数值分析项目一作业报告

刘陈若 3200104872
信息与计算科学 2001

1 程序编译和运行说明

本次项目作业采用 Makefile 文件对编译进行统一管理。具体地，在 Makefile 所在目录下输入`make run` 即可完成编译，得到`test.cpp`的可执行文件`test`，对其进行运行即可得到程序结果。

需要说明两点：首先，本项目作业使用 eigen3 进行线性方程组求解，并使用 json file 进行参数输入，它们分别以`#include <eigen3/Eigen/...>`和`#include <jsoncpp/json/json.h>`的形式被调用，因此在编译时需要保持相应的文件关系；其次，在使用 json file 进行参数输入时，部分参数可能需要修改，参数的含义以及具体的修改方式都将在下文中详细给出。

2 程序运行结果及简要分析

2.1 FD.h 设计思路

在`spline.h`头文件中，一共设计了`Function`，`Circle`和`FD`三个基类，派生出不同的 class 以满足项目要求。以下将分别展开设计思路叙述。

2.1.1 class Function

`Function`类一共设计了`operator()`（括号重载），`diff_x`,`diff_y`和`Laplace`四个虚函数，用于返回给定二元函数的值，其一阶偏导数的值以及-Laplace 算子作用后的值。`Function`类主要功能是：存储待求解函数 u 的真实表达式以便输入边界条件并进行误差分析。

2.1.2 class Circle

这个类实现了一个圆的基本功能，包括圆心和半径的存储，计算点 P 处的与圆心连线的单位外向量，计算直线与圆的交点（如果存在），以及返回圆心和半径。通过这个类，可以方便地进行圆形几何的计算和处理。其中，函数`get_normal`可以用于计算单位外法方向，函数`get_intersection`可以用于计算直线与圆的交点，函数`get_center`和函数`get_radius`可以用于获取圆心和半径的值。`Circle`类主要功能是存储项目要求中不规则区域的圆。

2.1.3 class FD

FD类主要功能是实现基于 FD method 的微分方程数值解的求解。FD类一共设计了solve, get_result和get_errornorm三个虚函数，分别用于进行 FD 算法，返回计算得到的格点处的数值解以及返回误差的 1,2 以及无穷范数。通过继承这个类，可以创建各种具体的数值解算法，例如针对单位正方形上的偏微分方程和针对单位正方形除去一个圆的偏微分方程。

- class FD_regular：该派生类用于支持单位正方形上的微分方程的 Dirichelet，Neumann 和 mixed 三种边界条件求解。FD_regular(Function &_u, double _h, int _condition)构造函数，用于接收真实函数值、网格宽度和边界条件等参数。

solve函数求解 $AU = F$ 的线性方程组，其中 A 和 F 分别为系数矩阵和右侧向量。我们加入边界作为 ghost point，并且按从左到右，从下到上的顺序给格点标序号以便后续计算。该函数在满足不同的边界条件下，使用不同的方法来构造 A 和 F: 当condition=1时，表示 Dirichlet 边界条件。此时，在矩阵 A 的内部区域中，使用五点差分公式来近似求解二阶偏微分方程。在边界上，将 u 的值作为已知量，构造出系数矩阵 A 和右侧向量 F；当condition=2时，表示 Neumann 边界条件。此时，在矩阵 A 的内部区域中，仍然使用五点差分公式来近似求解二阶偏微分方程。但是在边界上，我们根据 FD formula 得到的二阶精度的导数来作为导数值（左右边界为关于 x 的偏导数，上下边界为关于 y 的偏导数）的估计。需要注意的是，左右两边以及上下两边对应的系数是相反的；当condition=3时，表示 mixed 边界条件，在本程序中假设左右两边为 Neumann，上下两边为 Dirichlet，此时将以上两种方法结合使用即可构建方程组。

get_result和get_errornorm的作用已在前文叙述，此处不再赘述。

- class FD_irregular：该派生类用于支持单位正方形内部去掉一个圆的区域上的微分方程的 Dirichelet，Neumann 和 mixed 三种边界条件求解。FD_irregular构造函数，用于接收真实函数值，网格宽度，边界条件和圆的信息等参数。

