“神威•太湖之光”计算机系统
高性能扩展数学库 xMath 1.0
用户手册

杨超、刘芳芳、孙乔、敖玉龙、赵玉文、张佳佳

中国科学院软件研究所
并行软件与计算科学实验室

2016 年 6 月 15 日

目 录

1 系统概述

“神威•太湖之光”计算机系统高性能扩展数学库 xMath（众核版和片上多核版）是一套在国产申威 CPU 上运行，支持申威众核新处理器特点，具有单核组众核并行化和片上多核并行化特征且掌握源代码的扩展数学库。在该库的研制过程中，针对国产 CPU 研究一系列数学库性能优化关键技术和框架，并利用这些关键技术进行性能优化，以提升数学库整体的性能。

本软件，包括以下 5 个子模块：

- BLAS；
- LAPACK 3.5.0；
- FFT 信号处理子程序；
- 稀疏线性系统求解子程序包；
- ScaLAPACK 2.0.2。

其中 ScaLAPACK 模块本软件未做改动，请参考其用户手册。本用户手册，将分别对其余 4 个模块，进行介绍和函数说明等。

2 使用方法

2.1 链接方法

本库需要使用 sw5f90 进行链接，由于内部使用了从核函数，所以需要加上-hybrid 参数。示例如下：

1） 编译用户程序主核程序

```
sw5cc -host -o test.o test.c
```

2） 链接

```
sw5f90 -hybrid -o test test.o -L/path/to/xMath -lxMath
```

2.2 运行设置

本库需使用全部从核资源，提交任务时需要指定满从核数（64）；本库内部通过从核函数的局部变量使用从核 LDM，所以需要添加-b 参数，否则性能会极差。示例如下：

```
bsub –I –b –N 1 –q q_sw_zky –cgsp 64 ./test
```

由于本库使用了大量从核 LDM，所以有可能会覆盖用户编写的从核函数中使用的 LDM 静态变量（__thread_local 声明的变量）。如果出现此种情况，需要设置环境变量 XMATH_PRESERVE_LDM=1，此时本库会对 LDM 使用前和使用后分别进行保存和恢复操作。示例如下：

```
export XMATH_PRESERVE_LDM=1
bsub –I –b –N 1 –q q_sw_zky –cgsp 64 ./test
```

2.3 出错信息排查方法

用户遇到问题，首先需要查看用户手册，检查安装是否正确，运行参数配置是否正确，以及是否和其他的运行程序冲突。其次，如果看见错误提示信息和错误代码，可以查看用户手册，找到错误代码对应的错误原因，然后审查程序调用的参数是否合理。最后，如果采取上述方案仍然无法排除错误，请立即联系相关维护人员，由维护人员现场排错。

对于维护人员，追溯问题最基本的就是横向定位和纵向记录。为了规范，现场维护人员需要按照如下步骤来操作：

定位：首先，需要设计好测试用例。从简单到复杂，从小规模到大规模，从一般情况到特殊边界，总之必须保证覆盖到所有的分支。其次，尽量使用 gdb 工具，如果平台支持不足情况（例如此众核平台），就需要用深入源文件进行 printf 定位。printf 输出又分为定位错误位置的输出和审查程序内容的输出。然后，对错误的程序做减法，直至剩下的程序运行正确，然后逐步添加逻辑，在不影响已有的代码前提下，需要保证新添加的代码符合预期。按照这种方式迭加直到程序运行正确。最后，为了排除用户的输入数据不合理导致的错误，需要添加适当的代码来提示用户和开发人员。

记录：不管是开发过程还是维护过程都需要认真对待。开发过程中每次遇到错误，开发人员需要记录下错误的输出信息，错误的位置，错误的所引发的影响。对于整个团队来说，还需要同步和共享相关信息。需要分清错误是个人问题，还是集成问题。为了便于最终维护，一方面我们需要在源程序中做好注释，须知注释除了说明程序函数使用方法，还有一个重要功能就是标注错误相关信息以便后来人借鉴。另一方面需要一个版本控制工具，例如 git。采用中心共享模式：建立一个中心代码库，开发人员还需要建立本地代码库，协同是通过中心代码库来实现。维护人员现场排除错误后切记做好错误记录和用户反馈。

3 BLAS 模块

BLAS(Basic Linear Algebra Subprograms)是一个线性代数核心子程序的集合，主要包括向量和矩阵的基本操作。它是最基本和最重要的数学库之一，广泛应用于计算科学，特别是在科学建模中。目前世界上有关矩阵运算的软件几乎都调用 BLAS 库；重要的稠密线性代数算法软件包(如 EISPACK、LINPACK、LAPACK 和 ScaLAPACK 等)的底层都是以 BLAS 为支撑。BLAS 目前已成为线性代数领域的一个标准 API 库。

BLAS 分成三个等级：

- 1 级：进行向量-向量操作；
- 2 级：进行向量-矩阵操作；
- 3 级：进行矩阵-矩阵操作。

BLAS 子程序名的结构如下：

<character><name><mod>( )

<character>域指明数据类型：

s 单精度实数

c 单精度复数

d 双精度实数

z　双精度复数

一些子程序会使用组合的字符代码，如 sc 或者 dz。

<name>域在 BLAS level 1 中指明操作类型。在 BLAS level 2 和 3 中,则反应矩阵的类型：

ge　普通矩阵

gb　普通带状矩阵

sy　对称矩阵

sp　对称矩阵(packed storage)

sb　对称带状矩阵

he　Hermitian 矩阵

hp　Hermitian 矩阵(packed storage)

hb　Hermitian 带状矩阵

tr　三角矩阵

tp　三角矩阵(packed storage)

tb　三角带状矩阵

<mod>域提供了操作的附加细节信息。BLAS level 1 中的<mod>域可能是如下字符：

c　共轭向量

u　非共轭向量

g　Givens 变换

BLAS level 2 中的<mod>域可能是如下字符：

mv　矩阵-向量乘

sv　使用矩阵-向量操作解线性方程组

r　秩 1 更新

r2　秩 2 更新

BLAS level 3 中的<mod>域可能是如下字符：

mm　矩阵-矩阵乘

sm　使用矩阵-矩阵操作解线性方程组

rk　秩 k 更新

r2k　秩 2k 更新

下面的例子说明了如何解释 BLAS 子程序名：

ddot <d><dot>：双精度实数向量点积

cdotc　<c><dot><c>：复数向量点积,共轭

sgemv　<s><ge><mv>：矩阵向量乘,普通矩阵,单精度

ztrmm　<z><tr><mm>：矩阵-矩阵乘,三角阵,双精度复数

下面,分别对 BLAS Level 1,2 和 3 的各个函数进行说明,包括输入接口,数据类型,功能等等。

3.1 BLAS Level 1 函数说明

表 3-1 BLAS Level 1 函数汇总表

子程序组	数据类型	描述
?asum	s,d,sc,dz	向量元素求和
?axpy	s,d,c,z	标量-向量乘

?copy	s,d,c,z	向量复制
?dot	s,d	点积
?sdot	sd,d	扩展精度点积
?dotc	c,z	共轭点积
?dotu	c,z	非共轭点积
?nrm2	s,d,sc,dz	欧几里得范数
?rot	s,d,cs,zd	Plane 变换
?rotg	s,d,c,z	Givens 变换
?rotm	s,d	修改的 plane 变换
?rotmg	s,d	修改的 Givens 变换
?scal	s,d,c,z,cs,zd	向量-标量乘
?swap	s,d,c,z	向量-向量交换
i?amax	s,d,c,z	向量最大绝对值的下标
i?amin	s,d,c,z	向量最小绝对值的下标

3.1.1 ?asum

语法

FORTRAN 77：

```
res = sasum(n, x, incx)
res = scasum(n, x, incx)
res = dasum(n, x, incx)
res = dzasum(n, x, incx)
```

描述

?asum 函数用于计算一个实向量中各个元素之和,或者一个复数向量各个实数和虚数部分的元素之和：

res = |Re x(1)| + |Im x(1)| + |Re x(2)| + |Im x(2)|+ ... + |Re x(n)| + |Im x(n)|

其中 x 代表有 n 个元素的向量。

输入参数

n　　整数类型。定义向量 x 的元素数目。

x　　sasum 函数下为实数类型
　　dasum 函数下为双精度浮点类型
　　scasum 函数下为复数类型
　　Dzasum 函数下为双精度复数类型
　　数组,规模至少是(1+(n-1))*abs(incx)。

incx　　整数类型。定义检索向量的步长。

输出参数

res　　sasum 函数下结果为实数类型
　　dasum 函数下结果为双精度浮点类型
　　scasum 函数下结果为实数类型
　　dzasum 函数下结果为双精度类型
　　包含向量中各个元素实数部分和叙述部分元素之和。

3.1.2 ？axpy

语法

FORTRAN 77:

```
call saxpy(n, a, x, incx, y, incy)
call daxpy(n, a, x, incx, y, incy)
call caxpy(n, a, x, incx, y, incy)
call zaxpy(n, a, x, incx, y, incy)
```

描述

？axpy 函数定义了如下所示的向量-向量操作：

```
y := a*x + y
```

其中:a 代表标量,x 和 y 代表有 n 个元素的向量。

输入参数

n　　整数类型。定义向量 x 和 y 的元素数目。

a　　saxpy 函数下为实数类型
　　daxpy 函数下为双精度浮点类型
　　caxpy 函数下为复数类型
　　zaxpy 函数下为双精度复数类型
　　定义标量 a。

x　　saxpy 函数下为实数类型
　　daxpy 函数下为双精度浮点类型
　　caxpy 函数下为复数类型
　　zaxpy 函数下为双精度复数类型
　　数组,规模至少是(1+(n-1))*abs(incx)。

incx　　整数类型。定义向量 x 的元素增量。

y　　saxpy 函数下为实数类型
　　daxpy 函数下为双精度浮点类型
　　caxpy 函数下为复数类型
　　zaxpy 函数下为双精度复数类型
　　数组,规模至少是(1+(n-1))*abs(incy)。
　　incy　　整数类型。定义向量 y 的元素增量。

输出参数

y　　向量 y 的更新结果。

3.1.3 ？copy

语法

FORTRAN 77:

```
call scopy(n, x, incx, y, incy)
call dcopy(n, x, incx, y, incy)
call ccopy(n, x, incx, y, incy)
call zcopy(n, x, incx, y, incy)
```

描述

？copy 函数实现了如下定义的向量-向量操作：

y = x,

其中 x 和 y 是向量。

输入参数

n　　整数类型。定义向量 x 和 y 的元素数目。

x　　scopy 函数下为实数类型
　　dcopy 函数下为双精度浮点类型
　　ccopy 函数下为复数类型
　　zcopy 函数下为双精度浮点类型
　　数组,规模至少是(1+(n-1))*abs(incx)。

incx　整数类型。定义向量 x 的元素增量。

y　　scopy 函数下为实数类型
　　dcopy 函数下为双精度浮点类型
　　ccopy 函数下为复数类型
　　zcopy 函数下为双精度浮点类型
　　数组,规模至少是(1+(n-1))*abs(incy)。

incy　整数类型。定义向量 x 的元素增量。

输出

y　　如果 n 是正数,输出 y 是向量 x 的拷贝。否则参数未被改变。

3.1.4 ？dot

语法

FORTRAN 77:

res = sdot(n, x, incx, y, incy)

res = ddot(n, x, incx, y, incy)

描述

？dot 函数实现了如下描述的向量-向量间的规约操作:

res = Σ(x*y),

其中 x 和 y 代表向量。

输入参数

n　　整数类型。定义向量 x 和 y 的元素数目。

x　　sdot 函数下为实数类型
　　ddot 函数下为双精度浮点类型
　　数组,规模至少是(1+(n-1))*abs(incx)。

incx　整数类型。定义向量 x 的元素增量。

y　　sdot 函数下为实数类型
　　ddot 函数下为双精度浮点类型
　　数组,规模至少是(1+(n-1))*abs(incy)。

incy　整数类型。定义向量 y 的元素增量。

输出参数

res　　sdot 函数下结果为实数类型
　　ddot 函数下结果为双精度浮点类型

如果 n 是正数,结果是向量 x 和 y 的点积。否则 res 为 0。

3.1.5 ？sdot

语法

FORTRAN 77:

res = sdsdot(n, sb, sx, incx, sy, incy)

res = dsdot(n, sx, incx, sy, incy)

描述

？sdot 函数计算两个扩展精度向量的内积结果。上述两个函数都使用中间结果的扩展精度累积,但是 sdsdot 函数的最终输出结果是单精度浮点类型。而 dsdot 函数的输出结果是双精度浮点类型。sdsdot 函数还把标量 sb 加入了内积计算中。

输入参数

n　　整数类型。定义了输入向量 sx 和 sy 的元素数目。

sb　　实数类型。单精度标量加入内积计算中(只针对 sdsdot 而言)。

sx,sy　　实数类型。

数组,规模大小分别至少为 1+(n-1))*abs(incx)和 1+(n-1))*abs(incy)。　　包含单精度向量输入。

incx　　整数类型。定义向量 sx 的元素增量。

incy　　整数类型。定义向量 sy 的元素增量。

输出参数

res　　sdsdot 函数下结果为实数类型

dsdot 函数下结果为双精度浮点类型

如果 n 是正数,结果是 sx 和 sy 的点积,否则对于 sdsdot 函数,结果是 sb,对于 dsdot 函数结果是 0。

3.1.6 ？dotc

语法

FORTRAN 77:

res = cdotc(n, x, incx, y, incy)

res = zdotc(n, x, incx, y, incy)

描述

？dotc 函数实现如下所示的向量-向量运算：

res = Σ(conjg(x)*y)

其中 x 和 y 是含有 n 个元素的向量。

输入参数

n　　整数类型。定义了向量 x 和 y 的元素数目。

x　　cdotc 函数下为复数类型

zdotc 函数下为双精度浮点类型

数组,规模至少为(1 + (n -1)*abs(incx))。

incx　　整数类型。定义向量 x 的元素增量。

y　　cdotc 函数下为复数类型
　　zdotc 函数下为双精度浮点类型
　　数组,规模至少为(1 + (n -1)*abs(incy))。
incy　整数类型。定义向量 y 的元素增量。

输出参数

res　　cdotc 函数下结果为复数类型
　　zdotc 函数下为双精度浮点类型
　　如果 n 是正数,结果为共轭向量 x 和非共轭向量 y 的点积。否则结果是 0。

3.1.7 ？dotu

语法

FORTRAN 77:

```
res = cdotu(n, x, incx, y, incy)
res = zdotu(n, x, incx, y, incy)
```

描述

？dotu 函数实现了如下定义的向量-向量规约操作：

res = Σ(x*y)

其中 x 和 y 代表 n 个元素的复数向量。

输入参数

n　　整数类型。定义了向量 x 和 y 的元素个数。
x　　cdotu 函数下为复数类型
　　zdotu 函数下为双精度复数类型
　　数组,规模至少是(1+(n-1))*abs(incx)。
Incx　整数类型。定义向量 x 的元素增量。
y　　cdotu 函数下为复数类型
　　zdotu 函数下为双精度复数类型
　　数组,规模至少是(1+(n-1))*abs(incy)。
incy　整数类型。定义向量 y 的元素增量。

输出参数

res　　cdotu 函数下结果为复数类型
　　zdotu 函数下为双精度复数类型
　　如果 n 是正数,结果是 x 和 y 的点积,否则结果为 0。

3.1.8 ？nrm2

语法

FORTRAN 77:

```
res = snrm2(n, x, incx)
res = dnrm2(n, x, incx)
res = scnrm2(n, x, incx)
res = dznrm2(n, x, incx)
```

描述

？nrm2 函数实现了如下定义的向量规约操作：

res = ||x||,

其中：x 是一个向量,res 是向量 x 元素的模长。

输入参数

n　　整数类型。定义了向量 x 和 y 的元素个数。

x　　snrm2 函数下为实数类型
　　dnrm2 函数下为双精度浮点类型
　　scnrm2 函数下为复数类型
　　dznrm2 函数下为双精度复数类型
　　数组,规模至少是(1+(n-1))*abs(incx)。

incx　　整数类型。定义了向量 x 的元素增量。

输出参数

res　　snrm2 函数下结果为实数类型
　　dnrm2 函数下结果为双精度浮点类型
　　scnrm2 函数下结果为实数类型
　　dznrm2 函数下结果为双精度浮点类型
　　包含向量 x 的欧几里得范数。

3.1.9 ？rot

语法

FORTRAN 77:

```
call srot(n, x, incx, y, incy, c, s)
call drot(n, x, incx, y, incy, c, s)
call csrot(n, x, incx, y, incy, c, s)
call zdrot(n, x, incx, y, incy, c, s)
```

描述

给出两个复数向量 x 和 y,每个向量的各个元素被替换为：

x(i) = c*x(i) + s*y(i)

y(i) = c*y(i) - s*x(i)

输入参数

n　　整数类型。定义向量 x 和 y 的元素数目。

x　　srot 函数下为实数类型
　　drot 函数下为双精度浮点类型
　　csrot 函数下为复数类型
　　zdrot 函数下为双精度复数类型
　　数组,规模至少为(1 + (n-1)*abs(incx))。

incx　　整数类型。定义 x 的元素增量。

y　　srot 函数下为实数类型
　　drot 函数下为双精度浮点类型
　　csrot 函数下为复数类型
　　zdrot 函数下为双精度复数类型

数组,规模至少为(1 + (n-1)*abs(incy))。

incy　整数类型。定义 y 的元素增量。

c　srot 函数下为实数类型
drot 函数下为双精度浮点类型
csrot 函数下为实数类型
zdrot 函数下为双精度浮点类型
一个标量。

s　srot 函数下为实数类型
drot 函数下为双精度浮点类型
csrot 函数下为实数类型
zdrot 函数下为双精度浮点类型
一个标量。

输出参数

x　每个元素被替换为 c*x + s*y。

y　每个元素被替换为 c*y - s*x。

3.1.10 ？rotg

语法

FORTRAN 77:

```
call srotg(a, b, c, s)
call drotg(a, b, c, s)
call crotg(a, b, c, s)
call zrotg(a, b, c, s)
```

描述

给出一个点 p 的笛卡尔积坐标(a,b),该函数返回参数 a,b,c,以及 s 关于给定点旋转后的 y 坐标为 0 的结果。

输入参数

a　srotg 函数下为实数类型
drotg 函数下为双精度浮点类型
crotg 函数下为复数类型
zrotg 函数下为双精度复数类型
提供点 p 的 x 坐标。

b　srotg 函数下为实数类型
drotg 函数下为双精度浮点类型
crotg 函数下为复数类型
zrotg 函数下为双精度复数类型
提供点 p 的 y 坐标。

输出参数

a　包含参数 x 旋转之后的结果。

b　包含参数 z 旋转之后的结果。

c　srotg 函数下为实数类型

drotg 函数下为双精度浮点类型
crotg 函数下为实数类型
zrotg 函数下为双精度浮点类型
包含参数 c 旋转之后的结果。

s srotg 函数下为实数类型
drotg 函数下为双精度浮点类型
crotg 函数下为实数类型
zrotg 函数下为双精度浮点类型
参数 s 旋转之后的结果。

3.1.11 ？rotm

语法

FORTRAN 77:

```
call srotm(n, x, incx, y, incy, param)
call drotm(n, x, incx, y, incy, param)
```

描述

给出两个复数向量 x 和 y,每个向量的各个元素被替换为如下形式：

```
x(i) = H*x(i) + H*y(i)
y(i) = H*y(i) - H*x(i)
```

其中 H 是一个变换矩阵,它的值存放在 param(2)到 param(5)中。详细细节参见 param 讨论部分。

输入参数

n 整数类型。定义向量 x 和 y 的元素数目。

x srotm 函数下为实数类型
drotm 函数下为浮点双精度类型
数组,规模至少为(1 + (n -1)*abs(incx))

incx 整数类型。定义 x 的元素增量。

param srotm 函数下为实数类型
drotm 函数下为双精度浮点类型
数组,规模为 5。
param 数组的元素为：
param(1)包含一个 switch 变量 flag。param(2-5)各自为矩阵 H 的 h11,h21,h12,和 h22 元素。
依据 flag 的值,元素 H 的元素被设置为：

$$flag = -1. : H = \begin{bmatrix} h11 & h12 \\ h21 & h22 \end{bmatrix}$$

$$flag = 0. : H = \begin{bmatrix} 1. & h12 \\ h21 & 1. \end{bmatrix}$$

$$flag = 1. : H = \begin{bmatrix} h11 & 1. \\ -1 & h22 \end{bmatrix}$$

$$flag = -2. : H = \begin{bmatrix} 1. & 0. \\ 0. & 1. \end{bmatrix}$$

对于后三种情况,矩阵元素中的 1.,-1,,以及 0 是根据 flag 的值的得出的,不需要在 param 向量中设置。

输出参数

x　　每个元素被替换为 h11*x + h12*y。

y　　每个元素被替换为 h11*x + h12*y。

3.1.12 ？rotmg

语法

FORTRAN 77:

```
call srotmg(d1, d2, x1, y1, param)
call drotmg(d1, d2, x1, y1, param)
```

描述

给出一个输入向量的笛卡尔坐标(x1,y1),这些函数通过与一个变换矩阵 H 计算使得结果向量的 y 坐标为零。

$$\begin{bmatrix} x \\ 0 \end{bmatrix} = H \begin{bmatrix} \mathrm{sqrt}(d_1)x_1 \\ \mathrm{sqrt}(d_2)y_1 \end{bmatrix}$$

输入参数

d1　　srotmg 函数下为实数
　　drotmg 函数下为双精度浮点类型
　　提供输入向量的 x 坐标的比例因子。

d2　　srotmg 函数下为实数
　　drotmg 函数下为双精度浮点类型
　　提供输入向量的 y 坐标的比例因子。

x1　　srotmg 函数下为实数
　　drotmg 函数下为双精度浮点类型
　　提供输入向量的 x 坐标。

y1　　srotmg 函数下为实数
　　drotmg 函数下为双精度浮点类型
　　提供输入向量的 y 坐标。

输出参数

d1　　srotmg 函数下为实数
　　drotmg 函数下为双精度浮点类型
　　提供更新矩阵的第一个对角线元素。

d2　　srotmg 函数下为实数
　　drotmg 函数下为双精度浮点类型
　　提供更新矩阵的第二个对角线元素。

x1　　srotmg 函数下为实数
　　drotmg 函数下为双精度浮点类型

提供更新矩阵的第一个对角线元素。

param　srotmg 函数下为实数

drotmg 函数下为双精度浮点类型

数组,规模为 5,

param 数组的元素为：

param(1)包含一个 switch 变量 flag。param(2-5)各自为矩阵 H 的 h11,h21,h12, 和 h22　元素。

依据 flag 的值,元素 H 的元素被设置为：

$$flag = -1. : H = \begin{bmatrix} h11 & h12 \\ h21 & h22 \end{bmatrix}$$

$$flag = 0. : H = \begin{bmatrix} 1. & h12 \\ h21 & 1. \end{bmatrix}$$

$$flag = 1. : H = \begin{bmatrix} h11 & 1. \\ -1 & h22 \end{bmatrix}$$

$$flag = -2. : H = \begin{bmatrix} 1. & 0. \\ 0. & 1. \end{bmatrix}$$

对于后三种情况,矩阵元素中的 1.,-1,以及 0 是根据 flag 的值的得出的,不需要在 param 向量中设置。

3.1.13 ？scal

语法

FORTRAN 77:

```
call sscal(n, a, x, incx)
call dscal(n, a, x, incx)
call cscal(n, a, x, incx)
call zscal(n, a, x, incx)
call csscal(n, a, x, incx)
call zdscal(n, a, x, incx)
```

描述

？scal 函数实现下列定义的向量操作：

x = a*x

其中：a 是标量,x 是有 n 个元素的向量。

输入参数

n　整数类型。定义向量 x 的元素数目

a　sscal 和 csscal 函数下是实数类型

dscal 和 zdscal 函数下是双精度浮点类型

cscal 函数下是复数类型

zscal 函数下是双精度复数类型

定义标量 a。

x　　sscal 函数下是实数类型
dscal 函数下是双精度浮点类型
cscal 和 csscal 函数下是复数类型
zscal 和 zdscal 函数下是双精度复数类型
数组,规模至少为(1 + (n -1)*abs(incx))。

incx　整数类型。定义 x 的元素增量。

输出参数

x　　向量 x 的更新

3.1.14 ? swap

语法

FORTRAN 77:

```
call sswap(n, x, incx, y, incy)
call dswap(n, x, incx, y, incy)
call cswap(n, x, incx, y, incy)
call zswap(n, x, incx, y, incy)
```

描述

给出两个向量 x 和 y,? swap 函数返回向量 y 和 x 的交换结果,它们彼此互换。

输入参数

n　　整数类型,定义向量 x 和 y 的元素个数。

x　　sswap 函数下为实数
dswap 函数下为双精度浮点类型
cswap 函数下为复数类型
zswap 函数下为双精度浮点类型
数组,规模至少为(1 + (n-1)*abs(incx))。

incx　整数类型。定义 x 的元素增量。

y　　sswap 函数下为实数
dswap 函数下为双精度浮点类型
cswap 函数下为复数类型
zswap 函数下为双精度浮点类型
数组,规模至少为(1 + (n-1)*abs(incy))。

incy　整数类型。定义 y 的元素增量。

输出参数

x　　结果向量 x 是输入向量 y 的值

y　　结果向量 y 是输入向量 x 的值

3.1.15 i?amax

语法

FORTRAN 77:

```
index = isamax(n, x, incx)
index = idamax(n, x, incx)
index = icamax(n, x, incx)
index = izamax(n, x, incx)
```

描述

给一个向量 x,i?amax 函数返回实数向量 x 中的最大绝对值元素 x(i),或者复数向量中|Re(x(i))|+|Im(x(i))|最大加和结果。

如果 n 是非正数,返回 0。

如果有不止一个最大绝对值元素,返回第一个最大绝对值元素的索引值。

输入参数

n	整数。定义向量 x 的元素数目。
x	isamax 函数下是实数类型 idamax 函数下是双精度类型 icamax 函数下是复数类型 izamax 函数下是双精度复数类型 数组,规模至少是(1+(n-1)*abs(incx))。
incx	整数。定义向量 x 的元素增量。

输出参数

index	整数。向量 x 中最大绝对值元素的位置。

3.1.16 i?amin

语法

FORTRAN 77:

```
index = isamin(n, x, incx)
index = idamin(n, x, incx)
index = icamin(n, x, incx)
index = izamin(n, x, incx)
```

描述

给出一个向量 x,i?amin 函数返回向量 x 中绝对值最小的元素的位置,对于复数向量则返回|Re(x(i))|+|Im(x(i))|的最小值。

如果 n 不是正数,返回 0。

如果有不止一个元素满足绝对值最小,返回第一个绝对值最小的元素的索引值。

输入参数

n	整数。定义向量 x 的元素数目。
x	isamin 函数下是实数类型 idamin 函数下是双精度类型 icamin 函数下是复数类型 izamin 函数下是双精度复数类型 数组,规模至少是(1+(n-1)*abs(incx))。
incx	整数。定义向量 x 的元素增量。

输出参数

index	整数。向量 x 中最小绝对值元素的位置。

3.2 BLAS Level 2 函数说明

表 3-2 BLAS Level 2 函数汇总表

子程序组	数据类型	描述
?gbmv	s,d,c,z	带状矩阵-向量积
?gemv	s,d,c,z	矩阵-向量积
?ger	s,d	秩 1 更新
?gerc	c,z	共轭矩阵秩 1 更新
?geru	c,z	非共轭矩阵秩 1 更新
?hbmv	c,z	Hermitian 带状矩阵-向量积
?hemv	c,z	Hermitian 矩阵-向量积
?her	c,z	Hermitian 矩阵秩 1 更新
?her2	c,z	Hermitian 矩阵秩 2 更新
?hpmv	c,z	Hermitian packed 矩阵-向量积
?hpr	c,z	Hermitian packed 矩阵秩 1 更新
?hpr2	c,z	Hermitian packed 矩阵秩 2 更新
?sbmv	s,d	对称带状矩阵-向量积
?spmv	s,d	对称 packed 带状矩阵-向量积
?spr	s,d	对称 packed 矩阵秩 1 更新
?spr2	s,d	对称 packed 矩阵秩 2 更新
?symv	s,d	对称矩阵-向量积
?syr	s,d	对称矩阵秩 1 更新
?syr2	s,d	对称矩阵秩 2 更新
?tbmv	s,d,c,z	三角带状矩阵-向量积
?tbsv	s,d,c,z	系数矩阵为三角带状矩阵的线性方程组求解
?tpmv	s,d,c,z	三角 packed 矩阵-向量积
?tpsv	s,d,c,z	系数矩阵为三角 packed 矩阵的线性方程组求解
?trmv	s,d,c,z	三角矩阵-向量积
?trsv	s,d,c,z	系数矩阵为三角矩阵的线性方程组求解

