Design Document # Project02

李之琪

1 问题描述

在区域 $\Omega = (0,1)$ 或区域 $\Omega = (0,1)^2$ 上求解 Possion 方程

$$\begin{cases} -\Delta\varphi = f, & \text{on } \Omega \\ a\varphi + b\frac{\partial\varphi}{\partial n} = F, & \text{on } \partial\Omega \end{cases} \tag{1}$$

以下内容应由用户指定或给出：

(1) 边界条件类型：Dirichlet 条件, Neumann 条件或混合边界条件；

(2) 右端项：上式中的右端项函数 f 与边界函数 F；

(3) 限制算子：full-weighting 算子或 injection 算子；

(4) 插值算子：线性插值算子或二次插值算子；

(5) 松弛算子：weighted Jacobi 迭代或 Gauss-Seidel 迭代；

(6) 多重网格方法：V-cycle 或 full multigrid cycle；

(7) 迭代停止条件：最大迭代次数与相对残差限；

(8) 初值条件：零初值；

(9) 底层求解器：Jacobi 迭代。

我们在 $n = 32, 64, 128, 256$ 的网格上对多重网格算法进行测试，计算误差与收敛精度，并将其所需的 CPU 时间与直接使用 LU 分解的求解方法进行对比。

2 程序设计

我们给出在规则区域上用多重网格求解 Possion 方程的程序设计。

- `#include <RectDomain.h>`
- `#include <Tensor.h>`
- `using Real = double;`
- `template <class T>`
  `using Vector = std::vector<T>;`

class ScalarFunction

- 函数 $\mathbb{R}^{\mathtt{Dim}} \to \mathbb{R}$ 的基类，用来表达方程右端项函数和边界函数。
- **模板：** `template<int Dim>`:
  `Dim` 表示空间维数。
- **成员函数：**

  (1) `virtual const Real operator()(const Vec<Real,Dim>& pt) const = 0;`
  Public 成员函数
  输入： `pt` 表示 `Dim` 维空间中的一个点。
  输出： 函数在 `pt` 点处的函数值。
  作用： 计算函数在离散格点上的值。纯虚函数，需要在继承类中重载实现。

class FuncFiller

- 用函数填充网格上的离散格点。
- **模板：** `template<int Dim>`:
  `Dim` 表示空间维数。
- **成员变量：**

  (1) `RectDomain<Dim> domain`: **`Protected` 成员变量**，进行函数填充的规则网格。
- **成员函数：**

(1) `void fill(Tensor<Real,Dim>& res, const ScalarFunction<Dim>* pfunc) const;`
Public 成员函数
输入： `res` 为待填充的 `Dim` 维 Tensor；`pfunc` 为用来进行填充的函数的指针。
作用： 将函数 `*pfunc` 在对应位置上的函数值填入 `res`。

class GhostFiller

- 根据不同边界条件填充 ghost cell。
- **模板：** `template<int Dim>`:
  `Dim` 表示空间维数。
- `public:`
  `enum side{ low = -1, high = 1 };`
- **成员变量：**

(1) `RectDomain<Dim> domain`: **`Protected` 成员变量**, 进行 ghost cell 填充的规则网格。

- **成员函数：**

(1) `void fillOneSide(Tensor<Real,Dim>& res, int dim, side s, const char BCType, const ScalarFunction<Dim>* pfunc = nullptr) const;`
Private 成员函数
输入： `res` 为待填充的 `Dim` 维 Tensor；`dim` 和 `s` 表示填充 ghost cell 的位置；`BCType` 是填充 ghost cell 使用的边界条件类型，用 'D' 和 'N' 分别表示 Dirichlet 和 Neumann 边界条件；`pfunc` 为用来填充 ghost cell 的函数的指针，用空指针表示齐次边界条件。
作用： 根据给定的边界条件类型和边界函数，在 `res` 中对一条边界填充 ghost cell。

(2) `void fillAllSides(Tensor<Real,Dim>& res, const char* const BCTypes, const std::array<ScalarFunction<Dim>*,3> pfuncs = std::array<ScalarFunction<Dim>*,3>{}) const;`
Public 成员函数
输入： `res` 为待填充的 `Dim` 维 Tensor；`BCTypes` 是填充 ghost cell 使用的边界条件类型，用一个 `char` 数组表示；`pfuncs` 为用来填充 ghost cell 的函数指针的 array，`pfuncs[0]`、`pfuncs[1]` 和 `pfuncs[2]` 分别表示 `F`、`Fx` 和 `Fy` 函数，用空指针表示齐次边界条件。