Label函数返回一个 vector L，其每个元素代表一个需要求解的点以及其类型。具体来说，对于一个二维区域，其第 i 个元素为 $\{x_i, y_i, type_i\}$，其中 (x_i, y_i) 是对应序号 i 的点的坐标，$type_i$ 为点的类型：$type_i = 1$ 为区域内格点，$type_i = 2$ 为区域外 ghost point（四周存在一个区域内格点，且不在边界上的点），$type_i = 3$ 为正方形边界点，$type_i = 4$ 为恰好落在圆上的格点。函数中的实现逻辑是遍历整个区域内的点，根据其距离区域内挖去的圆的距离以及是否在边界上来确定其类型，并将其加入到返回的 vector 中。同时，再判断全体格点是否为 ghost point。最后返回所有点的信息。

Find_Label函数用于查找给定坐标在 vector L 中的位置和类型。如果该点不在 L 中，返回 $\{-1, -1\}$。

solve函数求解 $AU = F$ 的线性方程组，其中 A 和 F 分别为系数矩阵和右侧向量。我们对 vector L 中的元素进行便利，在不同的边界条件下，使用不同的方法来构造 A 和 F: 当condition=1时，表示 Dirichlet 边界条件。此时，在矩阵 A 的内部区域中，使用五点差分公式来近似求解二阶偏微分方

程。在边界上，将 u 的值作为已知量，构造出系数矩阵 A 和右侧向量 F。比较不同的是 $type_i = 2$ 的情况，此时我们从该点按上下右左的顺序依次搜索，找到与其相邻的（第一个）内点以及对应的圆与网格的交点，从而用线性插值的方式得到一个方程；当`condition=2`时，表示 Neumann 边界条件。此时，在矩阵 A 的内部区域中，仍然使用五点差分公式来近似求解二阶偏微分方程，在正方形边界上的做法也和`FD_regular`相同。但是在圆形边界上的情况会有所不同，此时将圆和网格交点对应的 $type_i = 2$ 的点处通过上下左右一次搜索，用周围的四个点对方向导数进行二阶精度表示，以近似交点处的方向导数；当`condition=3`时，表示 mixed 边界条件，在本程序中假设正方形边界为 Neumann，圆形边界为 Dirichlet，此时 $type_i$ 的各种取值对应的情况都可以用上文中相应方法得到解决。

`get_result`和`get_errornorm`的作用已在前文叙述，此处不再赘述。

注：1. 关于 Neumann 编制条件以及非规则区域的搜索方法较为繁琐（但是思路是很清晰的），本人文字表达能力有限，如果有没有表达清楚的地方烦请在`FD.h`中找到相应的代码，我已尽量作出完整的注释；2.Neumann 边值的情况由于得到的解不适定，需要增加额外非导数的条件，本程序用序号为 1 的点对应的真实值作为补充条件。

2.2 程序测试指南及结果展示

本节中，我将说明如何对程序`test.cpp`进行测试，并根据题目要求完成对函数进行求解和误差分析，以及对结果给出可视化的展示（图像均由 Python 绘制）。

2.2.1 test.json 参数说明

所有的任务都可以通过调整`test.json`中的参数得到解决。因此本节首先给出`test.json`中各参数的含义说明。

- `n`：单位正方形格点等分数（沿 x 以及 y 方向）。根据题目要求 $n = 8, 16, 32, 64$，一般无需修改。
- `function_label`：函数标号。根据题目要求，设计了三个函数进行求解，序号 1 代表 $u = e^{y+\sin x}$，序号 2 代表 $u = e^{-xy}$，序号 3 代表 $u = \sin(xy)$，可根据需求自行修改。
- `circle`：区域中需要挖掉的圆的信息。需要输入三个浮点数，第一个代表圆心的 x 坐标，第二个代表圆心的 y 坐标，第三个代表圆的半径。如果区域 ω 为单位正方形，则将半径设置为 0 即可。
- `boundary_condition`：输入 1，2, 或 3, 代表边界条件。1 表示 Dirichlet，2 表示 Neumann, 3 表示 mixed。