3.2.1 ?gbmv

语法

FORTRAN 77：

```
call sgbmv(trans, m, n, kl, ku, alpha, a, lda, x, incx, beta, y, incy)
call dgbmv(trans, m, n, kl, ku, alpha, a, lda, x, incx, beta, y, incy)
call cgbmv(trans, m, n, kl, ku, alpha, a, lda, x, incx, beta, y, incy)
call zgbmv(trans, m, n, kl, ku, alpha, a, lda, x, incx, beta, y, incy)
```

描述

?gbmv 函数用于计算如下的矩阵向量操作

y := *alpha***A***x* + *beta***y*

或

y := *alpha***A*'**x* + *beta***y*,

或

y := *alpha* *conjg(*A*')**x* + *beta***y*,

alpha 和 *beta* 是标量

x 和*y* 是向量

A是m*n的带状矩阵,kl是下三角,ku是上三角

输入参数

trans 一个字符。定义具体操作

当 trans= 'N' 'n', 否则 y := alpha*A*x + beta*y

当 trans= 'T' 或 't', 否则 y := alpha*A'*x + beta*y

当 trans= 'C' 或 'c', 否则 y := alpha *conjg(A')*x + beta*y

m 整数类型。矩阵 A 的行数,至少为 0

n 整数类型。矩阵 A 的列数,至少为 0

alpha sgbmv 中的实型

dgbmv 中的双精度浮点型

cgbmv 中的复数

指定标量 alpha 的值。

kl 整数类型。指定矩阵 A 中上对角线的条数。

kl 的值必须满足 kl>= 0。

ku 整数类型。指定矩阵 A 中上对角线的条数。

ku 的值必须满足 ku>= 0。

a sgbmv 中的实型

dgbmv 中的双精度浮点型

cgbmv 中的复数

zgbmv 中双精度复数

数组,维度为(Lda,n)。

输入前,数组的前(kl+ku+1)*n 部分必须包含矩阵元素。矩阵按列如下存储：主对角线存在数组第(ku+1)行,第一条上对角线从第 ku 行的第 2 个位置开始存储,第一个下对角线从第 ku+2 行的第 1 个位置开始存储 等等。数组 a 中和带状矩阵不一致的数(例如左上 ku*ku 的三角形)不会被引用。

下面的代码段将一个传统满矩阵存储转换为带状存储。

```
    do 20, j = 1, n
           k = ku + 1 - j
           do 10, i = max(1, j-ku), min(m, j+kl)
              a(k+i, j) = matrix(i,j)
           10 continue
      20 continue
```

lda 整数类型。指定调用程序中定义的数组 a 的第一维维度。lda 至少为 kl+ku+1。

x sgbmv 中的实型
dgbmv 中的双精度浮点型
cgbmv 中的复数
zgbmv 中双精度复数
数组,当 trans = 'N' 或 'n', 维度至少为(1 + (n - 1)*abs(incx));
否则,至少为(1 + (m -1)*abs(incx))。
输入前数组x必须包含向量x

incx 整数类型。指定x的增量。incx必须不为0。

beta sgbmv 中的实型
dgbmv 中的双精度浮点型
cgbmv 中的复数
zgbmv 中双精度复数
指定标量beta
当beta等于0时,y不需要在输入时设置。

y sgbmv 中的实型
dgbmv 中的双精度浮点型
cgbmv 中的复数
zgbmv 中双精度复数
数组,当 trans = 'N' 或 'n', 维度至少为(1 + (m - 1)*abs(incx));
否则,至少为(1 + (n -1)*abs(incx))。
输入前数组y必须包含向量y。

incy 整数类型。指定y的增量。incy必须不为0。

输出参数

y 更新后向量 y

3.2.2 ?gemv

语法

FORTRAN 77:

```
call sgemv(trans, m, n, alpha, a, lda, x, incx, beta, y, incy)
call dgemv(trans, m, n, alpha, a, lda, x, incx, beta, y, incy)
call cgemv(trans, m, n, alpha, a, lda, x, incx, beta, y, incy)
call zgemv(trans, m, n, alpha, a, lda, x, incx, beta, y, incy)
```

描述

?gemv 函数计算如下矩阵向量操作

y := *alpha***A***x* + *beta***y*,

或

y := *alpha***A*'**x* + *beta***y*,

或

y := *alpha**conjg(*A*')**x* + *beta***y*,

其中

Alpha 和 beta 是标量

X 和 y 是向量

A 是 m*n 的矩阵

输入参数

trans	一个字符. 指定具体操作 当 trans= 'N' 或 'n', 否则 y := alpha*A*x + beta*y; 当 trans= 'T' 或 't', 否则 y := alpha*A'*x + beta*y; 当 trans= 'C' 或 'c', 否则 y := alpha *conjg(A')*x + beta*y.
m	整数类型。指定数组 A 行数。m 至少为 0。
n	整数类型。指定数组 A 的列数。m 至少为 0。
alpha	sgemv 中的实数 dgemv 中的双精度浮点数 cgemv 中的复数 zgemv 中的双精度复数 指定标量 alpha
a	sgemv 中的实型 dgemv 中的双精度浮点型 cgemv 中的复数 zgemv 中双精度复数 数组,维度为(Lda,n)。 输入前，数组的前 m*n 个元素必须包括矩阵元素。
lda	整数类型。
x	sgemv 中的实数 dgemv 中的双精度浮点数 cgemv 中的复数 zgemv 中的双精度复数 数组,当 trans = 'N' 或 'n', 维度至少为(1 + (m - 1)*abs(incx)); 否则,至少为(1 + (n-1)*abs(incx))。输入前数组 x 必须包含向量 x
incx	整数类型。指定x的增量。incx必须不为0。
beta	sgemv 中的实数 dgemv 中的双精度浮点数 cgemv 中的复数 zgemv 中的双精度复数 指定标量 beta 当beta等于0时,y不需要在输入时设置。
y	sgemv 中的实数 dgemv 中的双精度浮点数 cgemv 中的复数 zgemv 中的双精度复数 数组,当 trans = 'N' 或 'n', 维度至少为(1 + (m - 1)*abs(incx)); 否则,至少为(1 + (n -1)*abs(incx))。输入前数组 y 必须包含向量 y。

incy	整数类型。指定y的增量。incy必须不为0。

输出参数

y	更新后向量 y

3.2.3 ?ger

语法

FORTRAN 77：

```
call sger(m, n, alpha, x, incx, y, incy, a, lda)
call dger(m, n, alpha, x, incx, y, incy, a, lda
```

描述

?ger 计算如下的矩阵向量操作

A := alpha*x*y'+ A,

其中

Alpha 是标量

x 是 m 个数的向量

y 是 n 个数的向量

A 是 m*n 的一般矩阵

输入参数

m	整数类型。指定矩阵 A 的行数。m 的值必须至少为 0。
n	整数类型。指定矩阵 A 的列数。n 的值必须至少为 0。
alpha	sger 中的实数 dger 中的双精度浮点数 指定标量 alpha
x	数组,维度至少为(1 + (m - 1)*abs(incx))。输入前,增量数组 x 必须包含 m 个数的向量 x
incx	整数类型。指定 x 中元素的增量。incx 的值必须不为 0。
y	sger 中的实数 dger 中的双精度浮点数 数组,维度至少为(1 + (n - 1)*abs(incx))。输入前,增量数组 y 必须包含 m 个数的向量 y
incy	整数类型。指定 y 中元素的增量。incy 的值必须不为 0。
a	sger 中的实数 dger 中的双精度浮点数 数组,维度(lda,n) 输入前,数组 a 的前 m*n 部分必须包含系数矩阵。
lda	整数类型。

输出参数

a	由更新后的矩阵覆盖

3.2.4 ?gerc

语法

FORTRAN 77：

```
call cgerc(m, n, alpha, x, incx, y, incy, a, lda)
call zgerc(m, n, alpha, x, incx, y, incy, a, lda
```

描述

?gerc 函数计算如下矩阵向量操作

A := alpha*x*conjg(y') + A,

其中

alpha 是标量

x 是 m 个数的向量

y 是 n 个数的向量

A 是 m*n 的矩阵

输入参数

m	整数类型。指定矩阵 A 的行数。m 的值至少为 0
n	整数类型。指定矩阵 A 的列数。n 的值至少为 0。
alpha	cgerc 中的复数 zgerc 中的双精度复数 指定标量 alpha
x	cgerc 中的复数 zgerc 中的双精度复数 数组,维度至少为(1 + (m - 1)*abs(incx))。输入前,增量数组 x 必须包含 m 个数的向量 x。
incx	整数类型。指定 x 中数的增量。incx 的值必须不为 0
y	cgerc 中的复数 zgerc 中的双精度复数 数组,维度至少为(1 + (n - 1)*abs(incy))。输入前,增量数组 y 必须包含 n 个数的向量 y。
incy	整数类型。指定 y 中数的增量。incy 的值必须不为 0。
a	cgerc 中的复数 zgerc 中的双精度复数 数组,维度为(lda,n)。 输入前,数组 a 的前 m*n 部分必须包含系数矩阵。
lda	整数类型。指定调用函数中定义的 a 的第一维维度。lda 的值必须至少为 max(1, *m*)。

输出参数

a	由更新后矩阵覆盖

3.2.5 ?geru

语法

FORTRAN 77：

call cgeru(m, n, alpha, x, incx, y, incy, a, lda)

call zgeru(m, n, alpha, x, incx, y, incy, a, lda)

描述

?geru 函数计算如下矩阵向量

A := alpha*x*y ' + A,

其中

alpha 是标量

x 是 m 个数的向量

y 是 n 个数的向量

A 是 m*n 的矩阵

输入参数

m	整数类型。指定矩阵 A 的行数。m 的值至少为 0。
n	整数类型。指定矩阵 A 的列数。n 的值至少为 0。
alpha	cgeru 中的复数 zgeru 中的双精度复数 指定标量 alpha
x	cgeru 中的复数 zgeru 中的双精度复数 数组,维度至少为(1 + (m - 1)*abs(incx))。 输入前,增加后数组 x 必须包含 m 个数的向量 x。
incx	整数类型。指定 x 中数的增量。incx 的值必须不为 0。
y	cgeru 中的复数 zgeru 中的双精度复数 数组,维度至少为(1 + (n - 1)*abs(incy))。 输入前,增加后数组 y 必须包含 n 个数的向量 y。
incy	整数类型。指定 y 中数的增量。incy 的值必须不为 0。
a	cgeru 中的复数 zgeru 中的双精度复数 数组,维度(Lda,n)。 输入前,数组 a 的前 m*n 部分必须包含矩阵元素
lda	整数类型。指定调用函数中定义的 a 的第一维维度。Lda 的值必须至少为 max(1, m)。

输出参数

a	由更新后矩阵覆盖。

3.2.6 ?hbmv

语法

FORTRAN 77：

call chbmv(uplo, n, k, alpha, a, lda, x, incx, beta, y, incy)

call zhbmv(uplo, n, k, alpha, a, lda, x, incx, beta, y, incy)

描述

?hbmv 函数计算如下矩阵向量操作

*y := alpha*A*x + beta*y*,

其中

alpha 和 beta 是标量

x 和 y 是 n 个数的向量

A 是一个 n*n 的 Hermitian 带状矩阵,有 k 个上对角线。

输入参数

uplo

一个字符。指定使用的是 Hermitian 带状矩阵上三角部分还是下三角部分：

如果 uplo = 'U'或'u',则使用的是上三角部分；

如果 uplo = 'L'或'l',则使用的是下三角部分。

n

整数类型。指定矩阵 A 的顺序。n 的值必须至少为 0。

k

整数类型。指定矩阵 A 的 super-diagonals 数。

k 的值必须满足 k>=0。

alpha

chbmv 中的复数

zhbmv 中的双精度复数

指定标量 alpha。

a

chbmv 中复数

zhbmv 中的双精度复数

数组,维度为(Lda,n)。

输入前,uplo = 'U' 或 'u’数组的前(k +1)*n 部分必须包含 Hermitian 矩阵上三角带状部分。矩阵按列存储：主对角线存在数组第(k+1)行,第一条上对角线从第 ku 行的第 2 个位置开始存储,第一个下对角线从第 k+2 行的第 1 个位置开始存储 等等。左上 k*k 的三角形不会被引用。

下面的代码段将一个传统满矩阵存储转换为带状存储。

```
do 20, j = 1, n
    k = k + 1 - j
        do 10, i = max(1, j-k), min(m, j+k)
        a(k+i, j) = matrix(i,j)
    10 continue
20 continue
```

输入前,uplo = 'L' 或 'l’数组的前(k +1)*n 部分必须包含 Hermitian 矩阵下三角带状部分。矩阵按列存储：主对角线存在数组第(k+1)行,第一条下对角线从第 2 行的第 1 个位置开始存储, 等等。右下 k*k 的三角形不会被引用。

下面的代码段将一个传统满矩阵存储转换为带状存储。

```
do 20, j = 1, n
    k = k + 1 - j
        do 10, i = max(1, j-k), min(m, j+k)
        a(k+i, j) = matrix(i,j)
```

```
                    10 continue
                20 continue
```

对角线上元素的虚数部分不需要设置且默认为 0

lda 整数类型。

指定调用函数中定义的 a 的第一维维度。lda 的值必须至少为(k+1)

x chbmv 中的复数

zhbmv 中的双精度复数

数组,维度至少为(1 + (*n* - 1)*abs(*incx*))。

输入前,增加后数组 x 必须包含向量 x

incx 整数类型。指定 x 中数的增量。incx 的值必须不为 0。

beta chbmv 中的复数

zhbmv 中的双精度复数

指定标量 beta。

y chbmv 中的复数

zhbmv 中的双精度复数

数组,维度至少为(1 + (*m* - 1)*abs(*incx*))。

输入前,增加后数组 x 必须包含向量 y。

incy 整数类型。指定 y 中数的增量。incy 的值必须不为 0。

输出参数

y 由更新后向量 y 覆盖。

3.2.7 ?hemv

语法

FORTRAN 77：

```
call chemv(uplo, n, alpha, a, lda, x, incx, beta, y, incy)
call zhemv(uplo, n, alpha, a, lda, x, incx, beta, y, incy)
```

描述

?hemv 函数计算如下矩阵向量运算

*y := alpha*A*x + beta*y*,

其中

alpha 和 beta 是标量

x 和 y 是 n 个元素的向量

A 是 n*n 的 Hermitian 矩阵

输入参数

uplo 一个字符。指定使用的是 Hermitian 带状矩阵上三角部分还是下三角部分：

如果 *uplo* = 'U'或'u',则使用的是上三角部分；

如果 *uplo* = 'L'或'l',则使用的是下三角部分。

n 整数类型。指定矩阵 A 的顺序。n 的值必须至少为 0。

alpha	chemv 中的复数 zhemv 中的双精度复数 指定 alpha 标量
a	chemv 中的复数 zhemv 中的双精度复数 数组,维度为(Lda,n)。 输入 *uplo* = 'U'或'u'前,数组 a 的前 n*n 的上三角部分必须包含 Hermitian 矩阵的上三角部分且 a 的下三角部分不会被引用；输入 uplo = 'L'或'l'前,数组 a 的前 n*n 的下三角部分必须包含 Hermitian 矩阵的 下三角部分且 a 的下三角部分不会被引用 对角线上的数的虚数部分不需要设置且默认为 0。
lda	数组矩阵。 指定调用函数中定义的 a 的第一维的维度。lda 的值必须至少为 max(1,n)。
x	chemv 中的复数 zhemv 中的双精度复数 数组,维度至少为(1 + (*n* - 1)*abs(*incx*))。输入前,增值数 x 必须包含 n 个数的向量 x。
incx	整数类型。指定 x 中数的增量。incx 的值必须不为 0。
beta	chemv 中的复数 zhemv 中的双精度复数 指定标量 beta。如果 beta 被指定为 0,不需设定 y 的输入值。
y	chemv 中的复数 zhemv 中的双精度复数 数组,维度至少为(1 + (*n* - 1)*abs(*incy*))。输入前,增加后数组 y 必须包含 n 个数的向量 y。
incy	整数类型。指定 y 中数的增量。incy 的值必须不为 0。

输出参数

y	由更新后的向量 y 覆盖。

3.2.8 ?her

语法

FORTRAN 77：

```
call cher(uplo, n, alpha, x, incx, a, lda)
call zher(uplo, n, alpha, x, incx, a, lda)
```

描述

?her 函数函数计算如下矩阵向量操作

A := *alpha***x**conjg(*x*') + *A*,

其中：

alpha 是实数标量

x 是 n 个数的向量

A 是 n*n 的 Hermitian 矩阵

输入参数

uplo	一个字符。指定使用的是 Hermitian 带状矩阵上三角部分还是下三角部分：如果 *uplo* = 'U'或'u',则使用的是上三角部分；如果 *uplo* = 'L'或'l',则使用的是下三角部分。
n	整数类型。指定矩阵 A 的顺序。n 的值必须至少为 0。
alpha	cher 中的复数 zher 中的双精度复数 指定 alpha 标量
x	cher 中的复数 zher 中的双精度复数 数组,维度至少为(1 + (*n* – 1)*abs(*incx*))。输入前,增值数 x 必须包含 n 个数的向量 x。
incx	整数类型。指定 x 中数的增量。incx 的值必须不为 0。
a	cher 中的复数 zher 中的双精度复数 数组,维度为(Lda,n)。 输入 uplo = 'U'或'u'前,数组 a 的前 n*n 的上三角部分必须包含 Hermitian 矩阵的上三角部分且 a 的下三角部分不会被引用；输入 uplo = 'L'或'l'前,数组 a 的前 n*n 的下三角部分必须包含 Hermitian 矩阵的 下三角部分且 a 的下三角部分不会被引用 对角线上的数的虚数部分不需要设置且默认为 0。
lda	整数类型。指定调用函数中定义的 a 的第一维的维度。lda 的值必须至少为 max(1,n)。

输出参数

a	当 *uplo* = 'U'或'u',数组 a 的上三角部分被更新的矩阵的上三角部分覆盖。 当 uplo = 'L' 或 'l',数组 a 的下三角部分被更新的矩阵的下三角部分覆盖。

3.2.9 ?her2

语法

FORTRAN 77：

```
call cher2(uplo, n, alpha, x, incx, y, incy, a, lda)
call zher2(uplo, n, alpha, x, incx, y, incy, a, lda)
```

描述

?her2 函数计算如下矩阵向量操作

A := *alpha* *x*conjg(*y'*) + conjg(*alpha*)**y* *conjg(*x'*) + *A*,

其中

alpha 是标量

x 和 y 是 n 个数的向量

A 是 n*n 的 Hermitian 矩阵

输入参数

uplo	一个字符。指定使用的是 Hermitian 带状矩阵上三角部分还是下三角部分：如果 *uplo* = 'U'或'u',则使用的是上三角部分；如果 *uplo* = 'L'或'l',则使用的是下三角部分。
n	整数类型。指定矩阵 A 的顺序。n 的值必须至少为 0。
alpha	cher2 中的复数 zher2 中的双精度复数 指定 alpha 标量
x	cher2 中的复数 zher2 中的双精度复数 数组,维度至少为(1 + (*n* - 1)*abs(*incx*))。输入前,增值数 x 必须包含 n 个数的向量 x。
incx	整数类型。指定 x 中数的增量。incx 的值必须不为 0。
y	cher2 中的复数 zher2 中的双精度复数 数组,维度至少为(1 + (*n* - 1)*abs(*incy*))。输入前,增值数 y 必须包含 n 个数的向量 x。
incy	整数类型。指定 y 中数的增量。incy 的值必须不为 0。
a	cher2 中的复数 zher2 中的双精度复数 输入 uplo = 'U'或'u'前,数组 a 的前 n*n 的上三角部分必须包含 Hermitian 矩阵的上三角部分且 a 的下三角部分不会被引用；输入 uplo = 'L'或'l'前,数组 a 的前 n*n 的下三角部分必须包含 Hermitian 矩阵的 下三角部分且 a 的下三角部分不会被引用 对角线上的数的虚数部分不需要设置且默认为 0。
lda	整数类型。指定调用函数中定义的 a 的第一维的维度。lda 的值必须至少为 max(1,n)。

输出

a	当 *uplo* = 'U'或'u',数组 a 的上三角部分被更新的矩阵的上三角部分覆盖。 当 uplo = 'L' 或 'l',数组 a 的下三角部分被更新的矩阵的下三角部分覆盖。 对角线上的数的虚数部分设置为 0。

3.2.10 ?hpmv

语法

FORTRAN 77：

```
call chpmv(uplo, n, alpha, ap, x, incx, beta, y, incy)
call zhpmv(uplo, n, alpha, ap, x, incx, beta, y, incy)
```

描述

?hpmv 函数计算如下矩阵向量操作

*y := alpha*A*x + beta*y*,

其中

alpha 和 beta 是标量,

x 和 y 是 n 个数的向量,

A 是 n*n 的 Hermitian 矩阵,压缩格式提供。

输入

uplo	一个字符。指定矩阵 A 的上三角部分还是下三角部分由压缩数组 ap 提供。如果 uplo = 'U'或'u',则的上三角部分由压缩数组 ap 提供。 如果 *uplo* = 'L'或'l',则矩阵 A 的下三角部分由压缩数组 ap 提供。
n	整数类型。指定矩阵 A 的顺序。n 的值必须至少为 0。
alpha	chpmv 中的复数 zhpmv 中的双精度复数 指定 alpha 标量
ap	chpmv中的复数 zhpmv 中双精度复数 数组,维度至少为((*n**(*n* + 1))/2)。输入*uplo* = 'U' 或'u'前,数组ap必须包含按列顺序压缩的Hermitian矩阵的上三角部分,ap(1)包含a(1,1),ap(2)和ap(3)分别包含a(1,2)和a(2,2)等等。输入*uplo* = 'L' 或者'l', 数组ap必须包含按列顺序压缩的Hermitian矩阵的下三角部分,ap(1)包含a(1,1),ap(2)和ap(3)分别包含a(1,2)和a(2,2)等等。 对角线上数的虚数部分不需要设置并默认为 0。
x	chpmv 中的复数 zhpmv 中的双精度复数 数组,维度至少为(1 +(*n* - 1)*abs(*incx*))。输入前,增加后数组 x 必须包含 n 个数的向量 x。
incx	整数类型。指定 x 中值的增量。incx 必须不为 0。
beta	chpmv 中的复数 zhpmv 中的双精度复数 指定标量 beta。 如果 beta 为 0 则 y 不需要在输入时设置。
y	chpmv 中的复数 zhpmv 中的双精度复数 数组,维度至少为(1 + (*n* - 1)*abs(*incy*))。 输入前,增加后数组 y 必须包含 n 个数的数组 y。
incy	数类型。指定 x 中值的增量。incy 必须不为 0。

输出参数

y	由更新后向量 y 覆盖。

3.2.11 ?hpr

语法

FORTRAN 77：

```
call chpr(uplo, n, alpha, x, incx, ap)
call zhpr(uplo, n, alpha, x, incx, ap)
```

描述

?hpr 函数计算如下矩阵向量操作

A := *alpha***x**conjg(*x*') + *A*,

其中

alpha 是实数向量

x 是 n 个数的向量

A 是 n*n 的 Hermitian 矩阵,由压缩形式提供

输入参数

uplo	一个字符。指定矩阵 A 的上三角部分还是下三角部分由压缩数组 a 提供。 如果 uplo = 'U'或'u',则的上三角部分由压缩数组 ap 提供。 如果 uplo = 'L'或'l',则矩阵 A 的下三角部分由压缩数组 ap 提供。
n	整数类型。指定矩阵 A 的顺序。n 的值必须至少为 0。
alpha	chpr 中的复数 zhpr 中的双精度复数 指定 alpha 标量
x	chpr 中的复数 zhpr 中的双精度复数 数组,维度至少为(1 + (*n* - 1)*abs(*incx*))。 输入前,增加后数组 x 必须包含 n 个数的向量 x。
incx	整数类型。指定 x 中数的增量。incx 必须不为 0。
ap	chpr 中的复数 zhpr 中的双精度复数 数组,维度至少为((*n**(*n* + 1))/2)。输入*uplo* = 'U' 或'u'前,数组ap必须包含按列顺序压缩的Hermitian矩阵的上三角部分,ap(1)包含a(1,1),ap(2)和ap(3)分别包含a(1,2)和a(2,2)等等。输入*uplo* = 'L' 或者'l', 数组ap必须包含按列顺序压缩的Hermitian矩阵的下三角部分,ap(1)包含a(1,1),ap(2)和ap(3)分别包含a(1,2)和a(2,2)等等。 对角线上数的虚数部分不需要设置并默认为 0。

输出参数

ap	当 *uplo* = 'U'或'u',数组 a 的上三角部分被更新的矩阵的上三角部分覆盖。 当 uplo = 'L' 或 'l',数组 a 的下三角部分被更新的矩阵的下三

角部分覆盖。
对角线上的数的虚数部分设置为 0。

3.2.12 ?hpr2

语法

FORTRAN 77：

```
call chpr2(uplo, n, alpha, x, incx, y, incy, ap)
call zhpr2(uplo, n, alpha, x, incx, y, incy, ap)
```

描述

?hpr2 函数计算如下矩阵向量操作

A := *alpha***x**conjg(*y*') + conjg(*alpha*)**y**conjg(*x*') + *A*,

其中

alpha 是个标量

x 和 y 是 n 个数向量

A 是 n*n 的 Hermitian 矩阵,由压缩形式提供。

输入参数

uplo
一个字符。指定矩阵 A 的上三角部分还是下三角部分由压缩数组 ap 提供。
如果 uplo = 'U'或'u',则的上三角部分由压缩数组 ap 提供。
如果 uplo = 'L'或'l',则矩阵 A 的下三角部分由压缩数组 ap 提供。

n
整数类型。指定矩阵 A 的顺序。n 的值必须至少为 0。

alpha
chpr2 中的复数
zhpr2 中的双精度复数
指定 alpha 标量

x
chpr2 中的复数
zhpr2 中的双精度复数
数组,维度至少为(1 + (*n* - 1)*abs(*incx*))。
输入前,增加后数组 x 必须包含 n 个数的向量 x。

incx
整数类型。指定 x 中数的增量。incx 必须不为 0。

y
chpr2 中的复数
zhpr2 中的双精度复数
数组,维度至少为(1 + (*n* - 1)*abs(*incy*))。
输入前,增加后数组 y 必须包含 n 个数的数组 y。

incy
整数类型。指定 y 中数的增量。incy 必须不为 0。

ap
chpr 中的复数
zhpr 中的双精度复数
数组,维度至少为((*n**(*n* + 1))/2)。输入*uplo* = 'U' 或'u'前,数组ap必须包含按列顺序压缩的Hermitian矩阵的上三角部分,ap(1)包含a(1,1),ap(2)和ap(3)分别包含a(1,2)和

a(2,2)等等。输入*uplo* = 'L' 或者'l', 数组ap必须包含按列顺序压缩的Hermitian矩阵的下三角部分,ap(1)包含a(1,1),ap(2)和ap(3)分别包含a(1,2)和a(2,2)等等。
对角线上数的虚数部分不需要设置并默认为 0。

输出参数

ap　当 *uplo* = 'U'或'u',数组 a 的上三角部分被更新的矩阵的上三角部分覆盖。
当 uplo = 'L' 或 'l',数组 a 的下三角部分被更新的矩阵的下三角部分覆盖。
对角线上的数的虚数部分设置为 0。

3.2.13 ?sbmv

语法

FORTRAN 77：

```
call ssbmv(uplo, n, k, alpha, a, lda, x, incx, beta, y, incy)
call dsbmv(uplo, n, k, alpha, a, lda, x, incx, beta, y, incy)
```

