

GSS 用户手册

v 2.0 beta

陈英时

gsp@grusoft.com

11/24/2007

GSS(GRUS SPARSE SOLVER) 是用于求解大型稀疏矩阵的软件包。采用最新的模型及算法，具有高效，稳定，通用等特点。对比测试表明：速度明显优于同类求解器，其中分解时间平均不到UMFPACK的一半。许多矩阵只有GSS可求解。

GSS支持多种数据类型和平台，提供多种接口和模式，已在多个行业得到应用。

GSS 用户手册	1
§1 简介	3
1.1 对称正定矩阵	3
1.2 对称不定矩阵	3
1.3 不对称矩阵	3
§2 多波前算法简介	4
§3 多波前算法的优点	6
§4 GSS 的主要特点	7
4.1 高效率计算	7
4.2 内存优化使用	7
4.3 稳定可靠	7
4.4 良好的通用性	7
4.5 多种调用模式	8
§5 测试结果	8
5.1 UMFPACK 测试集	8
§6 调用接口	13
6.1 简介	13
6.2 C 调用接口	13
6.3 FORTRAN 调用接口	15
§7 版本	17
参考文献	18

§ 1 简介

线性方程组求解方法主要有直接法和迭代法。以下主要讨论直接法。即对于

$$Ax=b \quad (1)$$

通过 A 的 LU 分解来得到 x 。当 A 中包含很多零元时，通常采用稀疏求解算法。

稀疏矩阵形态复杂，性质多变，因此求解的首要目标是稳健可靠。目前主流的求解器可以解出大部分矩阵，其中 GSS 和 UMFPACK 的可靠性最好，解的精度也较高，GSS 可解的矩阵最多。高效是另一个重要的目标。对称矩阵的速度以 GSS 和 PARDISO 为最好，非对称矩阵则 GSS 是最快的。就求解规模而言，这些求解器都可以求解百万阶以上的矩阵。值得注意的是，随着 64 位多核 CPU 的推出，PC 已具备求解百万阶矩阵的能力。

GSS 针对不同类型的稀疏矩阵采用不同的解法。

1.1 对称正定矩阵

GSS 采用 LL^T 分解 (cholesky 分解[15])。即对称正定阵 A 存在如下分解,其中 L 为下三角矩阵。

$$A=LL^T \quad (2)$$

cholesky 算法理论成熟，表现稳定，解的精度很高。稀疏求解中有多种高性能的实现，例如可分为 Left-looking、Right-looking 及 Crout 三类[11]。GSS 采用 Right-looking multi-frontal 算法，灵活、高效、并行性好。对某些正定矩阵，采用多波前/波前混合算法，以减少数据移动量，进一步提高速度。

1.2 对称不定矩阵

GSS 采用对称置换的 LDL^T 分解[15]。即

$$PA P^T = LDL^T \quad (3)$$

其中 L 是单位下三角阵， D 由阶数为 1 或 2 的对角块构成， P 是置换阵。

对称不定求解在理论上比较成熟，但具体实现还待提高。表现在稳健的算法性能较差，而高性能的算法不够稳健，无法解出各种类型的不定矩阵。例如 Bunch-Parlett 算法可以很好的解决满阵问题，但应用到稀疏矩阵上时，效率与稳定无法兼顾。GSS 正致力更高效稳健的不定阵算法。

1.3 不对称矩阵

GSS 采用不对称置换的 LU 分解。其基本形式如下：

$$PAQ=LU \quad (4)$$

其中 L 是单位下三角阵， U 是上三角阵， P, Q 是置换阵。

不对称矩阵的求解理论还在发展，还有很多可以改进的地方。GSS 在确保通用可靠的基础上，尽量提高效率。在符号分析中采用 block upper triangular 排序，QUOTIENT-GRAPH 模型，在数值分解中灵活运用多种主元方法。对于困难矩阵，并采用矩阵平衡，refine 等技术。实测中，综合性能较其它求解器有明显的优势，有一些矩阵只有 GSS 可以求解。

通过不对称置换把“大元素”置换到对角元，可提高速度和精度 [16]。GSS 采用网络流算法，这也是该算法首次用于“大元素”置换问题。

通过 block upper triangular 排序，可以得到一系列具有特殊性质的对角块，即 strong hall

matrix。随后的分析中充分利用这个特性，以提高效率。

依据矩阵的一些性质，GSS 采用 SIMPLE 和 HARD 两种策略。HARD 策略适用于非常不对称、对角元很多都是零、条件数很大等困难矩阵。在求解时会采用矩阵平衡技术，主元置换算法，动态符号分析及 iterative refine 等以提高解的精度，需要更多的时间。其它的矩阵采用 SIMPLE 策略，速度更快。

不对称矩阵数值分解过程中，仍需要一些符号分析，而符号分析算法的 flops 是很低的。GSS 采用很多技巧，将所需的符号分析工作减到最少。

§ 2 多波前算法简介

一、 稀疏求解器简介

目前主要的通用求解器有 UMFPACK[4]，PARDISO[9]，SuperLU[6]，WSMP[7]等。总体上比较稳定，可以求解大部分矩阵，效率也较高，精度一般要比迭代法高。

随着硬件的快速发展和 BLAS 等库的提出，许多过去要在大型机上求解的问题现在微机也能处理。[1]中的算例都在千阶左右，在今天看来，就规模而言已不足以作为算例。一般来说，P4 机完全可以胜任十万阶的矩阵，64 位多核 CPU 可以求解百万阶矩阵。[4]提到测试算例中的 BBMAT (38744 阶，分解后元素超过四千万)，1988 年在当时的巨型机 cray-2 上费时超过 1000 秒，而现在在一台普通的 P4 机上，GSS 只要 15 秒，在双核 CPU (AMD Athlon 64X2 4000+) 上，只要 7 秒多！