作用： 根据给定的边界条件类型和边界函数，在 `res` 中对全部边界填充 ghost cell。通过调用 `fillOneSide` 实现。

class Laplacian

- 规则区域上的算子类，专门针对 Laplacian 算子进行处理。
- **模板：** `template<int Dim>`:
  `Dim` 表示空间维数。
- **成员变量：**

  (1) `RectDomain<Dim> domain`：**Protected 成员变量**，进行算子作用的规则网格。

  (2) `Smoother<Dim>* psmoother`：**Protected 成员变量**，松弛算子的指针。

- **成员函数：**

  (1) `void apply(const Tensor<Real,Dim>& phi, Tensor<Real,Dim>& res) const;`
  Public 成员函数
  输入： `phi` 为待进行 Laplacian 算子作用的网格离散数据；`res` 为待填入数据的 `Dim` 维 Tensor，要填入的数据是对 `phi` 作用 Laplacian 算子的结果。
  作用： 在 `res` 中填充对 `phi` 作用 Laplacian 算子的结果。

  (2) `void smooth(const Tensor<Real,Dim>& phi, const Tensor<Real,Dim>& rhs, Tensor<Real,Dim>& res) const;`
  Public 成员函数
  输入： `phi` 为待进行松弛的网格离散数据；`rhs` 为离散形式的方程右端项；`res` 为待填入数据的 `Dim` 维 Tensor，要填入的数据是对 `phi` 进行一次松弛的结果。
  作用： 进行一次松弛，并将结果填入 `res`。直接调用成员变量 `psmoother` 的 `apply` 函数实现。

  (3) `void computeResidul(const Tensor<Real,Dim>& phi, const Tensor<Real,Dim>& rhs, Tensor<Real,Dim>& res) const;`
  Public 成员函数
  输入： `phi` 为待进行 Laplacian 算子作用的网格离散数据；`rhs` 为离散形式的方程右端项；`res` 为待填入数据的 `Dim` 维 Tensor，要填入的数据是残差计算的结果。
  作用： 计算 $-\Delta$ `phi` $=$ `rhs` 的残差 `rhs` $+ \Delta$ `phi`，并将结果填入 `res`。通过调用 `apply` 实现。

class Smoother

- 松弛算子的基类，针对 Laplacian 算子进行松弛操作。
- **模板：** `template<int Dim>`:
  `Dim` 表示空间维数。
- **成员变量：**

  (1) `RectDomain<Dim> domain`: **`Protected` 成员变量**，进行松弛操作的规则网格。
- **成员函数：**

  (1) `virtual void apply(const Tensor<Real,Dim>& phi, const Tensor<Real,Dim>& rhs, Tensor<Real,Dim>& res) const = 0;`
  Public 成员函数
  输入： 与 `Laplacian` 类的成员函数 `smooth` 一致。
  作用： 进行一次松弛，并将结果填入 `res`。纯虚函数，需要在继承类中进行实现。

class WeightedJacobi

- 针对 Laplacian 算子的 weighted Jacobi 松弛算子。
- **模板：** `template<int Dim>`:
  `Dim` 表示空间维数。
- **继承：** `class WeightedJacobi: public Smoother<Dim>`。
- **成员变量：**

  (1) `const Real weight = 2.0/3`: **`Protected` 成员变量**, weighted Jacobi 松弛的权重。
- **成员函数：**