描述

?sbmv 函数计算如下矩阵向量操作

*y := alpha*A*x + beta*y*,

其中

alpha 和 beta 是标量

x 和 y 是 n 个数的向量

A 是 n*n 的对称带状矩阵,k 条上对角线

输入参数

uplo　一个字符。指定使用的是带状矩阵 A 的上三角还是下三角部分
如果*uplo* = 'U' 或 'u' – 上三角部分
如果 *uplo* = 'L' 或 'l' – 下三角部分

n　整数类型。指定矩阵 A 的顺序。n 的值必须至少为 0。

k　整数类型。指定矩阵A中的上对角线的个数。
K的值必须满足k>=0.

alpha　ssbmv 中的实数
dsbmv 中的双精度浮点数
指定 alpha 标量。

a　ssbmv 中的实数
dsbmv 中的双精度浮点数
数组,维度为(Lda,n)。
输入前,uplo = 'U' 或 'u'数组的前(k +1)*n 部分必须包含对称矩阵上三角带状部分。矩阵按列存储：主对角线存在数组第(k+1)行,第一条上对角线从第 ku 行的第 2 个位置开始存储,第一个下对角线从第 k+2 行的第 1 个位置开始存储 等等。

左上 k*k 的三角形不会被引用。

下面的代码段将一个传统满矩阵存储转换为带状存储。

```
    do 20, j = 1, n
        k = k + 1 - j
            do 10, i = max(1, j-k), min(m, j+k)
            a(k+i, j) = matrix(i,j)
        10 continue
20 continue
```

输入前,uplo = 'L' 或 'l'数组的前(k +1)*n 部分必须包含对称矩阵下三角带状部分。矩阵按列存储：主对角线存在数组第(k+1)行,第一条下对角线从第 2 行的第 1 个位置开始存储,等等。右下 k*k 的三角形不会被引用。

下面的代码段将一个传统满矩阵存储转换为带状存储。

```
    do 20, j = 1, n
        k = k + 1 - j
            do 10, i = max(1, j-k), min(m, j+k)
            a(k+i, j) = matrix(i,j)
        10 continue
20 continue
```

lda 整数类型。

指定调用函数中定义的a的第一维维度。lda的值必须至少为(k+1)。

x ssbmv中的实数

dsbmv中的双精度浮点数

数组,维度至少为(1 + (*n* - 1)*abs(*incx*))。

输入前,增加后数组 x 必须包含 n 个数的向量 x。

incx 整数类型。指定 x 中数的增量。incx 必须不为 0。

beta ssbmv 中的实数

dsbmv 中的双精度浮点数

指定 beta 标量。

y ssbmv 中的实数

dsbmv 中的双精度浮点数

数组,维度至少为(1 + (*n* - 1)*abs(*incy*))。

输入前,增加后数组 y 必须包含向量 y。

incy 整数类型。指 y 中数的增量。incy 必须不为 0。

输出参数

y 由更新后向量 y 覆盖。

3.2.14 ?spmv

语法

FORTRAN 77：

```
call sspmv(uplo, n, alpha, ap, x, incx, beta, y, incy)
call dspmv(uplo, n, alpha, ap, x, incx, beta, y, incy)
```

描述

?spmv 函数计算如下矩阵向量操作

*y := alpha*A*x + beta*y*,

其中

alpha 和 beta 是标量,

x 和 y 是 n 个数向量,

A 是 n*n 对称矩阵,由压缩形式提供

输入参数

uplo	一个字符。指定矩阵 A 的上三角部分还是下三角部分由压缩数组 ap 提供。如果 *uplo = 'U*'或'*u*',则的上三角部分由压缩数组 ap 提供。 如果 uplo = 'L'或'*l*',则矩阵 A 的下三角部分由压缩数组 ap 提供。
n	整数类型。指定矩阵 A 的顺序。n 的值必须至少为 0。
alpha	sspmv 中的实数 dspmv 中的双精度复数 指定 alpha 标量。
ap	sspmv 中的复数 dspmv 中的双精度浮点数 数组,维度至少为((*n**(*n* + 1))/2)。输入*uplo* = 'U' 或'u'前,数组ap必须包含按列顺序压缩的Hermitian矩阵的上三角部分,ap(1)包含a(1,1),ap(2)和ap(3)分别包含a(1,2)和a(2,2)等等。输入*uplo* = 'L' 或者'l', 数组ap必须包含按列顺序压缩的Hermitian矩阵的下三角部分,ap(1)包含a(1,1),ap(2)和ap(3)分别包含a(1,2)和a(2,2)等等。
x	sspmv 中的实数 dspmv 中的双精度浮点数 数组,维度至少为(1 + (*n* - 1)*abs(*incx*))。 输入前,增加后数组 x 必须包含 n 个数的向量 x。
incx	整数类型。指定 x 中数的增量。incx 必须不为 0。
beta	sspmv 中的实数 dspmv 中的双精度浮点数 指定 beta 标量。 当 beta 被赋值为 0,则 y 不需输入时设置。
y	sspmv 中的实数 dspmv 中的双精度浮点数 数组,维数至少为(1 + (*n* - 1)*abs(*incy*))。 输入前,增加后数组 y 必须包含 n 个数的向量 y。
incy	整数类型。指定 y 中值的增量。Incy 必须不为 0。

输出参数

y	由更新后向量 y 覆盖。

3.2.15 ?spr

语法

FORTRAN 77：

```
call sspr(uplo, n, alpha, x, incx, ap)
call dspr(uplo, n, alpha, x, incx, ap)
```

描述

?spr 函数计算如下矩阵向量操作

a:= *alpha***x***x*'+ *A*,

其中

alpha 是实数标量

x 是 n 个数的向量

输入参数

uplo	一个字符。指定矩阵 A 的上三角部分还是下三角部分由压缩数组 ap 提供。如果 uplo = 'U'或'u',则的上三角部分由压缩数组 ap 提供如果 *uplo* = 'L'或'l',则矩阵 A 的下三角部分由压缩数组 ap 提供。
n	整数类型。指定矩阵 A 的顺序。n 的值必须至少为 0。
alpha	sspr 中的实数 dspr 中的双精度浮点数 指定 alpha 标量。
x	sspr 中的实数 dspr 中的双精度浮点数 数组,维度至少为(1 + (*n* – 1)*abs(*incx*))。 输入前,增加后数组 x 必须包含 n 个数的向量 x。
incx	整数类型。指定 x 中数的增量。incx 必须不为 0。
ap	cspr 中的复数 zspr 中的双精度浮点数 数组,维度至少为((*n**(*n* + 1))/2)。输入*uplo* = 'U' 或'u'前,数组ap必须包含按列顺序压缩的Hermitian矩阵的上三角部分,ap(1)包含a(1,1),ap(2)和ap(3)分别包含a(1,2)和a(2,2)等等。输入*uplo* = 'L' 或者'l', 数组ap必须包含按列顺序压缩的Hermitian矩阵的下三角部分,ap(1)包含a(1,1),ap(2)和ap(3)分别包含a(1,2)和a(2,2)等等。 对角线上数的虚数部分不需要设置并默认为 0。

输出参数

ap	当 *uplo* = 'U'或'u',数组 a 的上三角部分被更新的矩阵的上三角部分覆盖。 当 uplo = 'L' 或 'l',数组 a 的下三角部分被更新的矩阵的下三角部分覆盖。

3.2.16 ?spr2

语法

FORTRAN 77：

```
call sspr2(uplo, n, alpha, x, incx, y, incy, ap)
call dspr2(uplo, n, alpha, x, incx, y, incy, ap)
```

描述

?spr2 计算如下矩阵向量操作

A:= *alpha***x***y*'+ *alpha***y***x*' + *A*,

其中

alpha 是标量

x 和 y 是 n 个数的向量

A 是 n*n 的对称矩阵,由压缩格式提供。

输入参数

uplo	一个字符。指定矩阵 A 的上三角部分还是下三角部分由压缩数组 ap 提供。如果 uplo = 'U'或'u',则的上三角部分由压缩数组 ap 提供如果 uplo = 'L'或'l',则矩阵 A 的下三角部分由压缩数组 ap 提供。
n	整数类型。指定矩阵 A 的顺序。n 的值必须至少为 0。
alpha	sspr2 中的实数 dspr2 中的双精度复数 指定 alpha 标量。
x	sspr2 中的实数 dspr2 中的双精度复数 数组,维度至少为(1 + (*n* - 1)*abs(*incx*))。 输入前,增加后数组 x 必须包含 n 个数的向量 x。
incx	整数类型。指定 x 中数的增量。incx 必须不为 0。
y	sspr2 中的实数 dspr2 中的双精度复数 数组,维度至少为(1 + (*n* - 1)*abs(*incy*))。 输入前,增加后数组 x 必须包含 n 个数的向量 y。
incy	整数类型。指定 y 中数的增量。incy 必须不为 0。
ap	cspr2 中的复数 zspr2 中的双精度复数 数组,维度至少为((*n**(*n* + 1))/2)。输入*uplo* = 'U' 或'u'前,数组ap必须包含按列顺序压缩的Hermitian矩阵的上三角部分,ap(1)包含a(1,1),ap(2)和ap(3)分别包含a(1,2)和a(2,2)等等。输入*uplo* = 'L' 或者'l', 数组ap必须包含按列顺序压缩的Hermitian矩阵的下三角部分,ap(1)包含a(1,1),ap(2)和ap(3)分别包含a(1,2)和a(2,2)等等。 对角线上数的虚数部分不需要设置并默认为 0。

输出参数

ap	当 *uplo* ='U'或'u',数组 a 的上三角部分被更新的矩阵的

上三角部分覆盖。
当 *uplo* = 'L'或'l',数组 a 的下三角部分被更新的矩阵的下三角部分覆盖。

3.2.17 ?symv

语法

FORTRAN 77：

```
call ssymv(uplo, n, alpha, a, lda, x, incx, beta, y, incy)
call dsymv(uplo, n, alpha, a, lda, x, incx, beta, y, incy)
```

描述

?symv 函数计算如下矩阵向量操作
y := *alpha*A*x* + *beta*y*,
其中 alpha 和 beta 是标量
X 和 y 是 n 个数的向量
A 是 n*n 的对称矩阵。

输入参数

uplo	一个字符。指定矩阵 A 的上三角部分还是下三角部分由压缩数组 ap 提供。如果 *uplo* = 'U'或'u',则的上三角部分由压缩数组 ap 提供如果 *uplo* = 'L'或'l',则矩阵 A 的下三角部分由压缩数组 ap 提供。
n	整数类型。指定矩阵 A 的顺序。n 的值必须至少为 0。
alpha	ssymv 中的实数 dsymv 中的双精度浮点数 指定 alpha 标量。
a	ssymv 中的复数 dsymv 中的双精度浮点数 数组,维度为(Lda,n)。 输入 *uplo* = 'U'或'u'前,数组 a 的前 n*n 的上三角部分必须包含 Hermitian 矩阵的上三角部分且 a 的下三角部分不会被引用；输入 *uplo* = 'L'或'l'前,数组 a 的前 n*n 的下三角部分必须包含 Hermitian 矩阵的 下三角部分且 a 的下三角部分不会被引用
lda	整数类型。
x	ssymv 中的复数 dsymv 中的双精度浮点数 数组,维度至少为(1 + (*n* - 1)*abs(*incx*))。输入前,增量 数组 x 必须包含 n 个数的向量 x。
incx	整数类型。指定 x 中元素的增量。incx 的值必须不为 0。
beta	ssymv 中的实数 dsymv 中的双精度浮点数 指定标量 beta。 当 beta 被赋值为 0 时,y 不需要在输入时设置。
y	ssymv 中的实数

	dsymv 中的双精度浮点数 数组,维度至少为(1 + (*n* - 1)*abs(*incy*))。 输入前,增加后数组 y 必须包含 n 个数的向量 y。
incy	整数类型。指定 y 中数的增量 incy 的值必须不为 0。

输出参数

y	由更新后向量 y 覆盖。

3.2.18 ?syr

语法

FORTRAN 77：

```
call ssyr(uplo, n, alpha, x, incx, a, lda)
call dsyr(uplo, n, alpha, x, incx, a, lda)
```

描述

?syr 函数指定如下矩阵向量操作

*A: = alpha*x*x' + A* ,

其中

alpha 是实数标量

x 是 n 个数的向量

A 是 n*n 的对称矩阵。

输入参数

uplo	一个字符。指定矩阵 A 的上三角部分还是下三角部分由压缩数组 ap 提供。如果 *uplo* = 'U'或'u',则的上三角部分由压缩数组 ap 提供如果 *uplo* = 'L'或'l',则矩阵 A 的下三角部分由压缩数组 ap 提供。
n	整数类型。指定矩阵 A 的顺序。n 的值必须至少为 0。
alpha	ssyr 中的实数 dsyr 中的双精度复数 指定 alpha 标量。
x	ssymv 中的复数 dsymv 中的双精度浮点数 数组,维度至少为(1 + (*n* - 1)*abs(*incx*))。输入前,增量数组 x 必须包含 n 个数的向量 x。
incx	整数类型。指定 x 中元素的增量。incx 的值必
a	ssyr 中的复数 dsyr 中的双精度复数 数组,维度为(Lda,n)。 输入 *uplo* = 'U'或'u'前,数组 a 的前 n*n 的上三角部分必须包含 Hermitian 矩阵的上三角部分且 a 的下三角部分不会被引用；输入 *uplo* = 'L'或'l'前,数组 a 的前 n*n 的下三角部分必须包含 Hermitian 矩阵的 下三角部分且 a 的下三角部分不会被引用

lda	整数类型。

输出参数

a	当 *uplo* = 'U'或'u',数组 a 的上三角部分被更新的矩阵的上三角部分覆盖。 当 uplo = 'L' 或 'l',数组 a 的下三角部分被更新的矩阵的下三角部 分覆盖。

3.2.19 ?syr2

语法

FORTRAN 77：

```
call ssyr2(uplo, n, alpha, x, incx, a, lda)
call dsyr2(uplo, n, alpha, x, incx, a, lda)
```

描述

?syr 函数指定如下矩阵向量操作

*A: = alpha*x*x' + A* ,

其中

alpha 是实数标量

x 是 n 个数的向量

A 是 n*n 的对称矩阵。

输入参数

uplo	一个字符。指定矩阵 A 的上三角部分还是下三角部分由压缩数组 ap 提供。如果 *uplo* = 'U'或'u',则的上三角部分由压缩数组 ap 提供如果 *uplo* = 'L'或'l',则矩阵 A 的下三角部分由压缩数组 ap 提供。
n	整数类型。指定矩阵 A 的顺序。n 的值必须至少为 0。
alpha	ssyr2 中的实数 dsyr2 中的双精度复数 指定 alpha 标量。
x	ssyr2 中的复数 Dssyr2 中的双精度浮点数 数组,维度至少为(1 + (*n* - 1)*abs(*incx*))。输入前,增量数组 x 必须包含 n 个数的向量 x。
incx	整数类型。指定 x 中元素的增量。incx 的值必
y	ssyr2 中的实数 dsyr2 中的双精度浮点数 数组,维度至少为(1 + (*n* - 1)*abs(*incy*))。 输入前,增加后数组 y 必须包含 n 个数的向量 y。
incy	整数类型。指定 y 中数的增量 incy 的值必须不为 0。
a	ssyr2 中的复数 dsyr2 中的双精度浮点数 数组,维度为(lda,n)。

输入 *uplo* = 'U'或'u'前,数组 a 的前 n*n 的上三角部分必须包含对称矩阵的上三角部分且 a 的下三角部分不会被引用;
输入 *uplo* = 'L'或'l'前,数组 a 的前 n*n 的下三角部分必须包含对称矩阵的下三角部分且 a 的下三角部分不会被引用

lda 整数类型。指定调用函数中定义的 a 的第一维的维度。lda 的值 必须至少为 max(1,n)。

输出参数

a 当 *uplo* = 'U'或'u',数组 a 的上三角部分被更新的矩阵的上三角部分覆盖。
当 *uplo* = *'L'* 或 'l',数组 a 的下三角部分被更新的矩阵的下三角部分覆盖。

3.2.20 ?tbmv

语法

FORTRAN 77：

```
call stbmv(uplo, trans, diag, n, k, a, lda, x, incx)
call dtbmv(uplo, trans, diag, n, k, a, lda, x, incx)
call ctbmv(uplo, trans, diag, n, k, a, lda, x, incx)
call ztbmv(uplo, trans, diag, n, k, a, lda, x, incx)
```

描述

?tbmv 函数计算如下矩阵向量操作

x := *A***x*, 或 *x* := *A*'**x*, 或 *x* := conjg(*A*')**x*,

其中

x 是 n 个数的向量

A 是 n*n 的单位(非单位)上(下)带三角矩阵,有 k+1 条对角线

输入参数

uplo 一个字符。指定矩阵 A 是上三角矩阵还是下三角矩阵：
当 *uplo* = 'U' 或 'u',矩阵为上三角矩阵;
当 *uplo* = 'L' 或 'l',矩阵为下三角矩阵.

trans 一个字符。指定操作
当 *trans* = 'N' 或 'n', 否则 *x* := *A***x*;
当 *trans* = 'T' 或 't', 否则 *x* := *A*'**x*;
当 *trans* = 'C' 或 'c', 否则 *x* := conjg(*A*')**x*.

diag 一个字符。指定A是否为单位三角矩阵
当 *diag* = 'U' 或 'u' ,是单位三角矩阵
当 *diag* = 'N' 或 'n', 不是单位三角矩阵

n 整数类型。Spec 当 ies the 或 der of the matrix *A*.n 的值至少为 0。

k 整数类型. 当 *uplo* = 'U' 或 'u', k为矩阵A的上三角线个数;当*uplo* = 'L' 或 'l', k为矩阵A的下三角线个数。*K* 的值必须满足k>=0

a ctbmv 中复数

ztbmv 中的双精度复数
数组,维度为(Lda,n)。
输入前,uplo = 'U' 或 'u’数组的前(k +1)*n 部分必须包含矩阵上三角带状部分。矩阵按列存储：主对角线存在数组第(k+1)行,第一条上对角线从第 ku 行的第 2 个位置开始存储,第一个下对角线从第 k+2 行的第 1 个位置开始存储 等等。左上 k*k 的三角形不会被引用。
下面的代码段将一个传统满矩阵存储转换为带状存储。

```
do 20, j = 1, n
    k = k + 1 - j
        do 10, i = max(1, j-k), min(m, j+k)
        a(k+i, j) = matrix(i,j)
    10 continue
20 continue
```

输入前,uplo = 'L' 或 'l’数组的前(k +1)*n 部分必须包含矩阵下三角带状部分。矩阵按列存储：主对角线存在数组第(k+1)行,第一条下对角线从第 2 行的第 1 个位置开始存储, 等等。右下 k*k 的三角形不会被引用。
下面的代码段将一个传统满矩阵存储转换为带状存储。

```
do 20, j = 1, n
    k = k + 1 - j
        do 10, i = max(1, j-k), min(m, j+k)
        a(k+i, j) = matrix(i,j)
    10 continue
20 continue
```

对角线上元素的虚数部分不需要设置且默认为 0

lda　整数类型。
指定调用函数中定义的 a 的第一维维度。lda 的值必须至少为(k+1)

x　stbmv 中的实数
dtbmv 中的双精度浮点数
ctbmv 中的复数
ztbmv 中的双精度复数
数组,维度至少为(1 + (*n* - 1)*abs(*incx*))。
输入前,增加后数组 x 必须包含 n 个数向量 x。

incx　整数类型。指定 x 中数的增量。
incx 的值必须不为 0。

输出参数

x　由转换后向量 x 覆盖。

3.2.21 ?tbsv

语法

FORTRAN 77：

```
call stbsv(uplo, trans, diag, n, k, a, lda, x, incx)
call dtbsv(uplo, trans, diag, n, k, a, lda, x, incx)
call ctbsv(uplo, trans, diag, n, k, a, lda, x, incx)
call ztbsv(uplo, trans, diag, n, k, a, lda, x, incx)
```

描述

?tbsv 函数解决如下系统等式之一：

*A*x = b*, 或 *A'*x = b*, 或 conjg(*A'*)*x = b*,

其中

b 和 x 是 n 个数的向量

A 是 n*n 的单位(非单位)上(下)带三角矩阵,有 k+1 条对角线

此函数不检测奇点或类奇点,测试应该在调用前进行。

输入参数

uplo　一个字符。指定矩阵 A 是上三角矩阵还是下三角矩阵：
当 *uplo* = 'U' 或 'u',矩阵为上三角矩阵;
当 *uplo* = 'L' 或 'l',矩阵为下三角矩阵.

trans　一个字符。指定操作
当 *trans* = 'N' 或 'n', 否则 *A*x = b*;
当 *trans* = 'T' 或 't', 否则 *A'*x = b*;
当 *trans* = 'C' 或 'c', 否则 conjg(*A'*)*x = b*.

diag　一个字符。指定矩阵A是否为单位三角阵：
当 *diag* = 'U' 或 'u' A是单位三角阵。
当 *diag* = 'N' 或 'n',A不是单位三角阵

n　整数类型。指定矩阵A的顺序。n的值至少为0。

k　整数类型. 当 *uplo* = 'U' 或 'u', k为矩阵A的上三角线个数;当*uplo* = 'L' 或 'l', k为矩阵A的下三角线个数。*K* 的值必须满足k>=0

a　ctbmv 中复数
ztbmv 中的双精度复数
数组,维度为(Lda,n)。
输入前,uplo = 'U' 或 'u’数组的前(k +1)*n 部分必须包含矩阵上三角带状部分。矩阵按列存储：主对角线存在数组第(k+1)行,第一条上对角线从第 ku 行的第 2 个位置开始存储,第一个下对角线从第 k+2 行的第 1 个位置开始存储 等等。左上 k*k 的三角形不会被引用。
下面的代码段将一个传统满矩阵存储转换为带状存储。

```
    do 20, j = 1, n
        k = k + 1 - j
            do 10, i = max(1, j-k), min(m, j+k)
            a(k+i, j) = matrix(i,j)
        10 continue
20 continue
```

输入前,uplo = 'L' 或 'l’数组的前(k +1)*n 部分必须包

含矩阵下三角带状部分。矩阵按列存储：主对角线存在数组第(k+1)行,第一条下对角线从第2行的第1个位置开始存储，等等。右下 k*k 的三角形不会被引用。

下面的代码段将一个传统满矩阵存储转换为带状存储。

```
    do 20, j = 1, n
        k = k + 1 - j
            do 10, i = max(1, j-k), min(m, j+k)
            a(k+i, j) = matrix(i,j)
        10 continue
20 continue
```

对角线上元素的虚数部分不需要设置且默认为 0

lda 整数类型。

指定调用函数中定义的 a 的第一维维度。lda 的值必须至少为(k+1)

x stpsv 中的实数

dtpsv 中的双精度浮点数

ctpsv 中的复数

ztpsv 中的双精度复数

数组，维度至少为 (1 + (*n* - 1)*abs(*incx*))。

输入前，增加后数组x必须包含向量b的右边n个元素。

incx 整数类型。指定 x 中数的增量。

incx 的值必须不为 0。

输出参数

x 由解向量 x 覆盖。

3.2.22 ?trmv

语法

FORTRAN 77：

```
call strmv(uplo, trans, diag, n, a, lda, x, incx)
call dtrmv(uplo, trans, diag, n, a, lda, x, incx)
call ctrmv(uplo, trans, diag, n, a, lda, x, incx)
call ztrmv(uplo, trans, diag, n, a, lda, x, incx)
```

描述

?trmv函数计算如下矩阵向量操作

x := *A***x*, 或 *x* := A'**x*, 或 *x* := conjg(*A*')**x*,

其中

x是n个数的向量,

A是n*n的单位(非单位)上(下)带三角矩阵,有k+1条对角线

输入参数

uplo 一个字符。指定矩阵 A 是上三角矩阵还是下三角矩阵：

当 *uplo* = 'U' 或 'u',矩阵为上三角矩阵;

当 *uplo* = 'L' 或 'l',矩阵为下三角矩阵.

trans 一个字符。指定操作

	当 *trans* = 'N' 或 'n', 否则 *x* := *A***x*; 当 *trans* = 'T' 或 't', 否则 *x* := *A*'**x*; 当 *trans* = 'C' 或 'c', 否则 *x* := conjg(*A*')**x*.
diag	一个字符。指定矩阵A是否为单位三角阵： 当 *diag* = 'U' 或 'u' A是单位三角阵。 当 *diag* = 'N' 或 'n',A不是单位三角阵
n	整数类型。指定矩阵A的顺序。n的值至少为0。
a	strmv中的实数 dtrmv中的双精度浮点数 ctrmv中的复数 ztrmv中的双精度复数 数组,维度为(lda,n)。 输入 *uplo* = 'U'或'u'前,数组 a 的前 n*n 的上三角部分必须包含矩阵的上三角部分且 a 的下三角部分不会被引用；输入 *uplo* = 'L'或'l'前,数组 a 的前 n*n 的下三角部分必须包含矩阵的下三角部分且 a 的下三角部分不会被引用 当*diag* = 'U' 或 'u', a的对角线元素也不会被引用,但被假定 为单位数。
lda	整数类型。指定调用函数中定义的 a 的第一维维度。lda 至少为 max(1,n).
x	strmv中的实数 dtrmv中的双精度浮点数 ctrmv中的复数 ztrmv中的双精度复数 数组,维度至少为(1 + (*n* - 1)*abs(*incx*))。 输入前,增加后数组x必须包含n个数的向量x。
incx	整数类型。指定x中数的增量。 incx的值必须不为0。

输出参数

x	由转换后向量x覆盖。

3.2.23 ?trsv

语法

FORTRAN 77：

```
call strsv(uplo, trans, diag, n, a, lda, x, incx)
call dtrsv(uplo, trans, diag, n, a, lda, x, incx)
call ctrsv(uplo, trans, diag, n, a, lda, x, incx)
call ztrsv(uplo, trans, diag, n, a, lda, x, incx)
```

描述

?trsv函数解决如下系统等式之一：

*A***x* = *b*, 或 *A*'**x* = *b*, 或 conjg(*A*')**x* = *b*,

其中

b和x是n个数的向量

A 是 n*n 的单位(非单位)上(下)带三角矩阵,有 k+1 条对角线

此函数不检测奇点或类奇点,测试应该在调用前进行。

输入参数

uplo	一个字符。指定矩阵 A 是上三角矩阵还是下三角矩阵： 当 *uplo* = 'U' 或 'u',矩阵为上三角矩阵; 当 *uplo* = 'L' 或 'l',矩阵为下三角矩阵.
trans	一个字符。指定计算等式 当 *trans* = 'N' 或 'n', 否则 *A*x = b*; 当 *trans* = 'T' 或 't', 否则 *A'*x = b*; 当 *trans* = 'C' 或 'c', 否则 conjg(A')*x = b.
diag	一个字符。指定矩阵A是否为单位三角阵： 当 *diag* = 'U' 或 'u' A是单位三角阵。 当 *diag* = 'N' 或 'n',A不是单位三角阵
n	整数类型。指定矩阵A的顺序。n的值至少为0。
a	数组,维度为(lda,n)。 输入 *uplo* = 'U'或'u'前,数组 a 的前 n*n 的上三角部分必须包含矩阵的上三角部分且 a 的下三角部分不会被引用；输入 *uplo* = 'L'或'l'前,数组 a 的前 n*n 的下三角部分必须包含矩阵的下三角部分且 a 的下三角部分不会被引用 当 *diag* = 'U' 或 'u', a的对角线元素也不会被引用,但被假定为单位数。
lda	整数类型。指定调用函数中定义的a的第一维维度。 A的值必须至少为max(1,n)。
x	strsv中的实数 dtrsv中的双精度浮点数 ctrsv中的复数 ztrsv中的双精度复数 数组,维度至少为(1 + (n - 1)*abs(incx))。 输入前,增加后数组x必须包含向量b的右边n个元素。
incx	整数类型。指定x中数的增量。 incx的值必须不为0。

输出参数

x	由解向量 x 覆盖

3.3 BLAS Level 3 函数说明

表 3-3 BLAS Level 3 函数汇总表

子程序组	数据类型	描述
?gemm	s,d,c,z	矩阵-矩阵乘
?hemm	c,z	Hermitian 矩阵-矩阵乘
?herk	c,z	Hermitian 矩阵秩 k 更新
?her2k	c,z	Hermitian 矩阵秩 2k 更新
?symm	s,d,c,z	对称矩阵-矩阵乘

