高性能并行计算第一次实验报告

李子镔，鲁曦

2022 年 3 月 20 日

1 一点说明

本项目由鲁曦与李子镔共同完成。

2 任务描述 & 任务分解

根据要求，本次我们需要根据.json 文件中的输入信息，在正方形区域（可能去掉一个圆）上，通过有限差分法求解给定边界条件下的泊松方程的数值解。具体地，我们把本次实验任务分解为以下几个步骤：

1. 定义所需要的 json 文件属性，明确输入格式。
2. 设计网格类，定义必要的数据类型。用于记录 json 输入文件给出的数据及要求，以及进行后续的处理。
3. 根据边界值的类型 (Dirichlet，Neumann, 或者混合类型)，给出相应的线性方程组。
4. 求解方程组，将求解结果保存到网格中。
5. 将网格中的数据输出到文件中。
6. 在 Python 中对输出文件中的数值结果进行处理，生成图像。

3 代码简介

3.1 第三方库依赖

本项目使用的第三方库有：jsoncpp，Eigen。关于 jsoncpp 此处不再赘述，下面简单介绍 Eigen 库的安装和使用。

Eigen 是符合 BLAS/LAPACK API 标准的一个高性能线性代数纯头文件库。在 Ubuntu20.04 下安装 Eigen，只需要在命令行中输入如下安装命令：**sudo apt install libeigen3-dev**。在使用时，根据使用需要，添加 Eigen 库中相应的头文件 eigen3/Eigen/*** 即可。

3.2 代码结构

在我们的项目中，根据 OOP 的思想，我们自定义了网格节点类，网格类，以及表示边界值类型，节点类型的枚举类。并且，本项目中的大部分任务都由封装在网格类中的函数完成。

项目主要结构如下：

- **Grid.h**：定义了网格相关的一些类：
  - *GridNodeType*：用于指示每一个格点在要求解的区域的内部、外部、正方形边界还是圆形边界上。
  - *GridBdryType*：用于指示网格的边条件界类型，包括纯 Dirichlet 边界，方形 Dirichlet 边界圆形 Neumann 边界，方形 Neumann 边界圆形 Dirichlet 边界、纯 Neumann 边界。
  - *GridNode*：存储网格节点，包括函数值和边界类型两个信息。
  - *Grid*：存储网格所有信息，包括用于存储所有节点的二维数组，边界条件类型，网格的空间步长，以及是否要挖掉圆及其圆心半径。
- **Grid.cpp**：实现了网格所需的功能，包括根据 json 输入来构建网格，异常处理，检查网格合法性，构建要解的线性方程组的系数矩阵。
- **Array2D.h**：定义一个模板类二维数组作为小工具。
- **main.cpp**：程序主要流程，包括读取 json 文件，定义三个测试函数，调用 eigen 库求线性解方程组，将结果写入 txt 文件。

- **run.sh**：linux 脚本文件，可以调用 cmake 编译并运行。
- **input.json**：包含几个程序输入参数：网格的空间步长、挖掉的圆的圆心半径 (null 为不挖掉)、边值条件类型 (可选参数为 Dirihlet、Neumann、DirihleNeumann、NeumannDirihlet)、测试函数编号 (可选参数为 0、1、2) 以及输出结果的路径。
- **visualize.ipynb**：python 脚本，读取输出结果，作图并于理论解进行比较。
- **run_all.ipynb**：pyhton 脚本，调用系统命令一次性跑出每一种条件组合的结果。

3.3 编译说明

本项目使用 cmake 3.16.3 版本生成构建文件。如果希望使用 Ninja 生成器并行编译代码，可以在生成构建文件命令中加入选项-G ninja。否则默认将在./build 文件夹中生成 Makefile。通过执行./run.sh 脚本文件，可以在 Ubuntu20.04 下一键完成整个编译流程。最终生成的可执行二进制文件将在./build 文件夹中找到。

4 数值实验

我们对网格密度为 8，16，32，64（对应的节点数分别为 9×9、17×17、33×33、65×65），方形区域 Ω 的两种边界条件 (Dirichlet 与 Neumann) 和去掉一个圆的方形区域 $\Omega-D$ 的四种边界条件 (纯 Dirichlet 边界，方形 Dirichlet 边界圆形 Neumann 边界，方形 Neumann 边界圆形 Dirichlet 边界、纯 Neumann 边界)，与三个不同的测试函数

$$
\begin{aligned}
u_1(x,y) &= e^{y+\sin(x)},\\
u_2(x,y) &= (x-0.5)^2+(y-0.5)^2,\\
u_3(x,y) &= \sin(2\pi x)\sin(2\pi y),
\end{aligned}
$$

