有限差分法项目作业

俞茂桦
3210104374
信计 2101

30 March 2024

Contents

Chapter 1

Introduction

1.1 Objectives

该项目设计了利用有限差分法求解二阶 Possion 方程的程序. 利用 C++ 和五点差分完成计算 (目录下运行 make run 执行 main 程序读取 input 中输入并计算, 输出结果到 output) , 利用 python 实现绘图 (绘图结果在 image 中).

计算程序的定义域为 $(0,1)^2$ 或 $(0,1)^2$ 减去一个圆盘, 要求减去圆盘后剩余区域连通.

- 检测定义域连通性
- 在 $(0,1)^2$ 上计算带有 Dirichlet 边界条件的 Poission 方程
- 在 $(0,1)^2$ 上计算带有 Neumann 边界条件的 Poission 方程 (需要设定 Dirichlet 条件的点使得结果唯一)
- 在 $(0,1)^2$ 上计算带有 Mixed(部分 Dirichlet, 部分 Neumann) 边界条件的 Poission 方程
- 在 $(0,1)^2$ 减去一个圆盘的定义域上完成以上计算
- 可以计算 solution error 的 $1-norm, 2-norm, max-norm$
- 对解和 solution error 进行画图

Chapter 2

Design

在程序设计上, 通过将问题抽象, 主要实现了以下几个 class.

- Function: 函数的接口类, 要求实现 operator() 接口, 即可以通过函数 (参数) 的形式来获得函数值
- BoundaryCondition: 代表边界条件的类, 可以通过 Type() 获取某一点是 Dirichlet 条件还是 Neumann 条件, 通过 Value() 获取相应函数值, 这两个方法本质上都是某个 Function 的间接调用
- Cycle: 代表了定义域被减去的圆盘, 其中有 BoundaryCondition 作为成员, 并且可以判断一个点是否在圆盘上
- Domain: 代表了定义域, 将 BoundaryCondition 和 Cycleu 作为成员, 可以判断自身连通性
- Solver: 代表求解器, 将 Domain 作为成员, 实现有限差分法的求解.

此外,Tensor 模板类代表了张量,Solver 中的 Matrix 和 Vector 分别是其为二阶和一阶并且储存 double 值的情况.FunctionExtension 则是继承 Function 类的函数类的集合, 最终求解器接受这些函数作为输入进行初始化.ExpressionTree 中为表达树的实现, 可以解析带有 cmath 函数和自变量的字符串并求值. Assistance 中为一些辅助函数.JsonRead 中定义了读取 jsona 文件并接受所定义的函数进行 Solver 初始化, 求解, 输出的函数 (误差的 $max-norm, 2-norm, 1-norm$ 被依次打印在命令行中).

Chapter 3

Implementation

在完成问题的定义后, 求解器首先通过五点差分理论获得线性系统, 然后求解线性系统获得解, 再输出解以及利用解来计算各种 $q-norm$ 等.

将在此处介绍最艰难的部分, 即针对 Chapter1 中的各种问题该项目如何获得线性系统.

线性系统的定义主要由两个函数完成, 分别是 Define 和 DefineCoef.

DefineCoef 输入可离散的格点 (规则或不规则) 的序号 (第几列, 第几行), 依次返回其左侧, 上侧, 右侧, 下侧, 自身的系数以及线性系统右侧常数, 若左上右下的点中有点是非离散点, 则其系数为 0, 从而不会写入线性系统左侧的矩阵中.

Define 首先根据离散点总数完成线性系统左侧矩阵和右侧向量的初始化. 然后将离散点 (规则或不规则) 的格点序号输入到 DefineCoef 中, 然后将 DefineCoef 的输出赋予到线性系统相应的位置. 其中离散点格点序号和其在线性系统中的位置是通过一个 Hashmap 完成对应的.

于是针对各种情况的处理任务主要由 DefineCoef 完成, 下面具体介绍它是如何完成的.

3.1 DefineCoef

下面设沿着一个维度有 $n+1$ 个格点, 令 $h = 1/n$, 令 rhs 为 Poission 方程 $-\Delta u = f$ 的右端函数, 令 U_l 代表左侧点, 令 U_u 代表上侧点, 令 U_r 代表右侧点, 令 U_d 代表下侧点, 令 U_c 代表自身 (中心点).

DefineCoef 的实现思路是:

- 写出五点差分公式 (系数和常数值都储存在输出向量中), 若所有点都不在边界, 直接输出
- 否则, 将 *Neumann* 条件的点表示为其它点和常数的线性组合 (即通过 Taylor 展开的原理利用其它点估计该点的偏导数或方向导数), 得到相应的方程组
- 解出 *Neumann* 条件的点关于 *Dirichlet* 条件的点, 离散点和常数的线性组合.
- 将上一步结果代入五点差分公式, 再将 *Dirichlet* 条件的点的值代入
- 得到只关于离散点和常数的五点差分公式, 将其系数 (非离散点系数此时变为 0) 和常数值输出

3.1.1 五点差分公式

若输入点规则则根据

$$-\frac{U_l - 2U_c + U_r}{h^2} - \frac{U_u - 2U_c + U_d}{h^2} = rhs$$

初始化输出向量为 $[-1/h^2, -1/h^2, -1/h^2, -1/h^2, 4/h^2, rhs]$.

若不规则, 则通过 taylor 展开原理用相应的公式初始化向量.

如设 $U_l.x = U_c.x - \theta h$, 则对应的估计 U_{xx} 的公式为

$$\frac{\theta^2 h^2 + \theta h}{2} U_{cxx} = U_0 + (-1-\theta)U_1 + \theta U_2$$

这一步的 LTE 为 $O(h^2)$.(注意本篇 report 中不把左边的系数除到另一边)

3.1.2 非圆边界的 *Neumann* 条件

在这种情况下, 通过同一行或同一列的三个点来估计该值, 可以令 LTE 达到 $O(h^2)$.

由于另一侧的点可能在圆周上, 不妨设 U_l 是非圆周的边界上的不规则点,$U_r.x - U_c.x = \theta h$, 则 U_r 不在圆周上的情况等价于 $\theta = 1$, 所以这里保留 theta 给出估计 U_l 处方向导数的估计.

注意, 此项目中 *Neumann* 条件对应的方向导数方向为从定义域内部指向定义域外部, 这是在定义输入函数时容易写错的一点.

