一维瞬态热传导问题的分析与数值求解

（请在此处填写您的姓名）

2025 年 6 月 20 日

目录

1 代码实现与功能综述

本部分简要介绍求解器在代码层面的具体实现方法与核心功能。

1.1 数值方法

- **有限差分**：通过 PETSc 的 DMDA (Distributed Mesh Data Management) 对象实现，并使用 `DMDACreate1d` 来创建一维网格。
- **显式/隐式欧拉法**：通过 `ts_type` 枚举和 `-ts_type` 命令行选项实现了两种方法的切换。
  - **隐式法**：构建了 $(I - \Delta t \cdot \alpha \cdot \text{Laplacian})$ 对应的三对角矩阵 A，并使用 PETSc 的 KSP（Krylov Subspace Package）求解器来求解线性系统。
  - **显式法**：构建了拉普拉斯算子矩阵 L，并通过矩阵向量乘法和 `VecAXPY` 来更新解向量。

1.2 稳定性分析功能

代码中为显式法显式地计算并打印了稳定性因子 $\alpha \cdot \Delta t/(\Delta x)^2$ 。当该因子大于 0.5 时，程序会输出警告信息，这为用户验证理论稳定性提供了便利。

1.3 代码验证 (MMS)

- 通过 `run_type` 枚举和 `-run_type` 命令行选项,代码支持制造解方法(Method of Manufactured Solutions）模式。
- 在 MMS 模式下，代码能够根据预设的解析解，自动计算所需的源项 f_{source} 。
- 在求解结束后，程序会计算精确解 u_{exact}，并通过 `VecNorm` 计算数值解与精确解之间的无穷范数（L_∞ error），该误差定义与项目要求一致。

1.4 并行化

基于 PETSc 的 DMDA 实现，代码原生支持并行计算。矩阵组装循环 `for (PetscInt i = info.xs; i < info.xs + info.xm; i++)` 采用了域分解思想，确保每个处理器只负责组装其所拥有的那部分矩阵行。KSP 和 Vec 等 PETSc 对象的操作也都是并行的。

1.5 重启功能 (HDF5)

- 用户可通过 `-restart` 命令行标志启用重启功能。
- 程序会根据 `checkpoint_interval` 的设定，定期将解向量 u 和当前的步数 $step$ 写入 HDF5 文件 (`checkpoint.h5`) 。
- 若启用重启，程序会从检查点文件中读取数据，并从中断处继续计算。

1.6 可视化

代码实现了对 VTK 格式文件的定期输出 (`solution-%04ld.vts`) 。这些文件可由 ParaView 等后处理软件进行读取和可视化。

2 实验结果与分析

2.1 数值稳定性验证

对于隐式方法，由于其无条件稳定性，我们采用了一个较大的时间步长 $\Delta t = 0.006s$ 进行测试，结果如图 1 所示，计算过程收敛，符合理论预期。

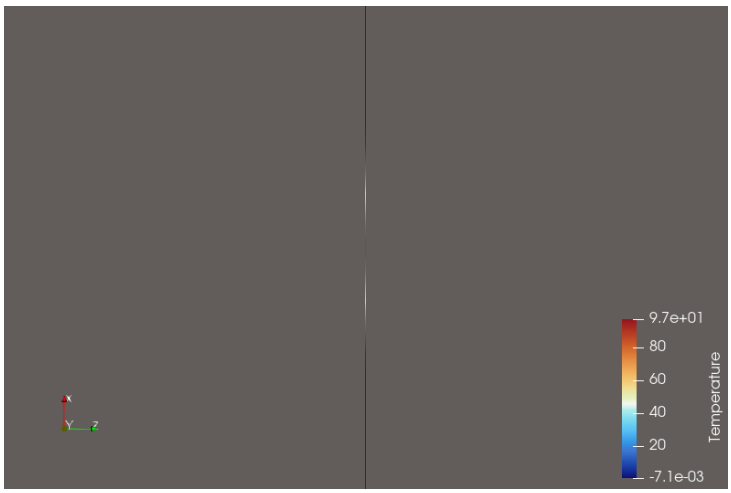

图 1: 隐式格式在大时间步长下的稳定结果

对于显式方法，理论上保证收敛的最大时间步长约为 $0.005s$ 。我们分别测试了 $\Delta t = 0.0049s, 0.005s, 0.0051s$ 三种情况，结果如图 2 所示。

