摘要

本次编程作业设计了多重网格求解 poisson 方程。自己设计了限制算子，插值算子，求解 Dircihlet 边界、Neumann 边界、混合边界条件的 Poisson 方程。

1 程序实现

1.1 类和成员

1. **Sparse_Matrix 类**首先我定义了 Sparse_Matrix 类，用来实现求解 Poission 方程，私有成员如下

```
        class Sparse_Matrix{
            private:
            vector<label> elements;
            int n;  //size of matrix
            public:
            Sparse_Matrix();
            Sparse_Matrix(const int &_n);
            Sparse_Matrix(const int &_n, const vector<label> &e);
            //Sparse_Matrix(Sparse_Matrix&& other) noexcept;
            //Sparse_Matrix& operator=(Sparse_Matrix&& other) noexcept;
            void setValues(const int &i, const int &j, const double &value); //set A(i,j)
            double operator()(const int &i, const int &j) const;
            Sparse_Matrix operator+(const Sparse_Matrix &B) const;
            Sparse_Matrix operator*(const Sparse_Matrix &B) const;
            Sparse_Matrix operator*(const double &a) const;
            Vector operator*(const Vector &v) const;
            vector<double> convert_to_vector() const;
            void Gauss_Seidel(Vector &initial, const Vector &b);
            int getdim() const;
            void print();

        };

```

Listing 1: 运算符实现

其中使用了稀疏矩阵的方法存储矩阵，只记录非零元素和非零元素的位置。并且实现了矩阵的移动构造、矩阵加法减法，矩阵的数乘，矩阵和矩阵、矩阵和向量的乘法以及 Gauss_Sediel 法求解 $Ax = b$。最后还实现了类似 matlab 按照行列标号取出矩阵的元素，如 $A(i, j)$ 取出矩阵的第 (i, j) 个元素。

2. **Vector 类：**定义了 Vector 类，用来储存向量，具体的成员变量和函数如下：

```
            class Vector{
                private:
                int n=0;  //length
                int m=0; //sqrt(n)
                vector<double> elements;
                public:
                Vector();
                Vector(const int &n);
                Vector(const vector<double> &e);
                Vector(const Vector& other);
                Vector(const int &n, const vector<double> &_e);
                Vector(Vector&& other) noexcept;
                void set_Value(const int &i, const double &value);
                void set_Value(const int &i, const int &j, const double &value);
                Vector& operator=(Vector&& other) noexcept;
                double operator()(const int &i) const;
```

```
17                Vector operator+(const Vector &v) const;
18                Vector operator-(const Vector &v) const;
19                Vector operator*(const double &a) const;
20                int getdim() const;
21                void print() const;
22                double operator()(const int &i, const int &j) const;
23                void go_zero(const int &k);
24                friend class Sparse_Matrix;
25                vector<double> getelements() const;
26                double infinity_norm() const;
27                double l2_norm() const;
28                double l1_norm() const;
29                void projection(); //投影到与kernel正交的子空间
30                void copy(const Vector& other);
31                void print_to_file(const string &filename);
32            };
33
```

Listing 2: 运算符实现

实现了向量的移动构造函数、向量的赋值（避免拷贝）向量的加法、减法，计算向量的范数。对于二维的 Poisson 问题对应的向量，可以视为一维向量对应的张量积，也支持 $v(i,j)$ 的操作取出网格 (ih, jh) 上的元素。

3. **限制算子和插值算子**具体的实现见 `report.tex`，保存在 `src` 下的 `restriction` 和 `prolongation` 文件夹下。

4. **Multigrid 类**

```
1                 template<int dim>
2                 class Multigrid{
3                     private:
4                     map<int, couplet> discretors; //int 代表网格数目，如2代表2的网格数，4代表4的网格数
5                     int n;  //网格个数
6                     Vector solutions;
7                     BoundaryCondition BC;
8                     int counter=1; //迭代次数
9                     string C;
10                    unique_ptr<Retriction<dim>> restriction;
11                    unique_ptr<Prolongation<dim>> prolongation;
12                    Vector w_Jacobi(int i, const Vector &initial);
13                    Vector V_cycle(const int &n, Vector& initial_guess, int nu1, int nu2);
14                    Vector FMG(const int &_n, int nu1, int nu2);
15                    vector<double> corsa_solve(const int &i);
16                    void create_grids_D(const Function &f, const Function &g,const int &i);
17                    void create_grids_N(const Function &f, const Function &g,const int &i);
18                    void create_grids_M(const Function &f, const Function &g,const int &i,
19                    const vector<int> &mixed=vector<int> {0,0,0,0});
20                    Vector error(const Function &f);
21
22                    public:
23                    Multigrid();
24                    Multigrid(const Function &f, const Function &g, BoundaryCondition bc, const int &i,
25                    const vector<int> &mixed=vector<int>{0,0,0,0});
26                    void solve(const string &restriction, const string &prolongation, const string &cycle,
    Vector& initial_guess, int nu1, int nu2,
27                    double tol,const double &value=0.0, int max_itr=50);
28                    void print();
29                    void print_to_file(const string &filename, const Function &f);
30                    void print_solution(const string &filename) const;
31                };
32
33
```

Listing 3: 运算符实现

本次作业的核心。实现了 FMG 和 V-cylce 两种方法，支持用户输入 $-\delta u = f$ 中的右端项 f，边界条件，网格数目，限制算子，插值算子，初值，相对误差，最大迭代次数。如果时 Neumann 边值条件，应当给出原函数在 $(0,0)$ 上的值。如果是 $Mixed$ 边界条件，还应额外给出在边界上的具体边界条件，用 `vector<int>` 给出边界条件，0 代表 Dirichlet 边界条件，1 代表 Neumann 边界条件。其中函数 `w_Jacobi` 实现松弛操作，之后在 $n = 4$ 的网格上用 Gauss-Sediel 迭代求出真实解。函数 `FMG V-cycle` 分别实现 FMG 和 v-cycle。

1.2 测试函数

测试函数全部定义在 `testFunction.h` 中。

1.3 输入和输出

由于测试用例很多，采用 `jsoncpp` 控制输入和输出。

- **输入：**输入文件放在 `input` 文件夹下，输入文件格式如下：

```
        {
        "dimension" :1,
        "grid_number" :256,
        "boundary_conditions":"Mixed",
        "mixed":[0,1],
        "restriction_operator": "full_weighting",
        "interpolation_operator": "linear",
        "cycle_type": "FMG",
        "max_iterations": 50,
        "relative_accuracy": 1e-8,
        "initial_guess": "zero"
      },

```

- **输出:** 最终程序运行的具体结果写在 `output` 文件夹下的 `output.json` 文件中，格式如下：

```
          {
        "boundary_condition": 1,
        "cycle": "FMG",
        "dimension": 1,
        "infinity_norm": 9.397917202669248e-06,
        "iterator times": 5,
        "l1_norm": 0.0016597179004913531,
        "l2_norm": 0.00011345851360180907,
        "number": 256
      },


```

由于本次测试数据规模很大，没有输出真实解和数值解，只输出了无穷范数和 l_1norm, l_2norm 来判断求解器的效果。