利用对左侧点的 Taylor 展开给出关于 x 轴 a 正方向的偏导数估计

$$\begin{aligned}&(-(1+\theta)^2 h + (1+\theta)h)U_{lx}\\ =&((1+\theta)^2 - 1)U_l - (1+\theta)^2 U_c + U_r\end{aligned}$$

该公式的 LTE 为 $O(h^2)$.

3.1.3 圆周上 *Neumann* 条件的情况

对于圆周上 *Neumann* 条件的点, 我们可以获得它指向圆心的方向导数, 其中方向导数为 $U_x \cos\beta + U_y \sin\beta$, 其中 β 为该圆周上的点与圆心连线的角度, 可以由该点坐标和圆心坐标求出, 这里不缀述.

我们只需要在给出 U_x 和 U_y 的估计即可将条件表示为点和常数的线性组合.

假设该在圆周上的点为 U_l, 设 $U_c.x - U_l.x = \theta h$, 此时还要考虑 U_u 和 U_d 是否在圆周上 (这是为了估计 U_y, 可以看下面的式子). 不妨设 U_d 在圆周上, 设 $U_c.y - U_d.y = \alpha h$, 而两个点都不在圆周上时, 只需代入 $\alpha = 1$.

通过对左侧点的 Taylor 展开, 给出以下估计式

$$\begin{aligned}&(-(1+\theta)^2\theta h + \theta^2(1+\theta)h)U_{lx}\\ =&((1+\theta)^2 - \theta^2)U_l - (1+\theta)^2 U_c + \theta^2 U_r\\ &(-\alpha^2 h - \alpha h)U_{ly}\\ =&(-\alpha^2)U_u + (\alpha^2 - 1)U_c + U_d\end{aligned}$$

上式中, 对 U_{lx} 的估计的 LTE 为 $O(h^2)$, 但是对 U_{ly} 的估计的 LTE 只有 $C(U_{lxy})h + O(h^2)$, 即若函数的在该点先后对 x 和 y 求偏导为 0, 则也可以达到 $O(h^2)$.

3.1.4 剩下的部分

剩下的部分是简单的, 对于如何求解 *Neumann* 条件的点关于其它点和常数的线性组合, 我的做法是对于每个非 *Neumann* 条件的点为 1 其它为 0 的情况以及它们都为 0 的情况都设置一个右端向量, 于是可以求解出相应的线性表示.

事实上有这种需求的情况主要是 *Neumann* 条件的点的线性表示中出现了其它 *Neumann* 条件的点, 而这种情况除了遇见圆上 *Neumann* 条件的点以外应该是比较极端的, 所以这是一个留以优化的地方.

将求解出的线性表示代入五点差分公式, 再将 *Dirichlet* 条件的点的值代入, 输出即可.

Chapter 4

Evaluation

关于测试连通性以及圆中是否有 4 个以上格点, 分别调用 $solver._domain.test_connection$ 和 $solver._domain._cycle.Isvalid()$ 即可.

在这里针对各种情况的计算结果.

如果你想要自己完成计算, 请按以下步骤操作:

- 在 FunctionExtension.h 中公开继承 Function 类, 实现其中的括号运算符.
- 仿照 input 中的示例填写 json 文件
- 在 main 函数中调用 JsonRead 函数, 会执行计算并将计算结果输出到 output 中, 四列数据分别代表该点的列数, 行数, 计算值, 绝对误差
- 在 plot 文件夹的 python 文件中实现画图 (画图函数已实现, 只需将文件的名字作为参数, 会自动检索 output 中同名文件, 输出结果在 image 中)

4.1 计算结果

4.1.1 计算实例命名

为方便查找想要看的计算结果, 这里给出相应的命名规范

- 开头为格点数减一的值, 如 $n8$,$n16$,$n32$,$n64$

- 其次为第几个函数, 如 $f1$,$f2$

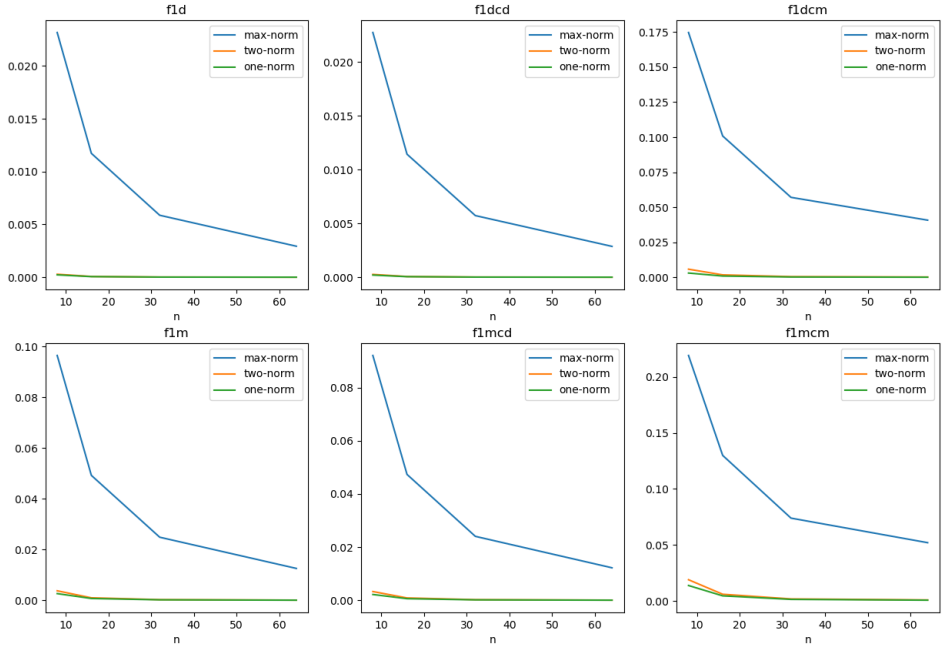
- 再然后为非圆上的边界条件类型 d 表示 *Dirichlet* 条件,m 表示 *Mixed* 条件 (由于 neumann 条件在常数意义下有无穷解, 需要设定一个 dirichlet 条件的点, 所以也视为 mixed)