采用直接法求解稀疏矩阵有两种常用的模式。第一种是直接求解 $Ax = b$ 。第二种是将求解分为三个阶段：符号分析，数值分解及回代求解。分阶段模式更灵活，适用于许多情况：例如偏微分方程组的求解，矩阵需要反复分解，求解，但它们的结构都是一样的。所以只要进行一次符号分析，然后是多次数值分解和求解。第一种模式也可以看作是第二种模式的特例。各个阶段的主要任务如下：

I. 符号分析。

包括矩阵结构分析、注入元(fill-in)排序、符号分解等。它的结果用于指引数值分解。也有文献 ([11]等)把矩阵结构分析单独列出作为一个阶段

II. 数值分解。

将矩阵分解为下三角矩阵 L 和上三角矩阵 U。其中包括选主元，置换，装配，更新等过程，有时还需要动态符号分析。

III. 求解。基于 LU 矩阵前代回代，有时需要多次回代以得到较高的精度，见[10] § 12.2。

二、 多波前法简介

多波前法 (multifrontal) 是直接法中比较稳定和高效的算法，为 UMFPACK，WSMP 等采用。原理是找出、构造稀疏矩阵中的密集子块(frontal)，frontal 的分解直接调用 BLAS。frontal 的生成，消去，装配，释放等都是由特定的数据结构来指引。这些数据结构随矩阵类型而变，也是不同 multifrontal 求解器的主要区别所在。

Multifrontal 的起源可推到 generalized element[1]，[3]把它看做是 submatrix-Cholesky 分解的一种特殊形式。这时分析和应用的重点还都是对称矩阵。指引结构是消去树 (Elimination Tree)，[2]更是充分挖掘对称性，推导、归纳了一系列精彩的特性，这些特性对求解器的开发影响深远。以至于处理非对称矩阵时，人为添加一些零元素，使其结构对称[8]。这种做

法不可避免将增加许多 fill-in。UMFPACK 的做法有很多独到之处，它采用 Elimination Tree，整体结构简明；但在实现 frontal 的过程中，结合了主元策略和类似于 AMD 的排序。从而既减少了 fill-in，又提高了精度。

随后提出了 Elimination DAG[12]作为 Tree 的一种推广，用以指引非对称矩阵的分解过程。虽然 DAG 更通用，但其实现要复杂很多，相应的主元策略也更困难。UMFPACK 的早期版本[17]就隐含使用了 DAG，后来还是采用了 TREE。WSMP[7]进一步将 DAG 分为 TASK DAG 和 DATA DAG。其总体性能与 UMFPACK 相当，具体矩阵各有长短。短时间内，TREE 和 DAG 还难言优劣。GSS 采用非对称的 frontal 数值分解阶段的主要指引结构是 DATA DAG。。GSS 1.0 在符号分析阶段用到了 TASK DAG，在 1.1 中添加了 QUOTIENT GRAPH 模型，不仅提高符号分析的速度，数值分解也快很多，尤其是一些不对称的矩阵。

如上所述，多波前法的思路很简单，但具体实现的算法与结构多变。为方便理解，以下描述最简单的一种情况：采用多波前法实现对称正定矩阵 LL^T 分解[26]。

以下记对称正定矩阵 A 的阶数为 N ，下标 i, k 表示矩阵第 i 行, k 列的元素，上标 T 表示矩阵的转置。 A 存在唯一的对角元素为正的下三角矩阵 L ，使 $A=LL^T$ [1]。记 L 矩阵第 k 列的结构为

$L(k) = \{i | L_{i,k} \neq 0, k \leq i \leq N\}$ 。波前的定义为连续的不同结构的列所构成的子矩阵。

多波前法实现 LL^T 分解的算法简要如下：

1 符号分析

通过符号分解得到 L 的结构，找出波前划分并排序。

波前划分算法如下。

(1) s 初始值为 1

(2) 寻找从第 s 列开始的波前 $F(s, K, t)$ ，最后一列 t 的定义如下

$$t = \max\{ t | t \geq s; L(s) = L(t) \cup \{s, s+1, \dots, t-1\} \}$$

(3) $s=t+1$ 。 $s>N$ 则退出，否则转(2)

2 数值分解：

通过分块算法得到 A 的 LL^T 分解。设 L 存在 M 个波前

(1) m 初始值为 1

(2) 第 m 个波前的处理包含集成，消元和修正过程。

为了提高效率，波前的消元过程采用分块算法，即将 $F(s, K, t)$ 如图 1 划分为 D, B

$$D = \{L_{i,k} | L_{i,k} \neq 0, s \leq k \leq i \leq t\}$$

$$B = \{L_{i,k} | L_{i,k} \neq 0, s \leq k \leq t; t \leq i\}$$

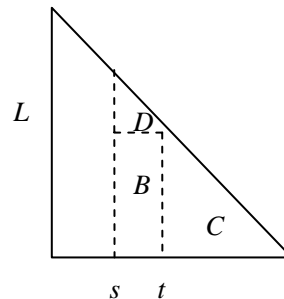


图 1、波前示意

将 D 分解为 $D_L D_L^T$ 后，修正 B 如下： $B' = B D_L^T$

修正 $C = \{L_{i,k} | L_{i,k} \neq 0, t \leq k, i \leq N\}$ 如下： $C' = C - B' B'^T$

(3) $m=m+1$ 。 $m>M$ 则退出，否则转(2)