2.2.2 test.cpp 输出说明

针对不同的参数运行`test.cpp`，得到的结果（包括误差范数，在固定点的误差，在格点处的值等）全部输出至文件`test_results.txt`中，根据需要进行选择即可。

2.2.3 数值求解结果展示

在求解方程之后，通过`get_result`可以得到每一个格点处的 u 的数值解并将其输出至`test_results.txt`中，据此可以绘制图像进行清晰地展示。由于对三个函数对应的所有 n 和边界条件进行绘图将会得到几十幅图像，一一列举并不现实，加之本人并没有足够的 Python 水平绘制出挖掉一个圆的立体图，此处仅选取部分数值结果图像进行展示。

具体来说，首先对 $n = 8, 16, 32, 64$，绘制正方形区域函数 $u = e^{y+\sin x}$ 对应 Dirichlet 边值条件的求解结果的图像如下所示。

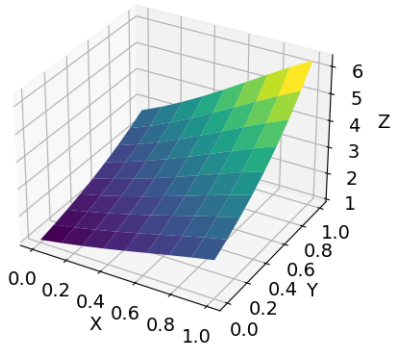

图 1: $n = 8$

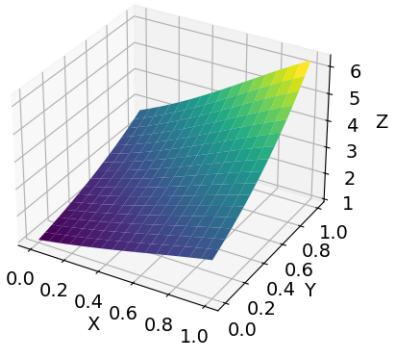

图 2: $n = 16$

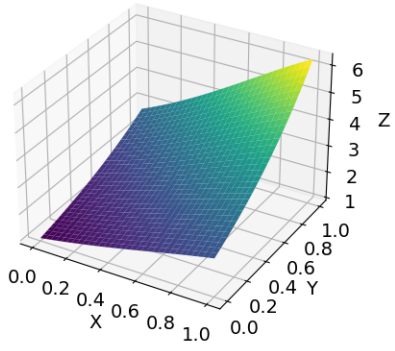

图 3: $n = 32$

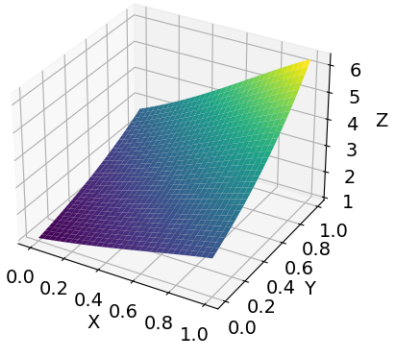

图 4: $n = 64$

其次，对 $n = 8, 16, 32, 64$，绘制正方形区域函数 $u = e^{-xy}$ 对应 Neumann 边值条件的求解结果的图像如下所示。

