# C++库的详细文档

本套库针对实验室的常见应用，利用C++的一些特性，做到同时保证运算的高效、使用方便以及使用这套库写出来的代码可读性强。本套库可以分成四大部分：量子力学，正交坐标系下的张量运算，算法集合，自旋体系。

## 量子力学

## 张量运算

本库提供张量运算的功能，但是要求张量必须是正交坐标系中的张量。张量运算位于头文件tensor.hpp中，使用时请#include <tensor.hpp>。

一般的张量运算库通常只支持同种数据类型之间的相互运算。比如说两个数据类型都是double类型的张量可以相互进行运算（加，减，数乘，张量积，并矢，缩并等），但是double类型的张量跟Operator类型的张量则不能相互之间进行运算，这就给我们的应用带来了极大的不方便。在实际应用的过程中，经常使用到不同类型的张量之间的运算，比如：自旋1与自旋2之间的dipole-dipole相互作用哈密顿量可以写成，其中与是自旋矢量算子，在编程的时候通常被实现为数据类型为Quantum::Operator类型的矢量。而则是两个自旋之间的耦合张量，通常被实现为数据类型为double类型的张量。数据类型为Quantum::Operator类型的矢量与数据类型为double类型的张量并不是相同的数据类型。为了实现这一点，本库中大量运用了C++11的新特性decltype进行模板元编程来实现。

所有张量运算的内容都位于名空间Tensor，如果要使用这些功能，需要在程序中加入代码：

using namespace Tensor;

### 类型

正交坐标系下的张量运算的核心是模板类template <typename scalar,int order,int dimension> tensor。其中order是张量的阶，默认为2，对于矢量order=1，对于二阶张量order=2。dimension是张量的维，默认为3。scalar是张量的域的类型（即张量所对应的标量的类型）。

例如：

* tensor<double>表示实二阶张量，数据用double存储。
* tensor<std::complex<double>,1>则表示复二阶矢量。
* tensor<Quantum::Operator,1>表示矢量算子。

该类在设计的时候，尽可能地使其行为与数组保持一直。

根据习惯，定义了一个常用的模板typedef：

template <typename scalar,int dimension> vector等价于tensor<scalar,1,dimension>，同样，模板中的dimension默认为3。注意的是，C++的标准模板库（STL中也有vector模板类，该类与这里的vector重名，所以写代码的时候可能需要明确指定是哪个vector，不然编译器会报错。使用Tensor::vector使用本库中的vector模板类，使用std::vector使用标准库中的模板类）例如：

* vector<Quantum::Operator>矢量算子。
* vector<double>实矢量，使用double存储数据。

每个tensor类（包括经过typedef得到的模板类）都有一个与之对应的数组类型，对于标量类型为scalar的M阶N维张量，这个类型是一个每一维大小都为N的M为数组。使用tensor:: array\_type来访问这个数组类型。例如：

* vector<double>::array\_type为double[3]
* tensor<Quantum::Operator>::array\_type为Quantum::Operator[3][3]

### 构造函数

tensor包含两个构造函数：不接受任何参数的默认构造函数，以及使用C++的统一初始化特性初始化的构造函数。默认构造函数的行为与C++中数组声明（但不初始化）的行为一致。统一初始化的行为则与C++中数组元素的统一初始化行为一致。例如：

* vector<double> a; //默认构造函数，创建三维矢量a，其值不确定
* vector<double> b = { 1.0, 2.0, 3.0 }; //统一初始化，创建三维矢量b，它的三个分量分别是1.0,2.0,3.0。

### 运算符

圆括号()运算符：括号运算符用于访问张量的分量。如果对象不是const类型，则圆括号的返回值是个左值（lvalue）。如果对象是const类型，则圆括号的返回值是个右值（rvalue）。对于N阶张量，该运算符接受N个参数，访问张量的第分量。tensor类的数据内部存储为一个变量名为components的数组（private变量，外部无法访问到），数组类型为该tensor类的array\_type。假设有tensor<T,M,N>类的变量a，则a访问的是。

方括号[]运算符：也可以使用方括号来访问张量的分量。按照数据在机器中的存储顺序来访问。由于张量内部数据存储采用的是多元数组，因此数据在机器中的存储顺序与多元数组相同。对于N阶M维的张量a，与访问的是同一个元素。与圆括号相同的一点是，如果对象不是const类型，则方括号的返回值是个左值（lvalue）。如果对象是const类型，则方括号的返回值是个右值（rvalue）。

例如，变量a,b的声明分别是tensor<double> a; vector<double> b;

* a(0,2) = 2.5;
* a[2] = 2.5; //效果同上
* b(1) = 4.3;
* b[1] = 4.3; //效果同上
* cout << a(2,1);
* cout << a[7]; //效果同上
* cout << b[0] << b[1] << b[2];
* cout << b(0) << b(1) << b(2); //效果同上

加减运算符：实现张量的加减运算。加减运算要求两个张量阶数维数都相同，但是张量的标量类型可以相同或者不同。假设现有tensor<T1,M,N>对象a、tensor<T2,M,N>对象b，T1与T2之间定义有加法运算，则a+b得到的是tensor<decltype(T1()+T2()),M,N>类的对象，记为c。它的元素。a-b得到的是tensor<decltype(T1()-T2()),M,N>类的对象，记为d。它的元素。