的所有组合进行了数值实验。我们将以作业要求中的测试函数（第一个测试函数）为例，对几种情况分类展示我们的数值结果。

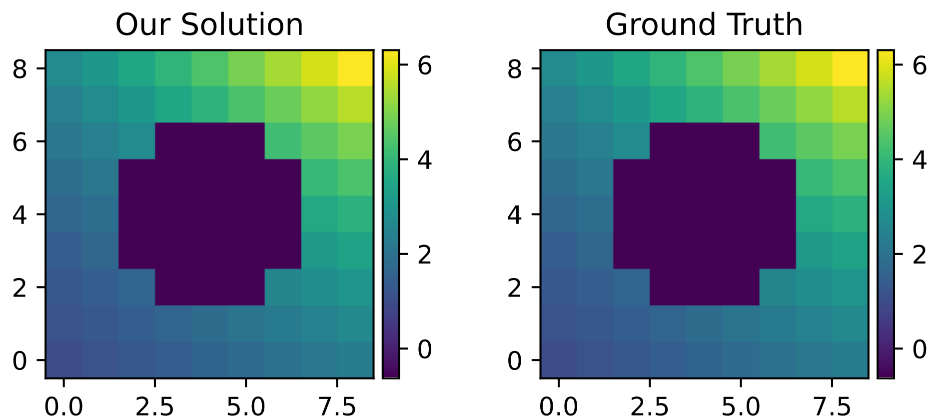

图 1: 测试函数 u_1，Dirichlet 边界条件，$n = 8$.

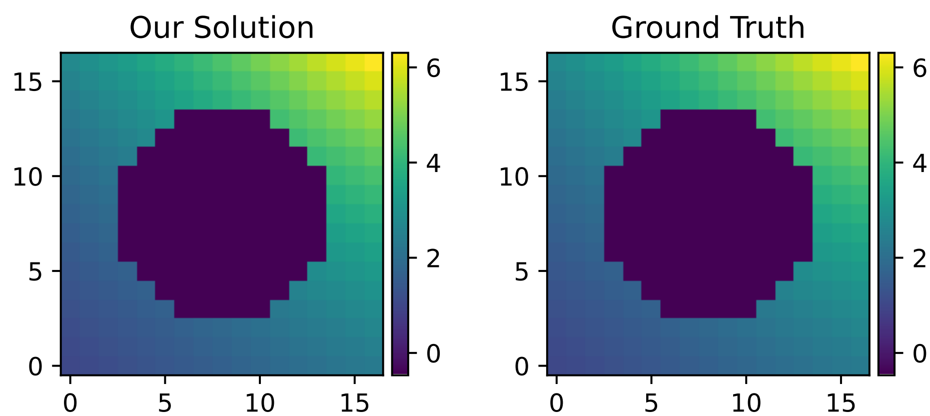

图 2: 测试函数 u_1，Dirichlet 边界条件，$n = 16$.

4.1 Dirichlet 边界条件下的求解

我们将网格点划分为四种类型：内部点，外部点，方形边界点，圆形边界点（其他边界条件下也作相同的分类）。标记剔除外部点后，我们只需要考虑其他三类点。对于内部点，我们采取标准的五点差分法近似 Laplace 算子。由于 Dirichlet 边界条件的特殊性，边界点上的数值已经给出，因此不需要考虑为其单独设计差分格式。

