高等程序设计 - Qt/C++

第 7 章：C++ 科学计算

王培杰

长江大学地球物理与石油资源学院

2025 年 11 月 24 日

目录

目录

C++ 标准库 cmath 简介

cmath 库

- C++ 标准库中的数学函数库
- 提供基础数学运算函数
- 包含三角函数、指数对数、幂函数等
- 无需额外安装，编译器自带
- 跨平台兼容性好
- 是学习科学计算的基础

cmath 主要函数分类

三角函数

- `sin`, `cos`, `tan` - 基本三角函数
- `asin`, `acos`, `atan` - 反三角函数
- `sinh`, `cosh`, `tanh` - 双曲函数
- `atan2` - 双参数反正切

指数与对数

- `exp` - 自然指数
- `log` - 自然对数
- `log10` - 常用对数
- `log2` - 以 2 为底的对数

幂函数与取整

- `pow` - 幂运算
- `sqrt` - 平方根
- `cbrt` - 立方根
- `ceil` - 向上取整
- `floor` - 向下取整
- `round` - 四舍五入

其他函数

- `abs`, `fabs` - 绝对值
- `fmod` - 取模
- `fmax`, `fmin` - 最值

cmath 使用示例

基本使用

- **头文件**：`#include <cmath>`
- **命名空间**：函数在 `std` 命名空间中
- **精度**：支持 `float`, `double`, `long double`

示例代码

- `double x = std::sin(3.14159 / 2);` // 计算 sin(π/2)
- `double y = std::pow(2.0, 3.0);` // 计算 2^3
- `double z = std::sqrt(16.0);` // 计算 $\sqrt{16}$
- `double w = std::log(2.71828);` // 计算 ln(e)
- `double angle = std::atan2(y, x);` // 计算角度

cmath 的局限性

功能限制

- 只提供基础数学函数
- 缺少特殊函数（贝塞尔、伽马等）
- 不支持高精度计算
- 不支持向量化运算
- 无统计分布函数

性能限制

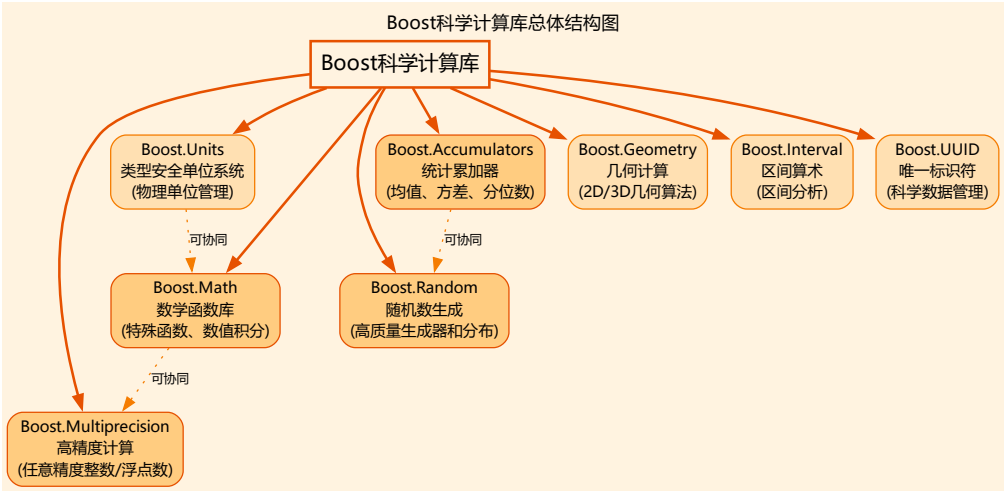
- 单元素计算，无 SIMD 优化
- 不适合大规模向量运算
- 无并行计算支持
- 性能优化有限

学习路径

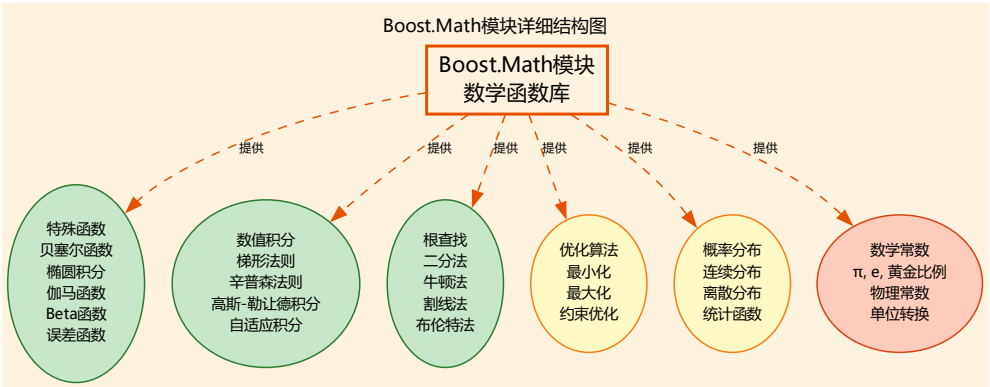
- cmath 是基础，需要掌握
- 进阶到 Boost 扩展功能
- 使用 Eigen 进行矩阵运算
- 使用 MKL 获得高性能

目录

Boost 简介


Boost科学计算库总体结构图
Boost科学计算库
Boost.Units
类型安全单位系统
(物理单位管理)
Boost.Accumulators
统计累加器
(均值、方差、分位数)
Boost.Geometry
几何计算
(2D/3D几何算法)
Boost.Interval
区间算术
(区间分析)
Boost.UUID
唯一标识符
(科学数据管理)
可协同
可协同
Boost.Math
数学函数库
(特殊函数、数值积分)
Boost.Random
随机数生成
(高质量生成器和分布)
可协同
Boost.Multiprecision
高精度计算
(任意精度整数/浮点数)

Boost.Math 模块



Boost.Math 详细功能

特殊函数

- **贝塞尔函数**:
  - `cyl_bessel_j(n, x)`
  - `cyl_bessel_i(n, x)`
- **伽马函数**:
  - `tgamma(x)` - 伽马函数
  - `lgamma(x)` - 对数伽马
- **椭圆积分**: `ellint_1(k)`, `ellint_2(k)`
- **误差函数**: `erf(x)`,`erfc(x)`

统计分布

- **连续分布**: 正态、t、卡方、F 分布
- **离散分布**: 二项、泊松、几何分布
- **操作**: `pdf()`,`cdf()`,`quantile()`

数值工具

- **求根算法**:
  - `bisect()` - 二分法
  - `newton()` - 牛顿法
- **插值方法**: 线性插值、三次样条
- **数值积分**: 梯形法则、辛普森法则

安装与使用

- **包管理器**:
  - Linux: `sudo apt-get install libboost-all-dev`
  - macOS: `brew install boost`
- **CMake**: `find_package(Boost REQUIRED COMPONENTS math)`
- **头文件**: `#include <boost/math/special_functions.hpp>`