  (1) `void apply(const Tensor<Real,Dim>& phi, const Tensor<Real,Dim>& rhs, Tensor<Real,Dim>& res) const;`
  Public 成员函数
  输入： 同基类一致。
  作用： 进行一次 weighted Jacobi 松弛，并将结果填入 `res`。

class PossionDirectSolver

- Possion 方程的直接求解器，用于在最底层的网格上进行 Possion 方程求解。
- **模板：** `template<int Dim>`:
  `Dim` 表示空间维数。
- **成员变量：**
  (1) `RectDomain<Dim> domain`：**`Protected` 成员变量**，进行 Possion 方程求解的规则网格。
- **成员函数：**
  (1) `void solve(Tensor<Real,Dim>& phi, const Tensor<Real,Dim>& rhs, const char* const BCTypes)`
  Public 成员函数
  输入： `phi` 为待求解方程的初值；`rhs` 为离散形式的方程右端项；`BCTypes` 是方程的边界条件类型，用一个 `char` 数组表示。
  作用： 在齐次边界条件下求解 Possion 方程 $-\Delta$ `phi` $=$ `rhs`，并将结果填入 `phi`。

class Restrictor

- 在相邻层级的网格上进行限制算子的基类。
- **模板：** `template<int Dim>`:
  `Dim` 表示空间维数。
- **成员变量：**
  (1) `RectDomain<Dim> coarseDomain`：**`Protected` 成员变量**，两级网格中较粗的网格。
  (2) `RectDomain<Dim> fineDomain`：**`Protected` 成员变量**，两级网格中较细的网格。
- **成员函数：**
  (1) `virtual void apply(const Tensor<Real,Dim>& data, Tensor<Real,Dim>& res) const = 0;`
  Public 成员函数
  输入： `data` 为细网格上的数据；`res` 为待填入数据的 `Dim` 维 Tensor，要填入的是对 `data` 作用限制算子后得到的粗网格上的数据。

作用： 对 `data` 作用限制算子并将结果填入 `res`。纯虚函数，需要在继承类中进行实现。

class Injection

- 从细网格到粗网格的 injection 算子。
- **模板：** `template<int Dim>`：
  `Dim` 表示空间维数。
- **继承：** `class Injection: public Restrictor<Dim>`。
- **成员函数：**

  (1) `void apply(const Tensor<Real,Dim>& data, Tensor<Real,Dim>& res) const;`
  Public 成员函数
  输入： 同基类一致。
  作用： 对 `data` 作用 injection 算子并将结果填入 `res`。

class FullWeighting

- 从细网格到粗网格的 full-weighting 算子。
- **模板：** `template<int Dim>`：
  `Dim` 表示空间维数。
- **继承：** `class FullWeighting: public Restrictor<Dim>`。
- **成员函数：**

  (1) `void apply(const Tensor<Real,Dim>& data, Tensor<Real,Dim>& res) const;`
  Public 成员函数
  输入： 同基类一致。
  作用： 对 `data` 作用 full-weighting 算子并将结果填入 `res`。

class Interpolator

- 在相邻层级的网格上进行插值算子的基类。
- **模板：** `template<int Dim>`：
  `Dim` 表示空间维数。

- **成员变量：**

  (1) `RectDomain<Dim> coarseDomain`：**Protected 成员变量**，两级网格中较粗的网格。

  (2) `RectDomain<Dim> fineDomain`：**Protected 成员变量**，两级网格中较细的网格。

- **成员函数：**

  (1) `virtual void apply(const Tensor<Real,Dim>& data, Tensor<Real,Dim>& res) const = 0;`
  Public 成员函数
  输入： `data` 为粗网格上的数据；`res` 为待填入数据的 `Dim` 维 Tensor，要填入的是对 `data` 作用插值算子后得到的细网格上的数据。
  作用： 对 `data` 作用插值算子并将结果填入 `res`。纯虚函数，需要在继承类中进行实现。

class LinearInterpolator

- 从粗网格到细网格的线性插值算子。
- **模板：** `template<int Dim>`：
  `Dim` 表示空间维数。
- **继承：** `class LinearInterpolator: public Interpolator<Dim>`。
- **成员函数：**

  (1) `void apply(const Tensor<Real,Dim>& data, Tensor<Real,Dim>& res) const;`
  Public 成员函数
  输入： 同基类一致。
  作用： 对 `data` 作用线性插值算子并将结果填入 `res`。

class QuadraticInterpolator

- 从粗网格到细网格的二次插值算子。
- **模板：** `template<int Dim>`：
  `Dim` 表示空间维数。
- **继承：** `class QuadraticInterpolator: public Interpolator<Dim>`。
- **成员函数：**

