Python 下大规模稀疏矩阵的 Cholesky 分解算法

张鹤潇

(计算机系，学号：2018011365，手机号：13310339961)

乐阳

(计算机系，学号：2018011359，手机号：18519660881)

摘　要: 在本文中，我们回顾了 Cholesky 分解的理论背景，介绍了基于符号分析、数值求解两阶段的稀疏矩阵 Cholesky 分解框架。我们利用 Cython 封装 C 库 SuiteSparse 实现的大规模稀疏矩阵 Cholesky 分解算法，为 Python 科学计算库 scipy 实现了正确而高效的稀疏矩阵分解功能。实验表明，调用我们实现的接口，与直接编写 C 程序相比，性能损失低于 5%。

关键词: Cholesky 分解; 稀疏矩阵分解; Cython 封装

1. 引言

Cholesky 分解是一种重要的矩阵分解方式，常被应用在方程组求解问题中，在数值分析领域有举足轻重的作用。很多实际问题中，矩阵的规模过大，为了对其进行 Cholesky 分解，需要专门针对稀疏矩阵的 Cholesky 分解算法。

scipy 和 numpy 是非常著名的 Python 科学计算库，遗憾的是，它们并没有提供稀疏矩阵 Cholesky 分解功能；而另一方面，以 C 语言实现的 SuiteSparse 库高效地实现了稀疏矩阵 Cholesky 分解算法，集成在其子模块 CHOLMOD 中。本文的主要工作是用 Cython 封装 CHOLMOD，对 scipy.sparse 稀疏矩阵进行 Cholesky 分解，从而使 scipy 具备了大规模稀疏矩阵的高效 Cholesky 分解功能[1]。与直接使用 C 语言编程相比，性能损失在 5% 以下。

本文的结构如下：第二节简要介绍稀疏 Cholesky 分解相关理论，包括 Cholesky 分解的一般方法、稀疏矩阵存储原理和稀疏 Cholesky 分解步骤；第三节介绍 CHOLMOD 库的算法策略和 Cython 封装的实现细节；第四节介绍实验设置和结果；第五节是总结。

2. 理论背景

本节将简要介绍稀疏 Cholesky 分解的理论背景。我们略去细节证明，聚焦于算法流程和基本思想。理解本节的内容，将有助于理解 CHOLMOD 库的使用方式。

2.1 Cholesky分解的通用算法

对称正定矩阵 A 的 Cholesky 分解是指将其表示为一个非奇异下三角矩阵 L 及其转置乘积的形式。由基本的线性代数知识可知，表示 $A = LL^T$ 存在且唯一。利用 Cholesky 分解，可以高效地求解对称线性系统，加速数值计算。

通用 Cholesky 分解算法的基本方法主要包括向上查看法（Up-Looking）、向左查看法（Left-Looking）、向右查看法（Right-Looking）等。各种现行的算法大多以这些基本方法为基础，针对各种数据场景和计算设备进行优化。CHOLMOD 实现的分解算法就基于向上查看法和向左查看法，下面对其详细介绍。

2.1.1 向上查看法

向上查看 Cholesky 分解算法在每轮迭代中计算出矩阵 L 的一行，因此也被称为行 Cholesky 算法。考虑对等式$A = LL^T$的如下分块：

$$\begin{bmatrix} L_{11} & \\ l_{12}^T & l_{22} \end{bmatrix} \begin{bmatrix} L_{11}^T & l_{12} \\ & l_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & a_{12} \\ a_{12}^T & a_{22} \end{bmatrix}$$

其中 l_{12}, a_{12} 为 $n-1$ 维向量； l_{22}, a_{22}为标量； A_{11} 是原矩阵左上角的 $n-1$ 阶子矩阵，其 Cholesky 分解为 L_{11}。