3 回代求解。

尽管构造这些波前需要一些额外的开销，但主要的运算量集中在数值分解中，由于 D, B, C 都是密集阵，可直接调用高效的 BLAS[22]，从总体上提高了效率。

以下是一个简单的 10 阶矩阵，图 2(a)是 A 的结构图，圆点代表非零元。图 2(b)是 $L + LT$ 的结构图，+ 代表注入元。共有 7 个波前： $\{a\} \{b\} \{c\} \{d\} \{e\} \{f, g\} \{h, i, j\}$ 。

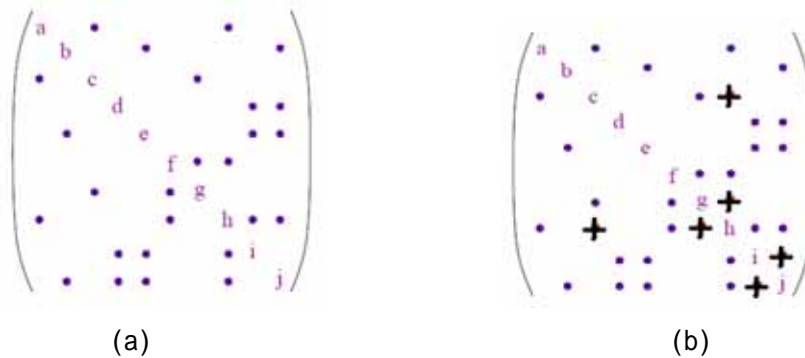


图 2 对称矩阵 LL^T 分解， L 的波前划分 $\{a\} \{b\} \{c\} \{d\} \{e\} \{f, g\} \{h, i, j\}$

§ 3 多波前算法的优点

多波前法只存储非零元的数值和位置，存储效率也较高。带宽，变带宽存储要记录轮廓线 (skyl ine) 之间的所有元素，往往很多都是零元素，效率很低。在固定的平台上，空间的节省意味着求解规模的增加，这是由于系统分配的内存是有上限的，一般取决于整数的字长。例如对于常用的 32 位操作系统，地址空间的上限只有 4G。因此对于一些大型矩阵，半带宽存储求解需要 64 位平台，而多波前法在 32 位平台上即可求解。

同一个波前不管含多少列，结构都是类似的。因此只需记录最左边那列的位置信息，其他列共用这些信息。波前越宽，节省的空间越多。而索引存储，链接表存储等都要记录每列非零元素的数值和位置，因此效率要低于多波前法。

波前的消去顺序通常是可变的，这使得多波前法灵活多变。例如图 2 中，波前 $\{a\}, \{b\}, \{d\}$ 彼此无关，并行算法中可以同时消去。还可通过排序以减少对内存的需求，具体见[3]。

多波前法与波前法[1]有很多区别，以下列举一二。

1、矩阵排序。波前法的排序目标是优化轮廓线之间的距离，以避免过大的波前。轮廓线之间一般还是会有零元。多波前法的排序目标则是尽量减少注入元，一般情况下其注入元要小于波前法。

2、波前排序。分析波前之间的关系，根据目标确定最优的排序，这些是多波前法所独有的，也更灵活。

3、并行性。波前算法的并行性一般局限于单个波前中。多波前法中独立的波前可以同时消去，并行潜力更大。

4、波前包含的行列。波前法中，用同一个波前遍历所有的行列，可以看作是动态的。多波前法中，符号分析完成后，每个波前对应的行列即固定，相对静态。即使动态调整，也只包含部分行列。

5、数据装配。两者都有数据移动，装配的过程，具体运算量取决于矩阵类型，有些矩阵波前法的装配量要少很多。

波前法，多波前法都还在发展中，并出现了混合算法[23]，值得关注。GSS 的实现中，也包括一些混合算法，供用户选用。

§ 4 GSS 的主要特点

GSS 采用最新的模型及算法，具有高效，节省内存，稳定，通用等特点。具体如下。（其中*标注的是 GSS 独有的最新功能，在专业版中提供）。

4.1 高效率计算

GSS 采用波前动态划分化算法等技术来优化 BLAS 使用，通过波前/多波前混合算法减少数据的装配。

BLAS 的效率和矩阵的规模正相关，即波前的规模越大，flops 越高。GSS 在波前划分的时候，尽可能生成大的波前，以发挥 BLAS 的效率。

数据移动是波前装配所必须的，虽然运算量不是很大，但其 flops 远低于 BLAS，因此用的时间并不少。对于很多矩阵，波前法的数据移动量远小于多波前法，GSS 采用混合算法来降低数据移动的数量。

4.2 内存优化使用

内存一直是大型稀疏矩阵 LU 分解的瓶颈，GSS 采用优化注入元排序，out-core 优化调度等算法来优化内存的使用。

注入元算法主要有两类，最小度排序（minimum degree）[18]和嵌套排序（nested dissection）[19]，常用的有 AMD[18]，METIS[19]，COLAMD[20]等。大量测试表明这些算法各有所长，对应的 LU 规模可相差数倍。METIS 总体上最好，但时间要多于最小度算法，是 GSS 的缺省排序算法。

*GSS 也提供其它排序算法及用户自定义排序。

*GSS 提供最小度排序和嵌套排序的混合算法。注入元排序已被证明为 NP 完全问题 [13]，上述两类算法的结果都只是近似最优。采用混合算法可以得到更好的结果。

当 LU 的规模超过内存时，操作系统会自动采用虚拟内存等技术，但这些技术并不适合大型稀疏矩阵，大量时间用于页面交换，CPU 的空置率高。GSS 开发了专用的 out-core 调度算法，动态调整 LU，充分发挥 CPU 的效率。