?syrk	s,d,c,z	三角矩阵秩 k 更新
?syr2k	s,d,c,z	三角矩阵秩 2k 更新
?trmm	s,d,c,z	三角矩阵-矩阵乘
?trsm	s,d,c,z	三角矩阵的线性矩阵-矩阵方程组求解

3.3.1 ？Gemm

语法

FORTRAN 77:

call sgemm(transa, transb, m, n, k, alpha, a, lda, b, ldb, beta, c, ldc)

call dgemm(transa, transb, m, n, k, alpha, a, lda, b, ldb, beta, c, ldc)

call cgemm(transa, transb, m, n, k, alpha, a, lda, b, ldb, beta, c, ldc)

call zgemm(transa, transb, m, n, k, alpha, a, lda, b, ldb, beta, c, ldc)

call scgemm(transa, transb, m, n, k, alpha, a, lda, b, ldb, beta, c, ldc)

call dzgemm(transa, transb, m, n, k, alpha, a, lda, b, ldb, beta, c, ldc)

描述

？Gemm 程序完成了一个针对一般矩阵的矩阵-矩阵操作。定义为

C := alpha*op(A)*op(B) + beta*C,

其中：

op(x) 是 op(x) = x, or op(x) = x', or op(x) = conjg(x')之一；

alpha 和 beta 是标量；

A, B 和 C 是矩阵；

op(A)是 m*k 矩阵,op(B)是 k*n 的矩阵,C 是 m*n 矩阵。

输入参数

Transa	CHARACTER 类型。 详细说明了矩阵乘法中使用的 op(A)的格式 如果 transa = 'N' or 'n',则 op(A) = A; 如果 transa = 'T' or 't', 则 op(A) = A'; 如果 transa = 'C' or 'c', 则 op(A) = conjg(A').
Transb	CHARACTER 类型。 详细说明了矩阵乘法中使用的 op(B)的格式 如果 transb = 'N' 则 'n', then op(B) = B; 如果 transb = 'T' 则 't', then op(B) = B'; 如果 transb = 'C' 则 'c', then op(B) = conjg(B').
m	整数类型。说明了矩阵 op(A)和矩阵 c 的行数。
n	整数类型。说明了矩阵 op(B)和矩阵 c 的列数。n 的值至少为 0。
K	整数类型。说明矩阵 op(A)的列数和矩阵 op(B)的行数。K 至少为 0。
alpha	sgemm 函数下为实数类型 dgemm 函数下为双精度类型 cgemm 和 scgemm 函数下为复数类型 zgemm 和 dzgemm 函数下为双精度复数类型 定义标量 alpha。
a	sgemm 和 scgemm 函数下为实数类型 dgemm 和 dzgemm 函数下为双精度类型 cgemm 函数下为复数类型 zgemm 函数下为双精度复数类型

数组,规模为(lda, ka), 当 transa= 'N' or 'n'时,ka 是 k 。否则 ka 是 m。如果 transa = 'N' or 'n', a 的前 m*k 部分必须包含矩阵 A,否则 a 的前 k*m 部分必须包含矩阵 A。

lda 整数类型。说明调用程序中 a 的第一维的维数。当 transa='N' or 'n'时。lda 至少为 max(1, m),否则 lda 至少为 max(1, k)。

b sgemm 函数下为实数类型
dgemm 函数下为双精度类型
cgemm 和 scgemm 函数下为复数类型
zgemm 和 dzgemm 函数下为双精度复数类型
数组, 规模为(ldb, kb), k 当 transb= 'N' or 'n'时,b 是 n。否则为 k。开始前, 如果 transb= 'N' or 'n', 数组 b 的前 k*n 部分必须包含矩阵 B,否则数组 b前 n*k 部分必须包含数组 B。

ldb 整数类型。说明了调用程序中 b 的第一维的维数。当 transb = 'N' or 'n' 时, ldb 至少为 max(1, k),否则 ldb 至少为 max(1, n)。

beta sgemm 函数下为实数类型
dgemm 函数下为双精度类型
cgemm 和 scgemm 函数下为复数类型
zgemm 和 dzgemm 函数下为双精度复数类型
定义标量 beta。当 beta 为 zero 时, 输入就不需要 c。

c sgemm 函数下为实数类型
dgemm 函数下为双精度类型
cgemm 和 scgemm 函数下为复数类型
zgemm 和 dzgemm 函数下为双精度复数类型
数组, 规模为 (ldc, n).在调用前, 数组 c 的前 m*n 部分必须包含矩阵 C, 但当 beta 等于 zero, 输入部分就不再需要 c 了。

Ldc 整数类型。 说明了调用函数中 c 的第一维的维数. Ldc 的值至少为 max(1, m).

输出参数

c 被矩阵结果 (alpha*op(A)*op(B) + beta*C)覆写。

3.3.2 ？hemm

语法

FORTRAN 77:

call chemm(side, uplo, m, n, alpha, a, lda, b, ldb, beta, c, ldc)

call zhemm(side, uplo, m, n, alpha, a, lda, b, ldb, beta, c, ldc)

描述

?hemm 程序执行了一个 Hermitian 矩阵-矩阵 运算。 该操作定义如下

C := alpha*A*B + beta*C

或

C := alpha*B*A + beta*C,

其中：alpha 和 beta 是标量,A 是一个 Hermitian 矩阵,B 和 C 是 m*n 矩阵。

输入参数

Side	CHARACTER 类型。表明 Hermitian 矩阵 A 在如下操作的左侧或右侧: 如果 side = 'L' or 'l',则 C := alpha*A*B + beta*C; 如果 side = 'R' or 'r',则 C := alpha*B*A + beta*C。
Uplo	CHARACTER 类型。 表明使用 Hermitian 矩阵 A 上三角或下三角形部分 If uplo = 'U' or 'u', 那么 Hermitian 矩阵 A 的上三角部分被使用了. If uplo = 'L' or 'l', 那么 Hermitian 矩阵 A 的下三角部分被使用了.
m	整数类型。说明矩阵 C 的行数。m 的值 至少为 0。
n	整数类型。说明矩阵 C 的列数 。n 的值至少为 0。
alpha	chemm 函数下为复数类型 zhemm 函数下为双精度复数类型 定义变量 alpha。
a	chemm 函数下为复数类型 zhemm 函数下为双精度复数类型 数组, 规模为 (lda,ka), ,其中，当 side ='L' 或 'l'时，ka 为 m 否则为 n .如果 side ='L' or 'l', 数组 a 的 m×m 部分必须包含 Hermitian 矩阵。例如，当 uplo = 'U' or 'u', 数组 a 的前 m×m 上三角部分必须包含上 Hermitian 矩阵的 三角部分 ,当 uplo = 'L' or 'l', 数组 a 的前 m×m 下三角部分必须包含 Hermitian 矩阵的下三角部分。 如果 side = 'R' or 'r', 数组 a 的 n×n 部分必须包含 Hermitian 矩阵。例如，当 uplo = 'U' or 'u',数组 a 得 前 n×n 的上三角部分必须包含 Hermitian 矩阵的上三角部分, 如果 uplo ='L' or 'l', 数组 a 的前 n×n 的下三角部分必须包含 Hermitian 矩阵的下三角部分.不需要设置对角线元素的虚数部分, 他们当成 0 处理。
Lda	整数类型。说明了调用函数中声明的 a 第一维的维数。当 side = 'L' or 'l' lda 至少为 max(1, m), 否则 lda 至少为 max(1,n)。
b	chemm 函数下为复数类型 zhemm 函数下为双精度复数类型 数组,规模为(ldb,n)。 在调用前,数组 b 的前 m*n 部分必须包含矩阵 B.
ldb	整数类型。说明了说明了调用函数中声明的 b 第一维的维数. ldb 的值至少为 max(1, m)。
beta	chemm 函数下为复数类型 zhemm 函数下为双精度复数类型 定义了标量 beta。 当 beta 为 zero, c 就不需要被设置了。
c	chemm 函数下为复数类型 zhemm 函数下为双精度复数类型 数组,规模为 (c, n). 在调用之前, 数组 c 的前 m*n 部分必须包含矩阵 C, 但当 beta 为 zero, c 不需要在调用时设置。
Ldc	整数类型。说明了说明了调用函数中声明的 c 第一维的维数。

输出参数

c	被 m*n 的更新矩阵覆写。

3.3.3 ？herk

语法

FORTRAN 77:

call cherk(uplo, trans, n, k, alpha, a, lda, beta, c, ldc)

call zherk(uplo, trans, n, k, alpha, a, lda, beta, c, ldc)

描述

？herk 函数实现 Hermitian 矩阵的矩阵矩阵操作。操作定义如下：

C := alpha*A*conjg(A') + beta*C,

or

C := alpha*conjg(A')*A + beta*C,

其中

alpha 和 beta 是实数标量,

C 是一个 n*n 的 Hermitian 矩阵,

在上述第一个式子中 A 是一个 n*k 的矩阵,在第二个式子中 A 是一个 k*n 的矩阵。

输入参数

uplo　　CHARACTER 类型。定义是用矩阵 C 的上三角部分还是下三角部分。

如果 uplo='U'或者'u',那么使用矩阵 C 的上三角部分。

如果 uplo='L'或者'l',那么使用矩阵 C 的下三角部分。

trans　　CHARACTER 类型。定义下述操作：

如果 trans='N'或者'n',那么 C:=alpha*A*conjg(A')+beta*C;

如果 trans = 'C' or 'c', then C:=alpha*conjg(A')*A+beta*C;

n　　整数。定义矩阵 C 的维数。n 至少是 0。

k　　整数。如果 trans='N'或者'n',k 定义了矩阵 A 的列数；如果 trans = 'C' or 'c',k 定义了矩阵 A 的行数。k 的值至少是 0。

alpha　　cherk 函数下是实数；

zherk 函数下是双精度；

标量 alpha。

a　　cherk 函数下是复数

zherk 函数下是双精度复数类型

数组,规模为(lda,ka),其中当 trans='N'或者'n'时 ka=k,否则 ka=n。如果 trans='N'或者'n',数组 a 的前 n*k 部分存储矩阵 A,否则数组前 k*n 的部分存储矩阵 A。

lda 整数。定义数组 a 的行维度。当 trans='N'或者'n'时,lda 至少是 max(1,n),否则 lda 至少　是 max(1,k)。

beta　　cherk 函数下是实数

zherk 函数下是双精度类型

标量被 beta。

c　　cherk 函数下是复数

zherk 函数下是双精度复数类型

数组,规模为(ldc,n)。

如果 uplo = 'U' or 'u',数组 c 的前 n*n 部分保存 Hermitian 矩阵的上三角元素,严格下三角部分不被引用。如果 uplo = 'L' or 'l',数组 c 的前 n*n 部分保存 Hermitian 矩阵的下三角部分元素,严格上三角部分不被引用。对角线元素的虚部不需要被设置,假定它们为 0。

ldc　　整数。定义 c 的行维度大小。ldc 的值至少为 max(1,n)。

输出参数

c　　如果 uplo = 'U' or 'u',c 的上三角部分被更新矩阵的上三角部分覆写。
如果 uplo = 'L' or 'l',c 的下三角部分被跟新矩阵的下三角部分覆写。
对角线元素的虚部设为 0。

3.3.4 ？her2k

语法

FORTRAN 77:

call cher2k(uplo, trans, n, k, alpha, a, lda, b, ldb, beta, c, ldc)

call zher2k(uplo, trans, n, k, alpha, a, lda, b, ldb, beta, c, ldc)

描述

该函数实现 Hermitian 矩阵的 rank-2k 矩阵矩阵操作。该操作定义为：

C := alpha*A*conjg(B') + conjg(alpha)*B*conjg(A') + beta*C,

或者

C := alpha *conjg(B')*A + conjg(alpha) *conjg(A')*B + beta*C,

其中：

alpha 是标量,beta 是一个实数标量,

c 是一个 n*n 的 Hermitian 矩阵,

在一种式子情况下 A 和 B 是 n*k 矩阵,在第二种式子中 A 和 B 是 k*n 矩阵。

输入参数

uplo　　CHARACTER 类型。定义是用矩阵 C 的上三角部分还是下三角部分。
如果 uplo='U'或者'u',那么使用矩阵 C 的上三角部分。
如果 uplo='L'或者'l',那么使用矩阵 C 的下三角部分。

trans　　CHARACTER 类型。定义下述操作：
如果 trans='N'或者'n',那么 C:=alpha*A*conjg(B') + alpha*B*conjg(A') + beta*C;
如果 trans = 'C' or 'c', 那么 C:=alpha*conjg(A')*B + alpha*conjg(B')*A + beta*C.

n　　整数。定义矩阵 C 的规则。n 至少是 0。

k　　整数。如果 trans = 'N' or 'n',k 定义矩阵 A 的列数,如果 trans = 'C' or 'c',k 定义矩阵 A 的行数。

alpha　　cher2k 函数下为复数类型；
zher2k 函数下为双精度复数类型；
标量 alpha。

a　　cher2k 函数下为复数类型；
zher2k 函数下为双精度复数类型；
数组,规模为(lda,ka),当 trans= 'N' or 'n'时,ka=k 否则 ka=n。如果 trans= 'N' or 'n',数组 a 的前 n*k 部分必须包含矩阵 A,否则数组 a 的前 k*n 部分必须包含矩阵 A。

lda　　整数。定义 a 的主维。当 trans= 'N' or 'n'时,lda 至少为 max(1,n),否则 lda 至少为 max(1,k)。

beta　　cher2k 函数下为复数类型；
zher2k 函数下为双精度复数类型；
标量 beta。

b　　cher2k 函数下为复数类型；
zher2k 函数下为双精度复数类型；
数组,规模为(ldb,kb),当 trans= 'N' or 'n'时,kb=k,否则 kb=n。如果 trans= 'N' or 'n',数组 b 的前 n*k 部分必须包含矩阵 B,否则数组 b 的前 k*n 部分必须包含矩阵 B。

ldb　　整数。定义 b 的主维。当 trans= 'N' or 'n'时,ldb 至少为 max(1,n),否则 ldb 至少为 max(1,k)。

c　　cher2k 函数下为复数类型；
zher2k 函数下为双精度复数类型；
数组,规模为(ldc,n)。如果 uplo = 'U' or 'u',数组 c 的前 n*n 部分保存 Hermitian 矩阵的上三角元素,严格下三角部分不被引用。如果 uplo = 'L' or 'l',数组 c 的前 n*n 部分保存 Hermitian 矩阵的下三角部分元素,严格上三角部分不被引用。
对角线元素的虚部不需要被设置,假定它们为 0。

ldc　　整数。定义 c 的行维度大小。ldc 的值至少为 max(1,n)。

输出参数

c　　如果 uplo = 'U' or 'u',c 的上三角部分被更新矩阵的上三角部分覆写。
如果 uplo = 'L' or 'l',c 的下三角部分被跟新矩阵的下三角部分覆写。
对角线元素的虚部设为 0。

3.3.5 ？ symm

语法

FORTRAN 77:

call ssymm(side, uplo, m, n, alpha, a, lda, b, ldb, beta, c, ldc)
call dsymm(side, uplo, m, n, alpha, a, lda, b, ldb, beta, c, ldc)
call csymm(side, uplo, m, n, alpha, a, lda, b, ldb, beta, c, ldc)
call zsymm(side, uplo, m, n, alpha, a, lda, b, ldb, beta, c, ldc)

描述

该函数实现对称矩阵间的操作,操作定义如下：
C := alpha*A*B + beta*C,
或者
C := alpha*B*A + beta*C,
其中：
alpha 和 beta 是标量,A 是对称矩阵,B 和 C 是 m*n 矩阵。

输入参数

side　　CHARATHER 类型。定义对称矩阵 A 出现在操作的左边还是右边。
如果 side = 'L' or 'l',那么 C := alpha*A*B + beta*C；
如果 side = 'R' or 'r',那么 C := alpha*B*A + beta*C.

uplo　　CHARATHER 类型。定义使用矩阵 A 的上三角部分还是下三角部分。
如果 uplo = 'U' or 'u',使用上三角部分；
如果 uplo = 'L' or 'l',使用下三角部分。

m　　整数。定义矩阵 C 的行数。m 至少是 0。

n　　整数。定义矩阵 C 的列数。n 至少是 0。

alpha　　ssymm 函数下是实数类型；
dsymm 函数下是双精度类型；
csymm 函数下是复数类型；
zsymm 函数下是双精度复数类型；
定义标量 alpha。

a　　ssymm 函数下是实数类型；
dsymm 函数下是双精度类型；
csymm 函数下是复数类型；
zsymm 函数下是双精度复数类型；
数组,规模为(lda,ka),当 side ='L' or 'l'时,ka=m,否则 ka=n。
当 side = 'L' or 'l'时,a 数组为 m*m 用于存储矩阵 A,当 uplo = 'U' or 'u'时,对称矩阵 A 的上三角部分必须存储于 a 中,严格下三角部分可以不涉及；当 uplo = 'L' or 'l' 时,对阵矩阵 A 的下三角部分必须存储于 a 中,严格上三角部分可以不涉及。
当 side = 'R' or 'r'时,a 数组为 n*n,当 uplo = 'U' or 'u'时,对称 矩阵 A 的上三角部分必须存储于 a 中,严格下三角部分可以不涉及；当 uplo = 'L' or 'l' 时，对阵矩阵 A 的下三角部分必须存储于 a 中,严格上三角部分可以不涉及。

lda　　整数。定义数组 a 的主维。当 side = 'L' or 'l'时,lda 至少为 max(1, m),
否则 lda 应满足 max(1,n)。

b　　ssymm 函数下是实数类型；
dsymm 函数下是双精度类型；
csymm 函数下是复数类型；
zsymm 函数下是双精度复数类型；
数组,规模为(ldb,n)。开辟 m*n 的空间存储矩阵 B。

ldb　　整数。定义数组 b 的主维。ldb 应满足 max(1,m)。

beta　　ssymm 函数下是实数类型；
dsymm 函数下是双精度类型；
csymm 函数下是复数类型；
zsymm 函数下是双精度复数类型；
定义标量 alpha。
当 beta 为 0 时,c 不需要输入。

c　　ssymm 函数下是实数类型；
dsymm 函数下是双精度类型；
csymm 函数下是复数类型；
zsymm 函数下是双精度复数类型；
数组,规模为(ldc,n)。开辟 m*n 的空间存储矩阵 C。当 beta 为 0 时,不需 要作为输入参数。

ldc　　整数。定义数组 c 的主维。ldc 应满足 max(1,m)。

输出参数

c　　被更新矩阵结果覆写。

3.3.6 ？syrk

语法

FORTRAN 77:

call ssyrk(uplo, trans, n, k, alpha, a, lda, beta, c, ldc)

call dsyrk(uplo, trans, n, k, alpha, a, lda, beta, c, ldc)

call csyrk(uplo, trans, n, k, alpha, a, lda, beta, c, ldc)

call zsyrk(uplo, trans, n, k, alpha, a, lda, beta, c, ldc)

描述

该函数实现对称矩阵间的操作,操作定义如下：

C := alpha*A*A' + beta*C,

或者

C := alpha*A'*A + beta*C,

其中：

alpha 和 beta 是标量,C 是 n*n 的对称矩阵,对于上述第一个式子而言,A 是 n*k 的矩阵,对于第二个式子,A 是 k*n 的矩阵。

输入参数

uplo　　CHARATHER 类型。定义使用矩阵 C 的上三角部分还是下三角部分。

如果 uplo = 'U' or 'u',使用上三角部分；

如果 uplo = 'L' or 'l',使用下三角部分。

trans　　CHARATHER 类型。定义下述操作：

如果 trans = 'N' or 'n', 那么 C := alpha*A*A' + beta*C;

如果 trans = 'T' or 't', 那么 C := alpha*A'*A + beta*C;

如果 trans = 'C' or 'c', 那么 C := alpha*A'*A + beta*C.

n　　整数。定义矩阵 C 维数。n 至少是 0。

k　　整数。当 trans = 'N' or 'n'时,k 代表矩阵 A 的列数,当 trans = 'T' or 't' or 'C' or 'c'时,k 代表矩阵 A 的行数。k 的值最小为 0。

alpha　　ssyrk 函数下为实数类型

dsyrk 函数下为双精度类型

csyrk 函数下为复数类型

zsyrk 函数下为双精度复数类型

定义标量 alpha。

a　　ssyrk 函数下为实数类型

dsyrk 函数下为双精度类型

csyrk 函数下为复数类型

zsyrk 函数下为双精度复数类型

数组,规模为(lda,ka),当 trans = 'N' or 'n'时,ka=k,否则 ka=n。当 trans = 'N' or 'n'时,数组 a 的 n*k 部分用于存储矩阵 A,否则 k*n 部分存储矩阵 A。

lda　　整数。定义数组 a 的主维,当 trans = 'N' or 'n'时,lda 至少为 max(1,n),否则 lda 至少为 max(1,k)。

beta　　ssyrk 函数下为实数类型；

dsyrk 函数下为双精度类型；

csyrk 函数下为复数类型；

zsyrk 函数下为双精度复数类型；

定义标量 beta。

c　　ssyrk 函数下为实数类型；
dsyrk 函数下为双精度类型；
csyrk 函数下为复数类型；
zsyrk 函数下为双精度复数类型；
数组,规模为(ldc,n)。如果 uplo = 'U' or 'u',数组 c 必须存储对称矩阵的上三角部分,严格下三角部分可以不被引用。
如果 uplo = 'L' or 'l',数组 c 必须存储对称矩阵的下三角部分,严格上三角部分可以不被引用。

ldc　　整数。定义矩阵 c 的主维。Ldc 至少为 max(1,n)。

输出参数

c　　当 uplo = 'L' or 'l',时,数组 c 中下三角部分被更新写入,当 uplo = 'U' or 'u'时,数组 c 中上三角部分被更新写入。

3.3.7 ?syr2k

语法

FORTRAN 77：

Call ssyr2k(uplo, trans, n, k, alpha, a, lda, b, ldb, beta, c, ldc)
Call dsyr2k(uplo, trans, n, k, alpha, a, lda, b, ldb, beta, c, ldc)
Call csyr2k(uplo, trans, n, k, alpha, a, lda, b, ldb, beta, c, ldc)
Call zsyr2k(uplo, trans, n, k, alpha, a, lda, b, ldb, beta, c, ldc)

描述

？Syr2k 函数对一个矩阵-矩阵的运算使用对称矩阵进行一个 rank-2k 的操作,这个操作的如下所示：

C :=alpha*A*B' +alpha*B*A' + beta*C,

或者

C := alpha*a'*B + alpha*b'*A + beta*C,

其中：

alpha 和 beta 是标量,

C 是 一个 n×n 的对称矩阵,

A 和 B 在第一个例子中是 n*k 的矩阵,在第二个矩阵中式 k*n 矩阵

输入参数

uplo　　CHARACTER*1。详细说明使用的是数组 c 的上三角或者是下三角。
如果 uplo = 'U'或者 'u',那么数组 c 的上三角矩阵被使用。
如果 uplo = 'L'或者'l',那么数组 c 的下三角矩阵被使用。

trans　　CHARACTER*1。详细说明使用的操作：
如果 trans = 'N' 或者 'n' ,那么 C :=alpha*A*B' +alpha*B*A' + beta*C,
如果 trans = 'T'或者't',那么 C := alpha*a'*B + alpha*b'*A + beta*C,
如果 trans = 'C'或者'c',那么 C := alpha*a'*B + alpha*b'*A + beta*C,

n　　整数类型。详细说明矩阵 c 的维数，n 的值至少是零。

k　　整数类型。当 trans = 'N'或者'n'时，k 表示矩阵 A 和 B 的列数，当 trans = 'T'或

者't'或者'C'或者'c'时，k 表示矩阵和 B 的行数。k 最小值是零。

alpha　REAL 对于 ssyr2k；
DOUBLE PRECISION 对于 dsyr2k；
COMPLEX 对于 csyr2k；
DOUBLE COMPLEX 对于 zsyr2k；
alpha 标量值

a　REAL 对于 ssyr2k；
DOUBLE PRECISION 对于 dsyr2k；
COMPLEX 对于 csyr2k；
DOUBLE COMPLEX 对于 zsyr2k；
数组,规模为(lda,k)。当 trans = 'N' 或者 'n'时,ka=k,否则为 n。当输入 trans = 'N' 或者 'n'时,数组 a 的 n×k 部分必须包含矩阵 A,否则数组 a 的 k×n 部分必须包含矩阵 A。

lda　INTEGER。a 的主维。
当 trans = 'N'或者'n'的时候,那么 lda 必须至少是 max(1, n),其他情况 lda 必须至少是 max(1, k)。

b　REAL 对于 ssyr2k；
DOUBLE PRECISON 对于 dsyr2k；
COMPLEX 对于 csyr2k；
DOUBLE COMPLEX 对于 zsyr2k；
数组,维度(ldb,kb)，当 trans = 'N'或者'n'时，kb=k 否则为 n。在 trans = 'N'或者'n'时,数组 b 的在前面的 n×k 部分必须包含 矩阵 B。在其他情况下，数组 b 前面的 k×n 部分必须包含矩阵 B。

ldb　INTEGER。详细说明了在调用子程序的时候 a 的第一维数值。当 trans = 'N'或者'n'的时候,那么 lda 必须至少是 max(1, n),其他情况 lda 必须至少是 max(1, k)。

beta　REAL 对于 ssyr2k
DOUBLE PRECISION 对于 dsyr2k
COMPLEX 对于 csyr2k
DOUBLE COMPLEX 对于 zsyr2k
beta 标量值

c　REAL 对于 ssyr2k
DOUBLE PRECISON 对于 dsyr2k
COMPLEX 对于 csyr2k
DOUBLE COMPLEX 对于 zsyr2k
数组,维度(ldb,kb)
在 uplo = 'U'或者'u'时，数组 c 的前面的 n×k 的上三角矩阵部分必须包含对称矩阵的上三角部分。
在 uplo = 'L'或者'l'时，数组 c 的前面的 n×n 的下三角矩阵部分必须包含对称矩阵的下三角部分。

ldc　INTEGER。详细说明了在调用子程序的时候 c 的主维。ldc 的数值至少是 max(1, n)。

输出参数

c　当 uplo = 'U'或者'u'的时候,数组 c 的上三角矩阵部分被更新的矩阵的上三角部分重写。

当 uplo = 'L'或者'l'的时候,数组 c 的下三角矩阵部分被更新的矩阵的下三角部分覆盖。

3.3.8 ？trmm

语法

FORTRAN 77:

call strmm(side, uplo, transa, diag, m, n, alpha, a, lda, b, ldb)

call dtrmm(side, uplo, transa, diag, m, n, alpha, a, lda, b, ldb)

call ctrmm(side, uplo, transa, diag, m, n, alpha, a, lda, b, ldb)

call ztrmm(side, uplo, transa, diag, m, n, alpha, a, lda, b, ldb)

描述

该函数实现三角矩阵操作,该操作被定义为：

B := alpha*op(A)*B

或者

B := alpha*B*op(A)

其中：

alpha 是标量,B 是 m*n 的矩阵,A 是一个单位或非单位,上三角或下三角矩阵。op(A)是 op(A) = A, 或者 op(A) = A ’ 或者 op(A) = conjg(A’)之一。