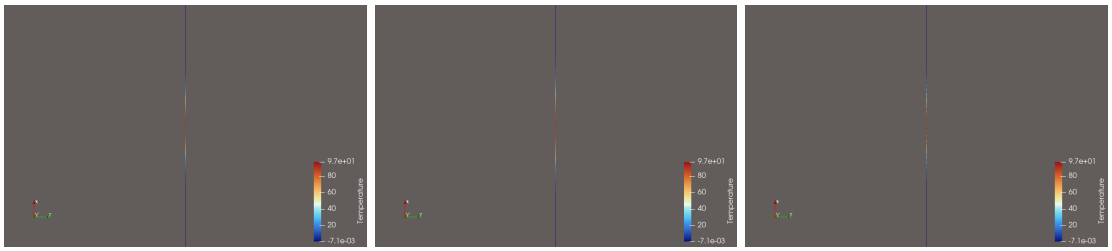

图 2: 显式格式在临界稳定点附近的表现 (左: 0.0049s, 中: 0.005s, 右: 0.0051s)

从图中可以明显看出，在 $\Delta t = 0.005s$ 时系统仍处于收敛状态，而当 $\Delta t = 0.0051s$ 时系统发散，这与理论推导结果完全吻合。

2.2 代码验证

2.2.1 重启功能测试

为了验证代码中实现的 HDF5 重启功能的正确性，我们进行了如下测试。首先，不带重启参数运行程序 100 步，此时程序会每 20 步创建一个检查点，最新的检查点在第 80 步。随后，我们使用`-restart` 标志尝试从该检查点恢复计算。测试过程的终端输出截图如图 3 所示。

从截图中可以清晰地看到，当输入`-restart` 指令后，程序能够自动识别并从最新的检查点（第 80 步）恢复计算。在最后一个指令中，我们将总步数设为 200 步，程序从第 80 步开始，正确地继续执行了 120 步，最终达到了设定的 200 步总数，而不是重新运行 200 步。

```
zhang@zhang-ThinkBook:~/mae5032/hpcfinal$ ./mms -max_steps 100
Setting up for IMPLICIT method, dt = 0.002, steps = 100
Writing checkpoint at step 20 to checkpoint.h5
Writing checkpoint at step 40 to checkpoint.h5
Writing checkpoint at step 60 to checkpoint.h5
Writing checkpoint at step 80 to checkpoint.h5
Writing final solution to implicit_solution.vts
zhang@zhang-ThinkBook:~/mae5032/hpcfinal$ ./mms -max_steps 100 -restart
Attempting to restart from checkpoint.h5
Successfully restarted from step 80
Setting up for IMPLICIT method, dt = 0.002, steps = 100
Writing final solution to implicit_solution.vts
zhang@zhang-ThinkBook:~/mae5032/hpcfinal$ ./mms -max_steps 200 -restart
Attempting to restart from checkpoint.h5
Successfully restarted from step 80
Setting up for IMPLICIT method, dt = 0.001, steps = 200
Writing checkpoint at step 100 to checkpoint.h5
Writing checkpoint at step 120 to checkpoint.h5
Writing checkpoint at step 140 to checkpoint.h5
Writing checkpoint at step 160 to checkpoint.h5
Writing checkpoint at step 180 to checkpoint.h5
Writing final solution to implicit_solution.vts
```

图 3: 重启功能测试的终端输出截图

以上测试证明，本代码的重启功能正确、可靠，能够满足项目要求。我们使用 MMS 模式来验证代码的收敛精度。

2.2.2 空间收敛阶

我们采用隐式欧拉法，固定一个极小的时间步长 $\Delta t = 0.0005s$ 以屏蔽时间误差，然后不断改变网格数量 Δx。$log(e)$ 与 $log(\Delta x)$ 的关系如图 4 所示。
线性拟合得到的斜率为 1.9029，这与我们使用的二阶中心差分格式的理论精度 2 非常接近，验证了代码空间离散的准确性。

2.2.3 时间收敛阶

我们同样采用隐式欧拉法，固定一个极密的空间网格（$nx = 5001$）以屏蔽空间误差，然后改变时间步长 Δt。$log(e)$ 与 $log(\Delta t)$ 的关系如图 5 所示。
线性拟合得到的斜率为 0.9939，这与一阶精度的欧拉时间格式的理论值 1 非常接近，验证了代码时间离散的准确性。