*对于 32 位系统来说，out-core 算法还可以突破 4G 的限制（Win32 下实际为 2G）。

4.3 稳定可靠

主元置换是以确保稳定性的重要手段。全选主元的时间复杂度比分解本身还要高[15]，只用于极少数情形。GSS 虽提供这种选项，但并不鼓励使用。列选主元得到了最普遍使用，是 GSS 的缺省选项。列选主元曾被认为可以提供足够的精度，但最近发现了一些反例[14]。rook pivoting[14]是最近才提出的。理论上可以保证和全选主元类似的精度，时间复杂度和列选主元类似。GSS采用rook pivoting作为列选主元的补充，提高某些矩阵的精度。

static pivoting[21]也是最近才提出，并用于SuperLU的并行版本和PARDISO等。对于不满足要求的对角元，改为较大的值，这样可以减少行列置换等需要的开销。但往往会降低解的精度，GSS只在一定的前提下选用。

*GSS采用新的矩阵平衡技术，对某些困难矩阵，精度可以提高几个数量级。

4.4 良好的通用性

GSS 提供 FORTRAN,C,MATLAB 等调用接口。GSS 支持各种操作平台，支持 32 位与

64 位系统。

GSS 支持压缩列(行)、等带宽,变带宽等多种存储格式。GSS 支持对称阵的下三角存储。

GSS 支持多种文件格式,包括

MATLAB format。

Rutherford/Boeing format。

Matrix Market forma。

Harwell Boeing format (<http://math.nist.gov/MatrixMarket/formats.html#hb>)。

基于 GSS 内核的 GSSMEX 可直接由 MATLAB 调用。

GSS 支持数据类型如下:浮点数,双精度浮点数,复数,双精度复数。

GSS 并行版支持共享内存的多(核)CPU 并行机;支持 INTEL Hyper-Threading。

4.5 多种调用模式

GSS 支持多种调用模式:包括一次符号分析,多次数值分解,多次求解的组合。可同时求解多个右端项。

支持 out-core 模式。当内存不足时,将部分 LU 输出到外存。

*GSS 支持 LU 部分更新模式。在微分方程求解,非线性求解等应用中,矩阵往往只有部分数据发生变化,只需更新其对应的 LU,从而大大提高效率。

*GSS 支持转置回代。即基于 A 的 LU 分解,求解 $A^T x = b$ 或 $A^H x = b$ 。

§ 5 测试结果

本节对比了 GSS、UMFPACK 和 INTEL PARDISO 的速度和精度。

UMFPACK[4]由 Florida 大学的 Timothy A. Davis 开发。著名的数学软件 MATLAB 也把 UMFPACK 作为求解器。UMFPACK 的设计考虑了多种数据类型的组合,可以在各种平台上编译。UMFPACK 得到广泛应用,许多求解器也把它作为衡量的依据。

INTEL PARDISO[9]是一个并行的求解器,它采用了 left and right-looking Level-3 BLAS supernode techniques,并行性良好。对比测试的版本来自于 INTEL MKL 10.0。

以下用一些广泛使用的算例来对比这三者的表现,总体而言 GSS 的性能是最优的,但每个求解器都有最快的算例。这三者采用了不同的模型和算法,当矩阵的特性与模型比较匹配时,自然速度更快,精度也更高。GSS 内部采用了多套模型和算法,表现最高效,稳定。

5.1 UMFPACK 测试集

[UF sparse matrix collection](#) 收集了进 2000 个矩阵,是目前最广泛的稀疏矩阵集。UMFPACK 测试集[4]从中选取了 35 个算例,具有广泛的代表性,这些算例也经常被用于其它求解器的测试。这 35 个算例被分为三个子集(表 1 到表 3): symmetric set, 2-by-2 set, unsymmetric set。UMFPACK 内部自动识别这些类并采取不同的算法。Sym 栏是矩阵的对称系数:定义为对称的非对角元素与所有非对角元素的比值。

测试环境：AMD Athlon 64X2 4000+。2G 内存。

操作系统：Windows 2000 sp4。

UMFPACK,GSS,INTEL PARDISO 都采用了缺省的参数。

表 4 是数值分解时间对比，用时最少的用下划线标出。表 5 是精度对比，精度最高的用下划线标出。精度对比采用如下方式：对于方程 $Ax=b$ ，令 $x_0=1.0$ ； $b=Ax_0$ ，将 $|x-x_0|$ 作为精度的衡量。

从表中可以看出：每个求解器都有最快的算例。这也印证了稀疏矩阵千变万化，目前并不存在一个绝对最优的求解器。

1、 GSS 与 UMFPACK 对比。

GSS 的速度明显快于 UMFPACK，其总的时间不到 UMFPACK 的一半。在对称算例和大算例中更为明显，往往要快数倍，甚至可提高一个数量级。仅有 EX40，TWO-TONE，RIM 三个算例要慢于 UMFPACK，最多慢 30%。

GSS 与 UMFPACK 可以解出所有算例，相当稳健。UMFPACK 的精度总体上好一些，GSS 与其基本一致，各有所长。其中 GARON2，RMA10 明显低于 UMFPACK，还待提高。

2、 GSS 与 PARDISO 对比

GSS 速度快于 PARDISO。总的时间仅为 PARDISO 的 20%，扣掉那些 PARDISO 失败的算例，总的时间约为 PARDISO 的 70%。PARDISO 在 2-by-2 set 上略快一些：2-by-2 set 中，PARDISO 在 GOODWIN，RIM 算例上快很多，而 GSS 求解 PSMIGR_1 要快很多。