输入参数

side	CHARACTER 类型。定义操作 op(A、)是在矩阵 B 的左边还是右边： 如果 side = 'L' or 'l',那么 B := alpha*op(A)*B； 如果 side = 'R' or 'r',那么 B := alpha*B*op(A)。
uplo	CHARACTER 类型。定义矩阵 A 是上三角还是下三角矩阵： 如果 uplo = 'U' or 'u',矩阵是上三角矩阵； 如果 uplo = 'L' or 'l',矩阵是下三角矩阵。
transa	CHARACTER 类型。定义 op(A)在矩阵乘法中的形式： 如果 transa = 'N' or 'n',那么 op(A) = A； 如果 transa = 'T' or 't',那么 op(A) = A’； 如果 transa = 'C' or 'c',那么 p(A) = conjg(A’)。
diag	CHARACTER 类型。定义矩阵 A 是否是单位三角矩阵： 如果 diag = 'U' or 'u' 则矩阵是单位对角线三角矩阵； 如果 diag = 'N' or 'n'则矩阵不是单位三角矩阵。
m	整数。定义矩阵 B 的行数,m 至少是 0。
n	整数。定义矩阵 B 的列数,n 至少是 0。
alpha	strmm 函数下为实数类型 dtrmm 函数下为双精度类型 ctrmm 函数下为复数类型 ztrmm 函数下为双精度复数类型 定义标量 alpha,当 alpha 为 0 时,a 未被引用而 b 不需要作为输入参数。
a	strmm 函数下为实数类型 dtrmm 函数下为双精度类型 ctrmm 函数下为复数类型

ztrmm 函数下为双精度复数类型

数组,规模为(lda,k),当 side = 'L' or 'l'时,k=m；当 side = 'R' or 'r'时,k=n。当 uplo = 'U' or 'u'时,数组 a 的 k*k 部分用于存数三角矩阵的上三角部分， 严格下三角部分未被引用。当 uplo = 'L' or 'l'时,数组 a 的 k*k 部分用于存储 下三角矩阵的部分,严格上三角部分未被引用。

当 diag = 'U' or 'u'时,a 的对角线元素也未被引用,但是假定为单位 1。

lda 整数。数组 a 的第一维度大小。当 side = 'L' or 'l'时,lda 满足 max(1,m),当 side = 'R' or 'r'时,lda 满足 max(1,n)。

b strmm 函数下为实数类型

dtrmm 函数下为双精度类型

ctrmm 函数下为复数类型

ztrmm 函数下为双精度复数类型

数组,规模为(ldb,n)m*n 数组 b 用于存储矩阵 B。

ldb 整数。定义数组 b 的第一维度大小。ldb 满足 max(1,m)。

输出参数

b 被变换矩阵结果覆写。

3.3.9 ？trsm

语法

FORTRAN 77:

```
call strsm(side, uplo, transa, diag, m, n, alpha, a, lda, b, ldb)
call dtrsm(side, uplo, transa, diag, m, n, alpha, a, lda, b, ldb)
call ctrsm(side, uplo, transa, diag, m, n, alpha, a, lda, b, ldb)
call ztrsm(side, uplo, transa, diag, m, n, alpha, a, lda, b, ldb)
```

描述

该函数解决下述矩阵方程：

op(A)*X = alpha*B,

或者

X*op(A) = alpha*B,

其中：

alpha 是标量,X 和 B 是 m*n 的矩阵,A 是一个单位或者非单位,上三角或者下三角矩阵,

op(A)是 op(A) = A, 或者 op(A) = A’ 或者 op(A) = conjg(A’)。

输入参数

side CHARACTER 类型。定义操作 op(A、)是在矩阵 B 的左边还是右边：

如果 side = 'L' or 'l',那么 op(A)*X = alpha*B；

如果 side = 'R' or 'r',那么 X*op(A) = alpha*B。

uplo CHARACTER 类型。定义矩阵 A 是上三角还是下三角矩阵：

如果 uplo = 'U' or 'u',矩阵是上三角矩阵；

如果 uplo = 'L' or 'l',矩阵是下三角矩阵。

transa CHARACTER 类型。定义 op(A)在矩阵乘法中的形式：

如果 transa = 'N' or 'n',那么 op(A) = A；

如果 transa = 'T' or 't',那么 op(A) = A’；

如果 transa = 'C' or 'c',那么 op(A) = conjg(A’)。

diag　CHARACTER 类型。定义矩阵 A 是否是单位三角矩阵：
如果 diag = 'U' or 'u' 则矩阵是单位对角线三角矩阵；
如果 diag = 'N' or 'n'则矩阵不是单位三角矩阵。

m　整数。定义矩阵 B 的行数,m 至少是 0。

n　整数。定义矩阵 B 的列数,n 至少是 0。

alpha　strsm 函数下为实数类型
dtrsm 函数下为双精度类型
ctrsm 函数下为复数类型
ztrsm 函数下为双精度复数类型
定义标量 alpha,当 alpha 为 0 时,a 未被引用而 b 不需要作为输入参数。

a　strsm 函数下为实数类型
dtrsm 函数下为双精度类型
ctrsm 函数下为复数类型
ztrsm 函数下为双精度复数类型
数组,规模为(lda,k),当 side = 'L' or 'l'时,k=m；当 side = 'R' or 'r'时,k=n。当 uplo = 'U' or 'u'时,数组 a 的 k*k 部分用于存数三角矩阵的上三角部分， 严格下三角部分未被引用。当 uplo = 'L' or 'l'时,数组 a 的 k*k 部分用于存储 下三角矩阵的部分,严格上三角部分未被引用。
当 diag = 'U' or 'u'时,a 的对角线元素也未被引用,但是假定为单位 1。

lda　整数。数组 a 的第一维度大小。当 side = 'L' or 'l'时,lda 满足 max(1,m),当 side = 'R' or 'r'时,lda 满足 max(1,n)。

b　strsm 函数下为实数类型
dtrsm 函数下为双精度类型
ctrsm 函数下为复数类型
ztrsm 函数下为双精度复数类型
数组,规模为(ldb,n)m*n 数组 b 用于存储矩阵 B。

ldb　整数。定义数组 b 的第一维度大小。ldb 满足 max(1,m)。

输出参数

b　被变换矩阵结果覆写。

3.4 BLAS 错误信息提示

BLAS 模块会对接口的输入情况进行检查，如果输入参数出错，会打印出错的函数名和第几个参数出错。

4 LAPACK 模块

4.1 基本概况

4.1.1 LAPACK

LAPACK 是采用标准 Fortran77 和 C 语言编写的线性代数库，用来求解大多数常见的线性数值代数问题。LAPACK 是 Linear Algebra PACKagre 字母的缩写。

4.1.2 LAPACK 的用途

LAPACK 能够解决，如线性方程组求解，线性最小二乘问题求解，特征值和奇异值等问题。LAPACK 还可以实现矩阵分解和条件数的估计等相关计算。

LAPACK 包含驱动程序（driver）、计算程序（computational）、辅助程序（auxiliary）。驱动程序用于解决一个完整问题。计算程序用于执行一个单独的计算任务。辅助程序用于执行分块算法的子任务、完成低水平计算。驱动程序是调用一系列的计算程序。总体上来说，计算程序能够完成比驱动程序更为广泛的任务。许多辅助程序可能会被数值分析者或软件开发人员用到，因此我们为这些程序提供了同 LAPACK 程序与驱动程序一样详细的源代码文档。

LAPACK 提供了计算稠密矩阵和带状矩阵的功能，但是没有提供计算稀疏矩阵的功能。

在所有地方，对实数型和复数型数据，LAPACK 提供相同的计算功能。

4.1.3 LAPACK 软件包组成

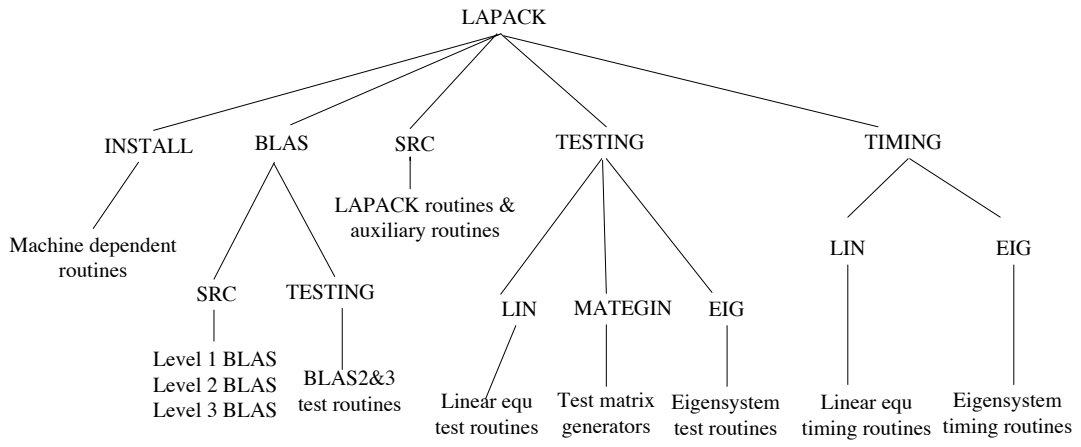

图4.1 LAPACK软件包的组成结构

我们在图 4.1 中给出了 LAPACK 软件包的组成结构，其中 SRC 是存放源程序代码的目录。LAPACK 软件包中 TESTING 子目录下的 LIN 子目录存放线性系统求解程序的测试程序正确性的源代码，EIG 子目录存放特征值问题求解程序的测试程序正确性的源代码，而 MAGTEN 子目录存放产生测试矩阵的源代码。TIMING 子目录下的 LIN 子目录存放线性系统求解程序

的测试程序性能的源代码，而 EIG 子目录存放特征值问题求解程序的测试程序性能的源代码。BLAS 子目录下的 SRC 子目录存放 BLAS 源程序代码，TESTING 子目录存放测试 BLAS 程序正确性的源代码。INSTALL 子目录存放安装 LAPACK 软件包所需的 make.inc，时间测试函数等文件。

4.1.4 LAPACK 的结构

4.1.4.1 程序的级别

LAPACK 中的子程序分类如下：

驱动程序，它们中的每一个都解决一个完整的问题，例如线性方程组的求解或者实对称矩阵的特征值的计算。推荐用户使用可以满足他们需要的驱动程序。计算程序，它们中的每一个都独立的解决一类计算任务，例如 LU 分解或将实对称矩阵规约成三对角形式。每一个驱动程序调用一系列的计算程序。用户（尤其是软件开发人员）可能直接的调用计算程序来完成任务，或者是一系列任务，这些任务不可能被驱动程序简单的完成。它们将在 4.2.4 节列出。

辅助程序,它们也可以依次分为如下三类:

- 完成分块算法的一些子任务的程序 - 尤其是完成算法非分块版本的那些程序;
- 完成一些公共的低级的计算程序，例如扩展矩阵， 计算矩阵范数,或者生成一个初等 Householder 矩阵； 数值分析员或者软件开发者有可能对其中的一些感兴趣，将来也考虑将这些加入到 BLAS 里面;
- 对 BLAS 的一些扩展, 例如实行复平面旋转的程序，或者关于复型对称矩阵的矩阵－向量相乘操作（严格意义上说 BLAS 本身并不是 LAPACK 中的一部分）。

对驱动程序和计算程序在本用户手册里都有详细描述，但没有辅助程序的详细描述。

4.1.4.2 数据类型和精度

LAPACK 为实型和复型提供了相同的功能。

对于大多数的计算，这里都有匹配程序，一个对应于实型，一个对应于复型，但是有一些例外。例如，对应于实对称不定线性方程系的程序，这里提供了复厄米特和复对称的程序，因为在实际应用中出现了这两类复型问题。然而，对实对称三对角矩阵找特征值的程序没有与之对应的复数类型，这里因为一个复型厄米特矩阵总可以规约成实对称三对角矩阵。

在大多数情况下，对于实型和复型匹配程序在编码上有很大的相关性。然而，在一些方面（尤其是非对称特征系问题）这种相关性是很弱的。

对于复型矩阵的双精度程序需要非标准的 Fortran 数据类型 COMPLEX*16，这对于经常使用双精度计算的大多数机器都是可用的。

4.1.4.3 命名机制

每一 LAPACK 程序的命名方法都是根据这个函数的说明编制的（在 Fortran 77 6 个字母名字的严格限制内）。

所有的驱动和计算程序都有相同形式的名字 **XYYZZZ**，对一些驱动程序第六个字母可能是空。

第一个字母，X，指出数据类型，含义如下：

S 实型

D 双精度

C 复型

Z COMPLEX*16 或者 双精度复型

一般当我们提到 LAPACK 程序，而不关心数据类型的时候，我们用“x” 代替第一个字母。所以 xGESV 指的是 SGESV, CGESV, DGESV 和 ZGESV 程序中的一个或者全部。

之后的两个字母，YY，指出矩阵类型（或是比较重要的矩阵特征）。这两个字母的编码大部分都应用到实型和复型矩阵；如表 4.1 列出。

表 4.1:在 LAPACK 命名方法中的矩阵类型

BD	双对角
DI	对角
GB	一般带状
GE	一般 (例如, 非对称,在一些情况下是矩形)
GG	一般矩阵, 一般性的问题(例如, 一对一般矩阵)
GT	一般三对角
HB	(复型) 厄米特带状
HE	(复型) 厄米特
HG	上 Hessenberg 矩阵, 一般性的问题(例如一个 Hessenberg 三角矩阵)
HP	(复型) 厄米特, 压缩存储
HS	上 Hessenberg
OP	(实型) 正交, 压缩存储
OR	(实型) 正交
PB	对称或者厄米特正定带状
PO	对称或者厄米特正定
PP	对称或者厄米特正定, 压缩存储
PT	对称或者厄米特正定三对角
SB	(实型) 对称带状
SP	对称,压缩存储
ST	(实型) 对称三对角

SY　　对称

TB　　三角带状

TG　　三角矩阵，一般性的问题（例如,一对三角矩阵）

TP　　三角，压缩存储

TR　　三角（或者在一些情况下 拟三角）

TZ　　梯形

UN　　（复型）单位

UP　　（复型）单位，压缩存储

若一类程序要对不同类型的矩阵完成相同的功能，可用“xyy”替换前三个字母。所以 xyySVX 指的是表 4.2 中列出的所有线性方程问题的专家驱动程序。

最后的三个字母指出所做的计算类型。例如，SGEBRD 是一个单精度程序，计算一个实型一般矩阵的双对角规约。

辅助程序的名字按照相似的命名方法编制的，仅仅是第二三个字母通常是 LA（例如，SLASCL 或者 CLARFG）。

这里只有两个例外。辅助程序实现的分块算法程序的非分块版本和分块算法的程序具有类似的名字，但第六个字母是“2”（例如 SGETF2 是分分块版本的 SGETRF）。一些程序可能被当作是 BLAS 的扩展，它们的名字是按照 BLAS 的命名策略（例如，CROT，CSTYR）。

4.1.5 驱动程序

这一部分描述 LAPACK 中的驱动程序。术语的详细描述和所实施的数值操作请参见 4.2.4 节。

4.1.5.1 线性方程

本库为解决线性方程问题提供了两种类型的驱动程序：

一类**简单**驱动(以 - SV 结尾)，通过分解 A 覆写 B，解方程 $AX = B$ 得到结果 X；

一类**专家**驱动(以 - SVX 结尾)，它们也可完成下列功能(有些是可选择的)：

- 解方程系 $A^T X = B$ or $A^H X = B$ (除非 A 是对称的或是厄米特矩阵)；
- 估计 A 的条件数， 检查它是否接近奇异，检查主元增长情况；
- 优化解决方案计算前向和后向边界误差 ；
- 如果 A 是不太容易扩展的，则平衡方程系。

专家驱动大约需要简单驱动两倍多的存储去完成额外的功能。

两种类型的驱动程序均可处理多个列向量所组成的右端项（矩阵 B 的各列）。

不同的驱动程序可以利用矩阵 A 的特殊性质或是存储机制，在表 4.2 列出。

这些驱动程序涵盖除矩阵求逆问题以外所有的线性问题的计算程序的功能。很少有需要去直接计算矩阵的逆，也不推荐用它作为解线性问题的一种方法。

表 4.2：线性方程的专家驱动

矩阵类型 和存储策略	操作	单精度		双精度	
		实型	复型	实型	复型
一般	简单驱动	SGESV	CGESV	DGESV	ZGESV
	专家驱动	SGESVX	CGESVX	DGESVX	ZGESVX
一般带状	简单驱动	SGBSV	CGBSV	DGBSV	ZGBSV
	专家驱动	SGBSVX	CGBSVX	DGBSVX	ZGBSVX
一般三对角	简单驱动	SGTSV	CGTSV	DGTSV	ZGTSV
	专家驱动	SGTSVX	CGTSVX	DGTSVX	ZGTSVX
对称/厄米特	简单驱动	SPOSV	CPOSV	DPOSV	ZPOSV
正定	专家驱动	SPOSVX	CPOSVX	DPOSVX	ZPOSVX
对称/厄米特	简单驱动	SPPSV	CPPSV	DPPSV	ZPPSV
正定(压缩存储)	专家驱动	SPPSVX	CPPSVX	DPPSVX	ZPPSVX
对称/厄米特	简单驱动	SPBSV	CPBSV	DPBSV	ZPBSV
正定带状	专家驱动	SPBSVX	CPBSVX	DPBSVX	ZPBSVX
对称/厄米特	简单驱动	SPTSV	CPTSV	DPTSV	ZPTSV
正定三对角	专家驱动	SPTSVX	CPTSVX	DPTSVX	ZPTSVX
对称/厄米特	简单驱动	SSYSV	CHESV	DSYSV	ZHESV
不定	专家驱动	SSYSVX	CHESVX	DSYSVX	ZHESVX
复对称	简单驱动		CSYSV		ZSYSV
	专家驱动		CSYSVX		ZSYSVX
对称/厄米特	简单驱动	SSPSV	CHPSV	DSPSV	ZHPSV
不定（压缩存储）	专家驱动	SSPSVX	CHPSVX	DSPSVX	ZHPSVX
复对称	简单驱动		CSPSV		ZSPSV
(压缩存储)	专家驱动		CSPSVX		ZSPSVX

4.1.5.2 线性最小二乘问题

线性最小二乘问题是：

$$\underset{x}{\text{minimize}} \ \|b - Ax\|_2 \tag{4.1}$$

这里 A 是一个 m×n 矩阵，b 是一个给定的 m 个元素的向量，x 是 n 个元素的解向量。在大多数情况下,m≥n 和 rank（A）＝n，在这种情况下问题（4.1）的解是唯一的，并且

这类问题也被称作为超定定线性方程问题寻找最小二乘解。

当 $m < n$ 和 rank（A）＝m 时，这里会有很多满足 $b-Ax=0$ 的 x，解就会有无限个。在这种情况下，我们通常会找出满足最小的$|x|_2$的唯一的解 x，这类问题被称作为欠定线性方程问题寻找最小范数解。

驱动程序 xGELS 解决问题（4.1），假定 rank（A）＝min（m，n），即 A 是满秩的。当 $m > n$ 时找到超定问题的最小二乘解，当 $m < n$ 时找到欠定问题的最小范数解。xGELS 用到 A 的 QR 或 LQ 分解，并且在问题陈述中允许 A 被替换为 A^T（或者 A^H，如果 A 是复型的）。

在通常的情况下，我们可能遇到 rank（A）<min（m，n），即 A 可能亏秩。我们寻找最小范数最小二乘解 x，x 将具有最小的$|x|_2$和$\|b-Ax\|_2$。

驱动程序 xGELSX, xGELSY, xGELSS, and xGELSD 解决问题(4.1)的一般形式，允许矩阵 A 是亏秩的；xGELSX 和 xGELSY 用到 A 的完全正交分解，xGELSS 用到 A 的奇异值分解，xGELSD 用到 A 的基于分治策略的奇异值分解算法。

子程序 xGELSF 是 xGELSX 的一个快速版本，但因为它调用分块算法完成完全正交分解所以需要更多的工作空间。xGELSX 是为了与 LAPACK2.0 版本兼容而保留，但是在这个用户手册中我们省略了这个程序的说明。

子程序 xGELSD 会比对应的旧的 xGELSS 明显的快，尤其是对较大问题的时候，但是可能需要更多的工作空间，这得依赖于矩阵维数。

表 4.3 列出了 LLS 驱动程序。

在一个简单的调用里面，这四个程序都允许处理右端向量 b 和相对应的解 x，把这些向量作为 B 和 X 的列单独存起来。注意不管怎样解问题 4.1，每一个右端向量是独立的；这与找一个矩阵 x，最小化$\|b-Ax\|_2$是不同的。

表 4.3：线性最小二乘问题的驱动程序

操作	单精度		双精度	
	实型	复型	实型	复型
解 LLS 用 *QR* 或 *LQ* 分解	SGELS	CGELS	DGELS	ZGELS
解 LLS 用完全正交分解	SGELSY	CGELSY	DGELSY	ZGELSY
解 LLS 用 SVD	SGELSS	CGELSS	DGELSS	ZGELSS
解 LLS 用分治策略的 SVD	SGELSD	CGELSD	DGELSD	ZGELSD

4.1.5.3 广义最小二乘问题

这里提供两种类型的广义最小二乘问题的驱动程序。

第一类是

$$\min_x \|c-Ax\|_2 \quad \text{subject to} \quad Bx=d \tag{4.2}$$

A 是一个 m×n 矩阵，B 是一个 p×n 矩阵，c 是一个给定的 m－向量，d 是一个给定的 p－向量，并且 p≤n≤m＋p。这叫作线性等式约束的最小二乘问题（LSE）。xGGLSE 程序用

广义的 RQ（GRQ）分解解决这类问题，并且假设 B 为行满秩(ran(B)=p)，矩阵列满秩。在这种假设条件下，LSE 问题有唯一解。

$$\min_x \|B^{-1}(d-Ax)\|_2$$

第二类广义最小二乘问题是

$$\min_x \|y\|_2 \quad \text{subject to} \quad d=Ax+By \tag{4.3}$$

这类 A 是一个 n×m 矩阵，B 是一个 n×p 矩阵，d 是一个给定的 n－向量，并且 m≤n≤m＋p。 这个有时候被称作 **广义(Gauss-Markov)线性模型问题**。当 $B=I$ 时，这个问题变成一个普通的最小二乘问题。当 B 为非奇异方阵时，GLM 问题等价于 **加权最小二乘问题** ：

$$\min_x \|B^{-1}(d-Ax)\|_2$$

xGGGLM 程序应用广义 QR（GQR）分解解决这里问题，这里假设设 A 为列满秩，矩阵（A，B）为行满秩。在这个假设条件下，这个问题是相容的，并且有唯一的解 x 和 y。广义最小二乘问题的驱动程序在表 4.4 中列出。

表 4.4: 广义最小二乘问题的驱动程序

操作	单精度		双精度	
	实型	复型	实型	复型
用 GRQ 解 LSE 问题	SGGLSE	CGGLSE	DGGLSE	ZGGLSE
用 GQR 解 GLM 问题	SGGGLM	CGGGLM	DGGGLM	ZGGGLM

4.1.5.4 标准特征值和奇异值问题

4.1.5.4.1 对称特征值问题（SEP）

对称特征值问题是求特征值，λ，和相应的特征向量，z≠0，

$$Az=\lambda z,\ A=A^T,\ A \text{ 是实型。}$$

对厄米特特征值问题我们有

$$Az=\lambda z,\ A=A^H$$

这两类问题特征值都是实型。

当我们计算得到了所有的特征值和特征向量，我们写成：

$$A=Z\Lambda Z^T$$

这里Λ是对角线元素都是特征值的对角矩阵，Z 是一个各个列都是特征向量的正交（或酉）矩阵。这就是经典的 A 的谱分解问题。

对于对称和厄米特特征系问题这里有如下四种类型的驱动程序。提供不同的驱动是为了更好的利用矩阵 A 的特殊的数据结构和存储方法，如表 4.5 列出的。

- 一类**简单**驱动 （以-EV 结尾）计算所有的特征值和（可以选择的）特征向量。

- 一类**专家**驱动 （以 –EVX 结尾）计算所有的或选择的一部分特征值和（可以选择的）特征向量。如果仅需要一小部分特征值或特征向量那么专家驱动会比简单驱动快。

- 一类**分治**驱动 （以 –EVD 结尾）解决类似简单驱动的问题。对于较大的矩阵会比简单驱动快的多，但是需要更多的工作空间。分治的名字指下面的算法（请看 4.2.4.4 节和 4.3.4.3 节）。

- 一类**相对健壮的描述**的驱动（RRR）(以 –EVR 结尾）计算所有或（稍后的版本）一部分特征值和（可以选择的）特征向量。这是在所有驱动中最快的算法（除了很少的一部分情况以外），并且用了最少的工作空间。RRR 的名字指下面的算法（请看 4.2.4.4 节和 4.3.4.3 节）。

4.1.5.4.2 非对称特征系问题（NEP）

非对称特征值问题是求特征值，λ，和相应的特征向量，v≠0，

$$Av = \lambda v。$$

一个实矩阵 A 可能会有复特征值，可能会出现一对复共扼特征值。更准确的说，应该叫向量 x 叫 A 的右特征向量，一个向量 u≠0 满足

$$u^H A = \lambda u^H$$

叫 A 的左特征向量。

这个问题可以通过 A 的 Schur 分解解决，在实型情况下定义如下

$$A = ZTZ^T,$$

这里 Z 是一个正交矩阵，T 是一个 1×1 和 2×2 对角分块的上拟对角矩阵，2×2 分块对应 A 的一对共扼特征值。在复型情况下 Schur 分解是

$$A = ZTZ^H,$$

这里 Z 是酉矩阵，T 是复型上三角矩阵。

Z 的列被称为 **Schur 向量**。对每一个 k（1≤k≤n），Z 的前 k 列构成对应 T 的对角上的前 k 个特征值的不变子空间的标准正交基。因为这个基是标准正交的，在许多实际应用中我们更倾向于计算 Schur 向量而不是特征向量。我们更可以定制 Schur 分解以使任意需要的 k 个特征值出现 T 对角上的前 k 个位置。

这里提供了两对驱动，一对针对 Schur 分解，另一对针对表 4.5 列出的特征值和特征向量：

- xGEES：一个简单驱动 计算 A 的所有或部分 Schur 分解可以选择重新排列特征值；

- xGEESX：一个专家驱动可以额外计算出一个选定的特征值子集平均数的条件数,还有对应的右不变子空间；

- xGEEV：一个简单驱动计算 A 的所有特征值，还有（可以选择的）左或右特征向量（或者两个都计算）；

- xGEEVX：一个专家驱动可以额外的平衡矩阵改善特征值和特征向量的调理，并且对特征值或右特征向量（或者两个都）计算条件数。

4.1.5.4.3 奇异值分解（SVD）

奇异值分解问题是对一个给定的 m×n 矩阵给出如下形式：

A＝UΣVT，（A＝UΣVT 是复型）

这里 U 和 V 是正交（酉）的，Σ是 m×n 对角矩阵，有实型对角元素，σi，并且使

$$\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_{\min(m,n)} \geq 0.$$

σi 是 A 的奇异值，U 和 V 的第一个 min（m，n）列是 A 的左和右奇异向量。

奇异值和奇异向量满足：

$$Av_i = \sigma_i u_i \quad \text{and} \quad A^T u_i = \sigma_i v_i \quad (\text{or} \quad A^H u_i = \sigma_i v_i \quad)$$

这里 u_i 和 v_i 分别是 U 和 V 的第 i 列。

这里对 SVD 问题有两种类型的驱动。原先的 LAPACK 只有下面描述的简单驱动，另外一个是改良算法的时候添加的。

- 一个**简单**驱动 xGESVD 计算所有的奇异值和（可以选择的）左和/或右奇异向量。

- 一个**分治**驱动 xGESDD 解决类似简单驱动的问题。对于较大的矩阵这个驱动更快，但是需要更多的工作空间。分治的名字指下面的算法（请看 4.2.4.4 节 和 4.3.4.3 节）.