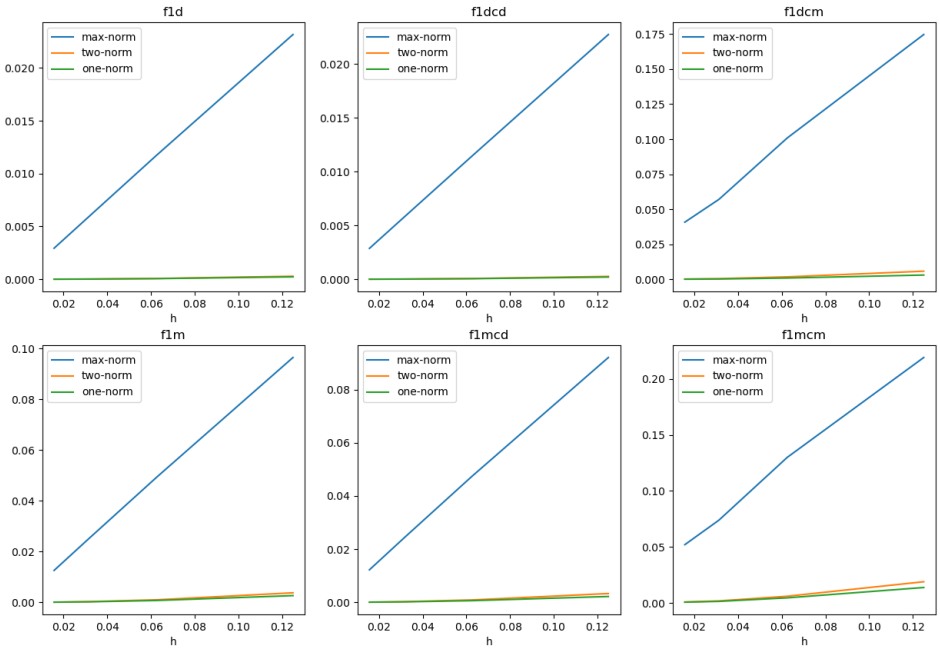
- 若接着一个 c, 表示定义域扣去了一个圆盘

- 若有 c, 则会跟随一个表示圆上边界条件类型的符号 (圆和非圆边界有一个有 Dirichlet 条件即可, 为方便, 测试时还是命名为 m 和 d)

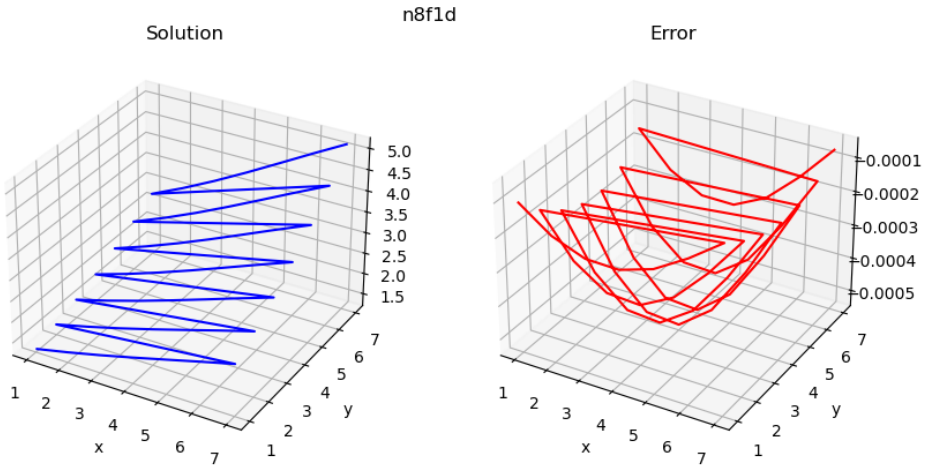
4.1.2 q-norm 随网格细分变化

结合随 n 变化的图像, 可知对算例总体而言, 随 n 变大, 误差变小. 从随 h 变化的图像, 关于 h 成线性关系. 以下第一张图为随 n 变化图像, 第二张图为随 h 变化图像.