Boost.Math 使用示例

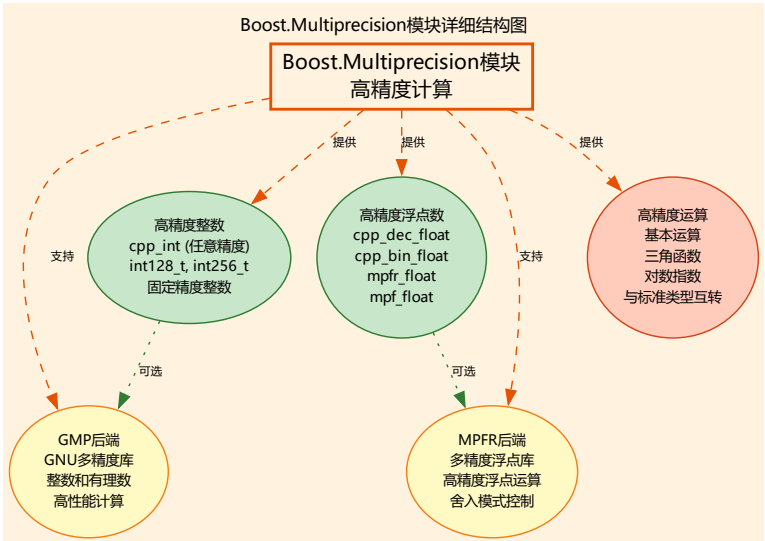
特殊函数示例

- `double gamma_val = boost::math::tgamma(5.0);` // 计算 $\Gamma(5)$
- `double bessel_j = boost::math::cyl_bessel_j(0, 1.0);` // $J_0(1)$
- `double erf_val = boost::math::erf(1.0);` // 误差函数

统计分布示例

- `boost::math::normal_distribution<> norm(0, 1);` // 标准正态分布
- `double prob = pdf(norm, 1.0);` // 概率密度
- `double cum = cdf(norm, 1.0);` // 累积分布

Boost.Multiprecision 高精度计算



Boost.Multiprecision 详细功能

高精度整数

- cpp_int - 任意精度整数
- int128_t, int256_t - 固定精度
- 支持标准运算符：+、-、*、/、%
- 支持高级函数：sqrt()
- 无溢出问题

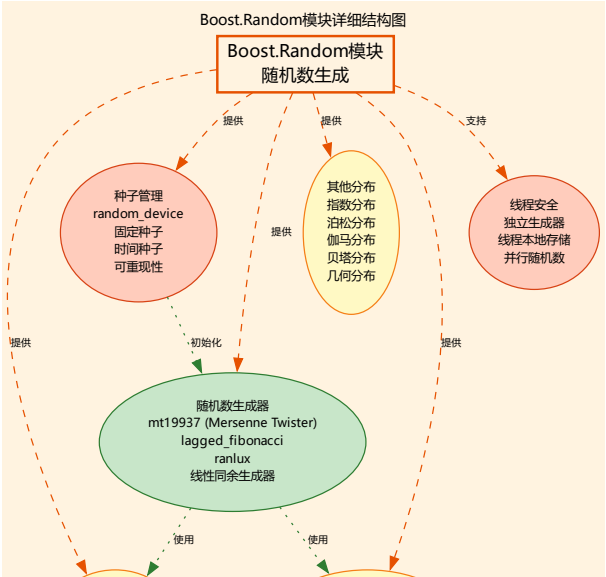
高精度浮点

- cpp_dec_float_50 - 50 位小数
- cpp_dec_float_100 - 100 位小数
- cpp_bin_float - 二进制浮点（更高精度）
- 精确的十进制表示，适合金融计算

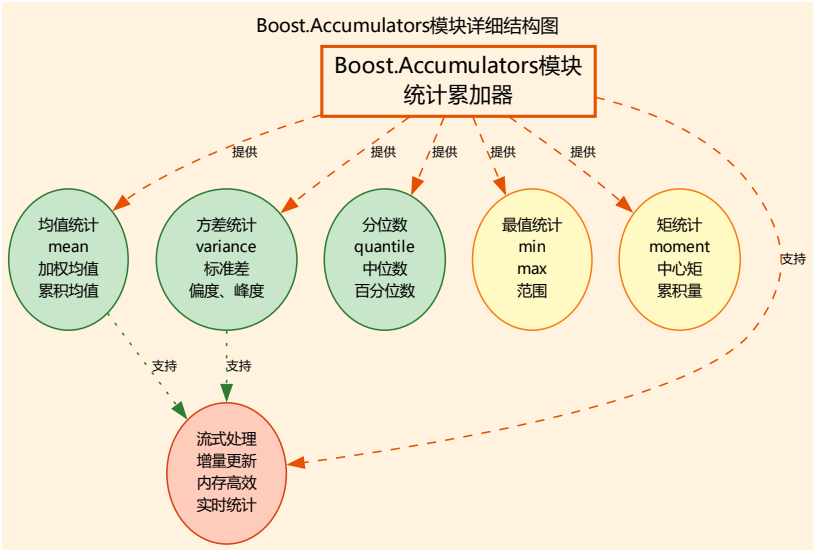
使用示例

```
#include <boost/multiprecision/cpp_int.hp
using namespace boost::multiprecision;
cpp_int n = 12345678901234567890;
cpp_int result = n * n; // 无溢出
cpp_dec_float_50 pi = 3.14159265358979323846264338327
```

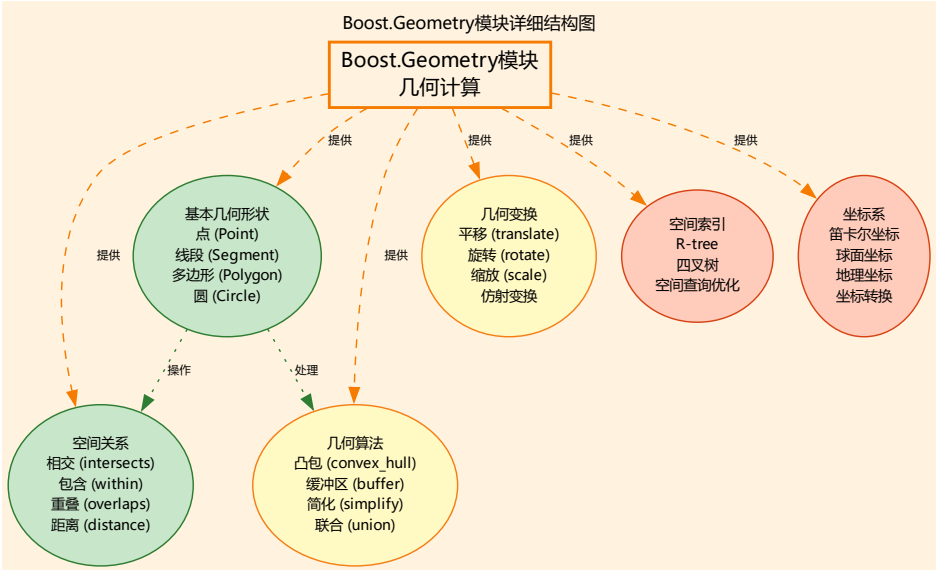
Boost.Random 随机数生成



Boost.Accumulators 统计累加器



Boost.Geometry 几何算法



Boost 安装与使用

安装方式

- Linux: `sudo apt install libboost-all-dev`
- macOS: `brew install boost`
- Windows: vcpkg 或源码编译
- 验证版本: `dpkg -l | grep libboost`