[1] 本文涉及的所有代码见 https://github.com/zhanghx0905/My-Cholmod

假设我们已经求出了前 $n-1$ 行的分解结果（即矩阵 L_{11}），则由分块矩阵乘法可以得到第 n 行的值：

$$l_{12} = L_{11} \backslash a_{12}$$
$$a_{22} = l_{12}^T l_{12} + l_{22}^2$$

其中 “\” 符号代表求解一个三角方程组。向上查看法的伪代码如下：

算法 1.（向上查看 Cholesky 分解）
输入：待分解矩阵 A
输出：分解结果 L
for k = 1 : n **do**
$L_{k,1:k-1} = L_{1:k-1,1:k-1} \backslash A_{1:k-1,k}$
$L_{k,k} = \sqrt{A_{k,k} - L_{k,1:k-1} L_{k:1:k-1}^T}$

向上查看法在经过一些特别的优化后可用于稀疏矩阵的分解，当矩阵特别稀疏时，该方法甚至比很多复杂方法更有效。此类算法被应用在很多现代数值计算软件中，包括 CHOLMOD。

2.1.2 向左查看法

向左查看 Cholesky 分解算法在每轮迭代中计算出矩阵L的一列，因此也被称为列 Cholesky 算法。在迭代进行至第 k 轮时，考虑对等式 $A = LL^T$ 的如下分块：

$$\begin{bmatrix} L_{11} & & \\ l_{12}^T & l_{22} & \\ L_{31} & l_{32} & L_{33} \end{bmatrix} \begin{bmatrix} L_{11}^T & l_{12} & L_{31}^T \\ & l_{22} & l_{32}^T \\ & & L_{33} \end{bmatrix} = \begin{bmatrix} A_{11} & a_{12} & A_{31}^T \\ a_{12}^T & a_{22} & a_{32}^T \\ A_{31} & a_{32} & A_{33} \end{bmatrix}$$

其中 $[L_{11}; l_{12}^T; L_{31}]$ 是已经求出的前 $k-1$ 列，$[l_{22}; l_{32}]$ 为待求列，l_{22} 为标量。根据分块矩阵乘法，可得待求列与已求列之间的关系：

$$l_{22} = \sqrt{a_{22} - l_{12}^T l_{12}}$$
$$l_{32} = \frac{a_{32} - L_{31} l_{12}}{l_{22}}$$

引入一个 $n-k+1$ 维的向量 c 可以将上述两步运算进一步合并

$$c = \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} a_{22} \\ a_{32} \end{bmatrix} - \begin{bmatrix} l_{12}^T \\ L_{31} \end{bmatrix} l_{12}$$
$$l_{22} = \sqrt{c_1}, l_{32} = c_2 / l_{22}$$

根据这一递推关系，可以得到向左查看算法：

算法 2.（向左查看 Cholesky 分解）
输入：待分解矩阵 A
输出：分解结果 L
for k = 1 : n **do**
$c_{k:n} = A_{k:n,k} - L_{k:n,1:k-1} L_{k,1:k-1}^T$
$L_{k,k} = \sqrt{c_k}$
$L_{k+1:n,k} = c_{k+1:n} / L_{k,k}$

向左查看法非常适合各种针对稀疏矩阵分解的优化。注意到，计算向量 c 的矩阵向量乘法只需要对在 l_{12}^T中非零的那些列进行，当我们通过其它手段——如将在后文中介绍的消去树——分析出 L 的非零元模式后，向量 c 的计算效率将大大增加。向左查看法借助消去树的指导，可以分为很多子任务，更适合并行计算。向左查看法很容易改造为分块的版本（将矩阵的相邻几列合并，见下）这构成了超节点法（Supernodal）Cholesky 分解的基础，在实际应用中非常有效。

$$\begin{bmatrix} L_{11} & & \\ L_{12}^T & l_{22} & \\ L_{31} & l_{32} & L_{33} \end{bmatrix} \begin{bmatrix} L_{11}^T & L_{12} & L_{31}^T \\ & L_{22} & L_{32}^T \\ & & L_{33} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} & A_{31}^T \\ A_{12}^T & A_{22} & A_{32}^T \\ A_{31} & A_{32} & A_{33} \end{bmatrix}$$

2.2 稀疏矩阵的存储结构

鉴于稀疏矩阵中非零元素占比很小，为了节省空间，常采用压缩存储数据结构，各种稀疏矩阵算法就是专门为稀疏矩阵特殊的存储格式设计的。常见的稀疏矩阵存储格式包括三元组（COO）、压缩稀疏行（CSR）、压缩稀疏列（CSC）等。