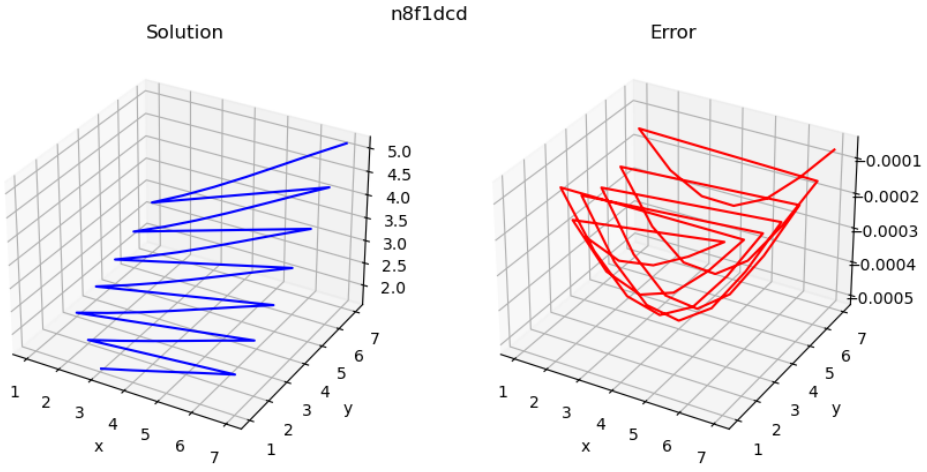



f1d
f1dcd
f1dcm
f1m
f1mcd
f1mcm
max-norm
two-norm
one-norm
h

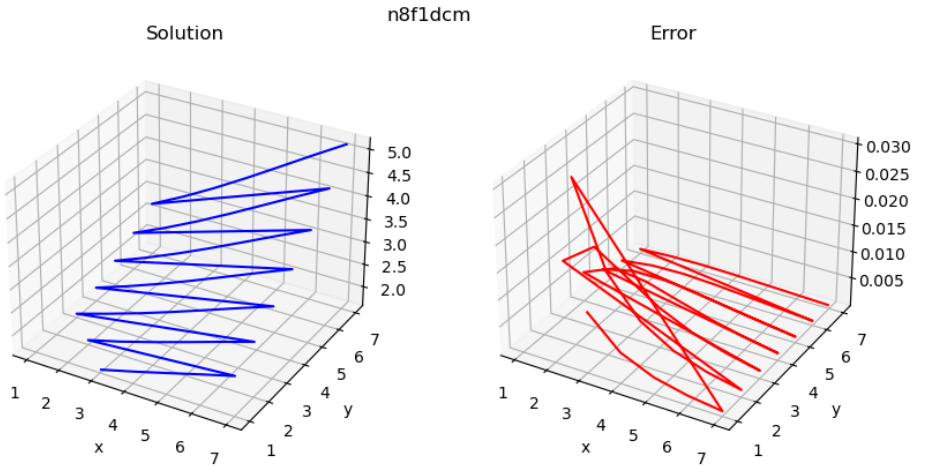
n8f1d


n8f1d
Solution
Error
x
y

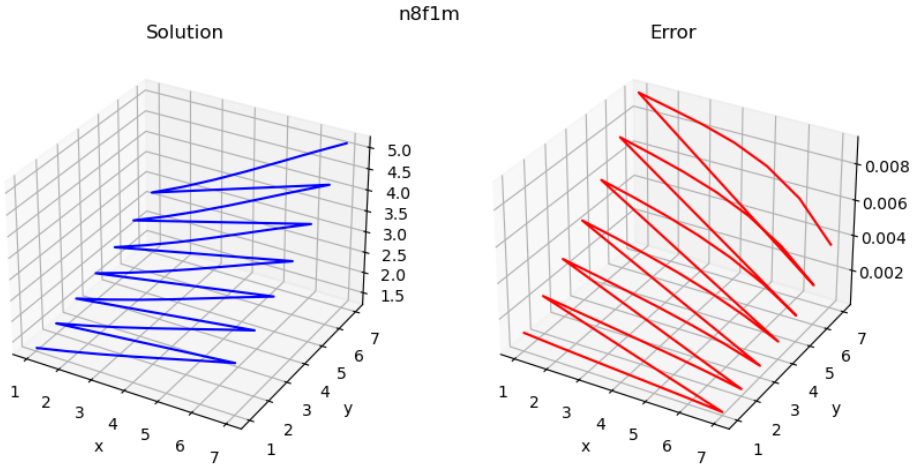
n8f1dcd



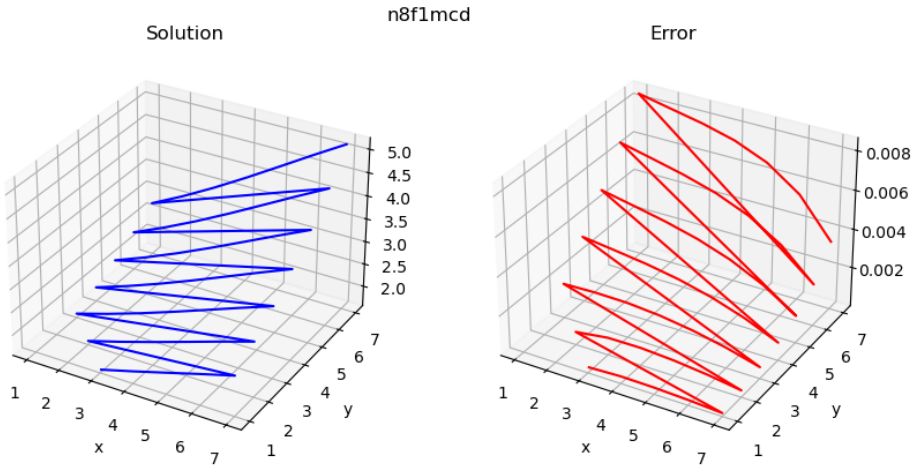
n8f1dcm



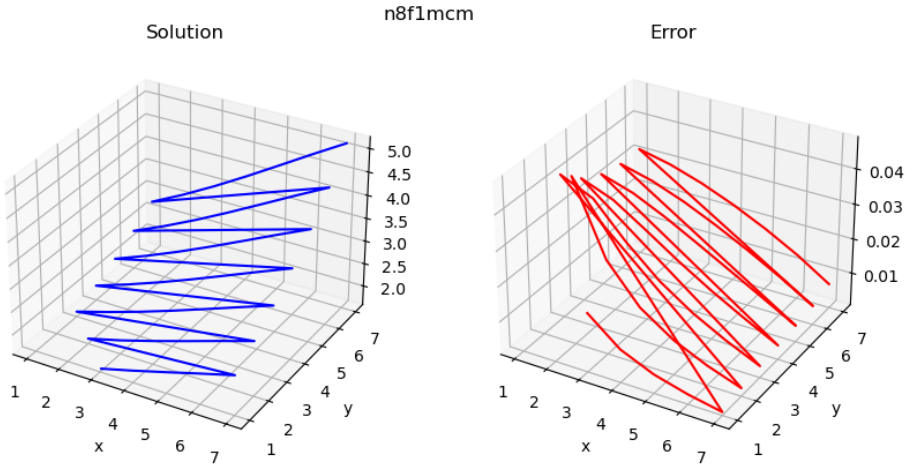
n8f1m



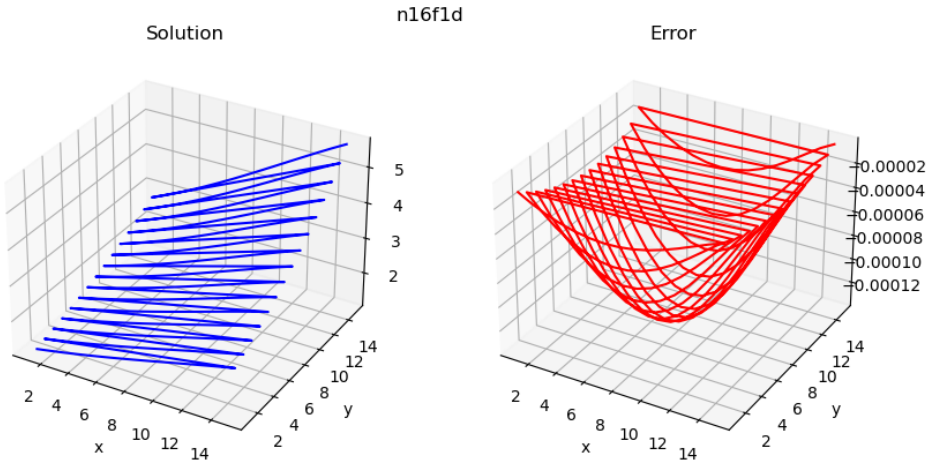
n8f1mcd