编译链接

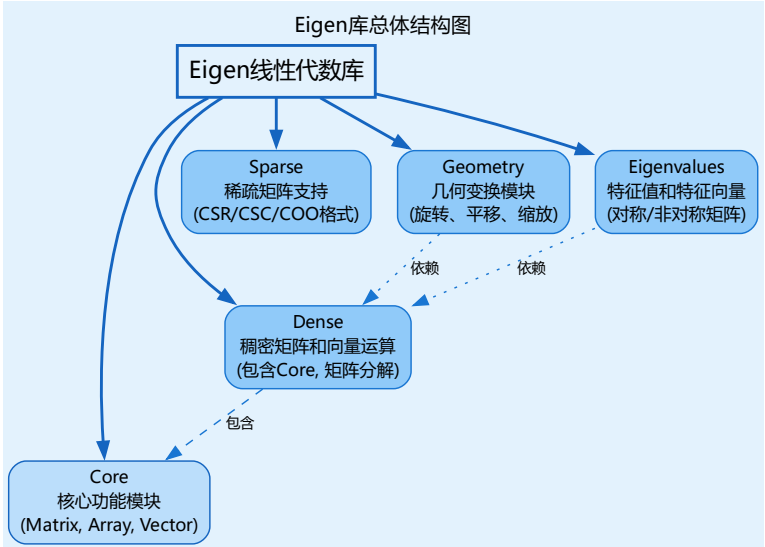
- 头文件库：无需链接
- 编译时添加: `-I/usr/include/boost`
- 需要链接的库: `-lboost_system -lboost_filesystem`

CMake 集成

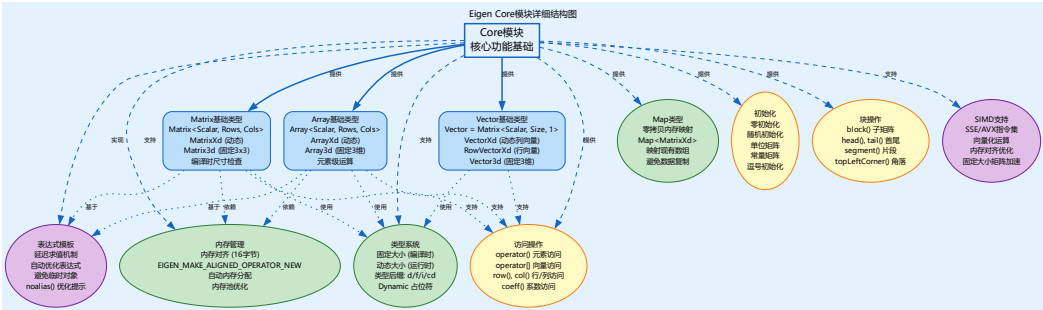
- `find_package(Boost REQUIRED)`
- `target_link_libraries(target Boost::boost)`
- 指定组件: `find_package(Boost REQUIRED COMPONENTS system filesystem)`

目录

Eigen 简介


Eigen库总体结构图
Eigen线性代数库
Sparse
稀疏矩阵支持
(CSR/CSC/COO格式)
Geometry
几何变换模块
(旋转、平移、缩放)
Eigenvalues
特征值和特征向量
(对称/非对称矩阵)
依赖
依赖
Dense
稠密矩阵和向量运算
(包含Core, 矩阵分解)
包含
Core
核心功能模块
(Matrix, Array, Vector)

Eigen 核心模块


Eigen Core模块详细结构图
Core模块
核心功能基础
提供
提供
提供
提供
提供
提供
提供
支持
支持
支持
支持
Matrix基础类型
Matrix<Scalar, Rows, Cols>
MatrixXd (动态)
Matrix3d (固定3x3)
编译时尺寸检查
Array基础类型
Array<Scalar, Rows, Cols>
ArrayXd (动态)
Array3d (固定3维)
元素级运算
Vector基础类型
Vector = Matrix<Scalar, Size, 1>
VectorXd (动态列向量)
RowVectorXd (行向量)
Vector3d (固定3维)
Map类型
零拷贝内存映射
Map<MatrixXd>
映射现有数组
避免数据复制
初始化
零初始化
随机初始化
单位矩阵
常量矩阵
逗号初始化
块操作
block() 子矩阵
head(), tail() 首尾
segment() 片段
topLeftCorner() 角落
SIMD支持
SSE/AVX指令集
向量化运算
内存对齐优化
固定大小矩阵加速
实现
支持
基于
基于
依赖
依赖
使用
使用
使用
支持
支持
支持
表达式模板
延迟求值机制
自动优化表达式
避免临时对象
noalias() 优化提示
内存管理
内存对齐 (16字节)
EIGEN_MAKE_ALIGNED_OPERATOR_NEW
自动内存分配
内存池优化
类型系统
固定大小 (编译时)
动态大小 (运行时)
类型后缀: d/f/i/cd
Dynamic 占位符
访问操作
operator() 元素访问
operator[] 向量访问
row(), col() 行/列访问
coeff() 系数访问

Eigen 基本类型

动态大小

- `MatrixXd` - 动态矩阵（double）
- `VectorXd` - 动态向量（double）
- `ArrayXd` - 动态数组（元素级运算）
- 运行时确定尺寸

固定大小

- `Matrix3d` - 3x3 矩阵（double）
- `Vector3d` - 3 维向量（double）
- `Matrix4f` - 4x4 矩阵（float）
- 编译时确定尺寸，性能更好

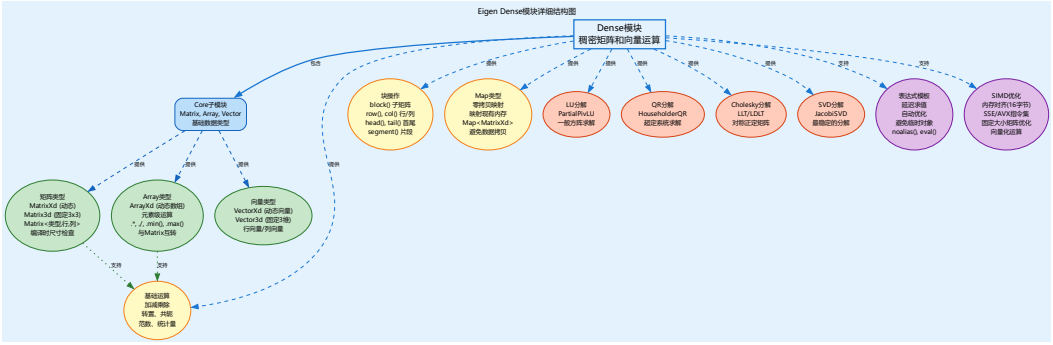
命名规则

- `Matrix` - 矩阵类型
- `Vector` - 向量类型
- `Array` - 数组类型（元素级）
- `X` - 动态大小
- `d` - double, `f` - float

类型转换

- `Matrix::array()` - 转为 Array
- `Array::matrix()` - 转为 Matrix
- 零开销转换

Eigen 稠密矩阵模块



Eigen 基本运算

矩阵运算

- 加减：`A + B`, `A - B`
- 乘法：`A * B`（矩阵乘法）
- 标量：`A * scalar`, `A / scalar`
- 转置：`A.transpose()`
- 共轭转置：`A.adjoint()`

元素访问

- `A(i, j)` - 矩阵元素
- `v[i]` - 向量元素
- `A.block(i,j,m,n)` - 子矩阵
- `A.row(i)`, `A.col(j)` - 行/列