求解结果如图1-4、7-9，颜色方块代表了方块中心的数值解或者真实解的数值大小，左边和右边分别是数值解与真实解的离散网格图像。

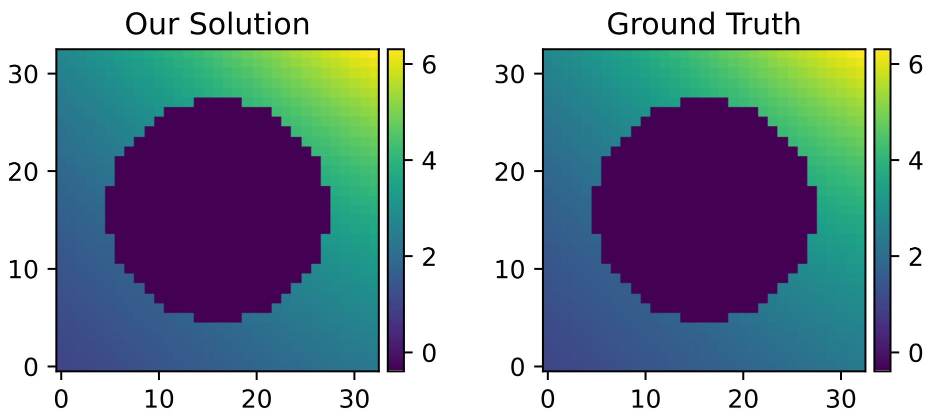

图 3: 测试函数 u_1，Dirichlet 边界条件，$n = 32$.

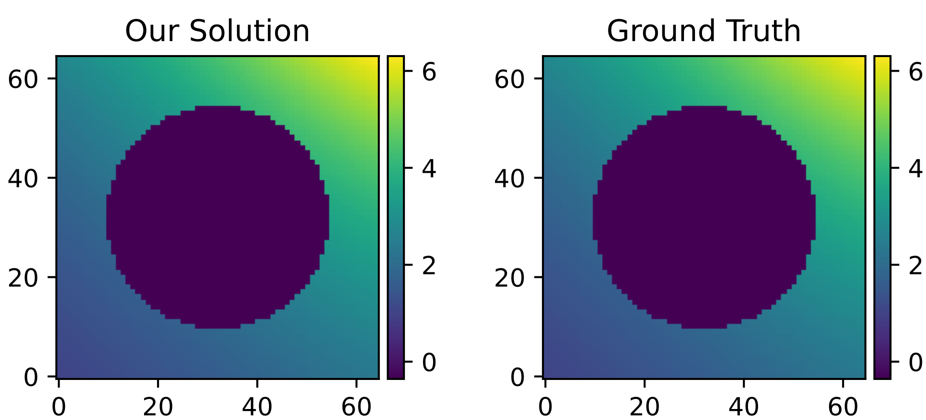

图 4: 测试函数 u_1，Dirichlet 边界条件，$n = 64$.

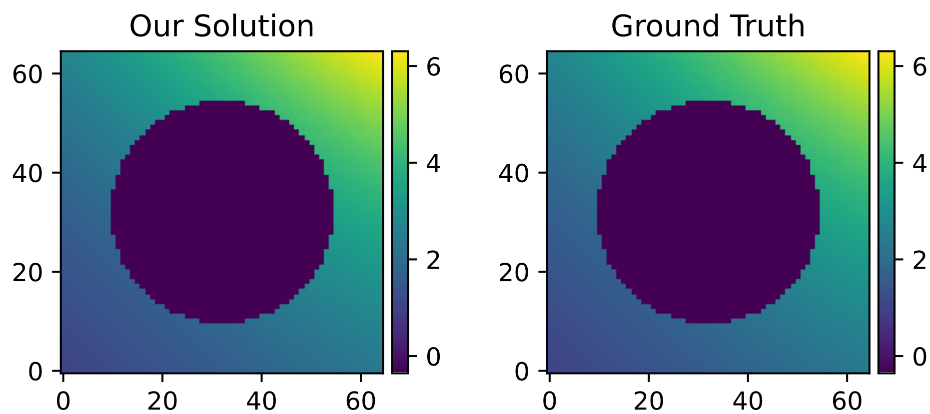

图 5: 测试函数 u_1，Dirichlet 方形边界加 Neumann 圆形边界，$n = 64$.