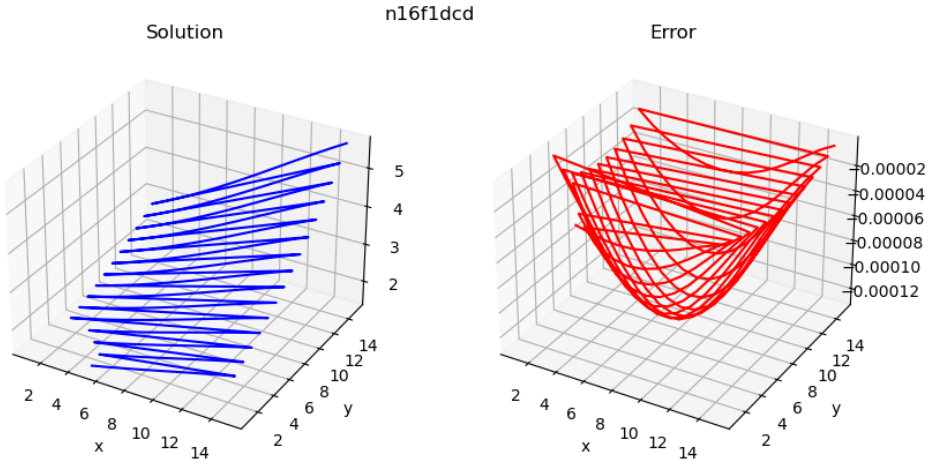
n8f1mcm



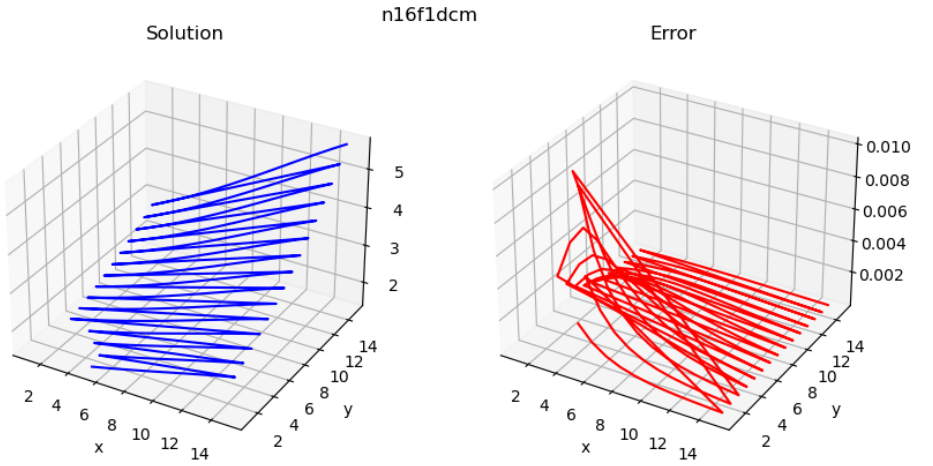
n16f1d



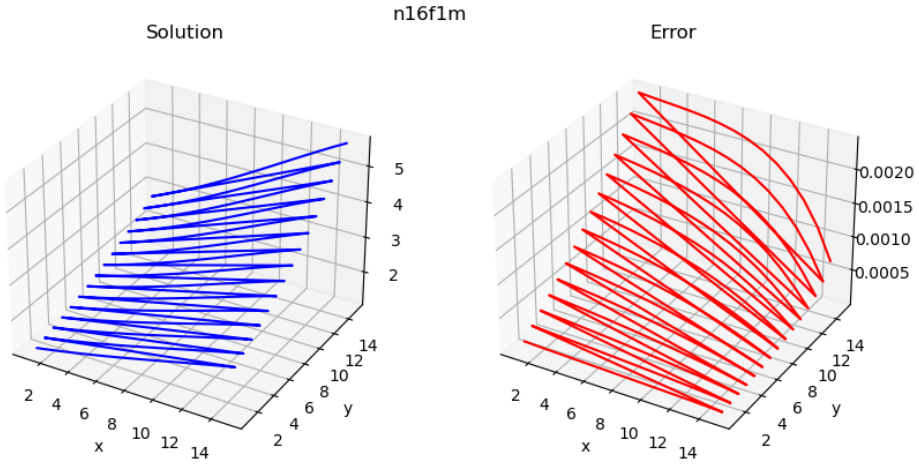
n16f1dcd



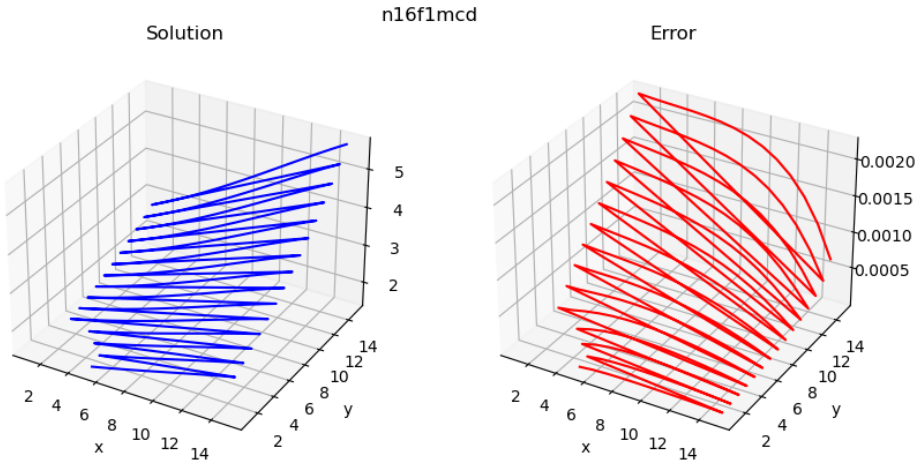
n16f1dcm



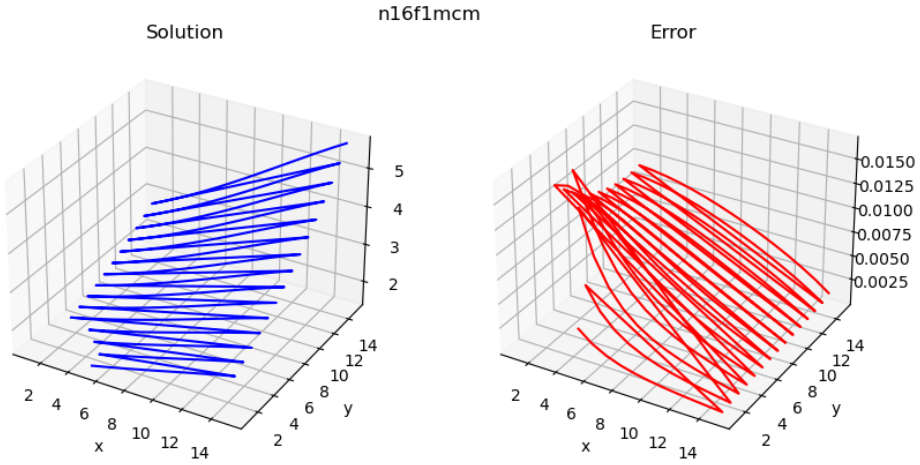
n16f1m



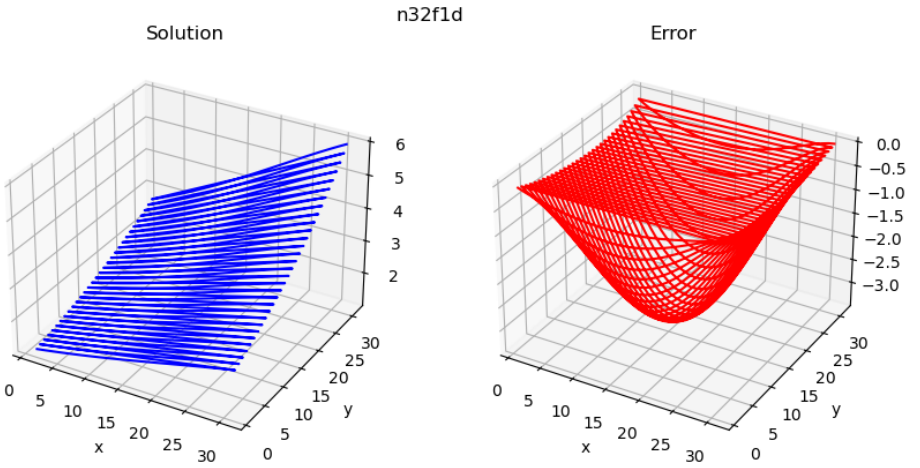
n16f1mcd



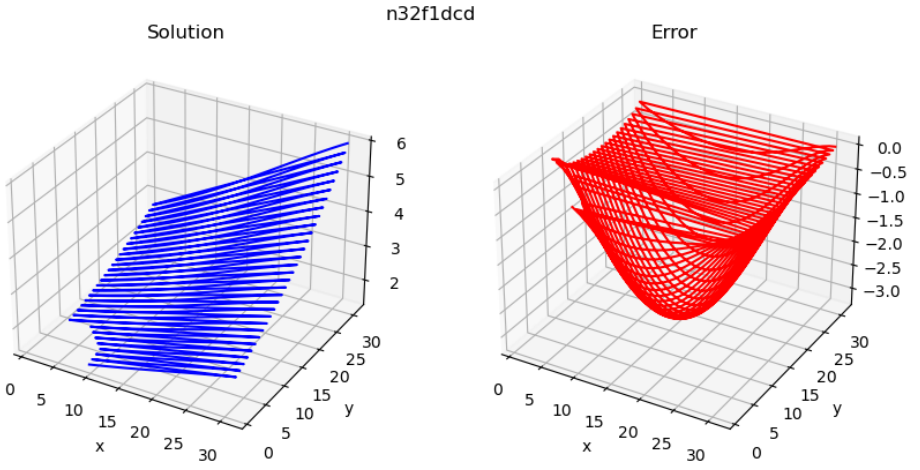
n16f1mcm



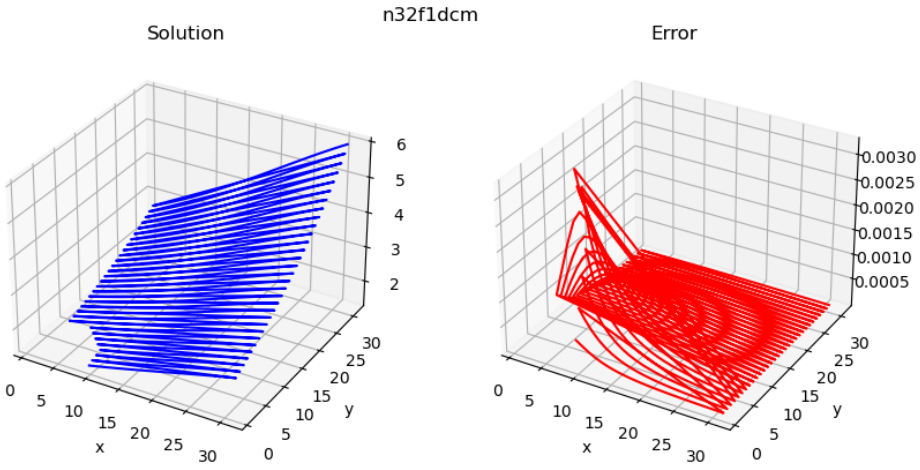