由于稀疏 Cholesky 分解算法经常会以列为单位进行操作和运算，因此压缩稀疏列（CSC）很适合作为稀疏 Cholesky 分解的基础数据结构。

CHOLMOD 库的稀疏矩阵就是基于压缩稀疏列结构，此种结构将非零元按列顺序存储，不需要记录每个元素的列号，只需记录每列第一个非零元的位置。因此压缩稀疏列矩阵需要三个数组：非零元素值（val）、行编号数组（row）、列指针数据（pcol）。下面是一个 CSC 格式的稀疏矩阵的例子：

$$A = \begin{bmatrix} 1 & 0 & 2 & 0 & 0 \\ 3 & 4 & 0 & 5 & 0 \\ 6 & 0 & 7 & 0 & 8 \\ 0 & 9 & 0 & 6 & 0 \\ 0 & 0 & 0 & 0 & 7 \end{bmatrix}$$

表 1 稀疏矩阵 A 的 CSC 格式存储

val	1	3	6	4	9	2	7	5	6	8	7
row	1	2	3	2	4	1	3	2	4	3	5
pcol	1	4	6	8	10						

2.3 稀疏Cholesky分解算法

稀疏矩阵的 Cholesky 分解是专门针对稀疏矩阵的大规模矩阵分解算法，本节简要介绍其基本步骤和相关概念。

2.3.1 稀疏 Cholesky 分解步骤

稀疏 Cholesky 分解算法一般分为符号分析和数值分解两个阶段。在符号分析阶段，算法根据矩阵 A 的非零模式进行一系列分析，为后继的数值分解阶段做准备，使数值求解的时间复杂度和空间复杂度都大幅下降；数值分解阶段将计算出分解的结果。符号分析的开销相对较小。

在整个算法中，一个绕不开的概念是消去树（elimination tree），下面对其作简要介绍。

2.3.2 消去树

消去树指导了 Cholesky 分解的符号分析阶段，也为数值分解提供了一个基本框架。

在符号分析阶段，一般用图来表示矩阵的非零模式。形式上，n 阶对称正定矩阵 A 诱导出一个包含 n 个节点的图 $G_A = (V_A, E_A)$，其中 $V_A = \{0, \cdots, n-1\}, E_A = \{(i,j) | 0 \le j < i < n \ a_{ij} \ne 0\}$。

对于 Cholesky 分解 $A = LL^T, A = (a_{ij}), L = (l_{ij})$，有如下两个命题成立：

命题 1. 如果 $a_{ij} \ne 0$，则 $l_{ij} \ne 0$。

命题 2. 如果$i < j < k$ 且 $l_{ji} \ne 0, l_{ki} \ne 0$，则 $l_{kj} \ne 0$。

从这两个命题可以知道，L 继承了 A 的所有非零模式，且其非零模式由 A 的非零模式唯一确定。命题 2 还反映出矩阵的每个非零元对应着图上一个节点到另一个节点的可达情况，因此 L 对应的图表示 G_L 中存在很多冗余边。具体而言，若在图 G_L 中有$(i,j) \in E_L$，则对所有$k > j, (i,k) \in E_L$，边 (i,k) 实际上是冗余的，因为从节点i 可以途经 j 到达 k 。

从上文的讨论可知，每个节点 i 只需要维护至多一条出边 $(i, parent(i))$，其中 $parent(i) = \min\{j: (i,j) \in E_L\}$ 称为该节点对应的父节点（父列）。按照这样的剪枝规则得到的图称为消去树（elimination tree）。

