MARS2D 程序设计文档

下文中 Vector 表示 std::vector，Point 表示 Vec<Real, Dim>。

1 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。

2 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 dt) = 0：
  输入： v 为速度场，pt 为当前点坐标，tn 为当前时间，dt 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 dt 时间。

  (2) virtual void timeStep(const VectorFunction<Dim> &v, Vector<Point> &pts, Real tn, Real dt)：
  输入： v 为速度场，pts 表示一列 Point ，tn 为当前时间，dt 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 dt 时间，调用单点版本的 timeStep 成员函数进行实现。

3 ButcherTableau

3.1 enum RK_Category1

- `enum RK_Category1{ERK=1, DIRK, ARK, nRK_Family}`。
- **作用：**表示 RK 方法的一般类型。

3.2 enum RK_Category2

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

3.3 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>
{...};
```

4 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 &v, const Point &pt, Real tn, Real dt)`：
  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`。

5 class MARS

- 殷集的界面追踪方法。

- **模板：** `template<int Dim, int Order>`：
  其中 `Dim` 表示维数，`Order` 表示殷集边界所用样条曲线的阶数。

- **成员变量：**

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

  (2) `Interval<1> chdLenRange`：殷集边界上相邻节点间弦长取值范围。

- **成员函数：**

  (1) `MARS()=delete`。

  (2) `MARS(TimeIntegrator *_TI, Real hL, Real rtiny=0.1)`：
  构造函数，使 `chdLenRange` 为 `[rtiny*hL,hL]`，`rtiny` 默认值为 0.1。

  (3) `void discreteFlowMap(const VectorFunction &v, Vector<Point> &pts, Real tn, Real dt)`：
  Private 成员函数
  输入： `v` 为速度场，`pts` 为示踪点列，`tn` 和 `dt` 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 TI->timeStep 的 Vector 版本将 pts 映射到 dt 时间后的示踪点列。

(4) Vector<unsigned int> removeSmallEdges(Vector<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) Vector<unsigned int> splitLongEdges(const VectorFunction &v, Vector<Point> &pts, const Curve<Dim, Order> &crvtn, Real tn, Real dt)：

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 dt 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 crv 中对应的曲线段上加点，利用 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)。

(6) void timeStep(const VectorFunction &v, YinSet<Dim,Order> &ys, Real tn, Real dt)：

输入： v 为速度场，ys 为殷集，tn 和 dt 的定义同时间积分方法中一致。

输出： void，原址更改 ys。

作用： 函数调用 discreteFlowMap, splitLongEdges，removeSmallEdges 和 fitCurve 实现 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 dt 时间后的殷集并赋值给 ys。

(7) void trackInterface(const VectorFunction &v, YinSet<Dim,Order> &ys, Real startTime, Real dt, 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，dt 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 dt 的 MARS 方法映射到 endTime 时刻的殷集并赋值给 ys。调用 timeStep 进行实现。