n32f1d



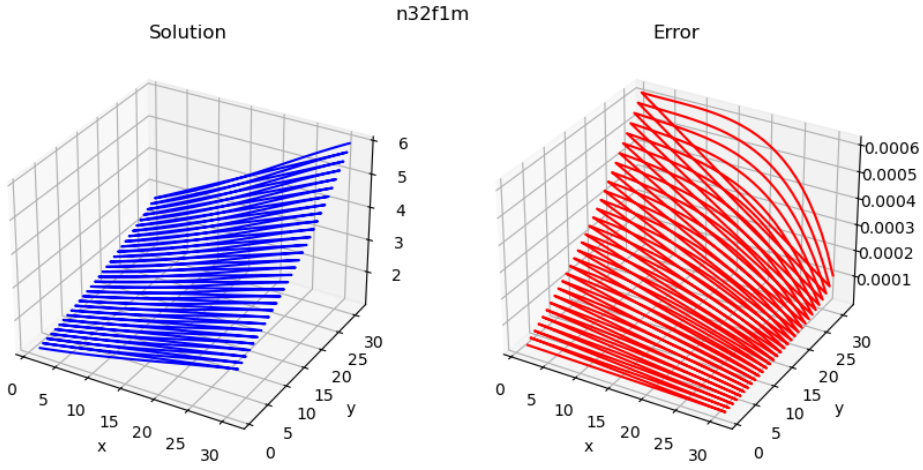
n32f1dcd



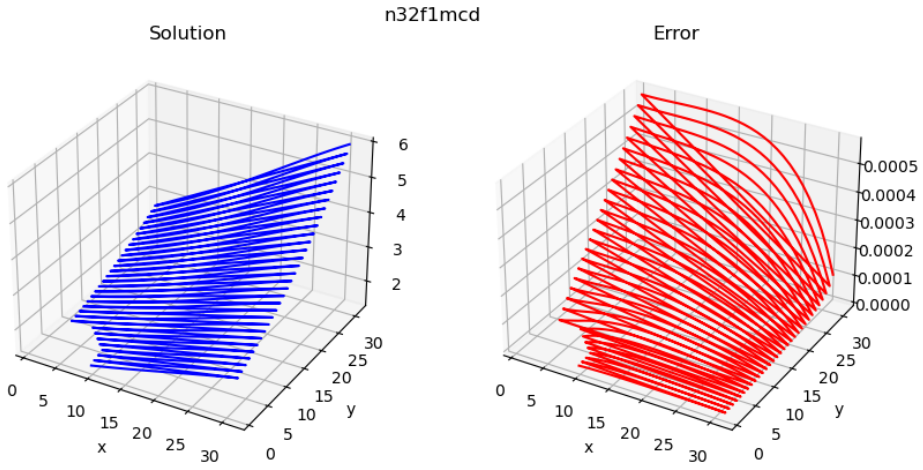
n32f1dcm



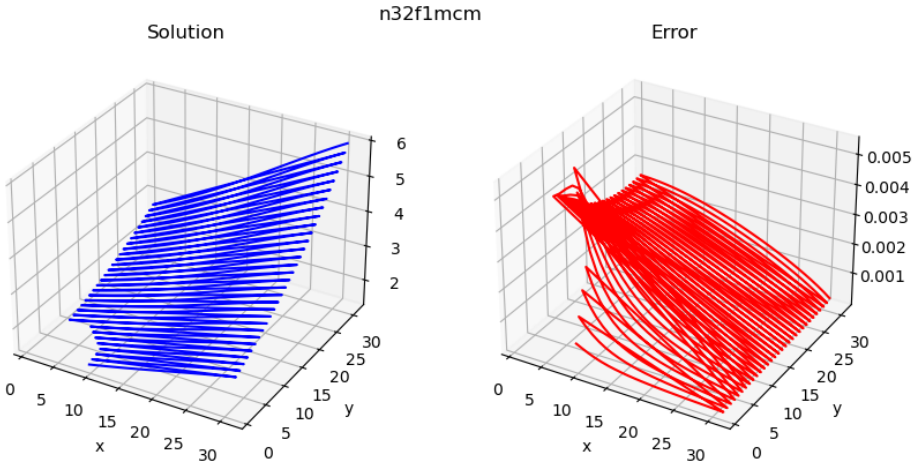
n32f1m



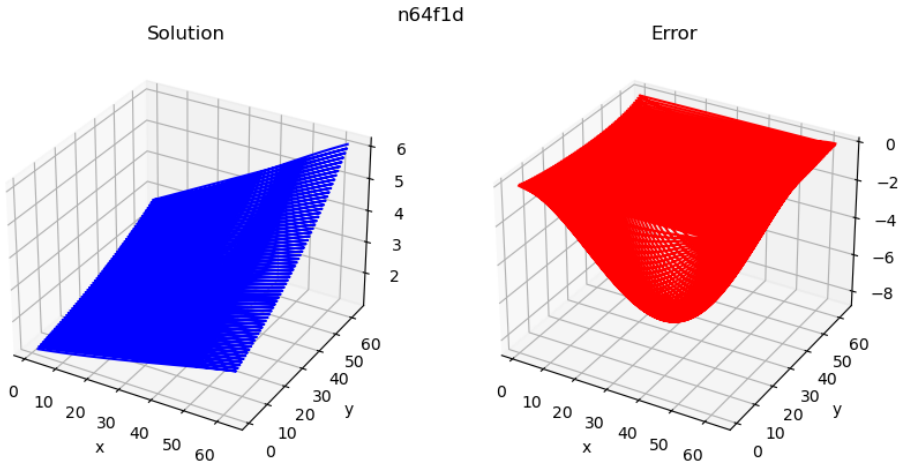
n32f1mcd



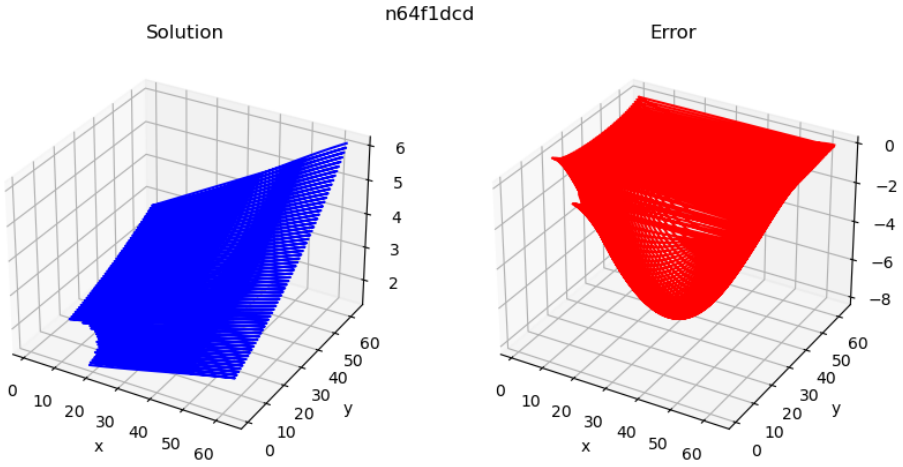
n32f1mcm



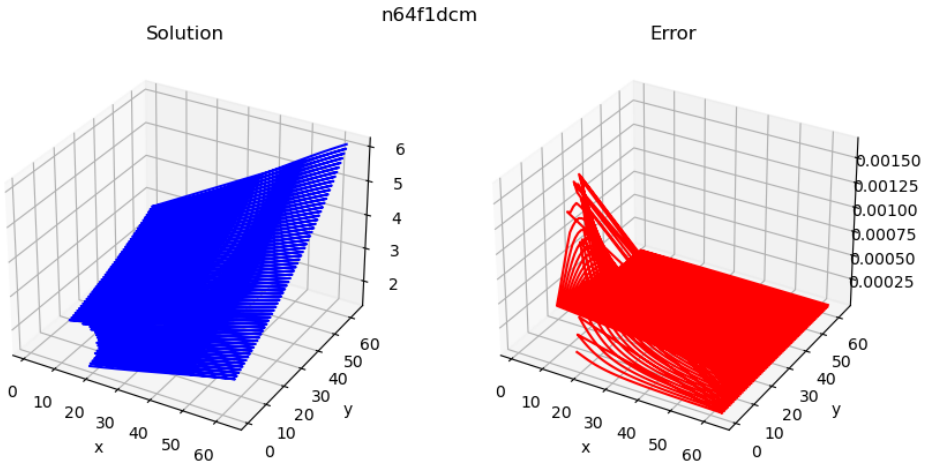
n64f1d



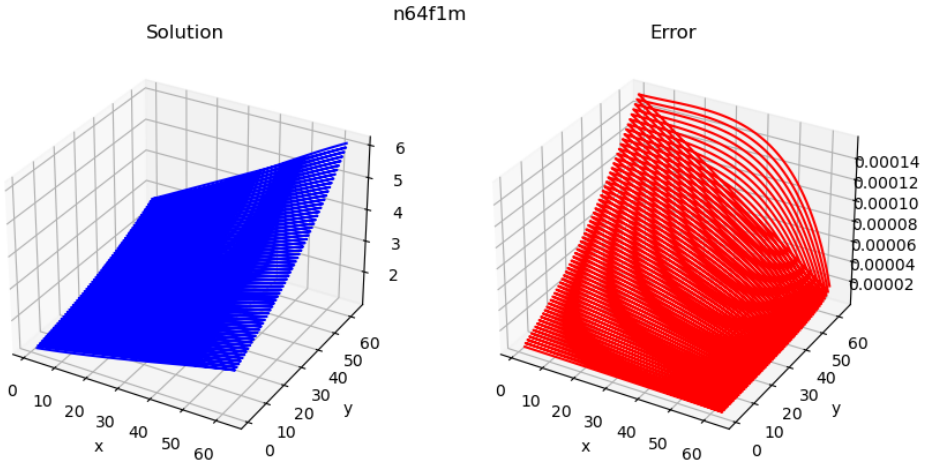
n64f1dcd