PARDISO 仅解出 unsymmetric set 13 个很多算例中的 3 个。就通用求解器的标准而言，PARDISO 的模型显然不适合不对称矩阵。

表 1 symmetric set of UMFPACK

Group	Name	n	Nonzeros(i n 1000's)	Sym.	description
NORRIS	TORSO2	115967	1033.5	0.992	2D human torso, electro-phys finite-diff
SIMON	OLAFU	16146	1015.2	1.000	Structure problem
SIMON	VENKAT01	62424	1717.8	1.000	Unstructured 2D Euler problem
BAI	AF23560	23560	460.6	0.944	airfoil
SIMON	RAEFSKY3	21200	1488.8	1.000	Fluid-structure, turbulence
ZHAO	ZHAO1	33861	166.5	0.922	electromagnetics
ZHAO	ZHAO2	33861	166.5	0.922	electromagnetics
FIDAP	EX11	16614	1096.9	1.000	3D fluid flow, cylinder and plate
SIMON	RAEFSKY4	19779	1316.8	1.000	Container buckling problem
WANG	WANG4	26086	177.2	1.000	3D MOSFET semiconductor
RONIS	XENON1	48600	1181.1	1.000	Zeolite, sodalite crystals
VANHEUKELEUM	CAGE10	11397	150.6	1.000	DNA electrophoresis
NORRIS	STOMACH	213360	3021.6	0.848	3D electro-physical, human duodenum

表 2 2-by-2 set of UMFPACK

Group	Name	n	Nonzeros(in 1000's)	Sym.	description
GOODWIN	GOODWIN	7320	324.8	0.635	Fluid mechanics, finite-element
AVEROUS	EPB2	25228	175.0	0.670	Plate-fin heat exchanger
GARON	GARON2	13535	373.2	0.999	2D finite-element, Navier-Stokes
GOODWIN	RIM	22560	1015.0	0.639	Fluid mechanics, finite-element
NORRIS	HEART2	2339	680.3	1.000	Quasi-static FEM, human heart
AVEROUS	EPB3	84617	463.6	0.667	Plate-fin heat exchanger
BOVA	RMA10	46835	2329.1	1.000	3D model of Charleston Harbor
NORRIS	HEART1	3557	1385.3	1.000	Quasi-static FEM, human heart
HB	PSMIGR_1	3140	543.2	0.479	Population migration

表 3 unsymmetric set of UMFPACK

Group	Name	n	Nonzeros(in 1000's)	Sym.	description
AT&T	ONETONE2	36057	222.6	0.116	Harmonic balance method
GRAHAM	GRAHAM1	9035	335.5	0.718	Navier-Stokes, finite-element
MALLYA	LHR34C	35152	764.0	0.002	Light hydrocarbon recovery
SHEN	E40R0100	17281	553.6	0.308	
MALLYA	LHR71C	70304	1528.1	0.002	Light hydrocarbon recovery
FIDAP	EX40	7740	456.2	1.000	Navier-Stokes, FEM (3D)
AT&T	ONETONE1	36057	335.6	0.076	Harmonic balance method
VAVASIS	AV41092	41092	1683.9	0.001	Unstructured finite-element
AT&T	TWOTONE	120750	1206.3	0.246	Harmonic balance method
HB	PSMIGR_2	3140	540.0	0.479	Population migration
SIMON	BBMAT	38744	1771.7	0.529	2D airfoil, turbulence
HOLLINGER	G7JAC200SC	59310	717.6	0.025	Economic modeling
HOLLINGER	MARK3JAC140SC	64089	376.4	0.061	Economic modeling

表 4 数值分解时间对比

SET	Matrix	数值分解时间			$\frac{GSS}{UMFPACK}$	$\frac{GSS}{PARDISO}$
		UMFPACK	PARDISO	GSS		
symmetric	TORSO2	2.766	1.109	<u>1.063</u>	0.38	0.96
	OLAFU	1.609	0.704	<u>0.594</u>	0.37	0.84
	VENKAT01	2	<u>0.781</u>	0.922	0.46	1.18
	AF23560	1.625	1.297	<u>1.188</u>	0.73	0.92
	RAEFSKY3	1.781	0.984	<u>0.937</u>	0.53	0.95
	ZHAO1	1.906	0.656	<u>0.578</u>	0.3	0.88
	ZHAO2	2.172	0.656	<u>0.579</u>	0.27	0.88
	EX11	2.765	1.39	<u>1.328</u>	0.48	0.96
	RAEFSKY4	4.656	1.781	<u>1.657</u>	0.36	0.93
	WANG4	3.453	1.437	<u>0.953</u>	0.28	0.66
	XENON1	8.156	4.172	<u>3.984</u>	0.49	0.95
	CAGE10	8.547	2.875	<u>1.719</u>	0.2	0.6
	STOMACH	31.75	18.75	<u>13.36</u>	0.42	0.71
2-by-2	GOODWIN	0.328	<u>0.063</u>	0.297	0.91	4.71
	EPB2	0.484	0.281	<u>0.234</u>	0.48	0.83
	GARON2	0.5	<u>0.157</u>	0.172	0.34	1.1
	RIM	1.438	<u>0.203</u>	1.687	1.17	8.31
	HEART2	0.344	0.234	<u>0.219</u>	0.64	0.94
	EPB3	1.031	<u>0.297</u>	0.422	0.41	1.42
	RMA10	2.031	<u>0.75</u>	0.875	0.43	1.17
	HEART1	0.985	<u>0.719</u>	<u>0.719</u>	0.73	1
	PSMIGR_1	4.188	9.781	<u>2.953</u>	0.71	0.3
unsymmetric	ONETONE2	0.344	0.859	<u>0.328</u>	0.95	0.38
	GRAHAM1	1.015	<u>0.109</u>	0.5	0.49	4.59
	LHR34C	0.625	227.016	<u>0.547</u>	0.88	0
	E40R0100	0.672	0.437	<u>0.282</u>	0.42	0.65
	LHR71C	1.25	<u>0.437</u>	1.078	0.86	2.47
	EX40	0.484	<u>0.282</u>	0.656	1.36	2.33
	ONETONE1	1.187	5.672	<u>0.625</u>	0.53	0.11
	AV41092	20.36	5.672	<u>1.485</u>	0.07	0.26
	TWOTONE	<u>3.032</u>	18.5	3.907	1.29	0.21
	PSMIGR_2	4.25	9.797	<u>3.11</u>	0.73	0.32
	BBMAT	14.219	13.094	<u>7.266</u>	0.51	0.55
	G7JAC200SC	18.094	24.812	<u>12.938</u>	0.72	0.52
	MARK3JAC140SC	16.531	3.625	<u>3.062</u>	0.19	0.84
sum		166.578	359.389	<u>72.224</u>	0.43	0.2

