有限元专题调研——稀疏矩阵储存方法与稀疏求解器

管唯宇

2017 年 12 月 17 日

1 稀疏矩阵储存方法

稀疏矩阵的储存方式主要有：

1. Coordinate Format (COO)

   此种方法将单个元素按照坐标储存，对于每一个元素，都使用 row, column, value 三个数来保存其坐标和值，所需空间为 3 倍非零元个数。

   但此种储存方法难以在储存后对数组进行读取和操作，因为多用于持久化保存而较少用于核心算法中。

2. Diagonal Storage Format (DIA) 对角线储存

   此种方法按照对角线的顺序储存矩阵元素，对于非零元集中于对角线附近的带状稀疏矩阵，尤其是对称矩阵，此种方法较为友好。

3. Compressed Sparse Row Format (CSR) 行压缩储存

   行压缩矩阵按照行储存稀疏向量。

   对于每一个矩阵行，CSR 算法按照稀疏向量的方法储存，即用两个坐标 (index, value) 来储存稀疏向量中的元素。对于矩阵整体，CSR 将前 k 行非零元总数储存在 columns 数组中，因为共 2* 非零元个数 + 矩阵行数大小。

4. Compressed Sparse Column Format (CSC) 列压缩储存

   CSC 储存方法类似 CSR 方法，只是将储存行作为稀疏矩阵变为将储存列作为稀疏矩阵。

5. Skyline Storage Format

   Skyline 储存方式是一种变带宽，储存当前列第一个非零元到最后一个非零元（或对角线）的所有元素。

Skyline 储存方式常与 LU 分解绑定，因为其在 LU 分解时不需要调整就能兼容，不会受到其他方法中填入元 (Fill in) [1] 的干扰。

但是，若不进行带宽优化，Skyline 储存方法有可能储存大量的零元素，其最坏的情况下将储存整个矩阵（如箭型矩阵）。

6. Block Compressed Sparse Row Format (BSR) 块压缩储存

   块压缩矩阵按照分块的方式储存矩阵，记录 values, columns, 块开始的指标和块结束的指标。

7. 其他储存方式

   ELLPACK (ELL) 类似行压缩

   Hybrid (HYB)

   dok matrix：基于 keys 的字典稀疏矩阵。

   lil matrix 基于行链接列表的稀疏矩阵，增量式创建稀疏矩阵的结构。

2 稀疏求解器概述

对于大型稀疏矩阵，大部分直接法求解器依然采用 LU 分解。

传统的求解器所采用的储存方案多为一维变带宽储存方式（Skyline），它有求解稳定可靠、精度高、有大量成熟的解决方案等有点，但其没有充分利用稀疏矩阵的特性，导致在带宽优化能力有限的前提下求解缓慢，占用空间较大。[5]

稀疏直接求解器利用稀疏矩阵的更高效的储存方式（如 CSC、CSR 储存方式），因此相比一维变带宽储存方式有更高效的储存效率，但是传统的变带宽储存方式（Skyline）并不会带来填入元的问题，因此在进行带宽优化之后即可进行正常的 LU 分解，如 CSC 之类的稀疏储存方式则会带来额外的填充元问题，因此需要在数值计算之前进行符号分解。

一般来说，稀疏直接求解器求解步骤分为 4 步 [4]：

Phase 1. 对稀疏矩阵进行预处理，初等变换以减少非零填入元素 (Fill in) 元素对矩阵结构的破坏, 即 $\boldsymbol{A}' = \boldsymbol{P}^T\boldsymbol{A}\boldsymbol{P}$。

Phase 2. 对处理后的矩阵进行符号分解，确定填充元的位置。

Phase 3. 进行数值分解，$\boldsymbol{A} = \boldsymbol{L}\boldsymbol{U}$

Phase 4. 回代过程，求解 $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{f}$, 即 $\boldsymbol{L}\boldsymbol{U}\boldsymbol{x} = \boldsymbol{f}$, $\boldsymbol{L}\boldsymbol{y} = \boldsymbol{f}$, $\boldsymbol{U}\boldsymbol{x} = \boldsymbol{y}$