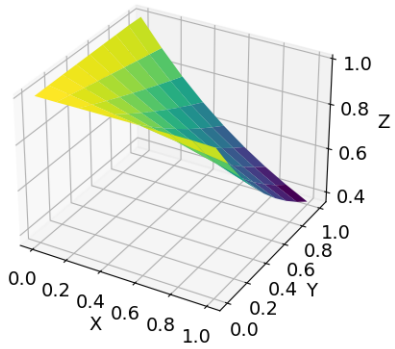

图 5: $n = 8$

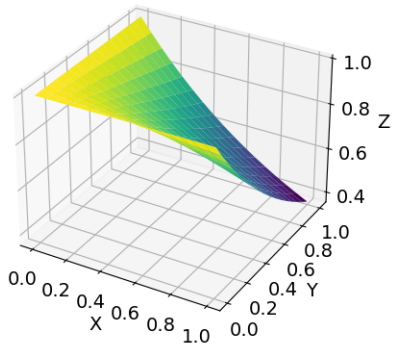

图 6: $n = 16$

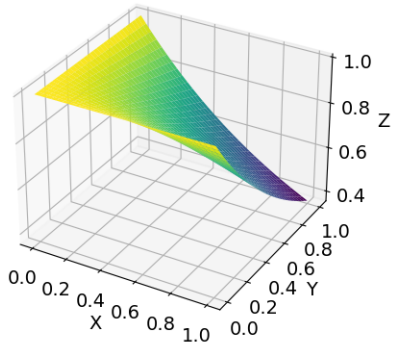

图 7: $n = 32$

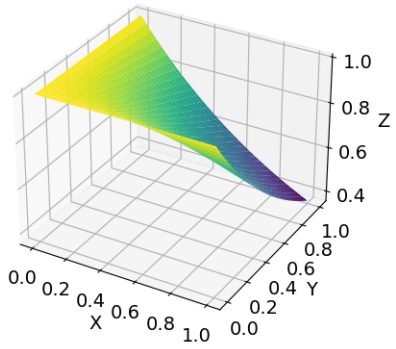

图 8: $n = 64$

最后，对 $n = 8, 16, 32, 64$，绘制正方形区域函数 $u = \sin(xy)$ 对应 Mixed 边值条件的求解结果的图像如下所示。

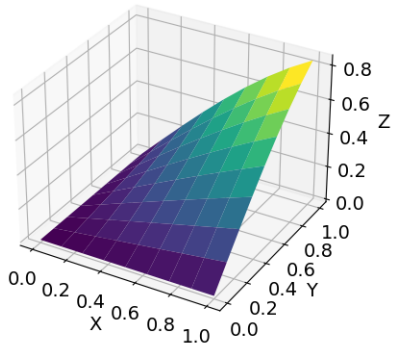

图 9: $n = 8$

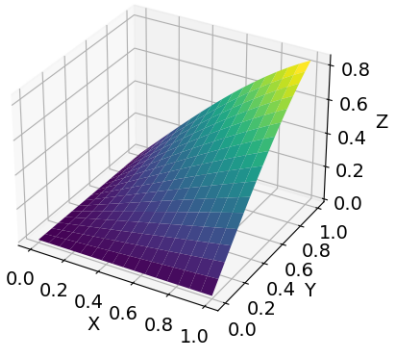

图 10: $n = 16$

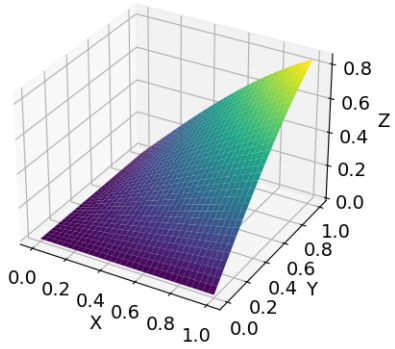

图 11: $n = 32$

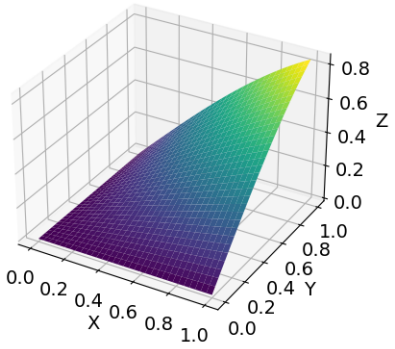

图 12: $n = 64$

从图像中我们可以很好地看出各个函数在单位正方形内的变化趋势，这从直观上验证了`FD.h`中关于 FD method 的算法的有效性，也展示了 FD method 能够较好的对原函数进行估计。对于不规则区域，接下来的误差分析中也可以从中反映算法的正确性和有效性。