(1) `void apply(const Tensor<Real,Dim>& data, Tensor<Real,Dim>& res) const;`
Public 成员函数
输入： 同基类一致。
作用： 对 `data` 作用二次插值算子并将结果填入 `res`。

class MultigridSolver

- 多重网格法求解 Poisson 方程的求解器。
- **模板：** `template<int Dim>`:
  `Dim` 表示空间维数。
- ```
  struct MGParam{
  int numPreIter;
  int numPostIter;
  int numBottomIter;
  Real reltol;
  int maxIter;
  };
  ```
- ```
  using VPR = Vector<Restriction<Dim>*>;
  using VPI = Vector<Interpolation<Dim>*>;
  ```
- **成员变量：**

  (1) `Vector<RectDomain<Dim> > vDomain`: **Protected 成员变量**，从细到粗一系列层级的规则网格。

  (2) `Vector<GhostFiller<Dim> > vGhostFiller`: **Protected 成员变量**，一系列层级的规则网格上的 ghost cell 填充器。

  (3) `Vector<Laplacian<Dim> > vLaplacian`: **Protected 成员变量**，一系列层级的规则网格上的算子。

  (4) `VPR vpRestriction`: **Protected 成员变量**，一系列相邻层级规则网格上的限制算子的指针。

  (5) `VPI vpInterpolation`: **Protected 成员变量**，一系列相邻层级规则网格上的插值算子的指针。

  (6) `PossionDirectSolver<Dim> bottomSolver`: **Protected 成员变量**，在最底层的网格上进行 Possion 方程求解的求解器。

(7) `char BCTypes[4]`: **Protected 成员变量**，边界条件类型。

(8) `MGParam param`: **Protected 成员变量**, 多重网格方法参数, 用一个结构体 `MGParam` 表示。

- **成员函数:**

(1) `MultigridSolver(const Vector<RectDomain<Dim> >& avDomain, const VPR& avpRestriction, const VPI& avpInterpolation);`

Public 成员函数

输入: 见以上成员变量的说明。

作用: 构造函数, 根据输入的一系列网格和相邻网格上的限制、插值算子生成 MultigridSolver 类。

(2) `void setParam(const MGParam& aparam);`

Public 成员函数

输入: 见以上成员变量的说明。

作用: 对相关参数进行设置。

(3) `void VCycle(int depth, Tensor<Real,Dim>& phi, const Tensor<Real,Dim>& rhs) const;`

Private 成员函数

输入: `depth` 为当前的 V-cycle 所在的层数; `phi` 为待求解问题的初值; `rhs` 为离散形式的方程右端项。

作用: 在齐次边界条件下用一次 V-cycle 求解 Possion 方程 $-\Delta$ `phi` = `rhs`，并将结果填入 `phi`。

(4) `void FMVCycle(int depth, Tensor<Real,Dim>& phi, const Tensor<Real,Dim>& rhs) const;`

Private 成员函数

输入: `depth` 为当前的 full multigrid V-cycle 所在的层数; `phi` 为待求解问题的初值; `rhs` 为离散形式的方程右端项。

作用: 在齐次边界条件下用一次 full multigrid V-cycle 求解 Possion 方程 $-\Delta$ `phi` = `rhs`，并将结果填入 `phi`。

(5) `Real solve(Tensor<Real,Dim>& phi, const Tensor<Real,Dim>& rhs, const std::array<ScalarFunction<Dim>*,3> pfuncs, bool useFMVCycle = 0) const;`

Public 成员函数

输入： phi 为待求解问题的初值；rhs 为离散形式的方程右端项；pfuncs 为用来填充 ghost cell 的函数指针的 array，pfuncs[0]、pfuncs[1] 和 pfuncs[2] 分别表示 F、Fx 和 Fy 函数，用空指针表示齐次边界条件；useFMVCycle 表示是否使用 full multigrid V-cycle 迭代进行方程的求解，默认使用 V-cycle 迭代。
输出： 迭代停止时的相对残差。
作用： 用 V-cycle 迭代或 full multigrid V-cycle 迭代求解 Possion 方程 $-\Delta$ phi $=$ rhs，并将结果填入 phi 中。通过调用 VCycle 或 FMVCycle 函数实现。迭代停止的条件是相对残差小于 reltol 或迭代次数达到 maxIter。