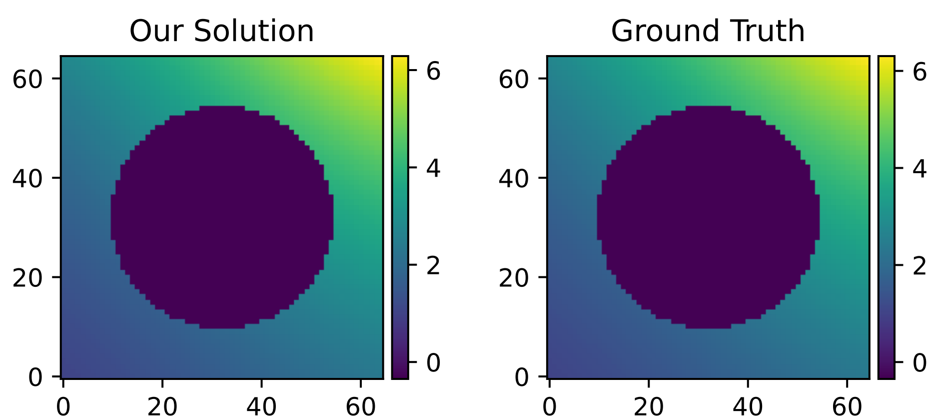

图 6: 测试函数 u_1，Dirichlet 圆形边界加 Neumann 方形边界，$n = 64$.

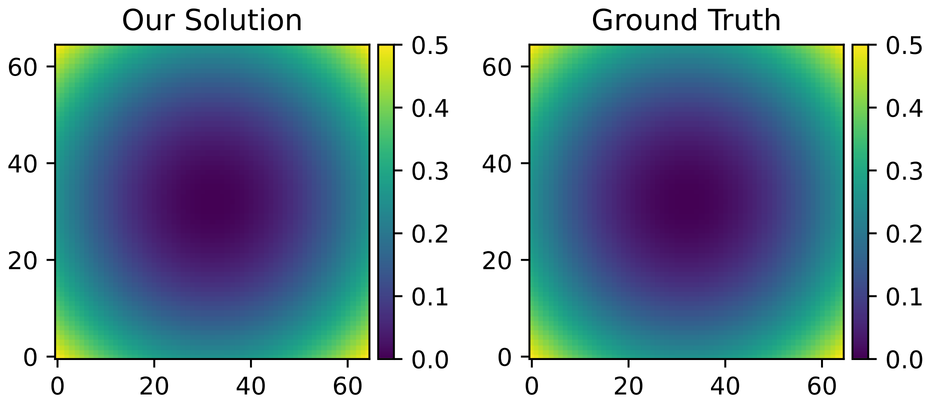

图 7: 测试函数 u_1，不挖掉圆，Dirichlet 边界条件，$n = 64$.

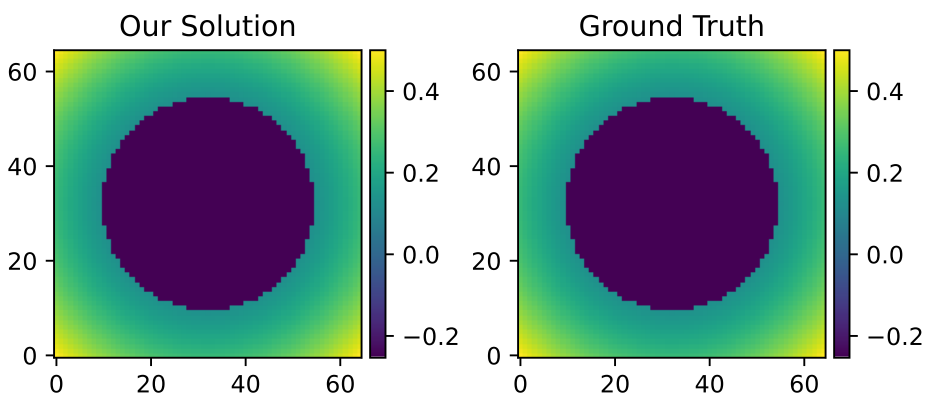

图 8: 测试函数 u_2，Dirichlet 边界条件，$n = 64$.