图 1 给出了某稀疏矩阵 A 及其分解结果 L，根据上述算法，可得矩阵 A 的消去树。

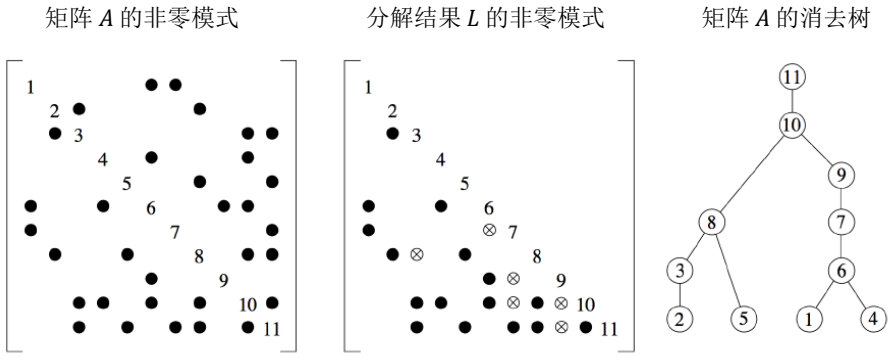

图 1 消去树与符号分析的案例

2.3.3 符号分析

在 Cholesky 分解过程，矩阵 L 在继承 A 的所有非零模式（命题 1）的同时会增加新的一些非零元，它们被称为填入元（fill-in）。填入元会造成分解结果的稀疏性下降，进而提高算法的空间和时间复杂度。

在符号分解阶段，首先会寻找一个排列阵 P（减少填入排列），使得 PAP^T 的分解结果中填入元尽量少。寻找到最优的减少填入排列是一个 NP 难问题，实践中一般采用启发式的算法，如最小度数（minimal degreee）、近似最小度数（approximate minimal degreee, AMD）、嵌套分割（nested dissection）等。

减小填入排列确定后，符号分析将对 A 进行符号分解，也就是计算出 L 的非零模式，以消去树的方式给出。

2.3.4 数值求解

稀疏 Cholesky 分解的数值方法主要有超节点法（supernodal）、多波前法（multifrontal）等。CHOLMOD 的实现为基于向左查看的超节点法。

在向左查看 Cholesky 分解算法中只考虑非零元，可以将其改造成处理稀疏矩阵的版本。从下面的伪代码中可以看出，稀疏 Choleksy 分解算法的每一步都是由非零元所主导的行或列的更新任务。

算法 3. （稀疏矩阵向左查看 Cholesky 分解）
输入：待分解矩阵 L （下三角部分）
输出：分解结果 L （就地分解）

```
for  j = 1 : n  do
    for  i = j : n  do
        L_ij = L_ij / sqrt(L_jj)
    for  k = j+1 : n  do
        if  L_kj ≠ 0  then
            for  p = k : n  do
                if  L_pj ≠ 0  then
                    L_pk = L_pk − L_pj L_kj
```

以稀疏向左查看 Cholesky 分解为基础，对多列构成的“块”执行向左查看方法，就得到了最基本的超节点法。

超节点法的核心思想是将稀疏结构相同或相似的列合并为超节点，列合并的规则由符号分析得到的消去树确定。稀疏矩阵的压缩存储节省了空间，但每次访问都要间接寻址，访存代价大大提高。合并稀疏结构相似的列，并在每个合并的块内采用稠密的方式记录非零元，可以抵消一部分压缩存储的负面影响。确定列合并方式后，采用分块向左查看法可以高效执行数值 Cholesky 分解。

3. 实现细节

3.1 CHOLMOD 中的 Cholesky 分解

使用 CHOLMOD 库完成一次 Cholesky 分解需要调用的例程主要有两个：cholmod_analyze 和 cholmod_factorize，前者进行符号分析（2.3.3），后者完成数值分解（2.3.4）。cholmod_analyze 会返回一个 Factor 结构体，该结构会作为 cholmod_factorize 的输入，指导后者计算出分解结果。调用 cholmod_analyze 时有两个主要的选项：寻找减小填入排列的方法以及数值分解的方法。

CHOLMOD 提供了多种寻找减少填入排列的算法，包括近似最小度数方法（CHOLMOD_AMD)，嵌套分割方法（CHOLMOD_METIS，CHOLMOD_NESDIS）等。CHOLMOD 的默认先尝试近似最小度数方法，如果效果不佳，则再尝试嵌套分割方法，详见算法 4。此外，用户还可以选择让 CHOLMOD 尝试所有搜索方式，并从中选取最优者。