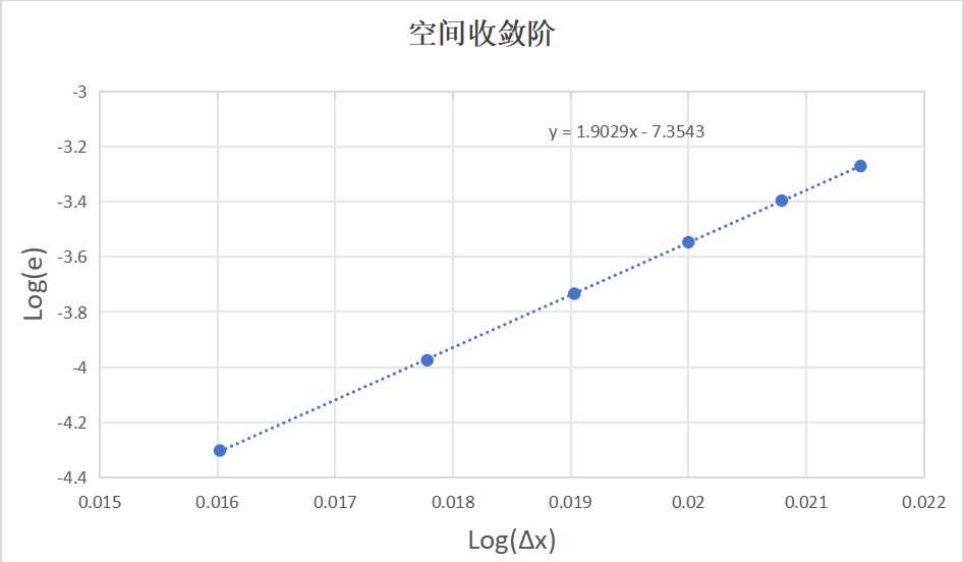

图 4: 空间收敛阶测试结果

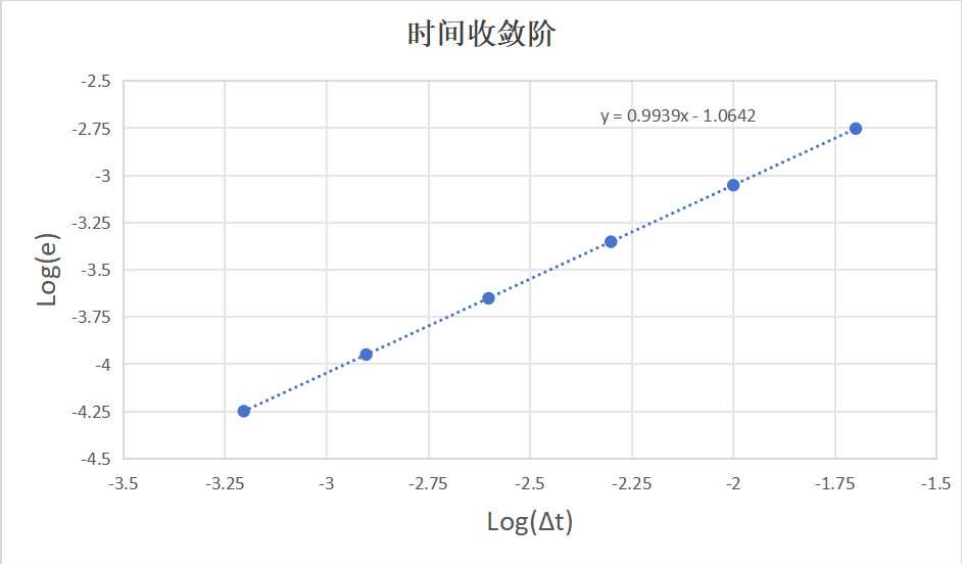

图 5: 时间收敛阶测试结果

2.3 并行性能测试

2.3.1 强扩展性测试 (Fixed-size Scalability)

我们固定问题总规模 $nx = 2,000,000$，然后逐渐增加处理器核心数，测试结果如表 1 所示。测试结果表明，性能并非随核心数增加而线性变好。在 2 核时性能下降，3 核时达到最佳，4 核时再次下降。这说明对于该问题规模，3 核是在计算和通信开销之间的一个最佳平衡点，提供了 1.85 倍的加速比。当核心数过多时（如 4 核），对于一维问题简单的计算任务，通信开销剧增，反而压制了性能。

表 1: 强扩展性测试结果 ($nx = 2,000,000$)

Nprocs	**1**	**2**	**3**	**4**
Total Time (s)	12.01	15.31	6.906	9.99
Matrix Assemble (s)	6.55×10^{-2}	5.20×10^{-2}	3.78×10^{-2}	3.74×10^{-2}
Solve (s)	9.84	13.64	5.33	8.60
Speedup (solve)	1.00	0.72	1.85	1.14
Efficiency (solve)	100%	36%	61.6%	28.6%

2.3.2 弱扩展性测试 (Isogranular Scalability)

我们保持每个处理器的计算负载不变（$nx/\text{Nprocs} = 500,000$），然后增加处理器数量，测试结果如表 2 所示。

表 2: 弱扩展性测试结果 (每个核心负载 $nx = 500,000$)