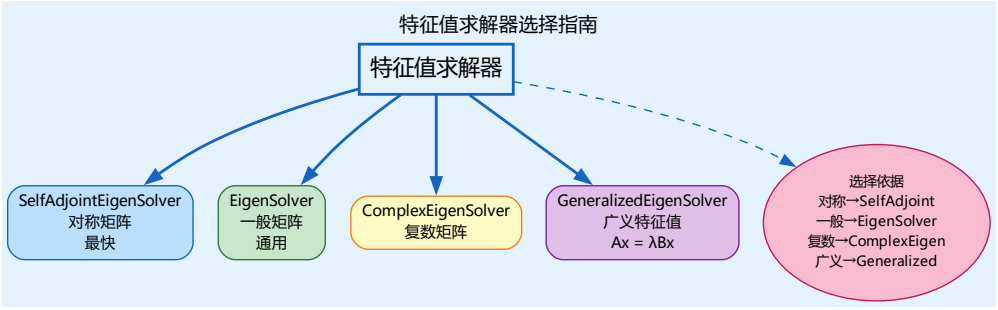
初始化方法

- `MatrixXd::Zero(m,n)` - 零矩阵
- `MatrixXd::Identity(m,n)` - 单位矩阵
- `MatrixXd::Random(m,n)` - 随机矩阵
- 逗号初始化：`A << 1,2,3,4,5,6;`

统计与范数

- `A.sum()`, `A.mean()` - 和、均值
- `A.norm()` - Frobenius 范数
- `A.minCoeff()`, `A.maxCoeff()` - 最值
- `A.determinant()` - 行列式

Eigen 矩阵分解


特征值求解器选择指南
特征值求解器
SelfAdjointEigenSolver
对称矩阵
最快
EigenSolver
一般矩阵
通用
ComplexEigenSolver
复数矩阵
GeneralizedEigenSolver
广义特征值
Ax = λBx
选择依据
对称→SelfAdjoint
一般→EigenSolver
复数→ComplexEigen
广义→Generalized

Eigen 矩阵分解方法

LU 分解

- `PartialPivLU<MatrixXd> lu(A);`
- `x = lu.solve(b);`
- 适用于一般方阵
- 复杂度：$O(n^3)$

Cholesky 分解

- `LLT<MatrixXd> chol(A);`
- `x = chol.solve(b);`
- 适用于对称正定矩阵
- 复杂度：$O(n^3/3)$，更快

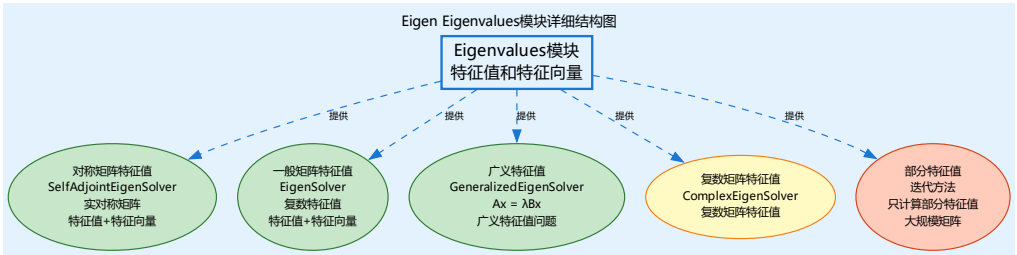
QR 分解

- `HouseholderQR<MatrixXd> qr(A);`
- `x = qr.solve(b);`
- 适用于超定系统
- 最小二乘问题

SVD 分解

- `JacobiSVD<MatrixXd> svd(A);`
- 最稳定的分解方法
- 适用于任意矩阵
- 可用于伪逆、降维

Eigen 特征值与特征向量模块



Eigen 使用示例

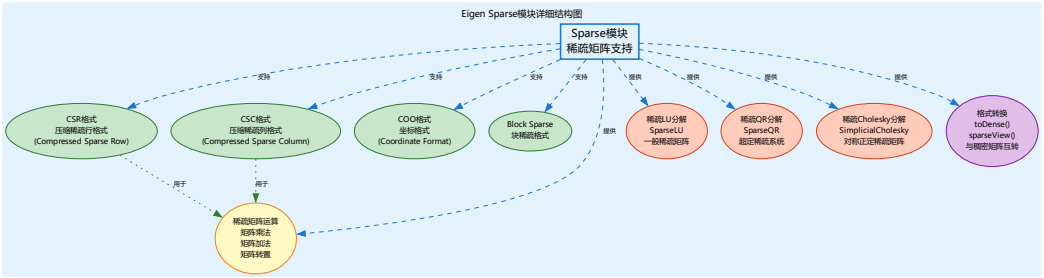
线性方程组求解

```
#include <Eigen/Dense>
using namespace Eigen;
MatrixXd A(3, 3);
A << 1, 2, 3, 4, 5, 6, 7, 8, 10;
VectorXd b(3);
b << 3, 3, 4;
VectorXd x = A.colPivHouseholderQr().solve(b);
```

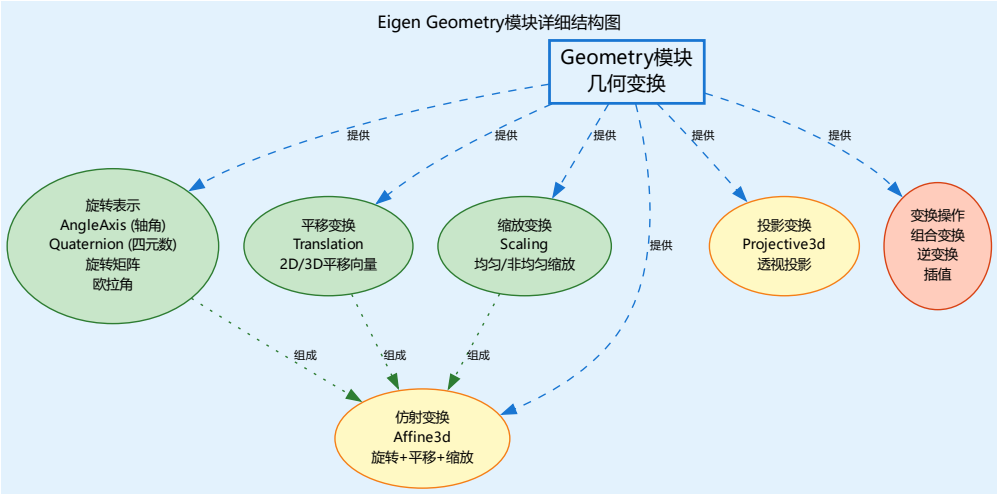
特征值计算

```
SelfAdjointEigenSolver<MatrixXd> eigensolver(A);
VectorXd eigenvalues = eigensolver.eigenvalues();
MatrixXd eigenvectors = eigensolver.eigenvectors();
```

Eigen 稀疏矩阵模块



Eigen 几何变换模块



Eigen 安装与配置

安装方法

- 下载源码解压即可
- 纯头文件库，无需编译
- 包管理器安装更方便
  - Linux: `apt-get install libeigen3-dev`
  - macOS: `brew install eigen`