算法 4.（CHOLMOD 选择减少填入排列的默认策略）
输入：待分解矩阵 A
输出：减少填入排列 P
$N_0 =$ 为待分解矩阵上三角部分非零元个数
使用近似最小度数方法寻找减少填入排列 P ，预计分解结果非零元个数为 N_1，后续浮点运算次数 M
if $N_1 > 5N_0 \wedge M > 500N_0$ **then**
尝试嵌套分割方法，选择更优的结果返回

CHOLMOD 支持的数值分解方法有两种模式：simplicial（基于向上查看法，2.1.1）和 supernodal（基于超节点法，2.3.4）。cholmod_analyze 根据选择的数值分解算法返回不同的符号分析结果——两种算法接受的数据格式不同，如超节点法需要超节点划分。CHOLMOD 也支持自动选择分解算法（算法 5）。如果在符号分析阶段执行的浮点运算次数与分解结果非零元个数的比值小于一个阈值（默认为 40），则选择 simpicial 方法，否则选择 supernodal 方法。

算法 5.（CHOLMOD 数值分解自动模式选择）
输入：待分解矩阵 A，符号分析结果（符号分析的浮点运算次数 M，分解结果的非零元个数 N）
输出：数值分解模式 mode
if $M/N < 40$ **then**
mode = simplicial
else
mode = supernodal

减少填入排列和分解方法都选定后，cholmod_analyze 例程将根据选定的方法分析分析以 cholmod_sparse 格式存储的原矩阵，得到的分析结果（消去树及其衍生信息）直接指导 cholmod_factorize 的计算。

3.2 将稀疏 Cholesky 分解嵌入 scipy.sparse

scipy.sparse 模块定义了多种稀疏矩阵类，而 CHOLMOD 使用的稀疏矩阵 cholmod_sparse 以 CSC 格式存储。为了让封装后的 CHOLMOD 能分解 scipy.sparse 稀疏矩阵，首先要实现后者与 cholmod_sparse 之间的相互转换。scipy.sparse 稀疏矩阵与 cholmod_sparse 稀疏矩阵相互转换的伪代码如下，在实现中尽可能直接操纵指针，避免了数据块的移动，以节省内存。

算法 6.（将 scipy.sparse 稀疏矩阵转换为 cholmod_sparse）
输入：scipy.sparse 稀疏矩阵 m
输出：cholmod_sparse 稀疏矩阵 out
if m 不为 CSC 格式 **then**
将 m 转为 CSC 格式
根据 m 中的数据，初始化 out 结构体
return out

算法 7.（将 cholmod_sparse 转换为 scipy.sparse CSC 稀疏矩阵）
输入：cholmod_sparse 稀疏矩阵 m
输出：scipy.sparse.csc_matrix 稀疏矩阵 out
根据 m 中的数据，初始化 out 对象
为 out 重写析构函数，保证 out 被析构时内存被正确释放
return out

实施 Cholesky 分解需要确定所使用的算法种类和列的排序方法，再依次调用 cholmod_analyze 和 cholmod_factorize 两个例程。封装后的 Cholesky 分解算法伪代码如下，函数返回 Factor 类对象，调用该类的接口

就能获得分解结果。

算法 8.（实施 Cholesky 分解）
输入：scipy.sparse 稀疏矩阵 A，分解模式 mode，排序方法 ordering_method
输出：分解结果 factor
将 A 转换为 cholmod_sparse 稀疏矩阵 m
根据 mode 和 ordering_method 初始化参数
调用 cholmod_analyze 进行符号分析，得到 factor
调用 cholmod_factorize 对 factor 进行分解
return factor

4. 实验与讨论

4.1 实验设置

实验的系统环境 Ubuntu 20.04，CPU 为 Intel(R) Core(TM) i7-8750H，SuiteSparse 版本为 5.7.1，CHOLMOD 版本为 3.0.14，Python 和 scipy 的版本分别为 3.8.5 和 1.6.3。实验以直接用 C 语言调用 CHOLMOD 为基线，验证调用 Python 接口分解稀疏矩阵的结果是否与之一致，并比较二者的性能。为了规避 OS 调度等因素的影响，性能测试的标准为多次运行取最优。