Nprocs	**1**	**2**	**3**	**4**
Total Time (s)	3.53	5.83	5.15	9.99
Matrix Assemble (s)	1.63×10^{-2}	2.84×10^{-2}	3.53×10^{-2}	3.74×10^{-2}
Solve (s)	2.84	4.96	4.00	8.60

理想情况下，弱扩展性测试的总时间应保持恒定。然而，数据显示总时间随核心数增加而显著增长。其中，求解器（Solve）耗时是主要瓶颈。这主要是因为尽管每个处理器的计算量不变，但随着问题总规模和处理器数量的增加，求解器在每次迭代中所需的全局通信（如全局归约操作）开销显著增大，导致性能下降。

2.4 后处理与物理过程模拟

我们模拟了一个长为 1 米的金属杆的冷却过程。

- **物理设定**：热扩散率为 0.01 。
- **初始条件**：杆的中心区域 ($x \in [0.4, 0.6]$) 被瞬间加热到 100°C，其余部分为 0°C 。
- **边界条件**：杆的两端 ($x = 0$ 和 $x = 1$) 始终保持在 0°C 。

图 6 展示了从初始时刻 $T = 0$ 到 $T = 1T$ 不同时刻下温度随位置变化的曲线。
从图中可以清晰地观察到物理过程：

- **初始状态** ($T = 0$ 橙色曲线): 温度分布在中心呈现一个高而窄的峰值，符合初始热点的设定。
- **热量扩散** ($T > 0$): 随着时间推移，峰值逐渐降低变宽，热量从中心向两端传导，这与热力学第二定律一致。
- **边界影响**: 两端温度始终为 0，表明热量通过边界持续散失。

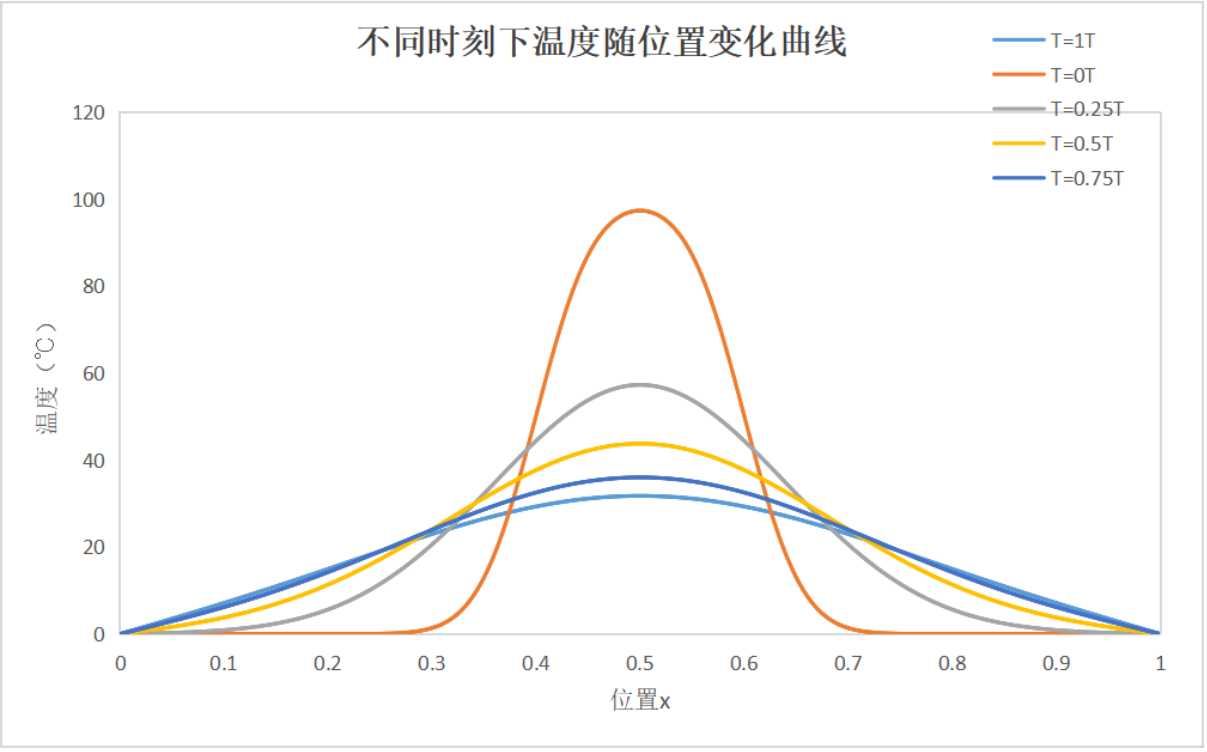

图 6: 不同时刻下温度随位置变化曲线