表 5 精度对比

SET	Matrix	精度			$\frac{GSS}{UMFPACK}$	$\frac{GSS}{PARDISO}$
		UMFPACK	PARDISO	GSS	$UMFPACK$	$PARDISO$
symmetric	TORSO2	4.115E-14	<u>3.588E-14</u>	7.627E-14	1.854	2.1259
	OLAFU	9.438E-09	<u>4.831E-09</u>	8.94E-09	0.947	1.8508
	VENKAT01	1.314E-13	<u>1.16E-13</u>	1.639E-13	1.248	1.4126
	AF23560	3.783E-13	<u>2.946E-13</u>	9.434E-13	2.494	3.202
	RAEFSKY3	<u>2.434E-12</u>	3.07E-12	5.891E-12	2.42	1.9193
	ZHAO1	8.485E-14	<u>7.227E-14</u>	2.194E-13	2.586	3.0362
	ZHAO2	<u>3.786E-12</u>	4.469E-12	3.84E-12	1.014	0.8593
	EX11	<u>1.663E-05</u>	1.823E-05	1.802E-05	1.084	0.9882
	RAEFSKY4	0.0004368	<u>0.0001855</u>	0.0001994	0.456	1.0748
	WANG4	2.887E-11	<u>2.065E-12</u>	2.693E-11	0.933	13.039
	XENON1	2.134E-11	<u>1.48E-11</u>	2.008E-11	0.941	1.3568
	CAGE10	2.894E-14	<u>1.592E-14</u>	4.247E-14	1.468	2.6679
	STOMACH	1.122E-13	<u>3.741E-14</u>	2.044E-13	1.821	5.4636
2-by-2	GOODWIN	<u>4.767E-10</u>	2.325E-09	1.007E-07	211.2	43.298
	EPB2	4.87E-13	3.119E-13	<u>2.457E-13</u>	0.504	0.7877
	GARON2	1.249E-11	<u>1.145E-11</u>	6.117E-05	5E+06	5E+06
	RIM	1296060	957416	<u>70194.5</u>	0.054	0.0733
	HEART2	<u>1.103E-12</u>	0.0246186	1.156E-12	1.048	5E-11
	EPB3	<u>8.149E-12</u>	9.754E-12	9.019E-12	1.107	0.9247
	RMA10	1.645E-11	<u>1.337E-11</u>	0.0004178	3E+07	3E+07
	HEART1	2.551E-11	0.0265926	<u>2.02E-11</u>	0.792	8E-10
unsymmetric	PSMIGR_1	<u>3.069E-11</u>	0.0339591	9.908E-11	3.229	3E-09
	ONETONE2	<u>1.108E-10</u>	-	1.371E-10	1.237	-
	GRAHAM1	<u>6.808E-09</u>	0.0004706	3.164E-08	4.647	7E-05
	LHR34C	0.364407	-	<u>0.217368</u>	0.596	-
	E40R0100	4.611E-10	-	<u>4.21E-10</u>	0.913	-
	LHR71C	0.152529	-	<u>0.0768207</u>	0.504	-
	EX40	2.517E-12	2.978E-12	<u>2.45E-12</u>	0.973	0.8226
	ONETONE1	<u>1.933E-10</u>	-	2.553E-10	1.321	-
	AV41092	<u>3.104E-12</u>	-	6.315E-10	203.4	-
	TWOTONE	<u>1.174E-10</u>	-	3.644E-09	31.03	-
	PSMIGR_2	1.134E-11	-	<u>4.919E-12</u>	0.434	-
	BBMAT	<u>1.237E-08</u>	1.339E-08	1.509E-08	1.22	1.1266
	G7JAC200SC	5.892E-07	-	<u>5.357E-07</u>	0.909	-
MARK3JAC140SC	3.4E-07	-	<u>3.185E-07</u>	0.937	-	

§ 6 调用接口

6.1 简介

GSS 的函数名有如下的结构

<函数>_<系统><数据>

<函数>列举如下:

	含义
GSS_init	初始化, 对系统, 矩阵等做必要的检查
GSS_symbol	符号分析
GSS_numeric	数值分解
GSS_solve	回代求解
GSS_clear	释放内存

<系统>列举如下:

	含义
i	32 位串行: 单 CPU,
m	32 位并行: 多 CPU, 多核 CPU, Hyper-Threading
l	64 位串行: 单 CPU
x	64 位并行: 多 CPU, 多核 CPU, Hyper-Threading

<数据>列举如下:

	含义	C	FORTRAN
s	实数 单精度	float	REAL
d	实数双精度	double	DOUBLE PRECISION
c	复数 单精度	自定义*	COMPLEX
z	复数 双精度	自定义*	DOUBLE COMPLEX

*标准 c 不支持复数, 可用自定义结构替代

GSS 标准接口采用压缩列 (compressed column storage) 存储格式。对于对称阵, 只需传入下三角矩阵的数据。压缩列格式的示例参见接口说明, 也可参见 http://www.netlib.org/linalg/html_templates/node92.html。