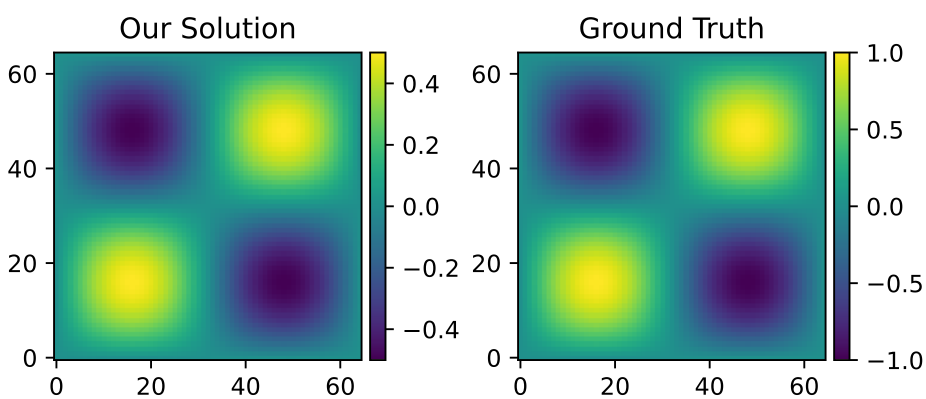

图 9: 测试函数 u_3，不挖掉圆，Dirichlet 边界条件，$n = 64$.

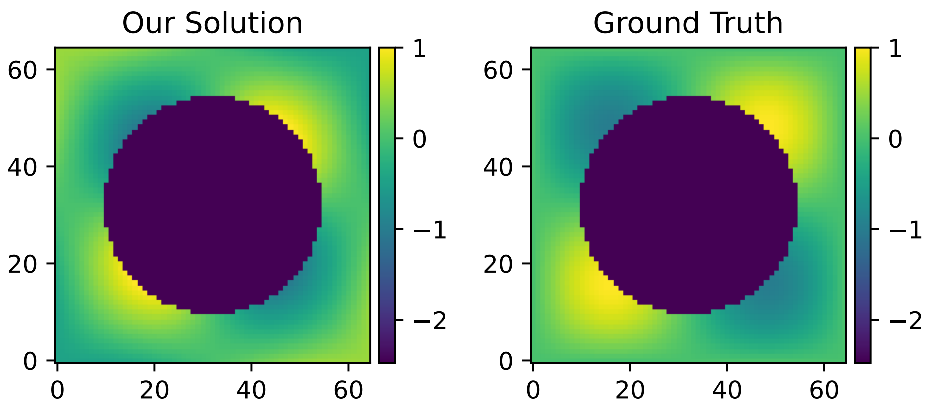

图 10: 测试函数 u_3，Neumann 方形边界加 Dirichlet 圆形边界，$n = 64$.

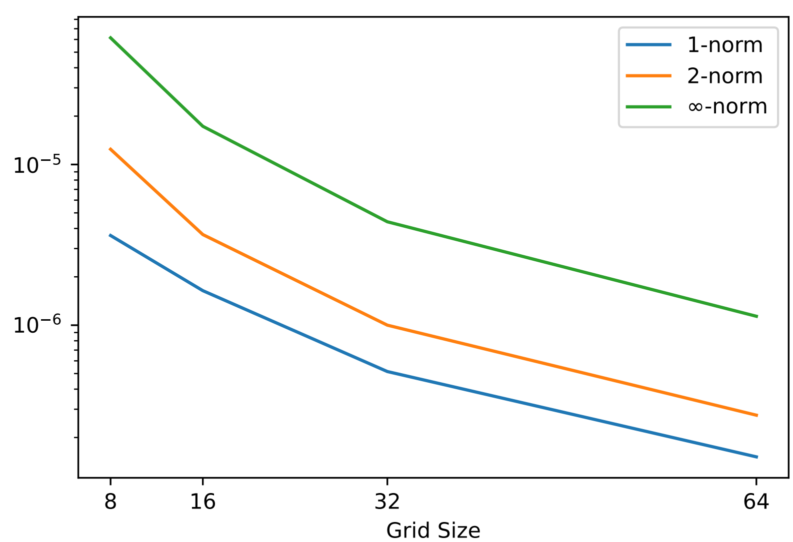

图 11: 测试函数 u_1，Dirichlet 边界的误差范数与 n 的关系.