例如，两个变量vector<int\*> a; vector<int> b;，则：

* a+b，b+a，a-b均是vector<int\*>类型的
* b-a编译无法通过

一元加减运算符：假设现有tensor<T,M,N>对象a，T与定义有单目加法运算，则+a得到的是tensor<decltype(+T()),M,N>类的对象，记为b。它的元素。-a得到的是tensor<decltype(-T()),M,N>类的对象，记为c。它的元素。

乘除运算符：乘除运算符实现张量的数乘操作，规则同张量的定义。注意的是，乘法运算符并不能计算张量的张量积，张量的点乘或者是叉乘。有意义的运算为scalar\*tensor, tensor\*scalar, tensor/scalar，而scalar/tensor，tensor/tensor以及tensor\*tensor则是无效的。假设有tensor<T1,M,N>类型的变量a，T2类型的变量b（T2不能是任意张量类型），则a\*b类型为tensor<decltype(T1()\*T2()),M,N>，b\*a的类型为tensor<decltype(T2()\*T1()),M,N>，a/b的类型为tensor<decltype(T1()/T2()),M,N>，b/a、a/a以及a\*a无法通过编译。例如，现有double w1; vector<Quantum::Operator> S则：

* w1\*S，S\*w1，S/w1类型都是vector<Quantum::Operator>。
* S\*S，S/S，w1/S无法通过编译。

### 方法

张量积：使用prod方法计算张量的张量积。这是一个不定参数个数的函数，可以接收任意多个参数，这些参数的类型必须是tensor模板类的某个实例。prod按照参数从左到右的顺序对参数中的张量进行张量积运算，并将结果作为返回值返回。要计算张量积，则使用。

例如，有3个张量vector<Quantum::Operator> S1; vector<Quantum::Operator> S2; Tensor<double> D12;，则：

* prod(S1,D12,S2); //返回张量积，结果是一个tensor< Quantum::Operator,4>。
* prod(S1, S2); //返回张量积，结果是一个tensor< Quantum::Operator >。
* prod(S1,S1,S2,S2); //返回张量积，结果是一个tensor< Quantum::Operator,4>。
* prod(S1); //编译失败

缩并：使用contract方法对张量进行缩并操作，使用方法为contract<i,j>(t)，其中i与j是编译时常量，t为一个超过二阶的tensor类型，该操作实现对张量t的第i,j个维度的缩并。维度的编号从0开始。例如有张量tensor<double,3> a;则：

* contract<0,1>(a); //对a的头两个维度进行缩并，结果为vector<double>。

点乘：使用dot计算张量的点乘积，这是一个不定参数个数的函数，可以接收任意多个参数，这些参数的类型必须是tensor模板类的某个实例。dot按照参数从左到右的顺序对参数中的张量进行点乘积运算，并将结果作为返回值返回。要计算点乘积，则使用。

例如，有3个张量vector<Quantum::Operator> S1; vector<Quantum::Operator> S2; Tensor<double> D12;，则：

* dot(S1,D12,S2); //返回点乘积，结果是一个tensor< Quantum::Operator,0>。
* dot(S1, S2); //返回点乘积，结果是一个tensor< Quantum::Operator,0>。
* dot(S1,D12); //返回点乘积，结果是一个vector< Quantum::Operator>。
* dot(S1); //编译失败

并联双点乘：使用ddotp计算并联双点乘，该函数接受两个参数a与b，返回结果为。如果a的类型为tensor<T1,M,N>，b的类型为tensor<T2,P,N>则返回值的类型为tensor<decltype(T1()\*T2()),M+P-4,N>。

串联双点乘：使用ddots计算串联双点乘，该函数接受两个参数a与b，返回结果为。如果a的类型为tensor<T1,M,N>，b的类型为tensor<T2,P,N>则返回值的类型为tensor<decltype(T1()\*T2()),M+P-4,N>。

叉乘：使用cross计算三维一阶张量（三维矢量）双点乘，该函数接受两个参数a与b，这两个可以具有不同的，返回结果为。如果a的类型为vector<T1 >，b的类型为vector<T2>则返回值的类型为vector<decltype(T1()\*T2())>。

矢量的模：使用abs计算矢量的模，该函数接受一个参数a，返回值为。

生成单位张量：使用模板函数I<T,dimension>()来生成二阶单位张量，其中T是标量类型，dimension是张量的维度。dimension可以不指定，这种情况下dimension为默认值3。

### 事例程序

#include <tensor.hpp>

#include <iostream>

using namespace Tensor;

int main() {

vector<int> a = { 1,2,3 };

vector<int> b = { 4,5,6 };

tensor<int> A = { 1,2,3,

4,5,6,

7,8,9 };

//calculate a dot A dot b

std::cout << dot(a,A,b) << std::endl;

//calculate the dot product of a and b

std::cout << dot(a,b) << std::endl;

//calculate the dyadic product of a,b,a and b (i.e. abab)

auto abab = prod(a,b,a,b);

//output the first component of abab

std::cout << abab(0,0,0,0) << std::endl;

}

## 算法集合

## 自旋体系