[1] 所谓填入元 (Fill in)，是指原矩阵 $\boldsymbol{A}$ 进行 LU 分解后的矩阵 $\boldsymbol{L}$ 在 $\boldsymbol{A}$ 零元的地方产生了非零元。

填充元的定义已在前文提到过。因为填充元的存在，求解的时空效率被严重影响。而通过交换矩阵的行列，可以大量减少填充元，从而大幅提高矩阵的求解效率。

另外，需要注意的是，数值分解过程中由于数值计算抵消出现的零并不能减少填充元。非零元仅指矩阵结构意义上的非零元。

举例说明，令

$$\boldsymbol{A} = \begin{pmatrix} 1.44118 & 0.157562 & 0.272554 & 0.13301 \\ 0.157562 & 1.34614 & 0 & 0 \\ 0.272554 & 0 & 0.505746 & 0 \\ 0.13301 & 0 & 0 & 0.490095 \end{pmatrix}$$

LU 分解得到

$$\boldsymbol{LU} = \begin{pmatrix} 1.44118 & 0.157562 & 0.272554 & 0.13301 \\ 0.109329 & 1.32891 & -0.0297979 & -0.0145418 \\ 0.189119 & -0.0224228 & 0.453533 & -0.0254808 \\ 0.0922925 & -0.0109426 & -0.0561829 & 0.476228 \end{pmatrix}$$

这是一个满阵。

而对 $\boldsymbol{A}$ 进行初等变换之后，得到

$$\boldsymbol{A'} = \begin{pmatrix} 0.490095 & 0. & 0. & 0.13301 \\ 0. & 0.505746 & 0. & 0.272554 \\ 0. & 0. & 1.34614 & 0.157562 \\ 0.13301 & 0.272554 & 0.157562 & 1.44118 \end{pmatrix}$$

再进行 LU 分解得到

$$\boldsymbol{LU'} = \begin{pmatrix} 0.490095 & 0. & 0. & 0.13301 \\ 0. & 0.505746 & 0. & 0.272554 \\ 0. & 0. & 1.34614 & 0.157562 \\ 0.271397 & 0.538915 & 0.117048 & 1.23976 \end{pmatrix}$$

可以看到，通过适量的重排，LU 分解可以大幅减少填充元。

3 填充元优化方法

对于有限大小的矩阵，显然存在一个最优的重排方法，矩阵的 LU 分解在这个排列下产生的填充元最小。但很可惜的是，最优的排序被证明是 N! 完全问题 [3]，因此对于每一个矩阵都计算其最优排列是不现实的，只能研究其近似最优解，尽量减少其填充元。

这个问题常用的方法是将研究矩阵 $\boldsymbol{A}$ 的填充元优化问题转换为其对应图 G 上的等价问题。[4]

3.1 将填充元优化问题转换为图问题

矩阵的对应图 G 是指，将稀疏矩阵的方程号作为节点，矩阵中的非零元为节点间的连线，这些点和线所组成的有向图。

对于对称矩阵，上述有向图退化为无向图。

关于矩阵填充元与矩阵的对应图 G 之间的关系，有如下定理：[1]

定理 1. *$L(i,j)$ 是填充元，当且仅当在矩阵 $\boldsymbol{A}$ 的对应图 G 中存在一个连接节点 i 和 j 的路径，路径上所有的点编号都不超过 $\min(i,j)$。*

通过这个定理，就可以从 $\boldsymbol{A}$ 的结构出发，在不进行任何数值计算的情况下计算出 $\boldsymbol{L}$ 的结构。

3.2 最小度量法 [4]

早期流行的有效算法是最小度算法，最小度量法的基本思想是不断挑选图 G 中的最小度量点，把它标定当前顺序号。标定后，按分解过程修改图 G，合并边、删除此点，重复至全部点被删除。最后所得到的顺序就是一个最小度量意义下的重排顺序。

最小度具有良好的局部最优性，算法实现方便、时空消耗低，但由于局部最优性的限制，优化质量对不同的问题参差不齐。

3.3 多重剖分法 [4]