2.2.4 固定点收敛速率分析

根据题目要求，在程序运行时，我们记录下求解结果与真实解在四个点 $(0.125, 0.125)$, $(0.125, 0.875)$, $(0.875, 0.125)$ 和 $(0.875, 0.875)$ 的绝对误差输出至`test_results.txt`中。

为了对收敛速率进行分析，我们需要对获得的绝对误差进行进一步处理。首先，修改`test.json`中`n`为`[8,16,32,64,128]`，增加网格数以便观察趋势；其次，绘制绝对误差序列前一项与后一项的比值，以便观察 n 翻倍之后绝对误差缩减的比例。

基于此，我们首先对规则区域进行收敛阶分析。对于三个函数，我们分别记录每个函数对应的三种边界条件的绝对误差序列的比值（取 $\log 2$）。由于有四个点，因此我们取四个点计算得到的平均值。据此绘制图像如图所示。

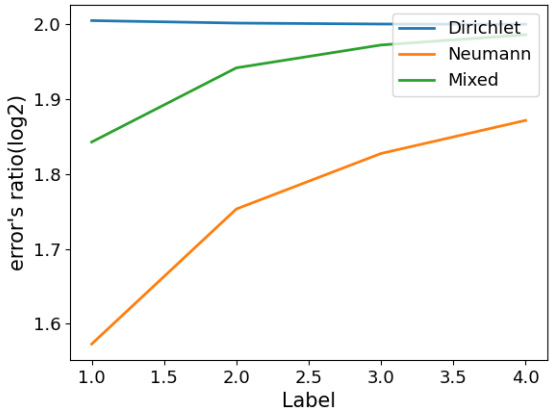

图 13: 规则区域 $u = e^{y+\sin x}$ 误差比值（$\log 2$）

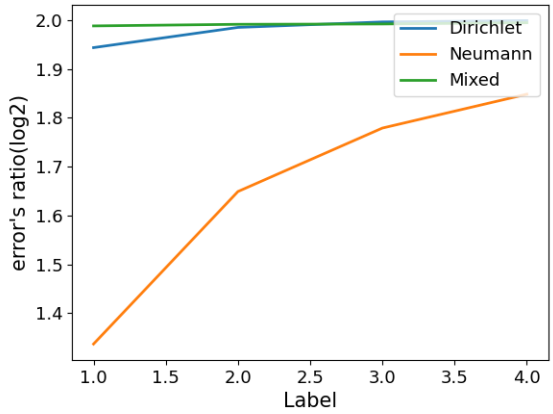

图 14: 规则区域 $u = e^{-xy}$ 误差比值（$\log 2$）

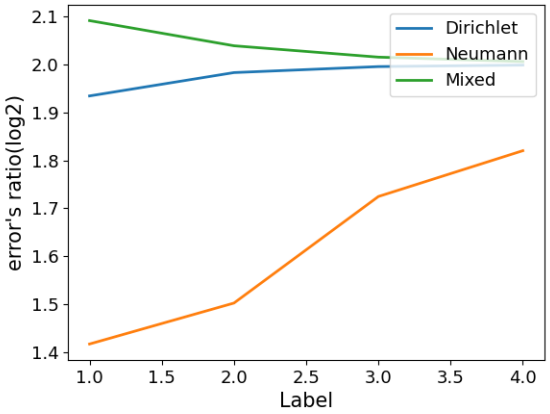

图 15: 规则区域 $u = \sin(xy)$ 误差比值（$\log 2$）

从图中可以较直观的看出，**三种边界条件的收敛速度关于 h 都是 2 阶的**。具体来说，相较之下

Dirichlet 和 Mixed 边值条件的收敛比值较为接近 2，且较为稳定；而 Neumann 边值条件的收敛阶相对较不稳定，但逐渐趋于 2。另一方面，根据`test_results.txt`的输出，Dirichlet 和 Mixed 边值条件的误差从量级上也小于 Neumann 边值条件的误差，这应该是因为纯 Neumann 的边值问题是不适定的，而本程序仅仅对其增加了一个新的条件，这只能确保该 PDE 有唯一解，但条件仍较弱，因此得到的数值解的误差较大。