表 4.5：标准特征值和奇异值问题的驱动程序

类型问题	功能和存储策略	单精度		双精度	
		实型	复型	实型	复型
SEP	简单驱动	SSYEV	CHEEV	DSYEV	ZHEEV
	分治驱动	SSYEVD	CHEEVD	DSYEVD	ZHEEVD
	专家驱动	SSYEVX	CHEEVX	DSYEVX	ZHEEVX
	RRR driver	SSYEVR	CHEEVR	DSYEVR	ZHEEVR
	简单驱动(压缩存储)	SSPEV	CHPEV	DSPEV	ZHPEV
	分治驱动(压缩存储)	SSPEVD	CHPEVD	DSPEVD	ZHPEVD
	专家驱动(压缩存储)	SSPEVX	CHPEVX	DSPEVX	ZHPEVX
	简单驱动(带状矩阵)	SSBEV	CHBEV	DSBEV	ZHBEV
	分治驱动（带状矩阵）	SSBEVD	CHBEVD	DSBEVD	ZHBEVD
	专家驱动(带状矩阵)	SSBEVX	CHBEVX	DSBEVX	ZHBEVX

	简单驱动(三对角矩阵)	SSTEV		DSTEV	
	分治驱动（三对角矩阵）	SSTEVD		DSTEVD	
	专家驱动(三对角矩阵)	SSTEVX		DSTEVX	
	RRR 驱动 (三对角矩阵)	SSTEVR		DSTEVR	
NEP	Schur 分解的简单驱动	SGEES	CGEES	DGEES	ZGEES
	Schur 分解的专家驱动	SGEESX	CGEESX	DGEESX	ZGEESX
	计算特征值/向量的简单驱动	SGEEV	CGEEV	DGEEV	ZGEEV
	计算特征值/向量的专家驱动	SGEEVX	CGEEVX	DGEEVX	ZGEEVX
SVD	简单驱动	SGESVD	CGESVD	DGESVD	ZGESVD
	分治驱动	SGESDD	CGESDD	DGESDD	ZGESDD

4.1.5.5 广义特征值和奇异值问题

4.1.5.5.1 广义对称正定特征系问题（GSEP）

这里提供计算以下三类问题的所有特征值和（可以选择的）特征向量：

1. $Az = \lambda Bz$
2. $ABz = \lambda z$
3. $BAz = \lambda z$

A 和 B 是对称或厄米特矩阵，B 是正定的。 所有这些问题的特征值 λ 都是实型的。 计算出的特征向量得到的矩阵 Z 满足 $Z^T AZ = \Lambda$（第 1，3 类问题）或者 $Z^{-1} A Z^T = I$(第 2 类问题)， Λ是对角元素都是特征值的对角矩阵。Z 还满足 $Z^T B Z = I$ （第 1，2 类问题）或 $Z^T B^{-1} Z = I$ （第 3 类问题）。

对广义对称和 厄米特特征系问题这里提供了三种类型的驱动。原先的 LAPACK 只提供了下面描述的简单和专家驱动，另外一类是在改良算法的时候添加的。

- 一类**简单**驱动 （以-GV 结尾）计算所有的特征值和（可以选择的）特征向量。
- 一类**专家**驱动 （以 -GVX 结尾）计算所有的或选择的一部分特征值和（可以选择的）特征向量。如果仅需要一小部分特征值或特征向量那么专家驱动会比简单驱动快。
- 一类分治驱动 （以 -GVD 结尾）解决类似简单驱动的问题。对于较大的矩阵会比简单驱动快的多，但是需要更多的工作空间。分治的名字指下面的算法（请看 4.2.4.4 节和 4.3.4.3 节）。

不同类型的驱动是为了更好的利用矩阵 A 和 B 的特殊的数据结构和存储机制，在表 4.6 中列出。

4.1.5.5.2 广义非对称特征问题（GNEP）

给定一对矩阵（A，B），A 和 B 是 n×n 方阵，**广义非对称特征系问题**就是求特征值 λ 和对应的特征向量 **x≠0 以使**

$$Ax = \lambda Bx,$$

或者是求特征值 μ 和对应的特征向量 y≠0 以使

$$\mu Ay = By.$$

注意如果 λ 和 μ 不等于 0 的时候在 μ=1/λ 和 $x=y$ 的时候这些问题是等价的。为了处理 λ 或 μ 等于 0 的情况或者接近于 0 的情况，LAPACK 程序返回两个值，α 和 β，对每一个特征值，满足 λ=α/β 和 μ=β/α。

更准确的说，x 和 y 被称为 **右特征向量。**向量 u≠0 或 v≠0 满足

$$u^H A = \lambda u^H B \quad \text{or} \quad \mu v^H A = v^H B$$

被称为**左特征向量**。

有些时候下面的等价的定义也用来指广义特征问题，对矩阵对（A，B）：A-B,这里 λ 是不定的，被称为矩阵束，或者只叫线束。所以我们可以说线束 A-B 的广义特征值和特征向量。

如果具有决定性的 A-λB 恰好对于所有的 λ 值等于 0，那么特征值问题被称为奇异的；否则称作正则的。（A，B）的奇异性以 α=β=0 （由于舍入，α 和 β 可能非常小）为标示。在这种情况下，特征值问题是病态的，事实上一些其他的非 0 的 α 和 β 值可能是不确定的[1,2,3]。LAPACK 的当前程序都只针对正则矩阵束。

广义非对称特征值问题可以通过矩阵对（A，B）的广义 Schur 分解来解决，在实型类型时定义如下

$$A = QSZ^T, \quad B = QTZ^T$$

Q 和 Z 是正交矩阵，T 是上三角矩阵，S 是对角为 1×1 和 2×2 分块的上拟三角矩阵，并且 2×2 分块对应（A，B）的特征值的复型共扼对。在复型类型的情况下，广义 Schur 分解是

$$A = QSZ^T, \quad B = QTZ^T$$

Q 和 Z 的列被称为**左和右广义 Schur 向量**，和 A 和 B 压缩子空间的扩展对。压缩子空间是一种广义的不变子空间：对每一个 k (1≤k≤n)，Z 的前 k 列扩展的右压缩子空间被 A 和 B 映射为 Q 的前 k 列扩展的左子空间。

更正式的，让 $Q = (Q_1, Q_2)$，$Z = (Z_1, Z_2)$为关于（S，T）的（1，1）一分块的 k 个特征值的集合的保角分划，举个例子，这里 Q_1 和 Z_1 都有 k 列，S_{11} 和 T_{11}都是 k×k 的，

$$\begin{pmatrix} Q_1^H \\ Q_2^H \end{pmatrix} (A-\lambda B) \begin{pmatrix} Z_1 & Z_2 \end{pmatrix} = S-\lambda T \equiv \begin{pmatrix} S_{11} & S_{12} \\ 0 & S_{22} \end{pmatrix} - \lambda \begin{pmatrix} T_{11} & T_{12} \\ 0 & T_{22} \end{pmatrix}.$$

那么子空间 $\mathcal{L} = \mathrm{span}(Q_1)$ 和 $\mathcal{R} = \mathrm{span}(Z_1)$ 构成一个关于 (S_{11}, T_{11}) 集合的压缩

子空间对（左和右），满足 $\mathcal{L} = A\mathcal{R} + B\mathcal{R}$ 和 $\dim(\mathcal{L}) = \dim(\mathcal{R})$ [4]。可以从（S，T）的 n 个特征值集合中定制广义 Schur 形式以使(S_{11}, T_{11})有任意需要的 k 个特征值的子集。

对于标准非对称特征问题，提供了两对驱动，一对针对广义 Schur 分解，另外一对针对表 4.6 列出的特征值和特征向量：

- xGEES：一类简单驱动 计算（A，B） 的所有或部分广义 Schur 分解，可以有选择的定制特征值；
- xGGESX：一类专家驱动可以额外计算出一个选择的特征值子集平均数的条件数,还有对应压缩子空间对；
- xGGEV：一类简单驱动计算（A，B）的所有广义特征值，还有（可以选择的）左或右特征向量（或者两个都计算）；
- xGEEVX：一类专家驱动可以额外的平衡矩阵改善特征值和特征向量的调理，并且对特征值或右特征向量（或者两个都）计算条件数。

在表 4.6 中为了节省空间在 Schur 分解，特征值/向量和奇异值/向量的前面省略了“广义”。

子程序 xGGES 和 xGGEV 分别是驱动程序 xGGES 和 xGGEV 的提高版本。xGEGS 和 xGEGV 为了保持和 LAPACK2.0 的兼容性仍然保留，但是在用户手册中略去了这些程序的参考说明。

4.1.5.5.3 广义奇异值分解（GSVD）

m×n 矩阵 A，p×n 矩阵 B 的**广义（或商）奇异值分解**给出如下分解对

$$A = U\Sigma_1[0,\ R]Q^T \quad \text{and} \quad B = V\Sigma_2[0,\ R]Q^T .$$

这类分解的矩阵有如下性质：

- U 是 m×m 的，V 是 p×p 的，Q 是 n×n 的，这三个矩阵都是正交的。如果 A 和 B 是复型的，这些矩阵就是酉矩阵而不是正交矩阵，并且 Q^T 在上面的分解对中应该被 Q^H 取代。
- R 是 $r\times r$ 的，上三角和非奇异。$[0,R]$ 是 $r\times n$（换句话， 0 是一个 $r\times n-r$ 0 矩阵）。 整数 r 是的秩，并且满足 $\boldsymbol{r \le n}$.
- $\sum_1$ 是 $m\times r$ 的，$\sum_2$ 是 $p\times r$ 的， 都是实型，非负，对角并且 $\Sigma_1^T\Sigma_1 + \Sigma_2^T\Sigma_2 = I$。得到 $\Sigma_1^T\Sigma_1 = \mathrm{diag}(\alpha_1^2,\ldots,\alpha_r^2)$ 和 $\Sigma_2^T\Sigma_2 = \mathrm{diag}(\beta_1^2,\ldots,\beta_r^2)$，这里 α_i 和 β_i 是介于 0 和 1 之间的数。比率

α_i/β_i叫做 A，B 对的广义奇异值。如果$\beta_i=0$，那么广义奇异值α_i/β_i是 **无穷大**.

$\sum_1$ 和$\sum_2$ 有以下详细的数据结构，这得依赖于是$m-r\geq 0$还是 $m-r<0$。在前面的情况下，$m-r\geq 0$，那么

$$\Sigma_1 = \begin{array}{c} \\ k \\ l \\ m-k-l \end{array}\begin{array}{c} \begin{array}{cc} k & l \end{array} \\ \begin{pmatrix} I & 0 \\ 0 & C \\ 0 & 0 \end{pmatrix} \end{array} \quad \text{and} \quad \Sigma_2 = \begin{array}{c} \\ l \\ p-l \end{array}\begin{array}{c} \begin{array}{cc} k & l \end{array} \\ \begin{pmatrix} 0 & S \\ 0 & 0 \end{pmatrix} \end{array}.$$

这里 l 是 B的秩，$k=r-l$，C 和 S 是对角矩阵满足 $C^2 + S^2 = I$，并且 S 是非奇异的。第二类情况是当 $m-r<0$ 时，

$$\Sigma_1 = \begin{array}{c} \\ k \\ m-k \end{array}\begin{array}{c} \begin{array}{ccc} k & m-k & k+l-m \end{array} \\ \begin{pmatrix} I & 0 & 0 \\ 0 & C & 0 \end{pmatrix} \end{array}$$

和

$$\Sigma_2 = \begin{array}{c} \\ m-k \\ k+l-m \\ p-l \end{array}\begin{array}{c} \begin{array}{ccc} k & m-k & k+l-m \end{array} \\ \begin{pmatrix} 0 & S & 0 \\ 0 & 0 & I \\ 0 & 0 & 0 \end{pmatrix} \end{array}.$$

一样的，l 时 B 的秩，$k=r-l$，C 和 S 是对角矩阵满足 $C^2 + S^2 = I$，S 是非奇异的，前 k 个广义奇异值是无穷大的，余下的 l 个广义奇异值是确定的。

这里有一些广义奇异值分解的比较重要的特殊情况。首先，如果 B 是方阵和奇异的，那么 r＝n 并且 A 和 B 的广义奇异值分解等同于 AB^{-1}的的奇异值分解，这里 AB^{-1}的奇异值与 A，B 对的广义奇异值相等：

$$AB^{-1} = (U\Sigma_1 RQ^T)(V\Sigma_2 RQ^T)^{-1} = U(\Sigma_1\Sigma_2^{-1})V^T \ .$$

$$\begin{pmatrix} A \\ B \end{pmatrix} = \begin{pmatrix} U & 0 \\ 0 & V \end{pmatrix}\begin{pmatrix} \Sigma_1 \\ \Sigma_2 \end{pmatrix} Q^T \ .$$

第二，如果$(A^T\ B^T)^T$ 的列是标准正交的，那么 $r=n$，$R=I$ 并且 A 和 B 的广义奇异值分解等同于$(A^T\ B^T)^T$ 的 CS（余弦-正弦)分解[5]：

$$\begin{pmatrix} A \\ B \end{pmatrix} = \begin{pmatrix} U & 0 \\ 0 & V \end{pmatrix}\begin{pmatrix} \Sigma_1 \\ \Sigma_2 \end{pmatrix} Q^T \ .$$

第三， $A^TA-\lambda B^TB$ 的广义特征值和特征向量可以用奇异值分解来表示。假设

$$X=Q\begin{pmatrix} I & 0 \\ 0 & R^{-1} \end{pmatrix} .$$

那么

$$X^TA^TAX=\begin{pmatrix} 0 & 0 \\ 0 & \Sigma_1^T\Sigma_1 \end{pmatrix} \text{ and } X^TB^TBX=\begin{pmatrix} 0 & 0 \\ 0 & \Sigma_2^T\Sigma_2 \end{pmatrix}.$$

所以，X 的列就是 $A^TA-\lambda B^TB$ 的特征向量，并且“非平凡”特征值就是广义奇异值的平方（参见 4.2.3.5.1 节)。“平凡”特征值就是那些对应的 X 的前 n－r 列，它们跨越 AT A 和 BT B 的公共零空间。“平凡特征值”并没有更好的定义 4.1。

一个简单驱动 xGGSVD 计算 A 和 B 的广义奇异值分解（如表 4.6)。这个方法基于[4,5,6]中描述的方法。

表 4.6: 广义特征值和奇异值问题的驱动程序

问题类型	功能和存储机制	单精度		双精度	
		实型	复型	实型	复型
GSEP	简单驱动	SSYGV	CHEGV	DSYGV	ZHEGV
	分治驱动	SSYGVD	CHEGVD	DSYGVD	ZHEGVD
	专家驱动	SSYGVX	CHEGVX	DSYGVX	ZHEGVX
	简单驱动(压缩存储)	SSPGV	CHPGV	DSPGV	ZHPGV
	分治驱动	SSPGVD	CHPGVD	DSPGVD	ZHPGVD
	专家驱动	SSPGVX	CHPGVX	DSPGVX	ZHPGVX
	简单驱动(带状矩阵)	SSBGV	CHBGV	DSBGV	ZHBGV
	分治驱动	SSBGVD	CHBGVD	DSBGV	ZHBGVD
	专家驱动	SSBGVX	CHBGVX	DSBGVX	ZHBGVX
GNEP	Schur 分解的简单驱动	SGGES	CGGES	DGGES	ZGGES
	Schur 分解的专家驱动	SGGESX	CGGESX	DGGESX	ZGGESX
	特征值/向量的简单驱动	SGGEV	CGGEV	DGGEV	ZGGEV
	特征值/向量的专家驱动	SGGEVX	CGGEVX	DGGEVX	ZGGEVX
GSVD	奇异值/向量	SGGSVD	CGGSVD	DGGSVD	ZGGSVD

4.1.6 计算程序

4.1.6.1 线性方程组

我们使用标准的记法表示一个线性方程组：

$$A\,x = b \tag{4.4}$$

其中，A 为一个系数矩阵，b 为右端项，x 是解向量。在(4.4)中，A 被假定为 n 阶方阵，但是一些个别的程序允许 A 为长方形矩阵。如果有不止一个右端项，我们记做：

$$A\,X = B \tag{4.5}$$

其中，B 的列是一系列相互独立的右端项，X 的各列为相应的解。基本的任务是在给定 A 和 B 的情况下，计算矩阵 X。

如果 A 是上（下）三角阵，(4.4)可以通过一个向前或向后置换的过程直接求解。否则，需要首先对 A 进行分解，使之成为上（下）三角矩阵（也可能是对角阵或置换阵），然后再求解。

矩阵 A 分解的形式取决于它的性质。LAPACK 基于分解的几种情况，提供了以下类型矩阵的程序：

- 一般矩阵（部分选主元的 LU 分解）：

  A＝PLU
  其中，P 是一个置换矩阵，L 是单位下三角阵（如果 m>n，则为下梯形阵），U 是上三角阵（如果 m<n，则为上梯形阵）。

- 一般带状矩阵，包括三对角矩阵（部分选主元的 LU 分解）：

  如果 A 是 m×n 矩阵，且下带宽为 kl，上带宽为 ku，那么，分解形式为：
  A＝LU
  其中，L 是一个下带宽为 kl 的单位下三角矩阵与置换矩阵的乘积，而 U 是一个上带宽为 kl＋ku 的上三角矩阵。

- 对称正定矩阵或厄米特矩阵，包括带状矩阵（乔累斯基分解）：

  $A=U^TU$ 或 $A=LL^T$（矩阵为对称矩阵）
  $A=U^HU$ 或 $A=LL^H$（矩阵为厄米特矩阵）
  其中，U 为一个上三角矩阵，L 为一个下三角矩阵。

- 对称正定三对角矩阵或厄米特三对角矩阵（LDL^T 分解）：

  $A=UDU^T$ 或 $A=LDL^T$（矩阵为对称矩阵）
  $A=UDU^H$ 或 $A=LDL^H$（矩阵为厄米特矩阵）
  其中，U 是一个单位上双对角矩阵，L 是单位下双对角矩阵，D 是对角矩阵。

- 对称非定矩阵或厄米特非定矩阵（对称非定分解）：

$A=UDU^T$ 或 $A=LDL^T$（矩阵为对称矩阵）

$A=UDU^H$ 或 $A=LDL^H$（矩阵为厄米特矩阵）

其中，U（或 L）是单位上（或下）三角矩阵与置换矩阵的乘积，D 为对称块对角矩阵，其中子块为一阶或二阶（块大小为 1 或 2）。

一般三对角矩阵的分解等同于 kl＝1 且 ku＝1 的带状矩阵的分解。上带宽（下带宽）为 k 的对称正定带状矩阵的分解形式与对称正定矩阵相同，只不过分解的矩阵 U（或 L）是一个上带宽（下带宽）为 k 的带状矩阵。带状矩阵使用 4.3.3.3 节所描述的压缩存储机制。LAPACK 也提供了程序实现如 4.3.3.2 节所描述的，对称矩阵（无论是正定还是非定）的压缩存储。

矩阵分解的主要应用是解决方程组问题，同时也包括一些相关的任务。无论如何，LAPACK 提供了各类型的矩阵和各存储格式的实现的程序（详见表 4.7 和 4.8）。下面列表将功能与相关计算程序的名称前 3 字符联系起来：

xyyTRF:

分解（不明确要求三角矩阵）

xyyTRS:

利用分解的结果（如果 A 自身为三角阵，则直接利用其自身），通过正向或逆向变换解决(2.5)问题。

xyyCON:

估计条件数；通常使用 Higham 修正版的 Hager 方法估计条件数，除了对称正定三对角矩阵可以直接估计，而达到相当的速度。

xyyRFS:

计算求解过程的误差界（由 xyyTRS 程序返回），并修正解以减少向后的误差（详见下面）。

xyyTRI:

利用分解的结果（如果 A 自身为三角阵，则直接利用其自身），计算 A 的逆（不提供带状矩阵求逆的程序，因为带状阵的逆不一定还是带状阵）；

xyyEQU:

计算比例因子以平衡 A（不提供三对角矩阵，对称非定矩阵和三角矩阵的此类程序），此程序实际上并不量度矩阵，这项工作可由辅助程序 xLAQyy 完成——可参见驱动程序 xyySVX 的代码作为使用样例。

下面注明了上面一些程序对其他程序的依赖关系：

xyyTRF:

如果 yy 属于集合{GE, GB, PO, PP, PB}，则可能对 xyyEQU 和 xLAQyy 已平衡好的矩阵进行处理。

xyyTRS:

需要 xyyTRF 的分解结果;。

xyyCON:

需要得到原始矩阵 A 的范数，以及 xyyTRF 返回的分解结果。

xyyRFS:

需要得到原始矩阵 A 和 B，xyyTRF 的分解结果和 xyyTRS 的解 X。

xyyTRI:

需要 xyyTRF 的分解结果。

RFS（“改进解法”）程序实现迭代改进，并计算求解过程向前和向后的误差界。迭代改进是在与输入结果相同的精度下进行的。值得注意的是，和传统做法一样，剩余的部分不会由额外的精度计算出。这样做的好处会在 4.2.4.4 节中讨论。

表 4.7：线性方程组的计算程序

矩阵类型和存储机制	操作	单精度		双精度	
		实型	复型	实型	复型
一般矩阵	分解	SGETRF	CGETRF	DGETRF	ZGETRF
	使用分解解方程组	SGETRS	CGETRS	DGETRS	ZGETRS
	求条件数	SGECON	CGECON	DGECON	ZGECON
	求误差界	SGERFS	CGERFS	DGERFS	ZGERFS
	利用分解求逆	SGETRI	CGETRI	DGETRI	ZGETRI
	平衡	SGEEQU	CGEEQU	DGEEQU	ZGEEQU
一般带状矩阵	分解	SGBTRF	CGBTRF	DGBTRF	ZGBTRF
	使用分解解方程组	SGBTRS	CGBTRS	DGBTRS	ZGBTRS
	求条件数	SGBCON	CGBCON	DGBCON	ZGBCON
	求误差界	SGBRFS	CGBRFS	DGBRFS	ZGBRFS
	平衡	SGBEQU	CGBEQU	DGBEQU	ZGBEQU
一般三对角矩阵	分解	SGTTRF	CGTTRF	DGTTRF	ZGTTRF
	使用分解解方程组	SGTTRS	CGTTRS	DGTTRS	ZGTTRS
	求条件数	SGTCON	CGTCON	DGTCON	ZGTCON
	求误差界	SGTRFS	CGTRFS	DGTRFS	ZGTRFS
对称正定矩阵或厄米特正定矩阵	分解	SPOTRF	CPOTRF	DPOTRF	ZPOTRF
	使用分解解方程组	SPOTRS	CPOTRS	DPOTRS	ZPOTRS
	求条件数	SPOCON	CPOCON	DPOCON	ZPOCON
		SPORFS	CPORFS	DPORFS	ZPORFS

	求误差界				
	使用分解求逆	SPOTRI	CPOTRI	DPOTRI	ZPOTRI
	平衡	SPOEQU	CPOEQU	DPOEQU	ZPOEQU
对称正定矩阵或厄米特正定矩阵（压缩存储）	分解	SPPTRF	CPPTRF	DPPTRF	ZPPTRF
	使用分解解方程组	SPPTRS	CPPTRS	DPPTRS	ZPPTRS
	求条件数	SPPCON	CPPCON	DPPCON	ZPPCON
	求误差界	SPPRFS	CPPRFS	DPPRFS	ZPPRFS
	使用分解求逆	SPPTRI	CPPTRI	DPPTRI	ZPPTRI
	平衡	SPPEQU	CPPEQU	DPPEQU	ZPPEQU
对称正定带状矩阵或厄米特正定带状矩阵	分解	SPBTRF	CPBTRF	DPBTRF	ZPBTRF
	使用分解解方程组	SPBTRS	CPBTRS	DPBTRS	ZPBTRS
	求条件数	SPBCON	CPBCON	DPBCON	ZPBCON
	求误差界	SPBRFS	CPBRFS	DPBRFS	ZPBRFS
	平衡	SPBEQU	CPBEQU	DPBEQU	ZPBEQU
对称正定三对角矩阵或厄米特正定三对角矩阵	分解	SPTTRF	CPTTRF	DPTTRF	ZPTTRF
	使用分解解方程组	SPTTRS	CPTTRS	DPTTRS	ZPTTRS
	求条件数	SPTCON	CPTCON	DPTCON	ZPTCON
	求误差界	SPTRFS	CPTRFS	DPTRFS	ZPTRFS

表 4.8：线性方程组的计算程序（续）

矩阵类型和存储机制	操作	单精度		双精度	
		实型	复型	实型	复型
对称非定矩阵或厄米特非定矩阵	分解	SSYTRF	CHETRF	DSYTRF	ZHETRF
	使用分解解方程	SSYTRS	CHETRS	DSYTRS	ZHETRS

	组				
	求条件数	SSYCON	CHECON	DSYCON	ZHECON
	求误差界	SSYRFS	CHERFS	DSYRFS	ZHERFS
	使用分解求逆	SSYTRI	CHETRI	DSYTRI	ZHETRI
复对称矩阵	分解		CSYTRF		ZSYTRF
	使用分解解方程组		CSYTRS		ZSYTRS
	求条件数		CSYCON		ZSYCON
	求误差界		CSYRFS		ZSYRFS
	使用分解求逆		CSYTRI		ZSYTRI
对称非定矩阵或厄米特非定矩阵（压缩存储）	分解	SSPTRF	CHPTRF	DSPTRF	ZHPTRF
	使用分解解方程组	SSPTRS	CHPTRS	DSPTRS	ZHPTRS
	求条件数	SSPCON	CHPCON	DSPCON	ZHPCON
	求误差界	SSPRFS	CHPRFS	DSPRFS	ZHPRFS
	使用分解求逆	SSPTRI	CHPTRI	DSPTRI	ZHPTRI
复对称矩阵（压缩存储）	分解		CSPTRF		ZSPTRF
	使用分解解方程组		CSPTRS		ZSPTRS
	求条件数		CSPCON		ZSPCON
	求误差界		CSPRFS		ZSPRFS
	使用分解求逆		CSPTRI		ZSPTRI
三角矩阵	解方程组	STRTRS	CTRTRS	DTRTRS	ZTRTRS
	求条件数	STRCON	CTRCON	DTRCON	ZTRCON
	求误差界	STRRFS	CTRRFS	DTRRFS	ZTRRFS
	求逆	STRTRI	CTRTRI	DTRTRI	ZTRTRI
三角矩阵（压缩存储）	解方程组	STPTRS	CTPTRS	DTPTRS	ZTPTRS
	求条件数	STPCON	CTPCON	DTPCON	ZTPCON
	求误差界	STPRFS	CTPRFS	DTPRFS	ZTPRFS

	求逆	STPTRI	CTPTRI	DTPTRI	ZTPTRI
三对角带状矩阵	解方程组	STBTRS	CTBTRS	DTBTRS	ZTBTRS
	求条件数	STBCON	CTBCON	DTBCON	ZTBCON
	求误差界	STBRFS	CTBRFS	DTBRFS	ZTBRFS

4.1.6.2 正交分解和线性最小二乘问题

LAPACK 提供了很多程序用于一般 m×n 的长方形矩阵 A 的分解，最终得到一个正交矩阵（如果是复型就得到酉矩阵）和一个三角矩阵（也可能是梯形矩阵）。

如果 $Q^T Q = I$，则实矩阵 Q 为正交矩阵；如果 $Q^H Q = I$，在复矩阵 Q 为酉矩阵。正交矩阵或酉矩阵具有使向量的二范数保持不变的重要性质：

$$\|x\|_2 = \|Qx\|_2, \quad \text{if } Q \text{ is orthogonal or unitary.}$$

这样，他们就可以用来保持数值的稳定性，因为他们不会扩大边界误差。

正交分解被用于解决线性最小二乘问题。他们也可以作为解决特征值或奇异值问题的第一步。

4.1.6.2.1 QR 分解

最常用而且最著名的分解就是 QR 分解。

$$A = Q \begin{pmatrix} R \\ 0 \end{pmatrix}, \quad \text{if } m \geq n,$$

其中，R 是一个 n×n 的上三角矩阵，Q 是一个 m×m 的正交矩阵（或酉矩阵）。如果 A 是 n 满秩的，那么 R 是非奇异的。有时，为了方便将分解记做：

$$A = \begin{pmatrix} Q_1 & Q_2 \end{pmatrix} \begin{pmatrix} R \\ 0 \end{pmatrix}$$

它可简化为：

$$A = Q_1 R$$

其中 Q1 由 Q 的前 n 列组成，而 Q2 为剩下的 m－n 列。

如果 m<n,R 就是梯形矩阵，分解可以记做

$$A = Q \begin{pmatrix} R_1 & R_2 \end{pmatrix}, \quad \text{if } m < n,$$

其中 R_1 为上三角矩阵，R_2 为长方形矩阵。

程序 xGEQRF 用于计算 QR 分解。矩阵 Q 没有被显式生成，而是被表示成如 2.3.4 节所介绍的一组反射向量的积形式。用户不需要了解这些表示的细节，因为 LAPACK 提供了可作用于 Q 的相关程序 xORGQR（或在复型情况下使用 xUNGQR），它可以产生全部或部分