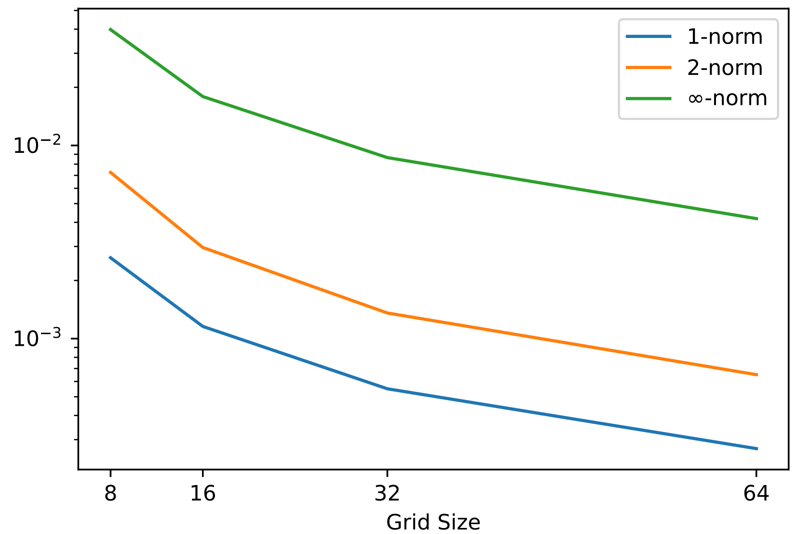

图 12: 测试函数 u_1，Dirichlet 方形边界加 Neumann 圆形边界的误差范数与 n 的关系.

4.2 混合边界条件下的求解

对于混合边界条件，我们考虑了两种类型：一种是方形边界采取 Neumann 边界条件，圆形边界采取 Dilichlet 边界条件；另一种是方形边界采取 Dirichlet 边界条件，圆形边界采取 Neumann 边界条件。无论是哪一种类型，如果边界点直接给定了解的数值，则不需要考虑其差分格式。如果只给了外法向方向的导数，则需要进行特别的处理。由于区域的特殊性，节点外法向 $\mathbf{n}$ 容易给出。方形区域边界点 $\mathbf{P}_{rect}$ 上，外法向 $\mathbf{n}=(n_1,n_2)$ 必然与坐标轴垂直，将从 $\mathbf{P}_{rect}$ 出发沿 $\mathbf{n}$ 方向最近的一个节点记为 $\mathbf{Q}_{rect}$。在我们设计的算法中，如果不相交或者 $\mathbf{P}_{rect},\mathbf{Q}_{rect}$ 的距离大于一个步长 h，则程序会抛出异常。如果正常执行，则我们利用如下差分格式估计 $\mathbf{P}$ 处的方向导数：

$$\frac{u(\mathbf{P_{rect}})-u(\mathbf{Q}_{rect})}{h}.$$

对于圆形区域的边界上的网格节点 $\mathbf{P}_{circ}$，记网格节点的两个坐标轴方向为 $\mathbf{e}_x,\mathbf{e}_y$。如果内积 $\langle\mathbf{n},e_x\rangle\geq 0$，则我们记 x 方向上的相邻节点 $\mathbf{Q}^{(1)}_{rect}=\mathbf{P}_{rect}-h\mathbf{e}_x$，否则则记 $\mathbf{Q}^{(1)}_{rect}=\mathbf{P}_{rect}+h\mathbf{e}_x$。类似地，我们可以得到 y 方向

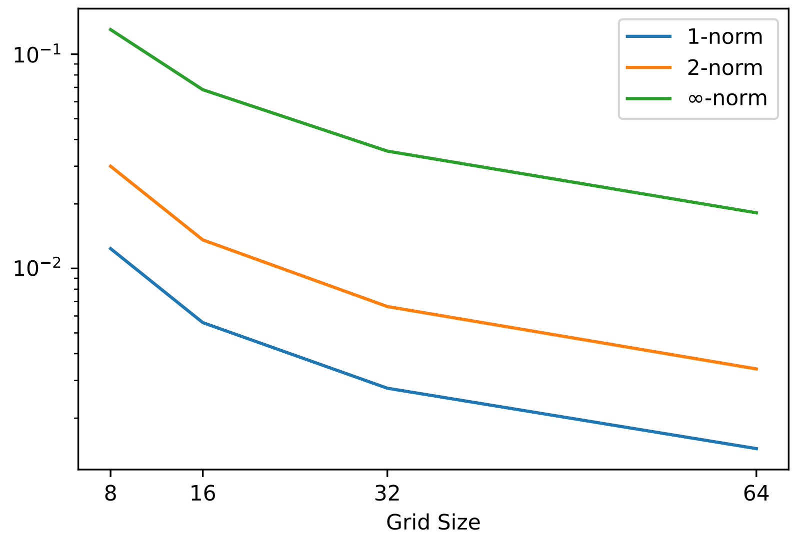

图 13: 测试函数 u_1，Neumann 方形边界加 Dirichlet 圆形边界的误差范数与 n 的关系.