实验中用到的数据罗列如表 2，均为对称正定矩阵。这些矩阵规模庞大，只能以稀疏形式存储。图 3 展示了它们的非零元模式。

表 2 测例矩阵的基本信息

测例	行/列数	非零元数
ted_B	10605	77592
s3rmt3m3	5357	106526
thermomech_dM	204316	813716
parabolic_fem	525825	2100225

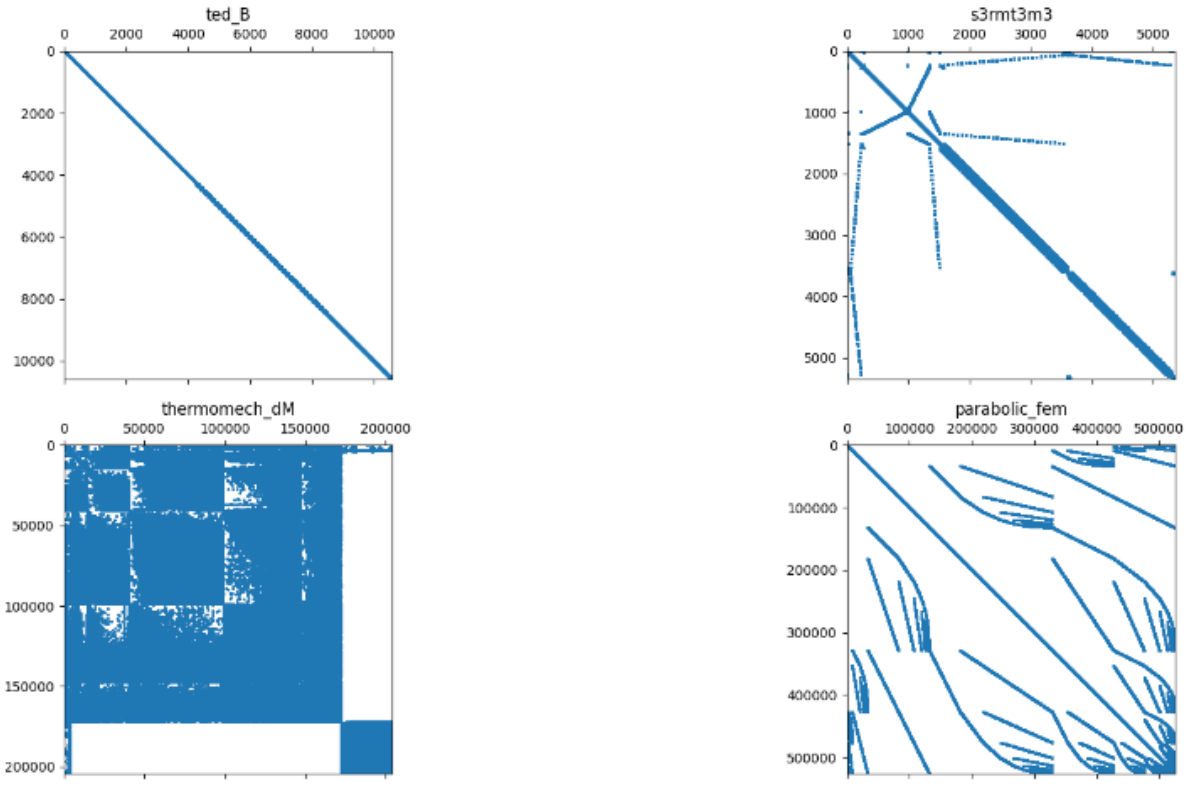

图 2 测例矩阵的非零元模式

4.2 实验结果

调用 Python 接口的分解结果与直接编写 C 程序得到的结果在误差范围内一致，确保了实验的正确性。

性能测试的结果如表 3 所示。由统计数据可知，与直接以 C 语言编程相比，调用 Python 接口进行分解花费的额外时间占原耗时比例在 5% 以内。

表 3 性能测试结果

测例	C 耗时	Python 耗时
ted_B	0.003957	0.004019
s3rmt3m3	0.023791	0.025165
thermomech_dM	0.686783	0.704579
parabolic_fem	5.768847	5.988630