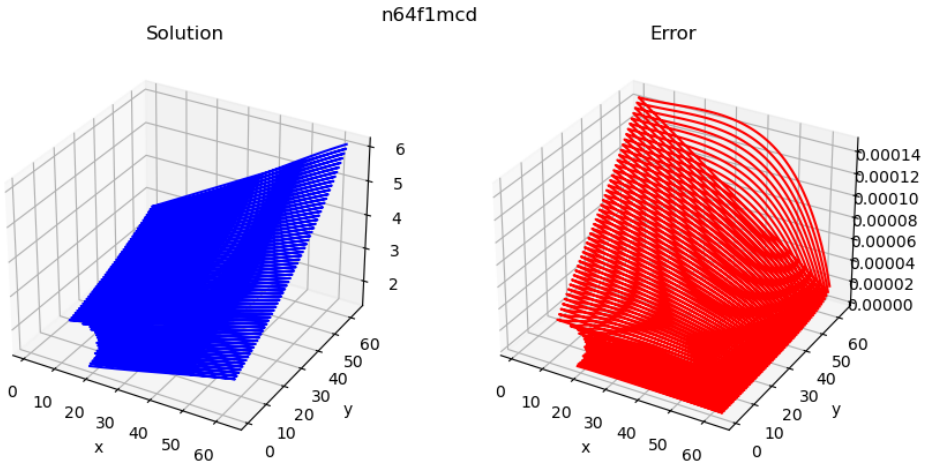
n64f1dcm



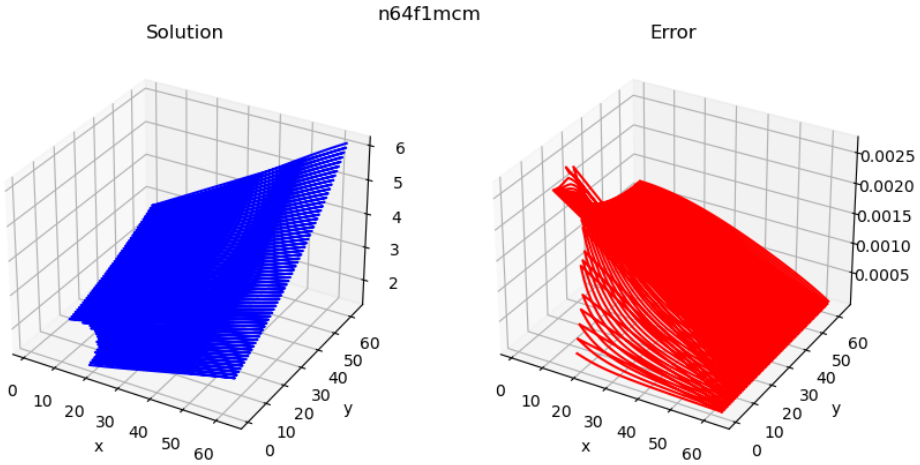
n64f1m



n64f1mcd



n64f1mcm


n64f1mcm
Solution
Error
x
y
0
10
20
30
40
50
60
2
3
4
5
6
0.0005
0.0010
0.0015
0.0020
0.0025

Chapter 5

Conclusions

结合上一章绘图, 发现在不同算例下, 误差分布有以下规律,

- 离 *Dirichlet* 条件点越远, 误差越大
- 离 *Neumann* 条件点越近, 误差越大

从 $max - norm$ 的变化来看, 误差都是关于 h 一阶收敛的.

5.1 Future work

目前程序存在以下缺点

- 程序在建立格点到线性系统的位置之间的关系时, 使用了两个 HashMap, 这导致了内存的浪费, 尤其是格点很多时.
- 程序对于所有情况的周边含有非离散点的点一视同仁, 都按照最复杂情况计算, 导致了运行效率上的损失.
- 程序在很多比较方面采用了字符串比较, 换成枚举会更好.
- 在调用函数时采用了不安全的函数指针.

- 在设计某些类时, 没有考虑并封装好其功能, 并且在它作为别的类的成员时, 没有考虑到外部访问它的需求, 在代码已经很多修改效率低下的情况下, 只能采取将其作为其它类的 public 成员的方法.

希望吸收此次作业的教训, 学习编程的经验, 将以后的作业完成的更好. 此外, 在算法精度上没有达到理想的二阶收敛等要求, 希望学习优秀的算法实现, 将数学原理和编程实现更好地融会贯通.

Bibliography