CMake 配置

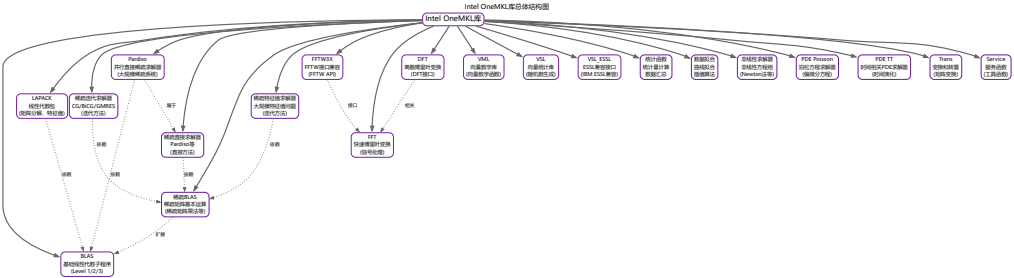
- `find_path(EIGEN3_INCLUDE_DIR Eigen3)`
- `include_directories(${EIGEN3_INC`
- 或使用 `find_package(Eigen3)`

头文件包含

- `#include <Eigen/Dense>` - 所有稠密矩阵
- `#include <Eigen/Sparse>` - 稀疏矩阵
- `#include <Eigen/Geometry>` - 几何变换

目录

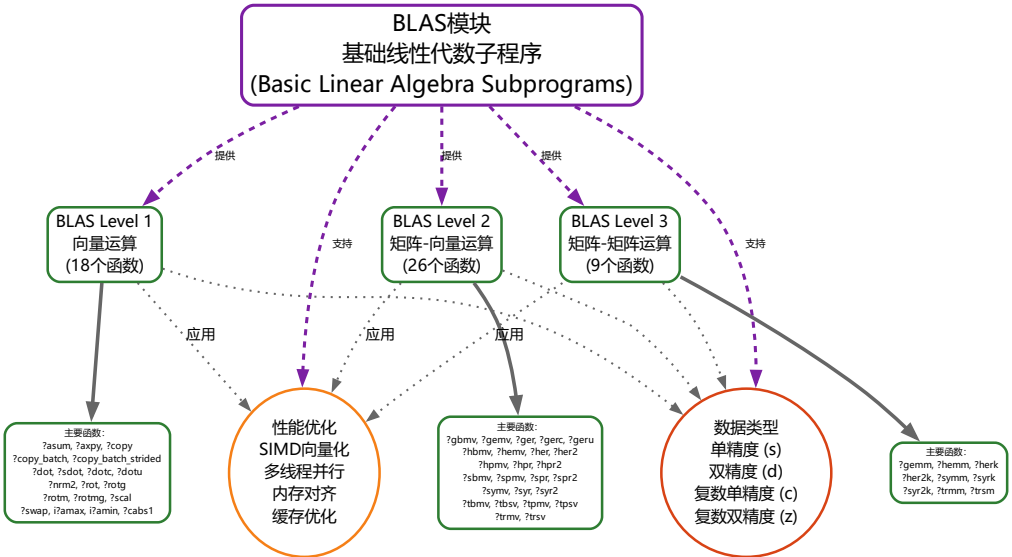
OneMKL 简介



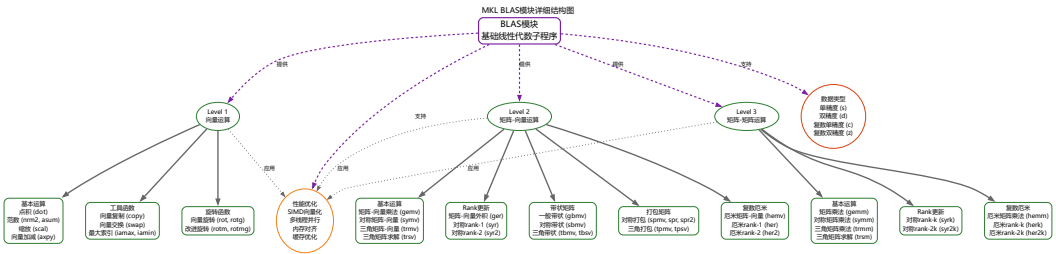
OneMKL 概览

Intel oneAPI Math Kernel Library（OneMKL）是 Intel 的高性能数学核心库，专为 x86/x86_64 平台深度优化。**模块**：BLAS、LAPACK、ScaLAPACK、FFT、DFT、FFTW3 接口、Pardiso 稀疏求解器、VML（向量数学库）、VSL（向量统计库）等。**并行**：自动利用多核 CPU、SIMD（AVX/AVX-512）及多线程，支持 OpenMP 与 TBB。**跨平台**：Windows、Linux、macOS；C/C++、Fortran、Python、DPC++。CMake 一行 `find_package(MKL)` 即可启用。