上的相邻节点 $\mathbf{Q}_{rect}^{(2)}$。同样地，如果 $\mathbf{Q}_{rect}^{(i)}$ 不存在或者与 $\mathbf{P}$ 的距离超过一个步长，我们的程序将抛出异常。在正常运行的情况下，根据 $\frac{\partial u}{\partial \nu} = \langle Du, \nu \rangle$，我们在 $\mathbf{P}_{rect}$ 处采用如下差分格式：

$$\frac{u(\mathbf{Q}_{rect}^{(1)}) - u(\mathbf{P}_{rect})}{h} \cdot |n_1| + \frac{u(\mathbf{Q}_{rect}^{(2)}) - u(\mathbf{P}_{rect})}{h} \cdot |n_2|.$$

求解结果如图5、6和10。

4.3 Neumann 边界条件下的求解

在 Neumann 边界条件下，求解器不能正常工作，数值实验结果如图14和图10所示。

这是因为 Neumann 边界条件本身就不能唯一确定方程组的解，以致于从根本上导致了，在离散差分格式中，其差分矩阵理论上是不可逆的。这一点在数值实验中体现为，要么差分矩阵不可逆，根本不能求解；要么由于机器精度导致差分矩阵接近奇异，条件数很大，使得求解错误。

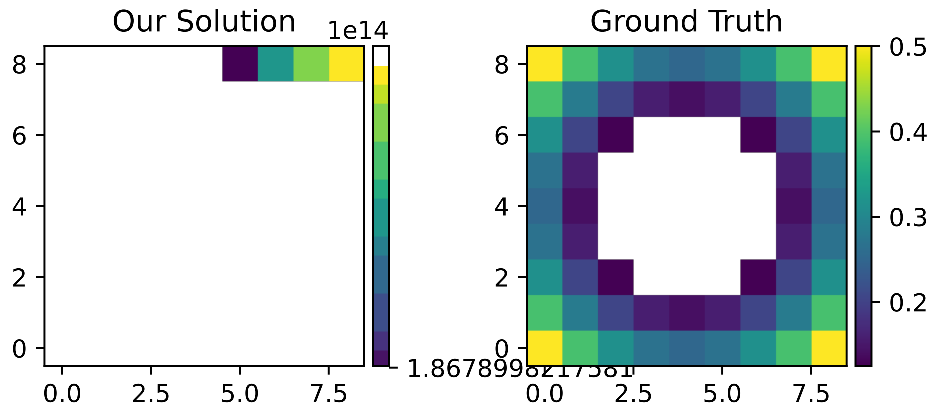

图 14: 测试函数 u_2，Neumann 边界条件，$n = 8$。左图中白色区域是 NaN。

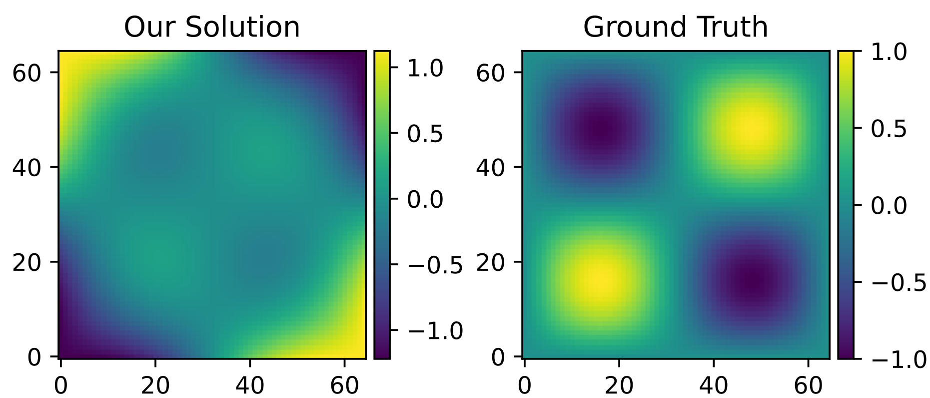

图 15: 测试函数 u_3，不挖掉圆，Neumann 边界条件，$n = 64$。