其次，我们对不规则区域进行收敛阶分析。不妨设挖去的圆心在 $(0.5, 0.5)$，半径为 0.25. 同样地，对于三个函数，我们分别记录每个函数对应的三种边界条件的绝对误差序列的比值如下图所示。

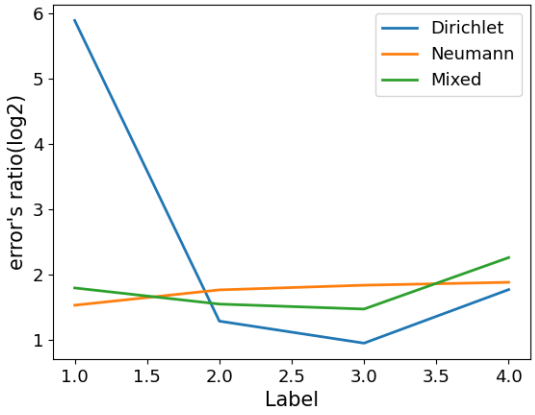

图 16: 不规则区域 $u = e^{y+\sin x}$ 误差比值（$\log 2$）

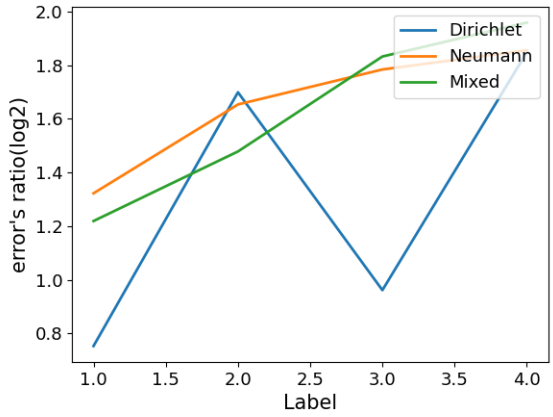

图 17: 不规则区域 $u = e^{-xy}$ 误差比值（$\log 2$）

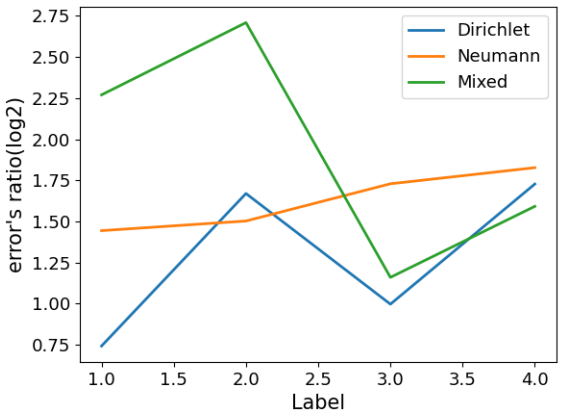

图 18: 不规则区域 $u = \sin(xy)$ 误差比值（$\log 2$）

从总体趋势来看，三**种边界条件的收敛速度关于 h 依然是 2 阶的**。区别在于不规则区域 Dirichlet 和 Mixed 边值条件的不稳定性相比规则区域变大了。观察`test_results.txt`的输出发现，这是因为此时某个别点的收敛速率严重偏离 2 阶，而其他点依旧是 2 阶收敛的。这可能是因为对于 ghost point，它的四周可能存在两个或者更多内点，但我们在 ghost point 只能设置一个边值条件，因此会使得得到的解具有较大的误差。