class MGFactory

- MultigridSolver 类的对象工厂。
- **模板：** template<int Dim>：
  其中 Dim 表示维数。
- 对象工厂的设计如下：

```
template <int Dim >
class MGFactory{
public:
  using CreateMultigridSolverCallback =
  std::unique_ptr<MultigridSolver< Dim > >
    (*)(RectDomain< Dim >, const char*);
private:
  using CallbackMap = std::map<std::string, CreateMultigridSolverCallback>;
public:
  static MGFactory& createFactory(){
    static MGFactory object;
    return object;
  }

  bool registerMultigridSolver(std::string multigridId, CreateMultigridSolverCallback createFn){
    return callbacks_.insert(typename CallbackMap::value_type(multigridId, createFn)).
      second;
  }

  bool unregisterMultigridSolver(std::string multigridId){
    return callbacks_.erase(multigridId) == 1;
  }

  template<class ... TS >
  std::unique_ptr<MultigridSolver< Dim > > createMultigridSolver(std::string
                                                multigridId, TS && ...args){
    auto it = callbacks_.find(multigridId);
    if (it == callbacks_.end())
      {
        throw std::runtime_error("Unknown MultigridSolver ID. ");
```

```
31        }
32      return (it->second)(std::forward< TS >(args)...);
33    }
34
35  private:
36    MGFactory() = default;
37    MGFactory(const MGFactory&) = default;
38    MGFactory& operator=(const MGFactory&) = default;
39    ~MGFactory() = default;
40
41  private:
42    CallbackMap callbacks_;
43
44  };
```

class TestMultigrid

- 对实现的多重网格进行各项测试，包括参数输入、对结果进行后处理、网格加密测试等。
- **模板：** `template<int Dim>`：
  其中 `Dim` 表示维数。
- **成员变量：**
- `std::unique_ptr<MultigridSolver<Dim> > pMG`：**Protected 成员变量**，通过对象工厂生成的 MultigridSolver 指针，用于后续处理与测试。
- `std::string multigridID`：**Protected 成员变量**，用字符串形式表示的多重网格 ID，用于生成 MultigridSolver 指针。
- **成员函数：**

(1) `TestMultigrid(const std::string& jsonfile);`

Public 成员函数

输入： `jsonfile` 以字符串格式给出一个 .json 文件的名字。

作用： 构造函数，根据输入的 .json 文件，通过对象工厂生成一个 MultigridSolver 指针作为成员变量。

(2) `Tensor<Real,Dim> solve(const ScalarFunction<Dim>* pfunc, const std::array<ScalarFunction<Dim>*,3> pfuncs, bool useFMVCycle = 0) const;`

Public 成员函数

输入： `pfunc` 为用来进行右端项填充的函数的指针；`pfuncs` 为用来填充 ghost cell

的函数指针的 array，pfuncs[0]、pfuncs[1] 和 pfuncs[2] 分别表示 F、Fx 和 Fy 函数，用空指针表示齐次边界条件；useFMVCycle 表示是否使用 full multigrid V-cycle 迭代进行方程的求解，默认使用 V-cycle 迭代。

输出： 以 Tensor 的形式输出求解结果。

作用： 用 V-cycle 迭代或 full multigrid V-cycle 迭代求解 Possion 方程 $-\Delta$ phi $=$ rhs。

(3) `Real computeError(const Tensor<Real,Dim>& res, const ScalarFunction<Dim>* pfunc, const int p = 0) const;`

Public 成员函数

输入： res 表示方程的求解结果；pfunc 表示精确解函数的指针；p 表示误差范数的类型，0 表示无穷范数，1 表示 1-范数，2 表示 2-范数。

输出： 求解结果与精确解间的误差范数。

作用： 计算求解结果与精确解间的误差范数。

(4) `void plot(const Tensor<Real,Dim>& res, const std::string &file) const;`

Public 成员函数

输入： res 表示方程的求解结果；file 以字符串格式给出一个 .m 文件的名字。

作用： 以 file 为名字的 .m 文件被写入 matlab 代码，可以通过在 matlab 下运行该文件得到求解结果的可视化结果。

(5) `void test(const ScalarFunction<Dim>* pfunc, const std::array<ScalarFunction<Dim>*,3> pfuncs, const int numEncryp, bool useFMVCycle = 0) const;`

