MARS2D 程序设计文档

下文中 Vector 表示 std::vector，List 表示 std::list，Point 表示 Vec<Real, Dim>。蓝色的文字表示是为了能够同时实现 Vector 和 List 而改变的。

1 Vector 和 List 的区别

1.1 Vector

在内存中分配一块连续的内存空间进行存储，相对于 List 更节省空间。支持随机访问，能通过 operator[] 进行对数据成员的访问，随机访问的时间复杂度为 $O(1)$，并且其 iterator 支持 operator+ 操作。但在进行插入和删除操作时会造成内存块的拷贝，效率较低，时间复杂度为 $O(n)$。

1.2 List

每个数据节点包括一个信息块、一个前驱指针和一个后驱指针，便于进行序列内部的插入和删除操作，时间复杂度为 $O(1)$。但不支持随机访问，访问指定数据的时间复杂度为 $O(n)$，其 iterator 也只支持 operator++。在创建大体量的数据时，List 比 Vector 要慢很多，因为其内存不是连续的。

1.3 将 Vector 替换为 List 对程序性能的影响

首先由于加减点时会调用容器的 emplace（插入成员函数）和 erase（删除成员函数），所以使用 List 在这方面可以降低时间复杂度。在随机访问方面，程序中仅在计算相邻点间弦长时调用了点列中的点坐标信息，而能否随机访问并不影响其时间复杂度。而对于 iterator 是否重载了 operator+，程序中在进行加点时确实调用了 Vector 迭代器的 operaotr+，但可以通过循环调用 operator++ 解决，时间复杂度应和加点个数同阶。在创建临时变量方面，List 会更慢一些，具体影响程度还需验证。

在实现上，程序中统一使用 iterator 来进行点列的遍历；Vector 和 List 的 emplace、erase 接口是一样的，不需要改变；在进行 iterator 移位时，统一用循环调用 operator++ 来实现。这样一套程序就可以同时实现 Vector 和 List 两个版本。

2 class VectorFunction

- 函数 $\mathbb{R}^{\mathtt{Dim}} \times \mathbb{R} \to \mathbb{R}^{\mathtt{Dim}}$ 的基类，可以作为速度场的基类使用。

- **模板：** `template<int Dim>`：
  `Dim` 表示空间维数。

- **成员函数：**

  (1) `virtual const Point operator()(const Point &pt, Real t) const = 0`：
  输入： `pt` 为当前点坐标，`t` 为当前时间。
  输出： `pt` 点处的速度场。

3 class TimeIntegrator

- 时间积分方法的基类。

- **模板：** `template<int Dim>`：
  `Dim` 表示空间维数。

- **成员函数：**

  (1) `virtual const Point timeStep(const VectorFunction<Dim> &v, const Point &pt, Real tn, Real k) = 0`：
  输入： `v` 为速度场，`pt` 为当前点坐标，`tn` 为当前时间，`k` 为时间步长。
  输出： 新的点坐标。
  作用： 使得 `pt` 在速度场 `v` 的作用下运动 `k` 时间。

  (2) `template<template<typename...>class Container>`
  `void timeStep(const VectorFunction<Dim> &v, Container<Point> &pts, Real tn, Real k)`：
  输入： `v` 为速度场，`pts` 表示一列 Point ，tn 为当前时间，`k` 为时间步长。
  输出： `void`，原址更改 pts。
  作用： 函数使得 pts 中的一列点在速度场 `v` 的作用下运动 `k` 时间，调用单点版本的 `timeStep` 成员函数进行实现。

4 ButcherTableau

4.1 enum RK_Category1

- `enum RK_Category1{ERK=1, DIRK, ARK, nRK_Family}`。

- **作用：**表示 RK 方法的一般类型。

4.2 enum RK_Category2

- `enum RK_Category2{ForwardEuler=1, ClassicRK4, nRK_Type}`。
- **作用：**表示 RK 方法的细分类型。

4.3 struct ButcherTableau

- **模板：**`template<RK_Category1 Type1, RK_Category2 Type2>`：
  `Type1` 表示 RK 的一般类型，如 ERK, DIRK, ARK 等；`Type2` 表示 RK 的细分类型，如 `Type1`=ERK 时，`Type2` 可以是 ForwardEuler, ClassicRK4 等。对于给定的 `RK_Category1` 中的一般类型，其数据结构是固定的。
- **特例化：**

(1)
```
template <>
struct ButcherTableau<ERK, ForwardEuler>
{
  static constexpr int nStages = 1;
  static constexpr Real a[nStages][nStages] = ...;
  static constexpr Real b[nStages] = ...;
  static constexpr Real c[nStages] = ...;
};
```

(2)
```
template <>
struct ButcherTableau<ERK, ClassicRK4>
{...};
```

5 class ExplicitRungeKutta

- 继承自 `TimeIntegrator<Dim>`，用于实现所有 ERK 方法。
- **模板：**`template<int Dim, RK_Category2 Type>`：
  `Dim` 表示空间维数，`Type` 是 ERK 方法的某个子方法。这里可以这么做是因为所有 ERK 方法的数据结构都是一致的，所以在 `class ExplicitRungeKutta` 中，只需要知道所用的是哪种子方法就可以建立对应的 Butcher 表，进而实现对应的 ERK 方法。

