**template<class VectorType = Vector<double>>**

**class Solver< VectorType >**

这是个迭代线性求解器的基类。这个类提供一个接口，一方面连接内存，一方面连接一个判断求解器是否收敛的对象。

**Requirements common to derived solver classed**

一般来说，迭代求解器不依赖于任何特别的矩阵结构或存储格式。相反地，它们往往只要求矩阵和向量定义了某些操作，如矩阵乘以向量，或向量间的内积等。因而，这个类及其派生类的具体实现都采用模板方式，即其实现由矩阵和向量的具体类型决定。而这些矩阵和向量必须满足某些共有的要求才能成为允许的类型。下面我们将列出这些要求。

下面展示的类并非任何具体的类，而是为了展现一种“signature”，所有具体的类都必须遵循这些规则。dealii自带的SparseMatrix和Vector类当然是满足下面的规范的，所以可以被视为下述signature的良好例子。此外，你可以参考step-20和step-22，或命名空间LinearSolvers下的一些类，学习如何自定义像matrix一样的类，用作线性求解器中的线性算子。

具体讲，能被传给一个线性求解器的矩阵类和向量类需要提供如下接口：

|  |
| --- |
| class Matrix |
| { |
| public: |
| // 作用于向量src，结果写入dst. |
| void vmult (VectorType& dst, const VectorType& src) const; |
| // 转置作用于某个向量，结果写入dst。这个函数只在某些迭代方法中会用到 |
| void Tvmult (VectorType& dst, const VectorType& src) const; |
| }; |
| class Vector |
| { |
| public: |
| // 调整当前对象的尺寸，使其与参数model\_vector的尺寸及布局一致，第二个参数指定是否在调整之后清楚当前对象，它必须有默认值为false |
| void reinit (const Vector& model\_vector, |
| const bool leave\_elements\_uninitialized = false); |
| // 当前对象与参数间的内积 |
| double operator \* (const Vector &v) const; |
| // 向量加法 |
| void add (const Vector &x); |
| // Scaled addition of vectors |
| void add (const double a, |
| const Vector &x); |
| // Scaled addition of vectors |
| void sadd (const double a, |
| const double b, |
| const Vector &x); |
| // Scaled assignment of a vector |
| void equ (const double a, |
| const Vector &x); |
| // Combined scaled addition of vector x into the current object and |
| // subsequent inner product of the current object with v. |
| double add\_and\_dot (const double a, |
| const Vector &x, |
| const Vector &v); |
| // 当前对象的元素都乘以一个相同的数 |
| Vector & operator \*= (const double a); |
| // 返回向量的l2范数 |
| double l2\_norm () const; |
| }; |

此外，对于某些求解器，必须有一个全局函数swap(VectorType& a, VectorType& b)用于交换两个向量的值。

最后，求解器还需要一个GrowingVectorMemory<VectorType>的实例，对于内置向量类型，这些实例化可由dealii库提供，但对于用户自定义向量类型，则必须显示地添加。否则，链接器会报错无法找到GrowingVectorMemory的构造和析构函数。

|  |
| --- |
| // 向量类的定义和实现 |
| class UserVector { ... }; |
| // 为向量类创建显示实例化。如果你的工程含有多个文件，包含头文件，这个实例化必须放在一个<code>.cc文件中，使得它只被包含一次。 |
| #include <deal.II/lac/vector\_memory.templates.h> |
|  |
| template class VectorMemory<UserVector>; |
| template class GrowingVectorMemory<UserVector>; |

用到的预处理器也必须有与矩阵相同的接口，即，尤其是他们必须有成员函数vmult用于表示预处理器的作用。

**AdditionalData**

**预处理器**

预处理器是用于加速线性系统的迭代解。典型的预处理器有Jacobi，Gauss-Seidel，或SSOR，但dealii也提供更复杂的，如Vanka或incomplete LU分解(ILU)。此外，稀疏直接求解器在需要的时候也可充当预处理器。

宽泛地讲，预处理器可看做算子，它作用于矩阵上用于改善条件数。其思想就是，经预处理后的系统，要比原先的系统好解得多。其具体含义依赖于矩阵的结构，我们在此无法泛泛而论。对于对称、正定的矩阵A和P，这意味着的谱条件数（最大特征值和最小特征值的商）要比A的小得多。

至于最简单的例子，Richardson迭代 (在SovlerRichardson中实现)，其预处理后的迭代过程为：



相应地，这里的预处理就相当于对残值施加一个线性操作，因而，预处理操作的实现就是vmult()。实际上，可以把任何像矩阵的，定义了vmult()和Tvmult()函数的对象都看做预处理器。

注：PreconditionerType含义：类似MatrixType，但vmult()和Tvmult()的含义不同。对于预处理器而言，vmult()的意思是把线性算子的某种近似**逆**施加到向量上去，而非施加线性算子本身。

当使用Krylov空间方法的时候。。。略去一段，没咋看懂。

**Relaxation methods**

许多预处理器需要把A分为两个矩阵，即A=P-N。在这种情况下，上述的Richardson方法可简化为：

从而完全避免了乘以A的操作。我们把这种把上一步的映射到下一步的的操作称为松弛算子。它们一般的接口在the RelaxationType concept中有介绍。