Public 成员函数

输入： pfunc、pfuncs 和 useFMVCycle 与 solve 中一致，numEncryp 是网格加密次数。

作用： 对 V-Cycle 或 full multigrid V-cycle 进行网格加密测试，并将结果输出到屏幕上。

3 UML 图

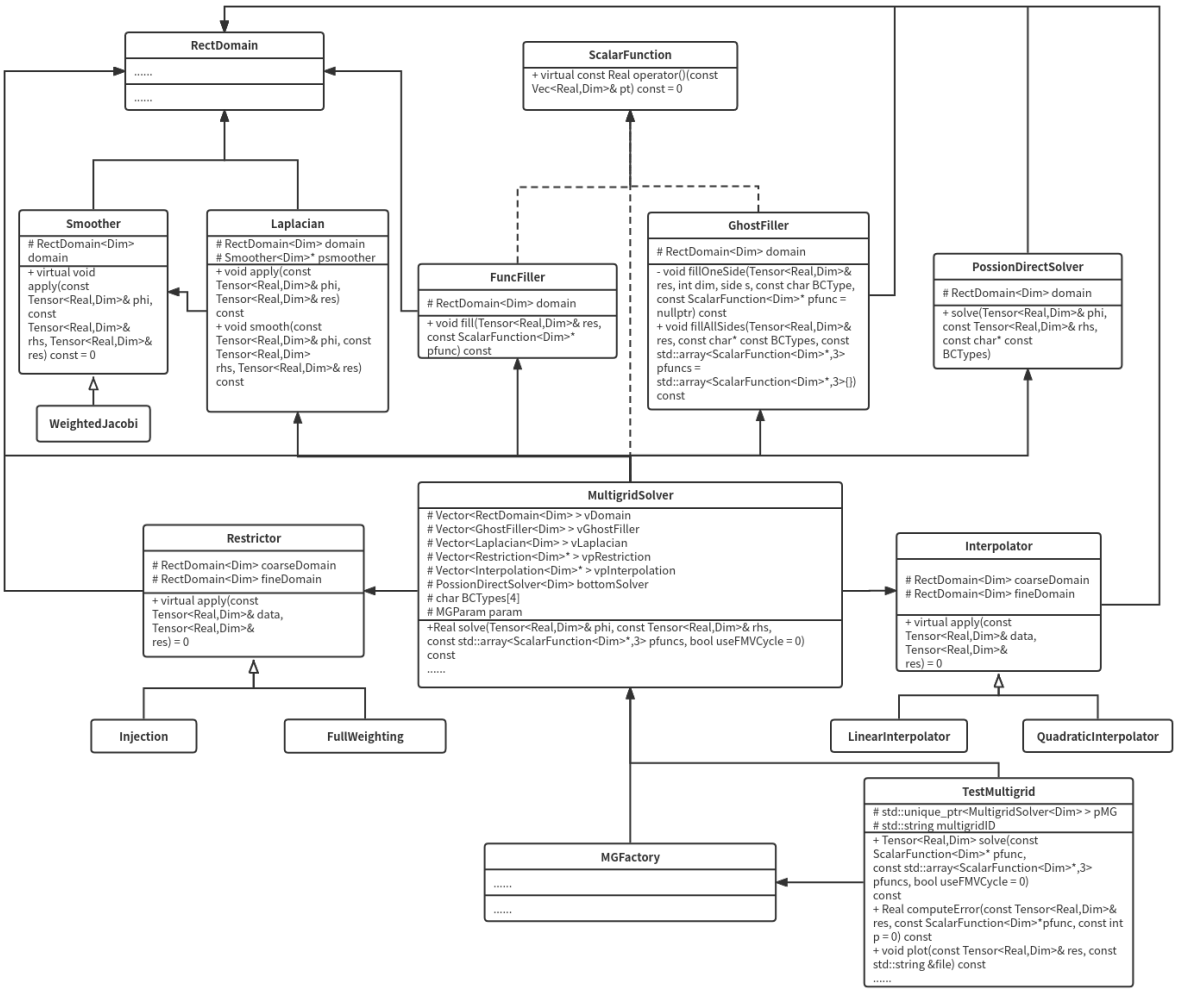

图 1: UML 图