多重剖分算法试图寻找全局最优解，避免了局部最优方法的质量不稳定情况。

该算法想法是寻找图 G 中的一个子集 $\boldsymbol{S}$，是的去除这些点后，剩下的点被分割成几个互不相连的分块。则原矩阵被分解为几个主对角线上的分块矩阵，分块后依然是主对角线上的分块矩阵。

4 快速直接解法

直接快速解法基于矩阵 A 的 LU 分解，核心算法是三重循环 [4]

$$A_{ij} = A_{ij} - \left(A_{ik}A_{kl}\right)/A_{kk}$$

采用不同的下标顺序，可以得到不同的算法。另外，根据内部数据访问的模式，还可以得到不同形式的算法。

不同的算法根据顺序有“向右算法”和“向左算法”这样不同的命名方式（或“向前算法”和“向后算法”）。不同形式的算法主要区别在于内存访问量不同和向量的检索方式不同。

在进一步介绍之前，我们首先需要了解消去树的概念。

4.1 消去树在快速直接算法中的应用

消去树（Elimination tree）在快速直接算法中起到了重要的作用。

消去树是一种数，其定义为 [2]

$$Parent[j] = \min\{i > j | l_{ij} \neq 0\}$$

关于消去树的性质，有如下定理 [2]：

定理 2. 如果 $l_{ij} \neq 0$，则 x_i 在消去树中是 x_j 中的祖先节点。

推论 1. 如果 $T_[x_i]$ 和 $T_[x_j]$ 是消除树中二独立的子数，则有，

$$\forall x_s \in T_[x_i], \forall x_t \in T_[x_j],\ l_{st} = 0$$

定理 3. 对于 $i > j$，则 l_{ij} 当且仅当在图 G 中存在一个路径

$$x_i, x_{p_1}, ..., x_{p_t}, x_j$$

使得 $\{x_{p_1}, ..., x_{p_t}, x_j\} \subseteq T[x_j]$

定理 4. $l_{ij} \neq 0$ 当且仅当

$$\exists x_k \subset T[x_i], s.t.\ a_{ik} \neq 0$$

关于消去树的更多性质、定理和具体算法，可以在文献 [2] 中找到。

消去树是快速直接解法中的一个重要的概念，直接快速解法几次大的进步都与消去树有关，以消去树为载体实现解法效能的提高。[4]

其中，多波前法（Multo-frontal, MF）最为成功，得到了广泛的应用。[4]

多波前法是向右算法的一个变体，当完成某一列的分解后，不立即对其右边的部分修改，而将修改的内容储存在一个稠密的矩阵中。通过这个名叫更新矩阵的缓存，多波前法可以达到更高的峰值速度，而且可以自然地移植到外存解法。但带来的缺点包括增加了额外的堆栈和运算量，以及堆栈储存量不定。[4]

其他的辅助方法还有超节点消去树快速直接解法，即“超方程”方法，通过合并结构完全相同的行列起到减少计算量的效果。结构完全相同的行列在图 G 中表现为具有完全相同的相邻节点的两个节点。[4] [6]

参考文献

[1] Timothy A Davis. *Direct methods for sparse linear systems*. SIAM, 2006.

[2] Joseph WH Liu. The role of elimination trees in sparse factorization. *SIAM Journal on Matrix Analysis and Applications*, 11(1):134–172, 1990.

[3] Mihalis Yannakakis. Computing the minimum fill-in is np-complete. *SIAM Journal on Algebraic Discrete Methods*, 2(1):77–79, 1981.

[4] 周洪伟, 吴舒, and 陈璞. 有限元分析快速直接求解技术进展. *力学进展*, 37(2):175–188, 2007.

[5] 梁峰 and 钱若军. 空间结构有限元分析的快速求解技术. *空间结构*, 9(4):3–8, 2003.

[6] 陈璞, 孙树立, and 袁明武. 有限元分析的直接快速解法. *力学 2000*”学术大会论文集, 2000.