

规则区域上的多重网格程序设计

李之琪

May 12, 2022

- 问题描述
- UML 图
- 程序设计
- 结果展示

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

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

问题描述

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 V-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。



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

class ScalarFunction

- 函数 $\mathbb{R}^{\text{Dim}} \rightarrow \mathbb{R}$ 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 BTypes, 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; BTypes 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 $-\Delta \phi = \text{rhs}$ 的残差 $\text{rhs} + \Delta \phi$ ，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 $-\Delta \phi = \text{rhs}$, 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**:
 - ④ `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 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 LinearInterpolation

- 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 LinearInterpolation: public Interpolator<Dim>`。
- 成员函数**:
 - ❶ `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 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 Poisson 方程 $-\Delta \phi = \text{rhs}$, 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 Poisson 方程 $-\Delta \phi = \text{rhs}$, 并将结果填入 `phi`。

程序设计

```
1  template <int Dim>
2  void MultigridSolver<Dim>::VCycle(int depth, Tensor<Real,Dim>
   & phi, const Tensor<Real,Dim>& rhs) const{
3      if (depth == (int)(vDomain.size()) - 1){
4          bottomSolver.Solve(phi, rhs, BCTypes);
5          return;
6      }
7      // relax numPreIter times
8      vGhostFiller[depth].fillAllSides(phi, BCTypes);
9      Tensor<Real,Dim> rsd(vDomain[depth].getGhostedBox());
10     vLaplacian[depth].computeResidual(phi, rhs, rsd);
11     vGhostFiller[depth].fillAllSides(rsd, BCTypes);
12     Tensor<Real,Dim> crsd(vDomain[depth+1].getGhostedBox());
13     vpRestriction[depth]->apply(rsd, crsd);
14     Tensor<Real,Dim> cphi(vDomain[depth+1].getGhostedBox());
15     cphi = 0.0;
16     VCycle(depth+1, cphi, crsd);
17     vpInterpolation[depth]->apply(cphi, phi);
18     // relax numPostIter times
19     vGhostFiller[depth].fillAllSides(phi, BCTypes);
20 }
```



```
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 Poisson 方程 $-\Delta \phi = \text{rhs}$ ，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。

程序设计

```
1  template <int Dim>
2  class MGFactory{
3  public:
4      using CreateMultigridSolverCallback = std::unique_ptr<MultigridSolver<Dim> >(*)(
          RectDomain<Dim>, const char*);
5  private:
6      using CallbackMap = std::map<std::string, CreateMultigridSolverCallback>;
7  public:
8      static MGFactory& createFactory(){
9          static MGFactory object;
10         return object;
11     }
12     bool registerMultigridSolver(std::string multigridId,
13         CreateMultigridSolverCallback createFn);
14
15     bool unregisterMultigridSolver(std::string multigridId);
16
17     template<class ...TS>
18     std::unique_ptr<MultigridSolver<Dim>>
19     createMultigridSolver(std::string multigridId, TS && ...args);
20
21 private:
22     MGFactory() = default;
23     MGFactory(const MGFactory&) = default;
24     MGFactory& operator=(const MGFactory&) = default;
25     ~MGFactory() = default;
26
27 private:
28     CallbackMap callbacks_;
29 };
```

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**:
 - ① `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。
 - ② `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 、 F_x 和 F_y 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 Poisson 方程 $-\Delta \phi = \text{rhs}$ 。

- 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。

结果展示

```
Grid 256X256:  
Iter 1 , relrsd = 0.0484418 , rsdratio = 0.0484418  
Iter 2 , relrsd = 0.0113953 , rsdratio = 0.235236  
Iter 3 , relrsd = 0.0028764 , rsdratio = 0.252421  
Iter 4 , relrsd = 0.000786047 , rsdratio = 0.273275  
Iter 5 , relrsd = 0.000229346 , rsdratio = 0.291771  
Iter 6 , relrsd = 6.72816e-05 , rsdratio = 0.293363  
Iter 7 , relrsd = 1.99039e-05 , rsdratio = 0.29583  
Iter 8 , relrsd = 6.04032e-06 , rsdratio = 0.303474  
Iter 9 , relrsd = 1.89266e-06 , rsdratio = 0.313338  
Iter 10 , relrsd = 5.93902e-07 , rsdratio = 0.313792  
Iter 11 , relrsd = 1.86809e-07 , rsdratio = 0.314545  
Iter 12 , relrsd = 5.909e-08 , rsdratio = 0.316312  
Iter 13 , relrsd = 1.90564e-08 , rsdratio = 0.322497  
Iter 14 , relrsd = 6.18376e-09 , rsdratio = 0.324499
```

Figure: 屏幕输出

结果展示

Domain: $(0, 1)^2$, BCtypes : D , D , D , D

n	32	ratio	64	ratio	128	ratio	256
$\ E\ _1$	4.391e-06	2.053	1.058e-06	2.028	2.594e-07	2.016	6.415e-08
$\ E\ _2$	6.653e-06	2.053	1.604e-06	2.027	3.932e-07	2.016	9.721e-08
$\ E\ _\infty$	3.095e-05	1.985	7.818e-06	1.996	1.960e-06	2.001	4.897e-07

Table: V-cycle test, Injection, LinearInterpolation, $F = e^{xy}$.

结果展示

Domain: $(0, 1)^2$, BCtypes : N , D , D , D

n	16	ratio	32	ratio	64
$\ E\ _1$	1.759e-05	2.037	4.287e-06	2.024	1.054e-06
$\ E\ _2$	2.289e-05	2.045	5.546e-06	2.026	1.361e-06
$\ E\ _\infty$	7.501e-05	1.975	1.908e-05	1.995	4.785e-06
CPU time(s)	0.009	4.887	0.267	5.749	14.375

Table: LU 分解, $F = e^{xy}$

n	16	ratio	32	ratio	64	ratio	128
$\ E\ _1$	1.759e-05	2.037	4.287e-06	2.024	1.054e-06	2.013	2.611e-07
$\ E\ _2$	2.289e-05	2.045	5.546e-06	2.027	1.361e-06	2.014	3.370e-07
$\ E\ _\infty$	7.501e-05	1.975	1.908e-05	1.995	4.785e-06	1.999	1.197e-06
CPU time(s)	0.048	1.131	0.106	1.566	0.313	2.009	1.258

Table: V-cycle test, Injection, LinearInterpolation, $F = e^{xy}$.

谢谢大家!