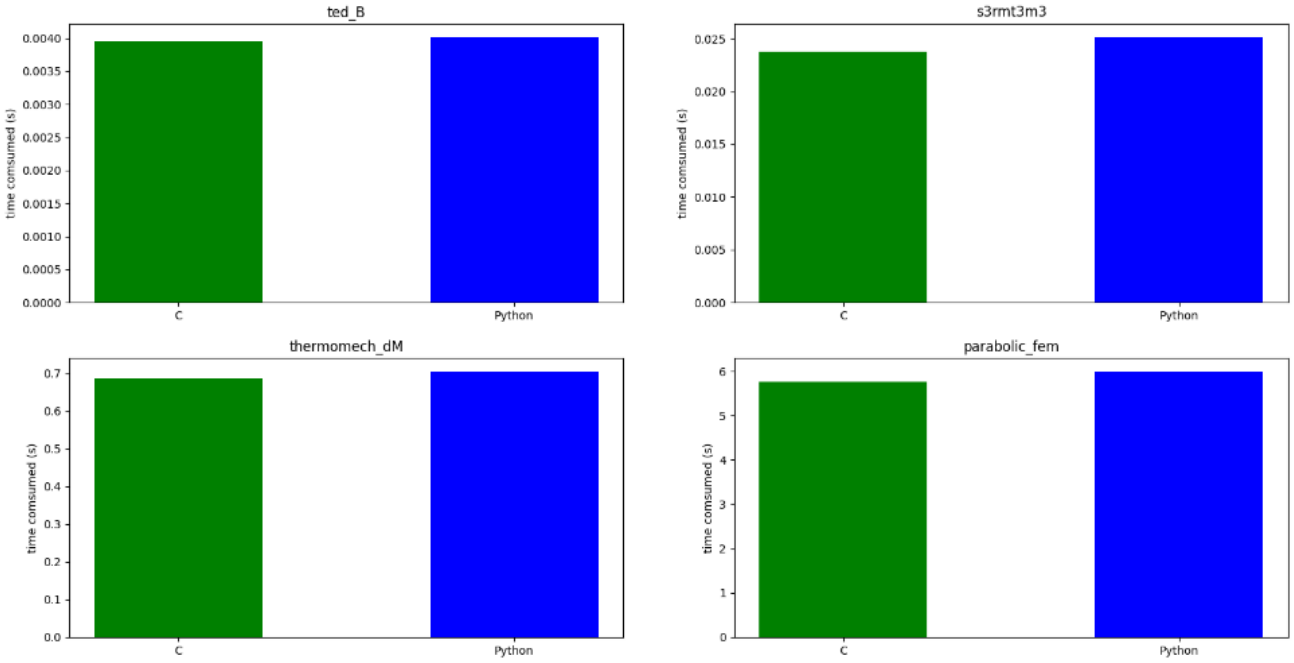

图 3 Python 和 C 接口效率对比图

5. 总结

本文利用 Cython 封装 SuiteSpars CHOLMOD 实现的大规模稀疏矩阵 Cholesky 分解模块，为 Python 科学计算库 scipy 实现正确而高效的稀疏矩阵分解算法，从而弥补了 scipy 和 numpy 没有稀疏矩阵 Cholesky 分解功能的缺憾。在文献调研和软件开发过程中，我们学习了用 Cython 调用 C 接口的技术，对于稀疏矩阵分解的基本方法有了更深的理解。

参考文献

[1] Davis, T., Rajamanickam, S., & Sid-Lakhdar, W. (2016). A survey of direct methods for sparse linear systems. Acta Numerica, 25, 383-566. doi:10.1017/S0962492916000076

[2] Timothy A. Davis, “User Guide for CHOLMOD: a sparse Cholesky factorization and modification package”, 2009

[3] 邹丹,窦勇,郭松.基于 GPU 的稀疏矩阵 Cholesky 分解[J].计算机学报,2014,37(07):1445-1454.

[4] 喻文健编著，《数值分析与算法》，清华大学出版社，2020.1