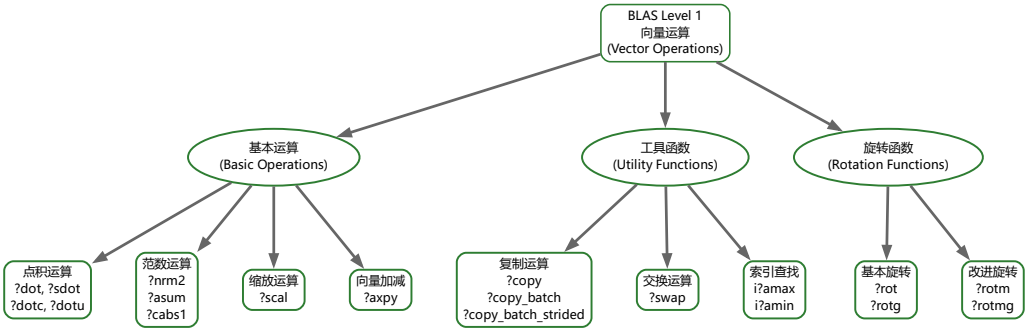
BLAS 基础线性代数子程序



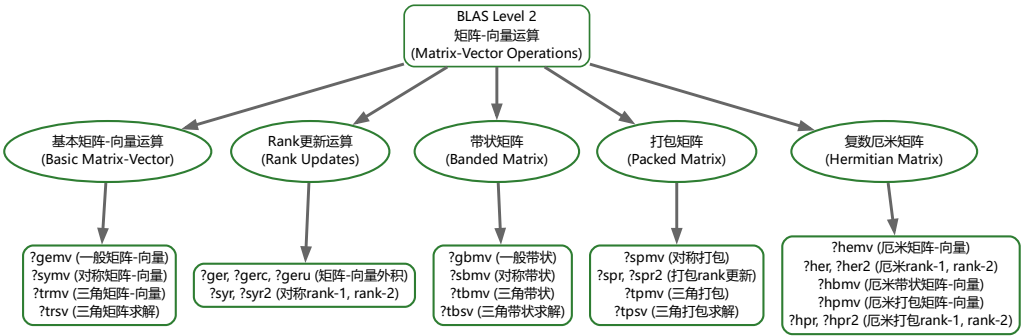
BLAS 详细结构


MKL BLAS模块详细结构图
BLAS模块
基础线性代数子程序
提供
提供
提供
支持
Level 1
向量运算
Level 2
矩阵-向量运算
Level 3
矩阵-矩阵运算
数据类型
单精度 (s)
双精度 (d)
复数单精度 (c)
复数双精度 (z)
支持
应用
应用
应用
基本运算
点积 (dot)
范数 (nrm2, asum)
缩放 (scal)
向量加减 (axpy)
工具函数
向量复制 (copy)
向量交换 (swap)
最大索引 (iamax, iamin)
旋转函数
向量旋转 (rot, rotg)
改进旋转 (rotm, rotmg)
性能优化
SIMD向量化
多线程并行
内存对齐
缓存优化
基本运算
矩阵-向量乘法 (gemv)
对称矩阵-向量 (symv)
三角矩阵-向量 (trmv)
三角矩阵求解 (trsv)
Rank更新
矩阵-向量外积 (ger)
对称rank-1 (syr)
对称rank-2 (syr2)
带状矩阵
一般带状 (gbmv)
对称带状 (sbmv)
三角带状 (tbmv, tbsv)
打包矩阵
对称打包 (spmv, spr, spr2)
三角打包 (tpmv, tpsv)
复数厄米
厄米矩阵-向量 (hemv)
厄米rank-1 (her)
厄米rank-2 (her2)
基本运算
矩阵乘法 (gemm)
对称矩阵乘法 (symm)
三角矩阵乘法 (trmm)
三角矩阵求解 (trsm)
Rank更新
对称rank-k (syrk)
对称rank-2k (syr2k)
复数厄米
厄米矩阵乘法 (hemm)
厄米rank-k (herk)
厄米rank-2k (her2k)

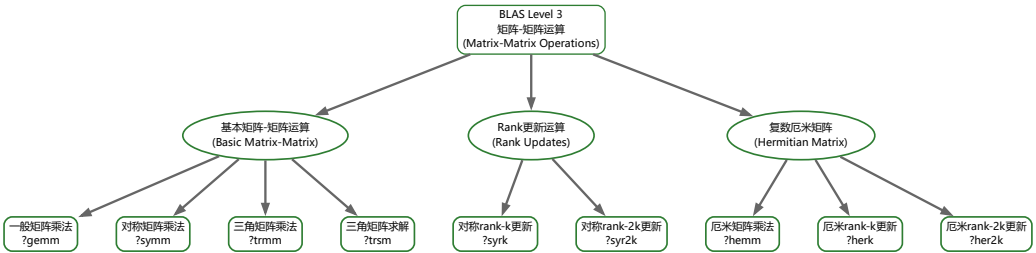
BLAS Level 1：向量运算



BLAS Level 2：矩阵-向量运算



BLAS Level 3：矩阵-矩阵运算



BLAS 函数列表

Level 1：向量运算

- ddot、sdot - 向量点积
- dnrm2、dasum - 向量范数
- dscal - 向量缩放
- daxpy - 向量线性组合
- dcopy - 向量复制
- idamax - 最大元素索引

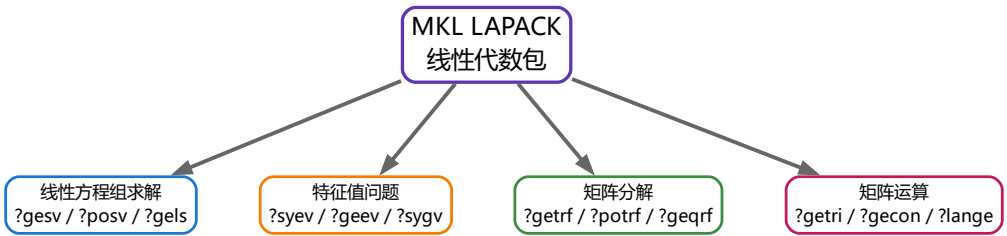
Level 2：矩阵-向量

- dgemv - 一般矩阵向量乘法
- dsymv - 对称矩阵向量乘法
- dtrmv - 三角矩阵向量乘法
- dger - 外积更新

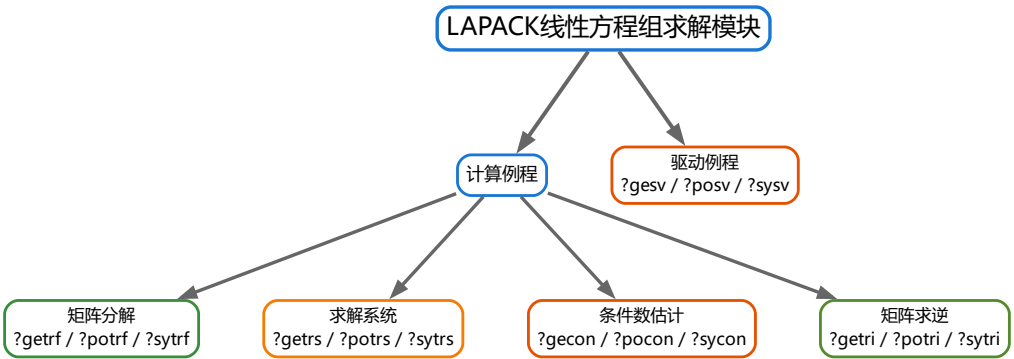
Level 3：矩阵-矩阵

- dgemm - 一般矩阵乘法
- dsymm - 对称矩阵乘法
- dtrmm - 三角矩阵乘法
- dsyrk - 对称秩 k 更新
- dtrsm - 三角矩阵求解

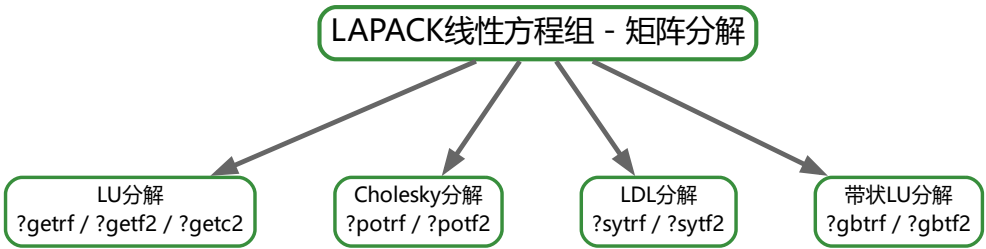
LAPACK 线性代数包



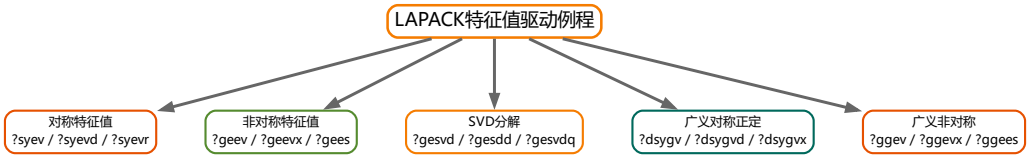
LAPACK 线性方程组求解



LAPACK 矩阵分解方法



LAPACK 特征值问题



LAPACK 主要功能

矩阵分解

- `dgesv` - LU 分解求解
- `dpotrf` - Cholesky 分解
- `dgeqrf` - QR 分解
- `dgesvd` - SVD 分解