2.3 误差范数及其收敛分析

根据题目要求，`test_results.txt`中还输出了不同函数、边值条件和 n 对应的误差的 1、2 和无穷范数，其定义为

$$\begin{aligned}\|g\|_1 &= h^2 \sum_i |g_i|,\\ \|g\|_2 &= \left(h^2 \sum_i |g_i|^2 \right)^{\frac{1}{2}},\\ \|g\|_\infty &= \max_i |g_i|.\end{aligned} \tag{1}$$

关于其绝对的值都可以在文本文档中详细查看，可以看出不论何种情况，误差的各范数都是很小的数值（小于 10^{-3}），这也从侧面印证了算法的正确性。

同样的，我们更加关注的是误差范数的收敛分析。当每个元素时二阶收敛时，根据类似 **Exercise 7.14** 的简单运算（注意此时 $N = \Theta(\frac{1}{h^2})$），我们可以得到

$$\begin{aligned}\|g\|_1 &= h^2 O(1) = O(h^2),\\ \|g\|_2 &= h[O(h^2)]^{\frac{1}{2}} = O(h^2),\\ \|g\|_\infty &= O(h^2),\end{aligned} \tag{2}$$

这也是我们所需要验证的。首先，修改`test.json`中`n`为`[8,16,32,64,128]`，增加网格数以便观察趋势；其次，绘制误差范数序列前一项与后一项的比值，以便观察 n 翻倍之后绝对误差缩减的比例。

基于此，我们首先对规则区域进行收敛阶分析。对于三个函数，我们分别记录每个函数对应的三种误差范数的比值（取 $\log 2$），据此绘制图像如图所示。（根据上一节分析，各点的收敛阶均为 2, 因此为了避免图像过多，不失一般性，此处仅展示 Dirichlet 边值条件的结果）。