的 Q，另外 xORMQR（xUNMQR）可以实现 Q 或 QT (QH)的左乘或右乘。

QR 分解可用于解决线性最小二乘问题(2.1)，当 m 大于或等于 n 且 A 满秩时

$$\|b-Ax\|_2 = \|Q^Tb-Q^TAx\|_2 = \left\| \begin{array}{c} c_1 - Rx \\ c_2 \end{array} \right\|_2, \quad \text{where } c \equiv \left(\begin{array}{c} c_1 \\ c_2 \end{array} \right) = \left(\begin{array}{c} Q_1^Tb \\ Q_2^Tb \end{array} \right) = Q^Tb;$$

其中 c 可以由 xORMQR（或 xUNMQR）计算，c_1 由它的前 n 个元素组成。而 x 是可以由 xTRTRS 计算的上三角方程组的解。

$$\boldsymbol{Rx = c_1}$$

剩余向量 r 可以从下列等式得到，

$$r = b - Ax = Q \left(\begin{array}{c} 0 \\ c_2 \end{array} \right),$$

可通过 xORMQR（或 xUNMQR）计算。剩余向量的元素平方和可以通过下面的公式，直接得到而不显式的生成 r。

$$|r|_2 = |b - Ax|_2 = |c_2|_2.$$

4.1.6.2.2 LQ 分解

LQ 分解形式如下：

$$A = \left(\begin{array}{cc} L & 0 \end{array} \right) Q = \left(\begin{array}{cc} L & 0 \end{array} \right) \left(\begin{array}{c} Q_1 \\ Q_2 \end{array} \right) = LQ_1, \quad \text{if } m \le n,$$

其中 L 是 m×m 的下三角矩阵，Q 是 n×n 正交矩阵（或酉矩阵），Q_1 由 Q 的的前 m 行组成，而 Q_2 由剩下的 n－m 行组成。

此分解由程序 xGELQF 实现，同样的 Q 由一组反射向量的积形式表示；程序 xORGLQ（或在复型情况下使用 xUNGLQ）可以产生全部或部分的 Q，另外 xORMLQ（xUNMLQ）可以实现 Q 或 Q^T (Q^H)的左乘或右乘。

A 的 LQ 分解本质上和 A^T (或 A^H)的 QR 分解相同。

$$A = \left(\begin{array}{cc} L & 0 \end{array} \right) Q \quad \Longleftrightarrow \quad A^T = Q^T \left(\begin{array}{c} L^T \\ 0 \end{array} \right).$$

$$x = Q^T \left(\begin{array}{c} L^{-1}b \\ 0 \end{array} \right)$$

LQ 分解可用于找到不定方程 A x = b（A 为 m×n 阶，其中 m<n）的最小范数解。结果可由下面的等式得到，并可由 xTRTRS 和 xORMLQ 实现。

4.1.6.2.3 选主列的 QR 分解

为了解决当 A 不满秩或情况不详时的最小二乘问题(4.1)，我们可以使用选主列的 QR 分解或者奇异值分解。（参见 4.2.4.6 节）

选主列的 QR 分解由下式给出：

$$A = Q \begin{pmatrix} R \\ 0 \end{pmatrix} P^T, \quad m \geq n,$$

其中 Q 和 R 同前，P 为一置换矩阵，选择它满足

$$|r_{11}| \geq |r_{22}| \geq \ldots \geq |r_{nn}|$$

此外，对于每个 k，有

$$|r_{kk}| \geq \|R_{k:j,j}\|_2 \quad \text{for } j = k+1, \ldots, n.$$

在精确的算法中，如果 rank（A）=k，则行列都是从 k+1 到 n 的子矩阵 R_{22} 为零。在数值计算中，目的应当是确定序号 k，使得 k 阶的主子矩阵性质优良，而 R_{22} 可以忽略。

$$R = \begin{pmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{pmatrix} \simeq \begin{pmatrix} R_{11} & R_{12} \\ 0 & 0 \end{pmatrix}.$$

k 是矩阵 A 的有效秩。参见 Golub and Van Loan [5]。

所谓最小二乘问题(4.1)的基本解可以从下面的分解中得到，

$$x = P \begin{pmatrix} R_{11}^{-1} \hat{c}_1 \\ 0 \end{pmatrix},$$

其中 $\hat{c}_1$ 只包括 $c = Q^T b$ 的前 k 个元素。

选主列的 QR 分解可以由 xGEQPF 或 xGEQP3 计算。这两个程序都是只计算分解而不确定矩阵的秩。xGEQP3 使用的是选主列 QR 分解的基于 3 级 BLAS 的版本，针对同一问题时它的速度比 xGEQPF 快很多。它们的区别可以在下面得到最好的描述。对于每一列，程序 xGEQPF 选出一列，置换它，然后计算一个反射向量，它可以使一部分元素为零，再通过 2 级 BLAS 操作将它运用到矩阵的其他部分。与之不同的是，程序 xGEQP3 只更新矩阵剩余部分的一行和一列（为下次选主列提供信息），而将剩余部分的更新留待一个列块全部完成之后再进行。这个反射向量的结果块通过 3 级 BLAS 操作应用到矩阵的其他部分。为了保持与 LAPACK2.0 的兼容性 xGEQPF 被保留了下来，但是我们在手册后面的部分将不会对此程序做介绍。

对于这些程序，矩阵 Q 所代表的矩阵与调用 xGEQRF 所产生的完全一样，因此，程序 xORGQR 和 xORMQR（或 xUNGQR 和 xUNMQR）可用于 Q。

4.1.6.2.4 完全正交分解

选主列的 QR 分解只有当 $R_{12} = 0$ 时，才能为我们计算亏秩最小二乘问题的最小范数解。但是，使用程序 xTZRQF（或 xTZRZF），通过对梯形矩阵（R_{11} ，R_{12}）从右到上的进行进一步的正交变换，R_{12} 可以被化为如下形式：

$$\begin{pmatrix} R_{11} & R_{12} \end{pmatrix} Z = \begin{pmatrix} T_{11} & 0 \end{pmatrix}.$$

这给出了完全正交分解

$$AP = Q\begin{pmatrix} T_{11} & 0 \\ 0 & 0 \end{pmatrix} Z^T$$

可以得到如下式形式的最小范数解。

$$x = PZ\begin{pmatrix} T_{11}^{-1}\hat{c}_1 \\ 0 \end{pmatrix}.$$

矩阵 Z 没有显式的生成，而是如 4.3.4 节所描述的那样由一组初步反射的积形式表示。用户不需要了解这些表示的细节，因为 LAPACK 提供了可作用于 Z 的相关程序 xORMRZ（xUNMRZ）可以实现 Z 或 Z^T (Z^H)的左乘或右乘。

程序 xTZRZF 是 xTZRQF 高效的分块版本。为了保持与 LAPACK2.0 的兼容性 xTZRQF 被保留了下来，但是我们在手册后面的部分将不会对此程序做介绍。

4.1.6.2.5 其他分解

QL 和 RQ 分解形式如下：

$$A = Q\begin{pmatrix} 0 \\ L \end{pmatrix}, \quad \text{if } m \geq n,$$

和

$$A = \begin{pmatrix} 0 & R \end{pmatrix} Q, \quad \text{if } m \leq n.$$

这些分解分别由 xGEQLF 和 xGERQF 计算，它们不如上面提到的 QR 分解或 LQ 分解常用，但是它们仍然有应用的领域，如计算广义 QR 分解。[9]

所有讨论过的分解程序（除了 xTZRQF 和 xTZRZF）都允许任意的 m 和 n，以便有时候矩阵 R 或 L 是梯形矩阵而非三角矩阵。选主元的程序只有 QR 分解。

表 4.9：正交分解的计算程序

分解和矩阵类型	操作	单精度		双精度	
		实型	复型	实型	复型
QR, 一般	选主元分解	SGEQP3	CGEQP3	DGEQP3	ZGEQP3
	不选主元分解	SGEQRF	CGEQRF	DGEQRF	ZGEQRF
	生成 Q	SORGQR	CUNGQR	DORGQR	ZUNGQR
	与 Q 相乘	SORMQR	CUNMQR	DORMQR	ZUNMQR
LQ, 一般	不选主元分解	SGELQF	CGELQF	DGELQF	ZGELQF
	生成 Q	SORGLQ	CUNGLQ	DORGLQ	ZUNGLQ

	与 Q 相乘	SORMLQ	CUNMLQ	DORMLQ	ZUNMLQ
QL, 一般	不选主元分解	SGEQLF	CGEQLF	DGEQLF	ZGEQLF
	生成 Q	SORGQL	CUNGQL	DORGQL	ZUNGQL
	与 Q 相乘	SORMQL	CUNMQL	DORMQL	ZUNMQL
RQ, 一般	不选主元分解	SGERQF	CGERQF	DGERQF	ZGERQF
	生成 Q	SORGRQ	CUNGRQ	DORGRQ	ZUNGRQ
	与 Q 相乘	SORMRQ	CUNMRQ	DORMRQ	ZUNMRQ
RZ, 梯形	不选主元分解（块算法）	STZRZF	CTZRZF	DTZRZF	ZTZRZF
	与 Q 相乘	SORMRZ	CUNMRZ	DORMRZ	ZUNMRZ

4.1.6.3 广义正交分解和最小二乘问题

4.1.6.3.1 广义 QR 分解

n×m 阶矩阵 A 和 n×p 阶矩阵 B 的广义 QR 分解（GQR）由下面一组分解表示：

$$A = QR \quad \text{and} \quad B = QTZ$$

其中 Q 和 Z 分别是 n×n 和 p×p 的正交矩阵（或酉矩阵）。R 的形式为：

$$R = \begin{array}{c} \\ m \\ n-m \end{array} \begin{array}{c} m \\ \begin{pmatrix} R_{11} \\ 0 \end{pmatrix} \end{array}, \quad \text{if} \quad n \geq m$$

或

$$R = \begin{array}{c} \\ n \end{array} \begin{array}{cc} n & m-n \\ \begin{pmatrix} R_{11} & R_{12} \end{pmatrix} \end{array}, \quad \text{if} \quad n < m$$

其中 R_{11} 为上三角矩阵。T 的形式为：

$$T = \begin{array}{c} \\ n \end{array} \begin{array}{cc} p-n & n \\ \begin{pmatrix} 0 & T_{12} \end{pmatrix} \end{array}, \quad \text{if} \quad n \leq p$$

或

$$T = \begin{array}{c} \\ n-p \\ p \end{array} \begin{array}{c} p \\ \begin{pmatrix} T_{11} \\ T_{21} \end{pmatrix} \end{array}, \quad \text{if} \quad n > p$$

其中，T_{12} 或 T_{21} 是上三角矩阵。

值得注意的，如果 B 为非奇异方阵，则 A 和 B 的广义 QR 分解隐含地给出了矩阵 $B^{-1}A$ 的 QR 分解：

$$B^{1}\ A = Z^{T}\ (\ T^{1}\ R\ )$$

而没有计算 B^{-1} 或 $B^{-1}A$。

程序 xGGQRF 计算广义 QR 分解时，首先计算 A 的 QR 分解，再计算 $Q^{T}B$ 的 QR 分解。这些正交矩阵（或酉矩阵）Q 和 Z 既可以显式的生成，也可以用于和其他矩阵相乘，就如同 QR 分解中的正交矩阵（或酉矩阵）一样（参见 4.3.4 节）。

广义 QR 分解可以用于解决广义(Gauss-Markov)线性模型问题（GLM）（参见 4.2.3 节和[5,Page 252]）。使用 A 和 B 的广义 QR 分解，我们可以重写方程式 d = A x + B y 为如下形式：

$$\begin{aligned} Q^{T}d &= Q^{T}Ax + Q^{T}By \\ &= Rx + TZy. \end{aligned}$$

我们将它划分成

$$\begin{array}{c} m \\ n-m \end{array}\left(\begin{array}{c} d_1 \\ d_2 \end{array} \right) = \begin{array}{c} m \\ n-m \end{array}\left(\begin{array}{c} R_{11} \\ 0 \end{array} \right) x + \begin{array}{c} m \\ n-m \end{array}\left(\begin{array}{cc} \overset{p-n+m}{T_{11}} & \overset{n-m}{T_{12}} \\ 0 & T_{22} \end{array} \right) \left(\begin{array}{c} y_1 \\ y_2 \end{array} \right)$$

其中

$$\left(\begin{array}{c} d_1 \\ d_2 \end{array} \right) \equiv Q^{T}d, \quad \text{and} \quad \left(\begin{array}{c} y_1 \\ y_2 \end{array} \right) \equiv Zy\ ;$$

$\left(\begin{array}{c} d_1 \\ d_2 \end{array} \right)$ 可由 xORMQR（或 xUNMQR）计算。

$$y_1 = 0 \quad \text{and} \quad y_2 = T_{22}^{-1}d_2$$

$$x = R_{11}^{-1}(d_1 - T_{12}y_2) \quad \text{and} \quad y = Z^{T}\left(\begin{array}{c} 0 \\ y_2 \end{array} \right),$$

设 $y_1=0$ 且 $y_2=T_{22}^{-1}d_2$,可以由 xTRSV, xGEMV 和 xORMRQ (或 xUNMRQ)的计算结果 $x = R_{11}^{-1}(d_1 - T_{12}y_2) \quad \text{and} \quad y = Z^{T}\left(\begin{array}{c} 0 \\ y_2 \end{array} \right)$,得到 GLM 的解。

4.1.6.3.2 广义 RQ 分解

n×m 阶矩阵 A 和 p×n 阶矩阵 B 的广义 RQ 分解（GRQ）由下面一组分解表示：

$$A = RQ \quad \text{and} \quad B = ZTQ$$

其中 Q 和 Z 分别是 n×n 和 p×p 的正交矩阵（或酉矩阵）。R 的形式为：

$$R = \begin{array}{c} n-m \\ m \end{array}\left(\begin{array}{cc} \overset{n-m}{0} & \overset{m}{R_{12}} \end{array} \right), \quad \text{if} \quad m \le n$$

或

$$R = \begin{matrix} & n \\ m-n & \\ n & \end{matrix} \begin{pmatrix} R_{11} \\ R_{21} \end{pmatrix}, \quad \text{if} \quad m > n,$$

其中 R_{12} 或 R_{21} 为上三角矩阵。T 的形式为：

$$T = \begin{matrix} & n \\ p-n & \\ & \end{matrix} \begin{pmatrix} T_{11} \\ 0 \end{pmatrix}, \quad \text{if} \quad p \geq n$$

或

$$T = \begin{matrix} & p & n-p \\ p & & \end{matrix} \begin{pmatrix} T_{11} & T_{12} \end{pmatrix}, \quad \text{if} \quad p < n$$

其中 T_{11} 是上三角矩阵。

值得注意的，如果 B 为非奇异方阵，则 A 和 B 的广义 QR 分解隐含地给出了矩阵 AB^{-1} 的 QR 分解：

$$\boldsymbol{A\,B^{-1} = (\,R\,T^{-1}\,)\,Z^{T}}$$

而没有计算 B^{-1} 或 AB^{-1}。

程序 xGGRQF 计算广义 RQ 分解时，首先计算 A 的 RQ 分解，再计算 BQ^T 的 RQ 分解。这些正交矩阵（或酉矩阵）Q 和 Z 既可以显式的生成，也可以用于和其他矩阵相乘，就如同 RQ 分解中的正交矩阵（或酉矩阵）一样（参见 4.2.4.2 节）。

广义 RQ 分解可以用于解决线性等式约束的最小二乘问题（LSE）。使用 B 和 A 的广义 QR 分解（注意 B 和 A 的角色互换），具体形式如下：

$$B = TQ \quad \text{and} \quad A = ZRQ.$$

我们把约束的线性方程式 Bx = d 写成：

$$\boldsymbol{T\,Q\,x = d}$$

它可以分解为：

$$\begin{matrix} & n-p & p \\ p & & \end{matrix} \begin{pmatrix} 0 & T_{12} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = d \quad \text{where} \quad \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \equiv Qx.$$

x_2 是上三角方程组 $T_{12}\,x_2 = d$ 的解。

$$\boldsymbol{T_{12}\,x_2 = d}$$

此外，

$$\begin{aligned} \|Ax - c\|_2 &= \|Z^T Ax - Z^T c\|_2 \\ &= \|RQx - Z^T c\|_2. \end{aligned}$$

我们将此表达式分解为：

$$\begin{array}{c} \\ n-p \\ p+m-n \end{array} \begin{array}{c} \begin{array}{cc} n-p & p \end{array} \\ \begin{pmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{pmatrix} \end{array} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} - \begin{pmatrix} c_1 \\ c_2 \end{pmatrix}$$

其中 $\begin{pmatrix} c_1 \\ c_2 \end{pmatrix} \equiv Z^T c$，它可由 xORMQR (或 xUNMQR)计算得到。

为了解决 LSE 问题，我们设

$$\boldsymbol{R_{11}\, x_1 + R_{12}\, x_2 - c_1 = 0}$$

其中 x_1 作为上三角系统 $R_{11}\, x_1 = c_1 - R_{12}\, x_2$ 的解。

$$\boldsymbol{R_{11}\, x_1 = c_1 - R_{12}\, x_2.}$$

最后，所求的解为

$$x = Q^T \begin{pmatrix} x_1 \\ x_2 \end{pmatrix},$$

它可由 xORMRQ (或 xUNMRQ)计算得到。

4.1.6.4 对称特征值

设 A 为一实对称 n 阶方阵或复厄米特 n 阶方阵。如果 Az=λz，则标量 λ 称为一个特征值，而非零向量 z 称为相应的特征向量。当 A 为实对称矩阵或复厄米特矩阵时，λ 恒为实数。

对称特征值程序的基本任务就是对于给定的矩阵 A 求特征值 λ，并可选择地计算相应的特征向量 z。

计算过程如下：

1. 归约实对称矩阵或复厄米特矩阵 A 为实三对角形式 T。如果 A 是实对称矩阵，则分解为正交矩阵 Q 和对称三对角矩阵 T 的分解 $A=QTQ^T$。如果 A 是复厄米特矩阵，则分解为酉矩阵 Q 和对称三对角矩阵 T 的分解 $A=QTQ^H$。

2. 计算实对称三对角矩阵 T 的特征值和特征向量。如果计算所有的特征值和特征向量，相当于将 T 分解为 $T=S\Lambda S^T$，其中 S 为正交矩阵，Λ 为对角矩阵。

在实型的情况下，分解 A = Q T Q^T 由程序 xSYTRD, xSPTRD 或 xSBTRD 中的一个计算得到，具体采用哪个根据矩阵的存储情况而定（参见表 4.10）。在复型情况下，类似的程序为 xHETRD, xHPTRD 和 xHBTRD。程序 xSYTRD (或 xHETRD)如 4.2.3.4 节那样，用一组基本反射的积形式表示矩阵 Q。程序 xORGTR（或 xUNMTR）用于提供显式的 Q 形式；这在使用程序 xSTEQR 由 QR 算法计算 A 的所有特征向量之前是需要的。程序 xORMTR（或 xUNMTR）用于在不显式生成 Q 的情况下，计算 Q 与其他矩阵的乘积；这可以用于将由 xSTEIN 计算得到的 T 的特征向量转化为 A 的特征向量。

当应用压缩存储时，显式形成 Q 和实型矩阵与 Q 相乘的两个函数为 xOPGTR 和 xOPMTR (复型情况下为 xUPGTR 和 xUPMTR)。

当 A 为带状矩阵且 xSBTRD（或 xHBTRD）将其规约为三对角形式，Q 是由 Givens 变换的积形式表示的，而非基本反射的积形式；如果 Q 要求显示的表示，则需要由归约程序生成。xSBTRD 是基于 Kaufman 的向量化算法的。

有一些程序用于计算 T 的特征值和特征向量，来封装计算全部特征值和部分或全部特征向量的情况。另外，一些程序在某个计算环境中或对于某些矩阵，执行速度较快。还有一些

程序比其他的更加精确。

详见 4.3.4.1 的讨论。

xSTEQR

此程序使用隐式位移 QR 算法。细节详见[10]。这个程序被用在名称后缀为－EV 和－EVX 的计算所有特征值和特征向量的驱动程序中（参见 4.3.4.1 节）。

xSTERF

此程序使用 QR 算法的一个平方根自由版本，同样交替使用 QR 和 QL 两种变换，而且只计算所有的特征值。细节详见[10]。这个程序被用在名称后缀为－EV 和－EVX 的计算所有特征值（无特征向量）的驱动程序中（参见 4.2.3.4.1 节）。

xSTEDC

此程序使用 Cuppen 的分治算法以寻找特征值和特征向量（如果只要求特征值，则直接调用 xSTERF）。对于大矩阵，xSTEDC 的速度数倍于 xSTEQR，但是需要额外的工作空间(2n2 或 3n2)。这个程序被用在名称后缀为－EVD 的计算所有特征值和特征向量的驱动程序中（参见 4.2.3.4.1 节）。

xSTEGR

此程序使用相关的健壮表示（RRR）的算法来寻找特征值和特征向量。此程序将 T 的具有很多 T－sI 变换的 LDLT 分解，用于靠近每组特征值的单步位移 s。对于每个转换，此算法都能为极小的特征值计算出精确的特征值组。除个别情况，xSTEGR 比其他所有程序都快，而且使用的工作空间最小。详情见 4.2.3.4.3 节。

xPTEQR

此程序仅用于对称正定三对角矩阵。它使用了一种结合了乔累斯基分解和双对角 QR 迭代（参见 xBDSQR）的方法，值得注意的是，它可能会比除 xSTEGR 外的所有程序都快。

xSTEBZ

此程序使用二分法计算部分的或全部的特征值。选项可以提供计算一个实数区间内的全部特征值或从 ith 到最大 jth 之间的所有特征值。它可以计算的很精确，也可以在可接受的的精确度下提高速度。此程序被应用于名称以－EVX 结尾的驱动程序中。

xSTEIN

在给定精确的特征值条件下，此程序可使用逆迭代计算部分的或全部的特征向量。此程序被用于名称以－EVX 结尾的驱动程序中。

详见表 4.10

表 4.10：对称特征值问题的计算程序

矩阵类型和存储机制	操作	单精度		双精度	
		实型	复型	实型	复型
稠密对称矩阵（或厄米特矩阵）	三对角归约	SSYTRD	CHETRD	DSYTRD	ZHETRD
压缩存储的对称矩阵（或厄米特矩阵）	三对角归约	SSPTRD	CHPTRD	DSPTRD	ZHPTRD

带状对称矩阵（带状厄米特矩阵）	三对角归约	SSBTRD	CHBTRD	DSBTRD	ZHBTRD
正交矩阵/酉矩阵	xSYTRD 规约后生成矩阵	SORGTR	CUNGTR	DORGTR	ZUNGTR
	xSYTRD 规约后，计算与矩阵的乘积。	SORMTR	CUNMTR	DORMTR	ZUNMTR
正交矩阵/酉矩阵（压缩存储）	xSYTRD 规约后生成矩阵	SOPGTR	CUPGTR	DOPGTR	ZUPGTR
	xSYTRD 规约后，计算与矩阵的乘积。	SOPMTR	CUPMTR	DOPMTR	ZUPMTR
对称三对角矩阵	通过 QR 方法计算特征值或特征向量。	SSTEQR	CSTEQR	DSTEQR	ZSTEQR
	使用自由基 QR 方法只计算特征值	SSTERF		DSTERF	
	使用分治的方法计算特征值和特征向量	SSTEDC	CSTEDC	DSTEDC	ZSTEDC
	使用 RRR 方法计算特征值和特征向量	SSTEGR	CSTEGR	DSTEGR	ZSTEGR
	使用二分法只计算特征值	SSTEBZ		DSTEBZ	
	逆迭代计算特征向量	SSTEIN	CSTEIN	DSTEIN	ZSTEIN
对称正定三对角矩阵	计算特征值和特征向量。	SPTEQR	CPTEQR	DPTEQR	ZPTEQR

4.1.6.5 非对称特征值问题

4.1.6.5.1 特征值，特征向量和 Schur 分解

设 A 为一个 n×n 矩阵。如果 $Av=\lambda v$，则称 λ 为特征值，非零向量 v 为相应的右特征向量。如果非零向量 u 满足 $u^HA=\lambda u^H$，则称之为左特征向量。本节介绍的程序的根本任务是对于给定的矩阵 A，计算它的所有特征值，如果需要的话，计算其相关的右特征向量 v 或左特征向量 u。

第二个基本任务就是计算矩阵 A 的 Schur 分解。如果 A 是复型的，则分解形式为 $A=ZTZ^H$，其中 Z 为酉矩阵，T 为上三角矩阵。如果 A 是实型的，则分解形式为 $A=ZTZ^T$，其中 Z 为正交矩阵，T 为拟上三角矩阵（其对角线上的块大小为 1×1 或 2×2）。Z 的列称作 A 的 Schur 向量。A 的特征值出现在 T 的对角线上，实矩阵 A 的复共轭特征值与 T 对角线上 2×2 的块有关。

这两个根本任务是由下面的步骤实现的：

1.

归约一般矩阵 A 为上海森伯格形式 H，H 的第一次对角线之下的元素皆为零。如果 A 为实矩阵则归约形式为 $A=QHQ^T$，其中 Q 为正交矩阵；如果 A 为复矩阵则归约形式为 $A=QHQ^H$，其中 Q 为酉矩阵。归约由程序 xGEHRD 执行，其中 Q 由 4.2.3.4 节所示的分解形式表示。程序 xORGTR（或 xUNMTR）用于提供显式的 Q 形式。程序 xORMTR（或 xUNMTR）用于在不显式生成 Q 的情况下，计算 Q 与其他矩阵的乘积

2.

上海森伯格矩阵 H 规约为 Schur 形式 T，并给出 Schur 形式 H=STST (H 为实矩阵) or H=STSH (H 为复矩阵)。矩阵 S（H 的 Schur 向量）可以有选择的计算。可以选择在第一步用 S 右乘矩阵 Q，以给出矩阵 Z＝QS，它是 A 的 Schur 向量。特征值可以由 T 的对角线得到。所有的这些由程序 xHSEQR 实现。

3.

