回顾：

标准Attention机制通过“查询（Query）”与“键（Key）”的相似度对“值（Value）”加权求和，实现信息聚合。自注意力中，输入序列同时作为Q、K、V。

计算流程：

1. $Scores = (QK^T)/\sqrt{d_k}$
2. $Attention\_Weights = \text{softmax}(Scores)$
3. $Output = Attention\_Weights \cdot V$

- FlashAttention核心技术详解：

FlashAttention通过分块计算（Tiling）和在线Softmax优化标准Attention，显著降低速度和显存消耗，几乎不损失精度。

1.分块计算（Tiling / Blocked Matrix Multiplication）

动机：标准Attention计算 $S = QK^T$ 产生 $N \times N$ 的巨大矩阵，当 N 很大时，无法完全放入GPU高速缓存SRAM，频繁访问HBM（GPU主显存）成为瓶颈。

方法：分块计算将 Q 和 K 分成小块（Tiles），每次只计算一小部分的分数矩阵。这样可以充分利用GPU的高速缓存，减少内存带宽的压力。

2. 在线Softmax（Online Softmax）与输出累积

动机：即使采用分块计算分数 S_{ij}，传统Softmax仍需计算出 Q_i 与所有 K_j 的全部分数后才能归一化。这意味着仍需存储 Q_i 对应的整行分数（大小为 $B_r \times N_{kv}$），这对于大的 N_{kv} 仍是挑战。在线Softmax解决了此问题。

方法：在线Softmax以流式方式处理每个查询。对于 Q 中的某一行 q_r，当其与 K 的各块 K_j 计算部分分数时，会动态维护和更新该行 q_r 的Softmax统计量：当前最大分数值 m_r 和Softmax分母的累积项 l_r。同时，输出 o_r 也会相应更新。

2. 实验内容

(1) 手动实现与验证：

- 在 `custom_flashatten.py` 中，使用Python和PyTorch张量操作，实现函数 `custom_tiled_attention_NO_ONLINE_softmax`，该函数模拟FlashAttention的分块计算。
- 阅读并理解同时实现了分块计算和在线softmax的函数 `custom_flash_attention_forward`。
- 在flash_atten_backward.py中实现 `flash_attention_backward`，该函数实现了分块计算的反向传播。

(2) 性能对比分析：

- **测试参数**：输入序列长度 `N`（例如 `512, 1024, 2048`）。`K_tile_size`（例如 `64, 128, 256`）。`Q_tile_size`
- **评估指标**：前向传播时间。反向传播时间。峰值GPU显存占用（若在GPU上运行）。峰值内存占用（若在CPU上运行）。
- **分析**：

- 验证自定义Attention（有/无在线Softmax）与标准Attention的数值一致性（前向和梯度）。

- 对比不同Attention实现（标准PyTorch、官方FlashAttention、自定义版本）在不同序列长度和 `block_size` 下的前向/反向时间及显存/内存占用。

- 对比你的实现与带在线Softmax的版本，分析**在线Softmax对降低峰值中间张量大小的贡献**。你的实现的峰值显存需求（理论上）应介于标准Attention和带在线Softmax的自定义版本之间，同时精度与标准Attention应更接近。

- 若使用CPU版本，分析自定义Flash计算和标准Attention计算的运行速度和内存差异，关注随序列长度N增加，三种方法内存占用的变化。

- 运行flash_attention_backward.py，验证反向传播的数值正确性。

(3)（可选）深入探索：

- 阅读FlashAttention V1 和 V2 的论文。浏览官方FlashAttention库的源码结构。
- 思考并分析论文中提到的除了分块和在线Softmax之外的其他关键技术（如kernel fusion, shared memory的优化使用, work scheduling, recomputation等）是如何进一步贡献于其极致的速度和显存效率的。

五、实验报告要求

1. **内容**：整理代码、结果图表，撰写实验报告。报告应包含：

- (a) **实现过程**：简述核心算法思路，附关键代码及解释。
- (b) **结果分析与讨论**：
  - **任务1**：对比高斯消去、高斯-赛德尔和SciPy库在不同参数下的求解速度、迭代次数、精度。分析稀疏性、对角占优性的影响。
  - **任务2**：对比不同实现的前向和反向传播时间、显存占用。分析分块计算和在线Softmax的贡献。
- (d) **（可选）思考与改进**：实验的不足或改进建议。

2. **提交要求**：

- (a) **提交格式**：按照课程主页要求提交单个压缩包，命名格式为：
  `{姓名}_{学号}_{实验名称}_v{版本号}.zip`
  例如：`张三_12345678_Exp2_v1.zip`
- (b) **压缩包结构实例**：

```
张三_12345678_Exp2_v1.zip
│
├── 张三_12345678_Exp2.pdf   (实验报告)
│
└── scripts/                      (所有可运行的Python代码)
    ├── sovle_le.py               (任务1代码)
    ├── custom_flashatten.py      (任务2自定义Python Attention实现)
    ├── atten_model.py            (任务2各种Attention模块定义)
    ├── main_test.py              (任务2主运行和测试脚本)
    └── (其他可能用到的辅助脚本或数据文件，若有)
```

祝实验顺利！