线性方程组

- `dgesv` - 一般矩阵求解
- `dposv` - 对称正定求解
- `dgels` - 最小二乘问题

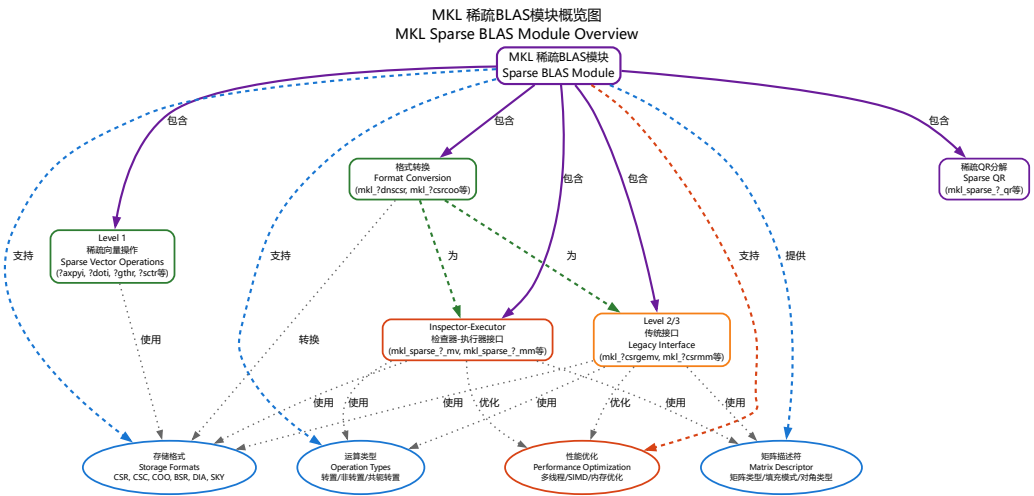
特征值问题

- `dsyev` - 对称矩阵特征值
- `dgeev` - 一般矩阵特征值
- `dsygv` - 广义特征值

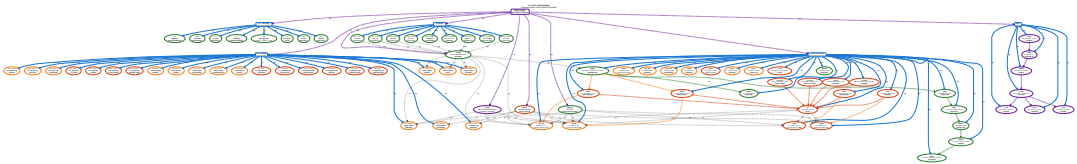
使用场景

- 大规模矩阵运算（>1000x1000）
- 高性能计算需求
- 科学计算、工程应用
- 需要极致性能的场景

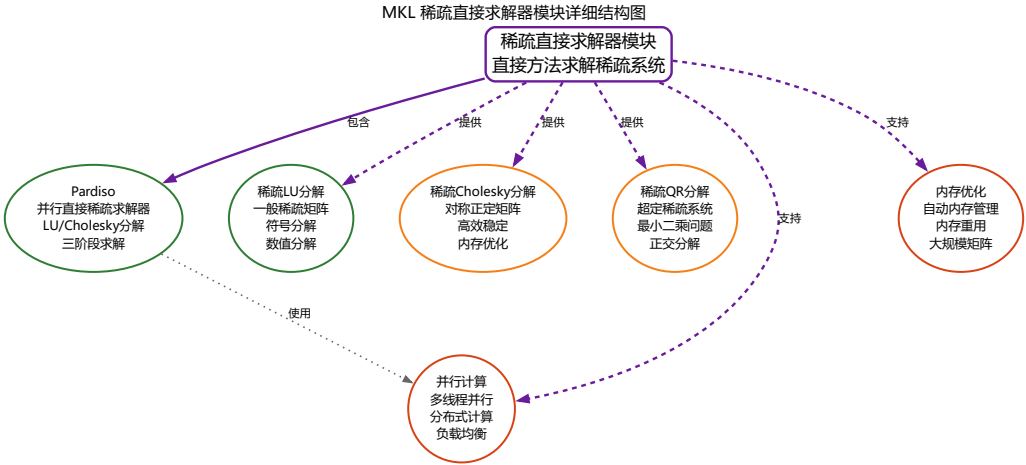
MKL 稀疏矩阵模块


MKL 稀疏BLAS模块概览图
MKL Sparse BLAS Module Overview
MKL 稀疏BLAS模块
Sparse BLAS Module
包含
包含
包含
包含
包含
格式转换
Format Conversion
(mkl_?dnscsr, mkl_?csrcoo等)
稀疏QR分解
Sparse QR
(mkl_sparse_?_qr等)
Level 1
稀疏向量操作
Sparse Vector Operations
(?axpyi, ?doti, ?gthr, ?sctr等)
支持
支持
为
为
支持
提供
使用
转换
Inspector-Executor
检查器-执行器接口
(mkl_sparse_?_mv, mkl_sparse_?_mm等)
Level 2/3
传统接口
Legacy Interface
(mkl_?csrgemv, mkl_?csrmm等)
使用
使用
使用
优化
使用
优化
使用
使用
存储格式
Storage Formats
CSR, CSC, COO, BSR, DIA, SKY
运算类型
Operation Types
转置/非转置/共轭转置
性能优化
Performance Optimization
多线程/SIMD/内存优化
矩阵描述符
Matrix Descriptor
矩阵类型/填充模式/对角类型