- `using ButcherTab = ButcherTableau<ERK, Type>;`

- **成员函数：**

  (1) `const Point timeStep(const VectorFunction<Dim> &v, const Point &pt, Real tn, Real k)`：
  输入： 同时间积分方法中的纯虚函数 `timeStep` 一致。
  输出： 同时间积分方法中的纯虚函数 `timeStep` 一致。
  作用：调用 `ButcherTab` 中的成员常量实现对应的 ERK 方法，进一步实现 `TimeIntegrator<Dim>` 中的纯虚函数 `timeStep`。

6 class MARS

- 殷集界面追踪方法的基类。

- **模板：** `template<int Dim, int Order>`：
  其中 `Dim` 表示维数，`Order` 表示殷集边界表示的阶数。

- **成员变量：**

  (1) `TimeIntegrator<Dim> *TI`：**Protected 成员变量**，时间积分方法基类指针。

- **成员函数：**

  (1) `MARS(TimeIntegrator<Dim> *_TI):TI(_TI){}`：
  构造函数。

  (2) `virtual void timeStep(const VectorFunction<Dim> &v, YinSet<Dim, Order> &ys, Real tn, Real k) = 0`：
  输入： `v` 为速度场，`ys` 为殷集，`tn` 和 `k` 的定义同时间积分方法中一致
  输出： `void`，原址更改 `ys`。
  作用： 纯虚函数，作为二维和三维 MARS 方法 `timeStep` 的公共接口，在继承类中进行实现。函数将 `tn` 时刻的殷集映射到 `k` 时间后的殷集并赋值给 `ys`。

  (3) `void trackInterface(const VectorFunction<Dim> &v, YinSet<Dim,Order> &ys, Real startTime, Real k, Real endTime)`：
  输入： `v` 为速度场，`ys` 为殷集，`startTime` 和 `endTime` 表示 MARS 方法作用的起止时间，`k` 为时间步长。
  输出： `void`，原址更改 `ys`。

作用： 在速度场 v 的作用下，将 startTime 时刻的殷集 ys 通过时间步长为 k 的 MARS 方法映射到 endTime 时刻的殷集并赋值给 ys。调用 timeStep 在此基类中进行实现。

7 class MARS2D

- 二维殷集的界面追踪方法。
- **模板：** template<int Order, template<typename...> class Container>:
  其中 Order 表示二维殷集边界所用样条曲线的阶数，Container 表示某种 STL 容器，用于存储示踪点列，目前考虑 Vector 和 List 两种。
- **继承：** class MARS2D: public MARS<2, Order>。
  using Base = MARS<2, Order>。
- **成员变量：**

  (1) Interval<1> chdLenRange：殷集边界上相邻节点间弦长取值范围。
- **成员函数：**

  (1) MARS2D(TimeIntegrator<2> *_TI, Real hL, Real rtiny=0.1):Base(_TI){...}:
  构造函数，使 chdLenRange 为 [rtiny*hL,hL]，rtiny 默认值为 0.1。

  (2) void discreteFlowMap(const VectorFunction<2> &v, Container<Point> &pts, Real tn, Real k)：
  Private 成员函数
  输入： v 为速度场，pts 为示踪点列，tn 和 k 的定义同时间积分方法中的一致。
  输出： void，原址更改 pts。
  作用： 函数调用 Base::TI->timeStep 的 Container 版本将 pts 映射到 k 时间后的示踪点列。

  (3) Vector<unsigned int> splitLongEdges(const VectorFunction<2> &v, Container<Point> &pts, const Curve<2, Order> &crvtn, Real tn, Real k)：
  Private 成员函数
  输入： v 为速度场，pts 为下一时刻的示踪点列，crvtn 为当前时刻对应的样条曲线，tn 和 k 的定义同时间积分方法中的一致。
  输出： pts 中新加入点的序号。
  作用： 函数在 pts 中寻找相邻点间距离过长的弦，并在 crvtn 中对应的曲线段上加

点，利用 Base::TI->timeStep 将新加入的点映射到下一时刻并添加到 pts 中 (注意这里调用的是 timeStep 的单点映射版本，避免对本身就存在于 pts 中的点进行重复操作)。

(4) `Vector<unsigned int> removeSmallEdges(Container<Point> &pts)`:

Private 成员函数

输入： pts 为下一时刻的示踪点列。

输出： 删除的节点在原 pts 中的序号。

作用： 函数将 pts 中相邻点间弦长过小的点在原址进行删除，满足不删除首尾两点、不连续删点的条件。

(5) `void timeStep(const VectorFunction<2> &v, YinSet<2,Order> &ys, Real tn, Real k)`:

输入： 同基类一致。

输出： 同基类一致。

作用： 实现基类中的纯虚函数，函数依次调用 discreteFlowMap, splitLongEdges, removeSmallEdges 和 fitCurve 实现二维 MARS 方法中的一个时间步，即预处理、流映射、后处理的复合，将当前时刻的殷集映射到 k 时间后的殷集并赋值给 ys。