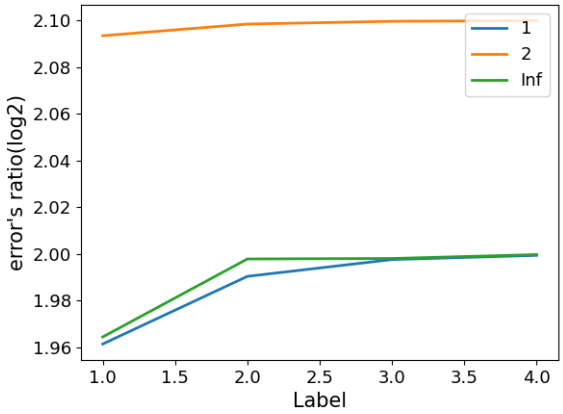

图 19: 规则区域 $u = e^{y+\sin x}$ 误差范数比值（$\log 2$）

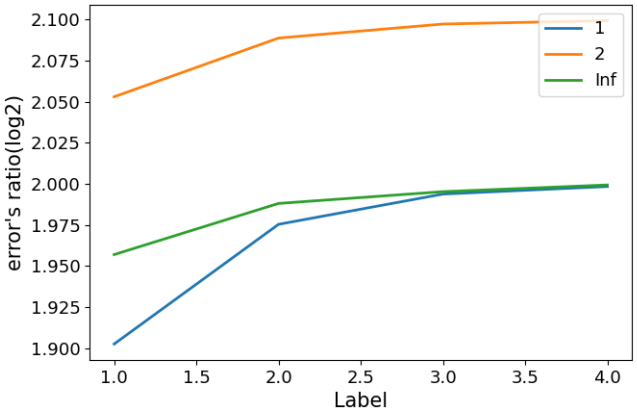

图 20: 规则区域 $u = e^{-xy}$ 误差范数比值（$\log 2$）

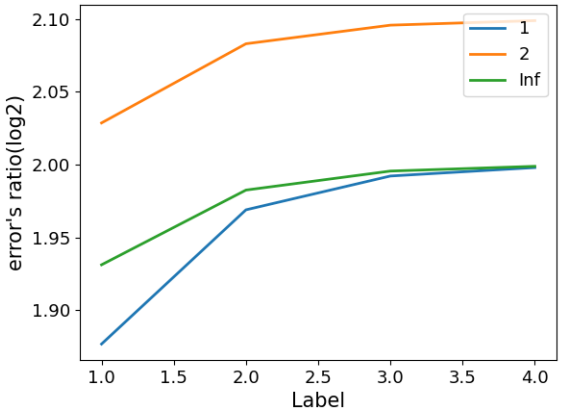

图 21: 规则区域 $u = \sin(xy)$ 误差范数比值（$\log 2$）

显然，**规则区域三种范数的收敛阶都稳定趋向于 2 附近**，这在计算机层面验证了前文所述的收敛速率。(不过三个函数的二范数都趋向于 2.1 左右，这个现象我并没有想明白是为什么)。

其次，我们对不规则区域进行收敛阶分析。不妨设挖去的圆心在 $(0.5, 0.5)$，半径为 0.25. 同样地，对于三个函数，我们分别记录每个函数对应的三种误差范数的比值（取 $\log 2$），据此绘制图像如图所示。(根据上一节分析，各点的收敛阶均为 2，因此为了避免图像过多，不失一般性，此处仅展示 Neumann 边值条件的结果)。

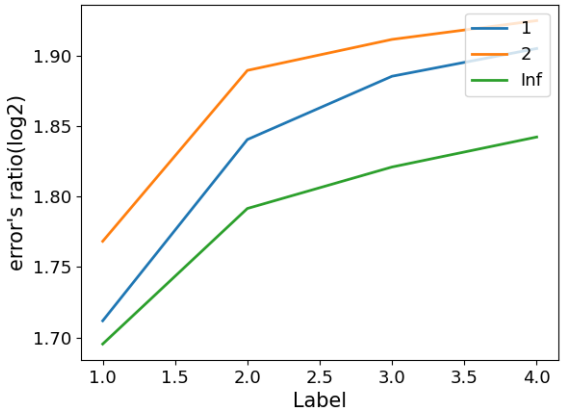

图 22: 不规则区域 $u = e^{y+\sin x}$ 误差比值（$\log 2$）

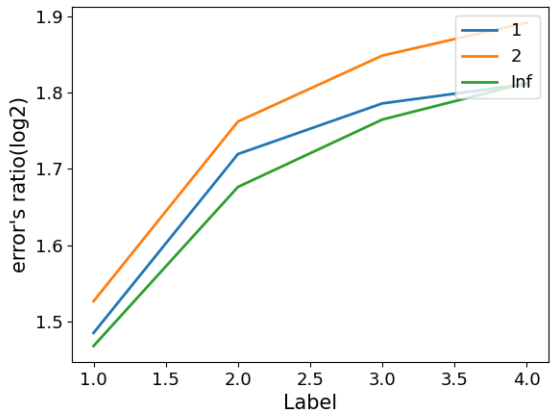

图 23: 不规则区域 $u = e^{-xy}$ 误差比值（$\log 2$）

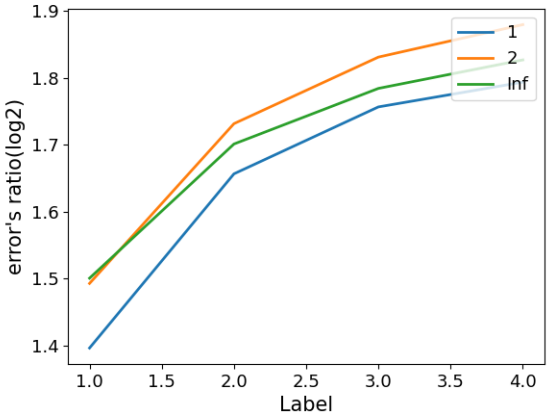

图 24: 不规则区域 $u = \sin(xy)$ 误差比值（$\log 2$）

不难看出，随着网格的加细，**不规则区域三种范数的收敛阶也都稳定趋向于 2 附近**，这在计算机层面验证了前文所述的收敛速率。