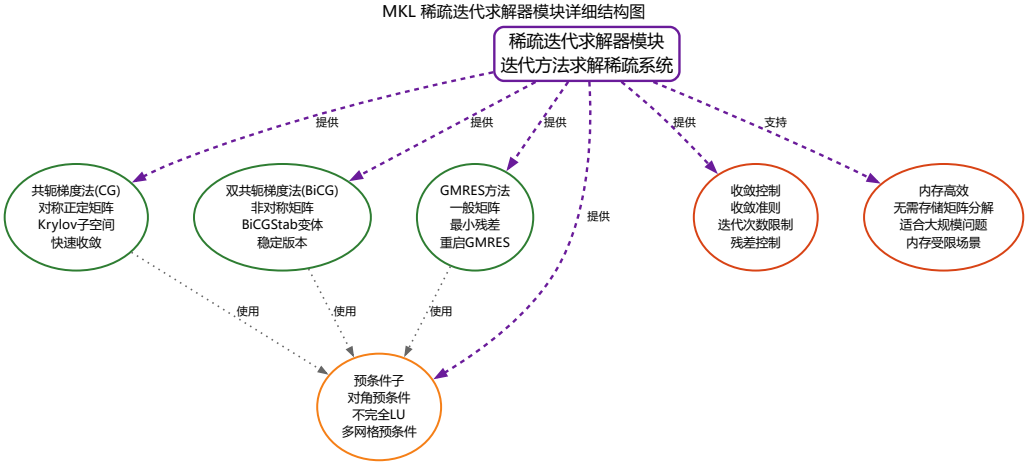
稀疏 BLAS 详细结构



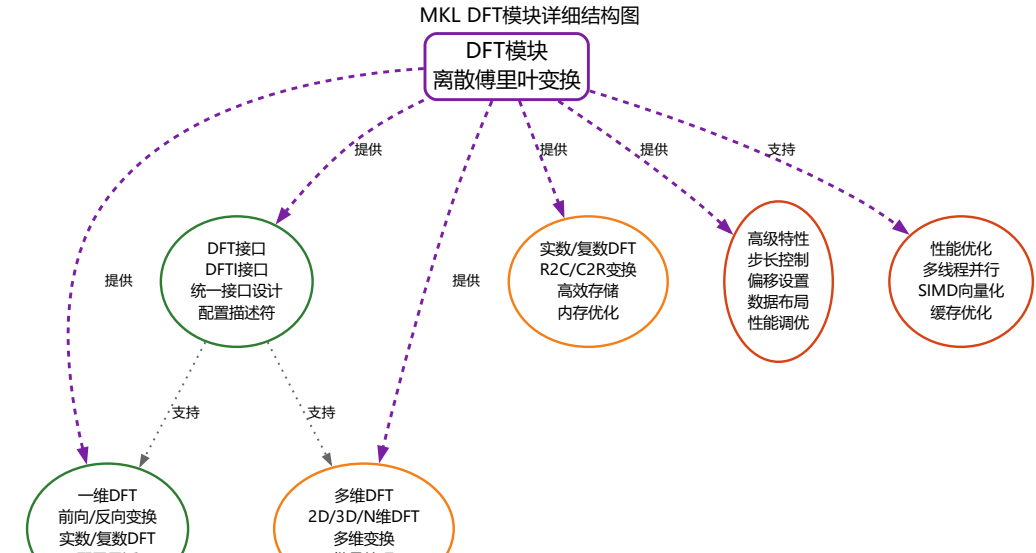
稀疏矩阵直接求解器


MKL 稀疏直接求解器模块详细结构图
稀疏直接求解器模块
直接方法求解稀疏系统
包含
提供
提供
提供
支持
支持
使用
Pardiso
并行直接稀疏求解器
LU/Cholesky分解
三阶段求解
稀疏LU分解
一般稀疏矩阵
符号分解
数值分解
稀疏Cholesky分解
对称正定矩阵
高效稳定
内存优化
稀疏QR分解
超定稀疏系统
最小二乘问题
正交分解
内存优化
自动内存管理
内存重用
大规模矩阵
并行计算
多线程并行
分布式计算
负载均衡

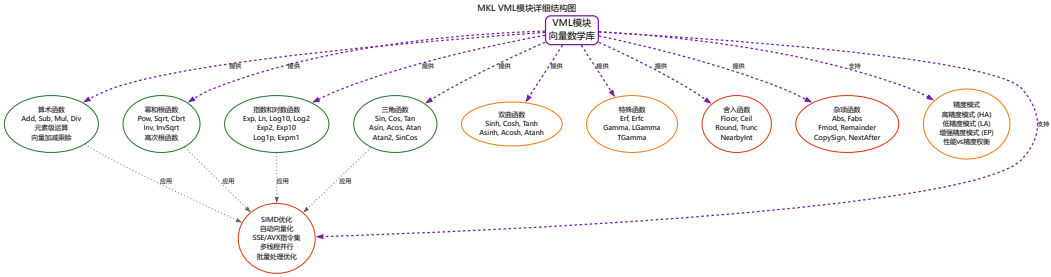
稀疏矩阵迭代求解器


MKL 稀疏迭代求解器模块详细结构图
稀疏迭代求解器模块
迭代方法求解稀疏系统
提供
提供
提供
提供
支持
共轭梯度法(CG)
对称正定矩阵
Krylov子空间
快速收敛
双共轭梯度法(BiCG)
非对称矩阵
BiCGStab变体
稳定版本
GMRES方法
一般矩阵
最小残差
重启GMRES
收敛控制
收敛准则
迭代次数限制
残差控制
内存高效
无需存储矩阵分解
适合大规模问题
内存受限场景
使用
使用
使用
提供
预条件子
对角预条件
不完全LU
多网格预条件

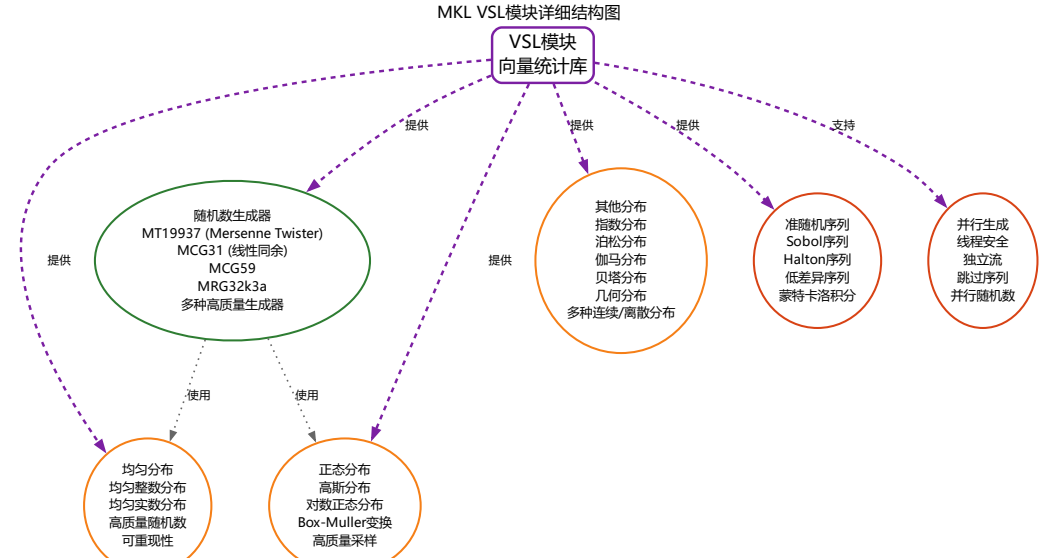
MKL FFT 快速傅里叶变换



MKL VML 向量数学库



MKL VSL 随机数生成库



OneMKL 安装与配置

安装方法

- **官方安装包**:
  - 从 Intel 官网下载
  - 运行安装程序
  - 设置环境变量 `MKLROOT`
- **包管理器**:
  - Linux: `apt-get install libmkl-dev`
  - macOS: `brew install mkl`

CMake 配置

- `find_package(MKL)`
- `target_link_libraries` 链接库
- 链接库示例:
  - `mkl_intel_lp64`
  - `mkl_core`
  - `mkl_sequential`

头文件包含

- `#include <mkl.h>` - 所有 MKL 函数
- `#include <mkl_blas.h>` - BLAS 函数
- `#include <mkl_lapacke.h>` - LAPACK 接口
- `#include <mkl_dfti.h>` - FFT 接口

总结

本章要点

- 掌握 C++ 标准库 cmath 基础
- 了解 Boost 科学计算组件
- 学会 Eigen 线性代数库的使用
- 了解 Intel OneMKL 高性能计算

学习路径

- 从标准库 cmath 开始学习基础
- 进阶到 Boost 扩展数学功能
- 使用 Eigen 进行矩阵运算
- 需要高性能时考虑 MKL

学习路径回顾

完整学习路径

cmath → Boost → Eigen → MKL

cmath

- 基础函数
- 起点
- 无需安装

Boost

- 扩展功能
- 特殊函数
- 高精度计算

Eigen

- 矩阵运算
- 线性代数
- 易用性好

MKL

- 高性能
- 优化加速
- 企业级

学习建议

- 循序渐进：按顺序学习每个库
- 实践为主：通过项目加深理解
- 对比学习：理解各库的适用场景
- 组合使用：发挥各库的协同优势

进一步学习

推荐资源

- C++ 标准库文档：`https://en.cppreference.com`
- Boost 官方文档：`https://www.boost.org`
- Eigen 官方文档：`https://eigen.tuxfamily.org`
- Intel OneMKL 文档：`https://software.intel.com/mkl`

实践项目建议

- 实现一个简单的线性方程组求解器
- 开发一个矩阵运算性能测试工具
- 构建一个科学计算工作流示例