6.2 C 调用接口

按C惯例, 数组下标从0开始。

以下声明中统一使用5个参数记录矩阵, 采用压缩列格式。

int nRow, nCol

矩阵的行数, 列数。

int *ptr, int *ind, double *val
(ptr, ind, val) 对应压缩列格式的列指针，行下标和元素值。每列的行下标递增排序。

对于N阶有nnz个非零元的矩阵来说:

ind, val的长度是nnz。按列顺序记录每个非零元的行标和数值。

ptr的长度是N+1。ptr[i]记录第i列第一个非零元的位置，最后一个元素ptr[N]=nnz。

这样第i列的长度就是ptr[i+1]-ptr[i]。

例如对于如下3阶的矩阵

1.0 0.0 5.0

0.0 3.0 6.0

2.0 4.0 7.0

则 ptr[4]={0,2,4,7};

ind[7]={0,2,1,2,0,1,2};

val[7]={1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0};

以下声明中的 FLOATING 指 GSS 支持的 float, double 等四种数据类型。

以下声明中的GRUS_OK定义为0。

int **GSS_init_?**

(int nRow, int nCol, int* ptr, int* ind, FLOATING *val, int type, double *setting);

初始化

参数：

type 矩阵类型，定义如下。

0： 缺省值为0

11： 对称正定矩阵。(ptr, ind, val)为矩阵下三角(含对角线)的数据。

12： 对称不定矩阵。(ptr, ind, val)为矩阵下三角(含对角线)的数据。

setting[32] 控制参数。

返回值：

初始化成功返回GRUS_OK, 否则返回错误代码

void* **GSS_symbol_?**(int nRow, int nCol, int* ptr, int* ind, FLOATING *val);

符号分析

参数：

返回值：

成功返回求解器的指针， 否则返回0x0

int **GSS_numeric_?**(int nRow, int nCol, int* ptr, int* ind, FLOATING *val, void *hSolver);

数值分解

参数：

hSolver 指向求解器的指针

返回值：

完成分解返回GRUS_OK, 否则返回错误代码

int GSS_solve_?

(void *hSolver, int nRow, int nCol, int *ptr, int *ind, FLOATING *val, FLOATING *rhs);

回代求解

参数：

hSolver 指向求解器的指针

rhs 方程组的右端项，返回的解也存储在rhs中

返回值：

完成求解返回GRUS_OK, 否则返回错误代码

int GSS_clear_? (void* hSolver);

释放求解器占用的内存

参数：

hSolver 指向求解器的指针

返回值：

完成释放返回GRUS_OK, 否则返回错误代码

6.3 FORTRAN 调用接口

按 FORTRAN 惯例，数组下标从 1 开始。参见 gss_6_demo.for，必须使用 use GSS_6_INTERFACE。

以下声明中统一使用5个参数记录矩阵，采用压缩列格式。

int nRow, nCol

矩阵的行数，列数。

int *ptr, int *ind, double *val

(ptr, ind, val) 对应压缩列格式的列指针，行下标和元素值。每列的行下标递增排序。

对于 N 阶有 nnz 个非零元的矩阵来说：

nRow=N, nCol=N。

ind, val 的长度是 nnz。按列顺序记录每个非零元的行标和数值。

ptr 的长度是 N+1。ptr[i]记录第 i 列第一个非零元的位置,最后一个元素

ptr[N+1]=nnz+1。这样第 i 列的长度就是 ptr[i+1]-ptr[i]。

例如对于如下 3 阶的矩阵

1.0 0.0 5.0

0.0 3.0 6.0

2.0 4.0 7.0

则 ptr[4]={1,3,5,8}

```
ind[7]={1,3,2,3,1,2,3}
val[7]={1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0}
```

以下声明中的 FLOATING 指 GSS 支持的 float, double 等四种数据类型。
以下声明中的 GRUS_OK 定义为 0。

```
function GSS_init_? ( nRow, nCol, ptr, ind, val, m_type, setting)
    integer GSS_init_?
    integer nRow,nCol,ptr(*),ind(*),m_type
    FLOATING val(*),
    double precision setting(32)
```

初始化

参数：

m_type 矩阵类型，定义如下。
0： 缺省值为 0
11： 对称正定矩阵。(ptr, ind, val)为矩阵下三角(含对角线)的数据。
12： 对称不定矩阵。(ptr, ind, val)为矩阵下三角(含对角线)的数据。
control [32] 控制参数。

返回值：

初始化成功返回 GRUS_OK, 否则返回错误代码

```
function GSS_symbol_?( nRow, nCol, ptr, ind, val )
    integer GSS_symbol_?
    integer nRow,nCol,ptr(*),ind(*)
    FLOATING val(*)
```

符号分析

参数：

返回值：

成功返回求解器的指针，否则返回 0x0

```
function GSS_numeric_? ( nRow, nCol, ptr, ind, val, hGSS )
    integer hGSS, GSS_numeric_?
    integer nCol, ptr(*), ind(*)
    FLOATING val(*)
```

数值分解

参数：

hGSS 指向求解器的指针

返回值：

完成分解返回GRUS_OK, 否则返回错误代码

```
function GSS_solve_?(hGSS,nRow,nCol,ptr,ind,val,rhs )
    integer hGSS, GSS_solve_?, nRow, nCol, ptr(*), ind(*)
    FLOATING val(*), rhs(*)
```

回代求解

参数：

hGSS 指向求解器的指针
rhs 方程组的右端项，返回的解也存储在rhs中

返回值：

完成求解返回GRUS_OK, 否则返回错误代码

```
function GSS_clear_? (hGSS)
    integer hGSS, GSS_clear_?
```

释放求解器占用的内存

参数：

hGSS 指向求解器的指针

返回值：

完成释放返回GRUS_OK, 否则返回错误代码

§ 7 版本

一、版本

通用版

用于求解各种类型的稀疏矩阵。

专业版

提供许多 GSS 独有的功能，速度更快。适用于难度较大的矩阵。
支持用户特定的要求。

体验版

为了方便用户，推广 GSS，特推出体验版。适用于十万阶以下的矩阵，可从 <http://www.grusoft.com> 等免费下载。

二、历史

GSS 2.0 发布于12/25/2007，改进如下：

1. 采用新的矩阵平衡技术，显著提高一些矩阵的求解精度，降低运算量。
2. LU部分更新。
3. out-of-core/in-core混合处理，完善了内存管理模块。

4. 波前/多波前混合算法
5. 完善iter-refine模块，可估计矩阵的条件数。

GSS 1.2发布于11/25/2005。主要改进如下：

6. 发布共享内存多CPU机上的并行版本。
7. 发布支持INTEL Hyper-Threading的并行版本。
8. 改进了数值分解算法，提高了分解对称或比较对称矩阵的速度。
9. 改进了static pivoting，取得较好的效果。
10. 增加iterative refine，提高解的精度。

GSS 1.1发布于9/12/2005。主要改进如下：

11. 符号分析增加QUOTIENT GRAPH模型。
12. 改进了对角元排序算法。
13. 改进了数值分解算法，大幅提高了非对称矩阵分解的速度。
14. 增加了行列比例化矩阵模块。
15. 增加了与INTEL PARDISO对比测试的结果。

GSS1.0 发布于7/20/2005。

参考文献

- [1] I.S.Duff, A.M.Erisman, and J.K.Reid. Direct Methods for Sparse Matrices. London:Oxford Univ. Press,1986.
- [2] J.W.H.Liu. The Role of Elimination Trees in Sparse Factorization. SIAM J.Matrix Anal.Applic.,11(1):134-172,1990.
- [3] J.W.H.Liu. The Multifrontal Method for Sparse Matrix Solution: Theory and Practice. SIAM Rev., 34 (1992), pp. 82--109.
- [4] T.A.Davis. A column pre-ordering strategy for the unsymmetric-pattern multifrontal method, ACM Trans. Math. Software, vol 30, no. 2, pp. 165-195, 2004.
- [5] T.A.Davis. UMFPACK Version 4.4 User Guide. 2005
- [6] DEMMEL, J. W., EISENSTAT, S. C., GILBERT, J. R., LI, X. S., AND LIU, J. W. H. A supernodal approach to sparse partial pivoting. SIAM J. Matrix Anal. Applic. 20, 3, 720-755. 1999.
- [7] GUPTA, A. Improved symbolic and numerical factorization algorithms for unsymmetric sparse matrices. SIAM J. Matrix Anal. Applic. 24, 529-552. 2002.
- [8] AMESTOY, P. R. AND PUGLISI, C. An unsymmetrized multifrontal LU factorization. SIAM J.Matrix Anal. Applic. 24, 553-569. 2002.
- [9] Intel Math Kernel Library Reference Manual
- [10] N.J.Higham. Accuracy and Stability of Numerical Algorithms. SIAM,2002
- [11] J.J.Dongarra,I.S.Duff, D.C.Sorensen, H.A.van der Vorst. Numerical Linear Algebra for

High-Performance Computers.

- [12] Elimination structures for unsymmetric sparse LU factors. Gilbert, John R.; Liu, Joseph WH SIAM J. Matrix Anal. Appl. 14, no. 2, 334--352, 1993.
- [13] Yannokakis M. Computing the minimum fill-in in NP-Complete. SIAM J. Algebraic Discrete Methods, 1981, 2:77~79
- [14] L.V.Foster. The growth factor and efficiency of Gaussian elimination with rook pivoting.
- [15] G.H.Golob, C.F.Van loan. Matrix Computations. The Johns Hopkins University Press. 1996
- [16] DUFF, I. S. AND KOSTER, J. On algorithms for permuting large entries to the diagonal of a sparse matrix. SIAM J. Matrix Anal. Applic. 22, 4, 973–996. 2001.
- [17] T. A. Davis and I. S. Duff, "An unsymmetric-pattern multifrontal method for sparse LU factorization", SIAM J. Matrix Analysis and Applications, vol. 19, no. 1, pp. 140-158, 1997
- [18] AMESTOY, P. R., DAVIS, T. A., AND DUFF, I. S. 1996a. An approximate minimum degree ordering algorithm. SIAM J. Matrix Anal. Applic. 17, 4, 886–905.
- [19] GKarypis, V.Kumar. METIS user's guide, version 4.0. 1998
- [20] Timothy A. Davis, John R. Gilbert, Stefan I. Larimore, and Esmond G. Ng. A column approximate minimum degree ordering algorithm. TR-00-005
- [21] I. S. Duff and S. Pralet, Towards a stable static pivoting strategy for the sequential and parallel solution of sparse symmetric indefinite systems.
- [22] D.Dodson and J.Lewis , Issues relating to extension of the Basic Linear Algebra Subprograms[J] , ACM SIGNUM Newsletter , 20(1):2-18 , 1985
- [23] T.A.Davis and I. S. Duff. A combined unifrontal/multifrontal method for unsymmetric sparse matrices, ACM Transactions on Mathematical Software (TOMS), v.25 n.1, p.1-20, March 1999
- [24] Miroslav Tuma. A note on the LDL. T. decomposition of matrices from saddle-point problems. SIAM J. Matrix Anal. Appl., 23(4):903–915, 2002
- [25] I. S. Duff and S. Pralet. Strategies for scaling and pivoting for sparse symmetric indefinite problems.RAL-TR-2004-020
- [26] 陈英时;吴文勇等. 采用多波前法求解大型结构方程组. 建筑结构(9) 138-140.2007.