在给定特征值的情况下，特征向量可以通过两种方法计算。xHSEIN 实现在 H 上的逆迭代以计算 H 的特征向量；xORMHR 可以被用来计算矩阵 Q 与特征向量的乘积以用于将它们转化为 A 的特征向量。xTREVC 计算 T 的特征向量，而且在矩阵 S 或 Z 给出的条件下，可有选择的将特征向量转化为 H 或 A。xHSEIN 和 xTREVC 都可以选择计算左特征向量或右特征向量。

其他辅助的任务可能在上面这些描述之前或之后实现。

4.1.6.5.2 平衡

程序 xGEBAL 可以用于在规约到海森伯格形式之前平衡矩阵 A。平衡包括两个步骤，它们都是可选的：

首先，xGEBAL 试图通过一个相似变换将 A 置换成块上三角矩阵形式：

$$PAP^T = A' = \begin{pmatrix} A'_{11} & A'_{12} & A'_{13} \\ 0 & A'_{22} & A'_{23} \\ 0 & 0 & A'_{33} \end{pmatrix}$$

其中，P 为一个置换矩阵，且 A'_{11} 和 A'_{33} 为上三角矩阵。这样除行列从 ILO 到 IHI 的中心对角块 A'_{22} 之外，此矩阵已经是 Schur 形式了。随后的操作 xGEBAL, xGEHRD 或 xHSEQR 都是仅需要应用在这些行列上；因此 ILO 和 IHI 成为 xGEHRD 和 xHSEQR 讨论的焦点。如果 ILO>1 或 IHI<n，则可以节约很多的工作。如果找到适当的置换（通常如此），xGEBAL 设 ILO＝1 且 IHI＝n，则 A'_{22} 为整个矩阵 A。

第二步，xGEBAL 为 A’提供了一个对角相似变换以使得 A’22 的行列尽可能标准：

$$A'' = DA'D^{-1} = \begin{pmatrix} I & 0 & 0 \\ 0 & D_{22} & 0 \\ 0 & 0 & I \end{pmatrix} \begin{pmatrix} A'_{11} & A'_{12} & A'_{13} \\ 0 & A'_{22} & A'_{23} \\ 0 & 0 & A'_{33} \end{pmatrix} \begin{pmatrix} I & 0 & 0 \\ 0 & D_{22}^{-1} & 0 \\ 0 & 0 & I \end{pmatrix}$$

这在某些情况下可以改进精度。

如果 A 通过了 xGEBAL 的平衡，接下来操作计算的特征向量就是平衡矩阵 A"的特征向量；xGEBAK 必须被调用以使这些特征向量转化为原矩阵 A 的特征向量。

4.1.6.5.3 不变子空间和条件数

Schur 形式取决于 T 对角线上的特征值顺序而且它也可以由用户选择。假设选择 $\lambda_1, \ldots, \lambda_j$, $1 \le j \le n$, 出现在 T 的左上角。这样 Z 的前 j 列就跨过了与 $\lambda_1, \ldots, \lambda_j$ 相关的 A 的右不变子空间。

下面的程序执行重排序，同时也计算特征值，特征向量和不变子空间的条件数：

1. xTREXC 将特征值（或 2×2 块）在对角线上从其原来的位置移动到任意位置。它可能被用于选择 Schur 形式中特征值的顺序。
2. xTRSYL 求解 Sylvester 矩阵方程 $AX \pm XB = C$ 中的 X，其中 A，B 和 C 给定且 A 和 B 为拟三角矩阵。他被用于程序 xTRSNA 和 xTRSEN，但是它仍具有独立的价值。
3. xTRSNA 计算 Schur 形式中矩阵 T 的特征值和/或右特征向量的条件数。这与得到 T 的原始矩阵 A 的特征值和右特征向量的条件数是相同的。用户可以计算所有特征值/特征向量的条件数或只计算某选定的子集的条件数。详细情况参见[11]。
4. xTRSEN 把 Schur 形式的矩阵 T 的一个选定的特征值子集移至 T 的左上角，并可以选择的计算它们平均值和右不变子空间的条件数。这些条件数与得到 T 的原始矩阵 A 的平均值和右不变子空间的条件数相同。详细情况参见[11]。

这些程序完整的列表详见表 4.11

表 4.11：非对称特征值的计算程序

矩阵类型和存储机制	操作	单精度		双精度	
		实型	复型	实型	复型
一般	海森伯格归约	SGEHRD	CGEHRD	DGEHRD	ZGEHRD
	平衡	SGEBAL	CGEBAL	DGEBAL	ZGEBAL
	还原转换	SGEBAK	CGEBAK	DGEBAK	ZGEBAK
正交/酉	海森伯格归约后生成矩阵	SORGHR	CUNGHR	DORGHR	ZUNGHR
	海森伯格后得到的矩阵做乘法	SORMHR	CUNMHR	DORMHR	ZUNMHR

海森伯格	Schur 分解	SHSEQR	CHSEQR	DHSEQR	ZHSEQR
	逆迭代求特征向量	SHSEIN	CHSEIN	DHSEIN	ZHSEIN
拟三角矩阵	特征向量	STREVC	CTREVC	DTREVC	ZTREVC
	Schur 分解重排序	STREXC	CTREXC	DTREXC	ZTREXC
	Sylvster 方程	STRSYL	CTRSYL	DTRSYL	ZTRSYL
	特征值/特征向量的条件数	STRSNA	CTRSNA	DTRSNA	ZTRSNA
	特征值组/不变子空间的条件数。	STRSEN	CTRSEN	DTRSEN	ZTRSEN

4.1.6.6 奇异值分解

设 A 为一 m×n 阶一般实矩阵。A 的奇异值分解形式为 $A = U\Sigma V^T$，其中 U 和 V 是正交矩阵，且 $\Sigma = \mathrm{diag}(\sigma_1, \ldots, \sigma_r)$, $r = \min(m,n)$, $\sigma_1 \geq \cdots \geq \sigma_r \geq 0$。如果 A 是复矩阵，则它的 SVD 分解为 $A = U\Sigma V^H$，其中 U 和 V 为酉矩阵，且 Σ 像上面一样是实对角矩阵。其中 σ_i 被称为奇异值，V 的前 r 列称右奇异向量，U 的前 r 列称为左奇异向量。

列在表 4.12 的本节所描述的程序，是用来计算此分解的。这个分解分为以下几步：

1. 如果 A 为实矩阵，则 A 被归约成双对角形式：$A=U_1\ B\ V_1^T$（如果 A 为复矩阵，在 A 被分解为 $A=U_1\ B\ V_1^H$），其中 U_1 和 V_1 是正交矩阵（或酉矩阵），且 B 为实上双对角矩阵（$m \geq n$ 时）或实下双对角矩阵（m<n 时），以致于 B 只有主对角线和上次对角线（如果 $m \geq n$）或下次对角线（如果 m<n）。
2. 计算双对角矩阵 B 的 SVD 分解：$B = U_2\Sigma V_2^T$，其中 U_2 和 V_2 为正交矩阵且 Σ 是如上所描述的对角矩阵。A 的奇异向量为 $U = U_1\ U_2$ and $V = V_1\ V_2$。

双对角矩阵的归约使用程序 xGEBRD 或 xGBBRD（带状矩阵）。

程序 xGEBRD 以基本反射向量的乘积形式表示 U_1 和 V_1，如 4.2.3.4 节所描述的。如果 A 为实矩阵则矩阵 U_1 和 V_1 可能由程序 xORGBR 明确计算，或使用程序 xORMBR 在不形成 U_1 和 V_1 的情况下实现与其他矩阵的相乘。如果 A 为复矩阵，则相应的程序为 xUNGBR 和 xUNMBR。

如果 A 是带状矩阵且被 xGBBRD 规约为双对角形式，U_1 和 V_1 由 Givens 变换的乘积表示，而不是基本反射向量的乘积。如果要求计算 U_1 和 V_1，则由 xGBBRD 显式生成。xGBBRD 使用类似于 xSBTRD 中使用的向量化算法（参见 Kaufman [12]）。当带宽很小时，xGBBRD 可能比 xGEBRD 快得多。

双对角矩阵的 SVD 分解可由程序 xBDSQR 或程序 xBDSDC 计算。当要求计算奇异向量时，xBDSQR 使用 QR 迭代，否则使用 dqds 算法[13]。它可以将奇异向量计算的非常精确。

xBDSDC 使用 Cuppen 变换的分治的方法寻找奇异值和奇异向量[14]。如果想得到的是大矩阵的向量，则它比 xBDSQR 快得多。当只要求奇异值时，它如 xBDSQR 一样，使用 dqds 算法[13]。分治的方法不能保证在接近完全相对精度下计算奇异值，而事实上，xBDSDC 通常精度不比 xBDSQR 差。xBDSDC 使用压缩格式的 U_2 和 V_2 表示奇异向量，以 $O(n\log n)$ 的空间代替原来 n^2 的空间。接下来，U_2 和 V_2 可以由程序 xLASDQ 显示生成，也可以使用 xLASD0 在不形成 U_1 和 V_1 的情况下实现与其他矩阵的相乘。

如果 $m \gg n$，按如下步骤计算将更有效：先执行 xGEQRF 提供的的 A 的 QR 分解，之后计算 n×n 矩阵 R 的 SVD 分解直至 $A = QR$ 且 $R = U\Sigma V^T$，这样就得到了 A 的 SVD 分解 $A = (QU)\Sigma V^T$。同样的，如果 $m \ll n$，则先执行 xGELQF 提供的 A 的 LQ 分解效率较高。这初步的 QR 分解和 LQ 分解由驱动 xGESVD 和 xGESDD 执行。

SVD 分解可能用于寻找亏秩最小二乘问题的最小范数解(4.1)。A 的有效秩 k 可以由超过一定大小阀值的奇异值的个数确定。设 $\hat{\Sigma}$ 为 Σ k×k 阶的主子矩阵，$\hat{V}$ 是由矩阵 V 前 k 列组成的。这样解就可以由下式给出：

$$x = \hat{V}\hat{\Sigma}^{-1}\hat{c}_1$$

其中 $\hat{c}_1$ 由 $c = U^T b = U_2^T U_1^T b$ 的前 k 个元素组成。$U_1^T b$ 可以由 xORMBR 计算，而 xBDSQR 有一个向量与 U_2^T 相乘的选项。

表 4.12：特征值分解的计算程序

矩阵类型和存储机制	操作	单精度		双精度	
		实型	复型	实型	复型
一般	双对角归约	SGEBRD	CGEBRD	DGEBRD	ZGEBRD
一般带状	双对角归约	SGBBRD	CGBBRD	DGBBRD	ZGBBRD
正交/酉	双对角归约后生成矩阵	SORGBR	CUNGBR	DORGBR	ZUNGBR
	双对角归约后实型矩阵相乘	SORMBR	CUNMBR	DORMBR	ZUNMBR
双对角	使用 QR 或 dqds 方法实 SVD 分解	SBDSQR	CBDSQR	DBDSQR	ZBDSQR
	使用分治的方法实现 SVD 分解。	SBDSDC		DBDSDC	

4.1.6.7 广义对称正定特征值

本节的内容是广义特征值问题 $Az = \lambda Bz$，$ABz = \lambda z$，和 $BAz = \lambda z$（其中 A，B 都是实型对称矩阵或复型厄米特矩阵且 B 为正定矩阵）的求解。每一个这样的问题都可以使用一个 B 的乔累斯基分解（形如 $B=LL^T$ 或 $B=U^TU$ (厄米特矩阵为 LL^H 或 U^HU)）归约成标准对称特征值问题。 对于 $Ax = \lambda Bz$ 的情况，如果 A 和 B 是带状的，则可以使用一个利用 $B = LL^T$ 的快速算法，$Az = \lambda Bz \quad \Rightarrow \quad (L^{-1}AL^{-T})(L^Tz) = \lambda(L^Tz).$ 因此 $Az = \lambda Bz$ 的特征值就是 $Cy = \lambda y$ 的特征值，其中 C 为对称矩阵，满足 $C = L^{-1} A L^{-T}$ and $y = L^T z$ 。C 为厄米特矩阵时，它满足 $C = L^{-1} A L^{-H}$ and $y = L^H z$ 。

表 4.13 归纳了这三种问题如何分别归约成标准形式 $Cy = \lambda y$，如何从归约问题得到的特征向量 y 恢复为原特征向量 z。此表针对的是实型问题，复型问题只需将转置改为共轭转置。

表 4.13：广义对称正定特征值问题转化为标准问题

	问题类型	B 的分解	归约	特征向量恢复
1.	$Az = \lambda Bz$	$B = LL^T$	$C = L^{-1} A L^{-T}$	$z = L^{-T} y$
		$B = U^TU$	$C = U^{-T} A U^{-1}$	$z = U^{-1} y$
2.	$ABz = \lambda z$	$B = LL^T$	$C = L^T A L$	$z = L^{-T} y$
		$B = U^TU$	$C = U A U^T$	$z = U^{-1} y$
3.	$BAz = \lambda z$	$B = LL^T$	$C = L^T A L$	$z = L y$
		$B = U^TU$	$C = U A U^T$	$z = U^T y$

给定矩阵 A 和 B 的乔累斯基分解，程序 xyyGST 用相关的标准问题 $Cy = \lambda y$ 中的矩阵 C 覆盖 A（参见表 4.14）。这可以使用 4.2.4.4 节描述的程序解决。不需要特殊的程序将标准问题的特征向量 y 恢复成广义问题的特征向量 z，因为这些计算只是 2 级或 3 级 BLAS 的简单应用。

如果问题是 $Az = \lambda Bz$ 且矩阵 A 和 B 是带状的，如上定义的矩阵 C 通常是满的。我们可以通过如下修改 C 定义的方法将此问题转化为带状的标准问题。

$$C = X^TAX, \quad \text{where } X = U^{-1}Q \quad \text{or} \quad L^{-T}Q,$$

其中 Q 是选择用来使 C 的带宽不大于 A 的正交矩阵。Q 由 Givens 变换的乘积表示。这就是 Crawford 算法。如果 X 要求计算，那么它必须由归约程序显式生成。

当 A 和 B 都是带状时，解决方法可以进一步改进以使得对于要求形成 C 的问题工作量减少一半。我们使用“分裂的乔累斯基方法” $B = S^T S$ （或复型 $S^H S$ ）代替标准的 B 的标准乔累斯基分解 $U^T U$ 或 $L L^T$，其中

$$S = \begin{pmatrix} U_{11} & \\ M_{21} & L_{22} \end{pmatrix}$$

U_{11} 为上三角矩阵，L_{22} 为阶约为 n/2 的下三角矩阵；S 的带宽与 B 相同。在 B 被程序 xPBSTF 分解之后，带状广义特征值问题 $Az = \lambda Bz$ 到一个标准的带状问题 $Cy = \lambda y$ 的归约可以由程序 xSBGST （复型情况为 xHBGST）实现。此程序实现了一个向量化形式的算法，它由 Kaufman [12]提出。

表 4. 14：广义对称正定特征值问题的计算程序

矩阵类型和存储机制	操作	单精度		双精度	
		实型	复型	实型	复型
对称/厄米特	归约	SSYGST	CHEGST	DSYGST	ZHEGST
对称/厄米特（压缩存储）	归约	SSPGST	CHPGST	DSPGST	ZHPGST
对称带状/厄米特带状	分裂的乔累斯基分解	SPBSTF	CPBSTF	DPBSTF	ZPBSTF
	归约	SSBGST	DSBGST	CHBGST	ZHBGST

4.1.6.8 广义非对称特征问题

4.1.6.8.1 特征值，特征向量和广义 Schur 分解

假设 ***A*** 和 ***B*** 都是 ***n*×*n*** 矩阵。 一个称为广义特征值的标量 λ 和一个是矩阵对（A，B）的相应右广义向量的非 0 列向量 x，如果 $Ax = \lambda Bx$。 满足 $y^H A = \lambda y^H B$ 的非 0 列向量 y 被称为与 λ 对应的左广义向量。(简单的说，在不容易引起混淆的情况下我们将省略“广义”一词 。） 如果 ***B*** 是奇异的，我们将有无限特征值 $\lambda = \infty$，我们的意思是 ***Bx* = 0**。 注意如果 ***A*** 是非奇异的，那么等价问题 $\mu Ax = Bx$ 就被完全的明确定义了，并且相应的无限特征值μ=0 。 在 4.2.3.7 节描述的广义对称确定特征系问题只有有限实特征值。广义非对称特征值问题可以有实型，复型和无限特征值。为了处理有限（包括 0）和无限特征值问题，LAPACK 程序返回了两个值，α 和 β。 如果 β 是非 0 的那么 λ=α/β 是一个特征值。如果 β 是 0 那么 $\lambda = \infty$是(***A*, *B***)的一个特征值。 (由于舍入可能会使一个为 0 的 β 变成一个很小的非 0 值，而使特征值 $\lambda = \infty$成为一个很大的值。)这些程序的基本任务就是对一个给定的矩阵对（A，B）计算所有的 n 对 (α, β) 和 ***x*** 和/或 ***y*** 。

如果具有决定性的 $A - \lambda B$ 对所有的 λ 恰好等于 0，这类奇异值问题被称为奇异的；否则就是正则。（A，B）的奇异性被α=β=0 标示（由于舍入，α和 β可能会很小）。在这种情况下，特征值问题非常病态的，事实上 α和 β 的一些其他非 0 值可能是不确定的（更进一步的信息可以参看 4.11.1.4 节）[1,2,3]。

另一个基本任务就是计算矩阵对（A，B）的**广义 Schur 分解。**如果是复型的，那么它们的广义 Schur 分解就是 $A = QSZ^H$ 和 $B = QTZ^H$,这里 Q 和 Z 是酉矩阵,S 和 T 是上三角的。LAPACK 程序可以规格化 T 使其初值为非负对角矩阵。在这种形式下，请注意，特征值可以很容易的从对角线计算出来：$\lambda_i = s_{ii}/t_{ii}$（如果 $t_{ii} \neq 0$）和 $\lambda_i = \infty$（如果 $t_{ii} = 0$），所以 LAPACK 程序返回 $\alpha_i = s_{ii}$ 和 $\beta_i = t_{ii}$。

广义 Schur 形式依赖于（S，T）对角线上的特征值的顺序。这个顺序可以被用户自己选择。

如果 A 和 B 是实型的, 那么它们的广义 Schur 分解就是 $A = QSZ^T$ 和 $B = QTZ^T$, 这里 Q 和 Z 是正交的,S 是拟上三角并且其对角线上是 1×1 和 2×2 分块， T 是对角线元素为非负的上三角矩阵。典型矩阵对（S，T）的结构如下所示，n=6:

$$S = \begin{pmatrix} \times & \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times & \times \\ 0 & 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & 0 & \times & \times \\ 0 & 0 & 0 & 0 & \times & \times \end{pmatrix}, \qquad T = \begin{pmatrix} \times & \times & \times & \times & \times & \times \\ 0 & \times & 0 & \times & \times & \times \\ 0 & 0 & \times & \times & \times & \times \\ 0 & 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & 0 & \times & 0 \\ 0 & 0 & 0 & 0 & 0 & \times \end{pmatrix}$$

(*S*,*T*)的 **1 x 1** 对角块(那些在(1,1) 和(4,4) 位置上的) 包含**(*A*,*B*)**的实特征值， **(*S*,*T*)**的 **2 x 2** 对角块(那些在(2:3,2:3) 和(5:6,5:6) 位置上的) 包含**(*A*,*B*)**复型特征值的共扼对。T 的 **2 x 2** 对角块和相应的 S 的 2×2 块均产生在对角线上。 这样的排列可以使我们只对实数操作,甚至（A，B）的一些特征值是复型的。注意对于实特征值与对所有在复型情况下的特征值一样, 与特征值对应的 α_i 和 β_i 值可以很容易的从 S 和 T 的对角线计算出来。与（S，T）的 2×2 对角块的复型特征值对应的 α_i 和 β_i 值是这样计算的，首先计算块的复型共扼特征值 λ 和 $\bar{\lambda}$, 然后如果块被变成复型广义 Schur 形式将会计算出 β_i 和 β_{i+1} 的值，最后相乘得到 $\alpha_i = \lambda\beta_i$ 和 $\alpha_{i+1} = \bar{\lambda}\beta_{i+1}$。

Q 和 Z 的列叫做广义 Schur 向量和 A 与 B 压缩子空间的扩展对[4]。压缩子空间是不变子空间的一般化：Z 的前 k 列扩展的右压缩子空间被 A 和 B 映射为 Q 的前 k 列扩展的左压缩子空间。与出现在 S 和 T 的左上角前 k 个特征值对应的压缩子空间对将在 2.3.5.2 说明。

计算将按照下列步骤进行：

1. 矩阵对 (A,B) 被规约成**广义上Hessenberg 形式**form. 如果 A 和 B 是实型的, 这个分解就是 $A = UHV^T$ 和 $B = URV^T$ 这里 H 是上 Hessenberg（第一个子对角下面是 0), R 是上三角,U 和 V 是正交的。 如果 A 和 B 是复型的,分解就是 $A = UHV^H$ 和 $B = URV^H$ ，U 和 V 是酉矩阵, H 和 R 和前面的一样。 子程序 xGGHRD 计算这个分解, 它计算 H 和 R, 以及可以选择的 U 和/或 V. 注意这个和 xGEHRD 相对应（对标准非对称特征系问题来说), xGGHRD 不以因子的形式计算 U 和 V 。

2. 子程序 xHGEQZ 将矩阵对(H, R) 规约成广义 Schur 形式 $H = QSZ^T$ 和 $R = QTZ^T$（当 H 和 R 都是实型）或者 $H = QSZ^H$ 和 $R = QTZ^H$（当 H 和 R 都是复型）。 还计算出 α_i 和 β_i 的值,这里 $\lambda_i = \alpha_i / \beta_i$ 就是特征值。矩阵 Z 和 Q 可以有选择的计算或不计算。
3. 子程序 xTGEVC 计算矩阵对（S，T）的左和/或右特征向量。我们可以有选择的通过把 (UQ, VZ) （或 (Q, Z)）传到子程序 xTGEVC 将(S, T)的右特征向量转换为(A, B) 的右特征向量(或者是(H, R)的）。

4.1.6.8.2 平衡

xGGBAL 程序可以用来在矩阵对（A，B）规约成广义 Hessenberg 形式之前平衡矩阵对。平衡过程涉及两步，其中任何一步都是可以选择的：

1. 首先，xGGBAL 尝试等价变换置换（A，B）使其成为分块上三角形式：

$$P_1(A,B)P_2 = (A', B') = \left(\begin{bmatrix} A'_{11} & A'_{12} & A'_{13} \\ 0 & A'_{22} & A'_{23} \\ 0 & 0 & A'_{33} \end{bmatrix}, \begin{bmatrix} B'_{11} & B'_{12} & B'_{13} \\ 0 & B'_{22} & B'_{23} \\ 0 & 0 & B'_{33} \end{bmatrix} \right)$$

这里 P_1 和 P_2 是置换矩阵， A'_{11}， A'_{33}， B'_{11} 和 B'_{33} 是上三角的。 所以在矩阵对行和列的 ILO 到 IHI 的主对角块 A'_{22} 和 B'_{22} 的以外已经是广义 Schur 形式了。接着的操作 xGGBAL, xGGHRD 或 xHGEQZ 只需针对这些行和列。所以 ILO 和 IHI 被作为参数传到 xGGHRD 和 xHGEQZ。如果 ILO > 1 或 IHI < n 这可以节省很多的工作量。如果没有找到合适的置换(经常是这种情况)，xGGBAL 就设置 ILO = 1 和 IHI = n， A'_{22} 就是整个 A ， B'_{22} 就是整个 B。

2. 然后，xGGBAL 对 (A', B') 应用对角等价变换试着将关于特征值的矩阵范数变的小一些并尝试减小由于舍入带来的不精确结果[15]:

$$\begin{aligned} (A'', B'') &= D_1(A', B')D_2 \\ &= \begin{bmatrix} I & 0 & 0 \\ 0 & D'_1 & 0 \\ 0 & 0 & I \end{bmatrix} \left(\begin{bmatrix} A'_{11} & A'_{12} & A'_{13} \\ 0 & A'_{22} & A'_{23} \\ 0 & 0 & A'_{33} \end{bmatrix}, \begin{bmatrix} B'_{11} & B'_{12} & B'_{13} \\ 0 & B'_{22} & B'_{23} \\ 0 & 0 & B'_{33} \end{bmatrix} \right) \begin{bmatrix} I & 0 & 0 \\ 0 & D'_2 & 0 \\ 0 & 0 & I \end{bmatrix} \end{aligned}$$

在一些情况下可以提高后面操作的精度。

如果是用 xGGBAL 来平衡矩阵对(A, B)，那么后面接着的操作计算的特征向量就是平衡矩阵对(A′,B′)的特征向量。之后可以通过调用 xGGBAK 将它们转换成原始矩阵对(A, B) 的特征向量。注意这些操作可以在一些情况下可以提高后面操作的速度与精度；然而，对角变换那步可能偶尔会使矩阵束的范数变大而导致精度降低。

4.1.6.8.3 压缩子空间和条件数

广义 Schur 的形式取决于（S，T）对角线上的特征值的阶数这个可以被用户选择。假设对（S，T）的左上角，用户选择的是 $(\alpha_1,\beta_1),\ldots,(\alpha_j,\beta_j),1\le j\le n$。那么由 UQ 和 VZ 的前 j 列扩展的左和右压缩子空间就是与 $(\alpha_1,\beta_1),\ldots,(\alpha_j,\beta_j)$ 对应的。

下面的程序计算这个重组并且计算特征值的条件数，特征向量和压缩子空间：

1. xTGEXC 可以将在（S，T）广义 Schur 形式对角线上的特征值对（或者一对 2×2 分块）从原来的位置移动到任何一个位置。还可以选择在广义 Schur 形式中出现的特征值的阶数。这个重组是由正交（酉）变换矩阵实现的[16,17]。
2. xTGSYL 对 L 和解决广义 Sylvester 方程 $AR - LB = sC$ 和 $DR - LE = sF$，给定 A 和 B 是上(拟)三角的，D 和 E 是上三角的。还可以解决转置问题（复型情况下是共扼转置问题）$A^T X + D^T Y = sC$ 和 $-X B^T - Y E^T = sF$，对 X 和 Y。 在计算过程中为了避免上溢还可以设置扩展因子 s。用户可以选择的， xTGSYL 计算两个矩阵对(A,B) 和 (D,E)之间“分离”的 Frobenius norm-based 估计。 xTGSYL 被程序 xTGSNA 和 xTGSEN 用到，但是它也是独立的。
3. xTGSNA 计算矩阵对（S，T）广义 Schur 形式的特征值的条件数和/或左和右特征向量。这些同原始矩阵对（A，B）的特征值的条件数和特征向量是一样的，（S，T）就是由矩阵对（A，B）生成的。用户可以对所有的特征值（或者一个子集）的条件数和相关特征向量。详细信息参见[17]。
4. xTGSEN 可以移动广义 Schur 形式的矩阵对（S，T）的一个选择的子集至（S，T）的左上角，或者可以选择的计算这些值的平均值的条件数和相关的压缩子空间对。 这些和原始矩阵对（A，B）的特征值平均值的条件数和压缩子空间是一样的， (S,T) 也是由（A，B）生成的。详细信息参见[17]。

完整的程序列表请参见表 4.15，这里，为了节省空间，“广义”一词被省略了。

表 4.15：广义非对称特征系问题的计算函数

矩阵类型和存储机制	操作	单精度		双精度	
		实型	复型	实型	复型
一般	Hessenberg 规约	SGGHRD	CGGHRD	DGGHRD	ZGGHRD
	平衡	SGGBAL	CGGBAL	DGGBAL	ZGGBAL
	back transforming	SGGBAK	CGGBAK	DGGBAK	ZGGBAK
Hessenberg	Schur 分解	SHGEQZ	CHGEQZ	DHGEQZ	ZHGEQZ
(拟)三角	特征向量	STGEVC	CTGEVC	DTGEVC	ZTGEVC
	重组 Schur 分解	STGEXC	CTGEXC	DTGEXC	ZTGEXC

	Sylvester 方程	STGSYL	CTGSYL	DTGSYL	ZTGSYL
	特征值/向量的条件数	STGSNA	CTGSNA	DTGSNA	ZTGSNA
	特征值集合/压缩子空间的条件数	STGSEN	CTGSEN	DTGSEN	ZTGSEN

4.1.6.9 广义（或商）奇异值分解

m×n 的矩阵 A 和 p×n 的矩阵 B 的**广义（或商）奇异值分解**问题已经在 4.3.5 部分描述了。这一部分描述的程序，是用来计算分解的。计算过程按下面两部执行：

1. xGGSVP 用来把矩阵 A 和 B 约成三角形式：

$$U_1^T A Q_1 = \begin{array}{r} \\ k \\ l \\ m-k-l \end{array} \begin{array}{c} \begin{array}{ccc} n-k-l & k & l \end{array} \\ \begin{pmatrix} 0 & A_{12} & A_{13} \\ 0 & 0 & A_{23} \\ 0 & 0 & 0 \end{pmatrix} \end{array}$$

$$V_1^T B Q_1 = \begin{array}{r} \\ l \\ p-l \end{array} \begin{array}{c} \begin{array}{ccc} n-k-l & k & l \end{array} \\ \begin{pmatrix} 0 & 0 & B_{13} \\ 0 & 0 & 0 \end{pmatrix} \end{array}$$

这里 A_{12} and B_{13} 是非奇异上三角的， A_{23} 是上三角的。 如果 $m-k-l < 0$， $U_1^T A Q_1$ 的下面的 0 块是不出现的， A_{23} 是一个上梯形。 U_1, V_1 和 Q_1 是 正交矩阵(或酉矩阵如果 A 和 B 是复型的)。

2. 两个 1×1 上三角矩阵 A_{23} 和 B_{13} 的广义奇异值分解是通过 xTGSJA[2.2]计算的:

$$A_{23} = U_2 C R Q_2^T \quad \text{and} \quad B_{13} = V_2 S R Q_2^T \ .$$

这里 U_2, V_2 和 Q_2 是正交的(或酉) 矩阵, C 和 S 都是非负实对角矩阵满足 $C^2 + S^2 = I$， S 是非奇异的， R 是上三角并且非奇异的。

表 4.16：广义奇异值分解的计算程序

操作	单精度		双精度	
	实型	复型	实型	复型
A 和 B 的三角规约	SGGSVP	CGGSVP	DGGSVP	ZGGSVP
一对三角矩阵的 GSVD	STGSJA	CTGSJA	DTGSJA	ZTGSJA

对数值秩的确定由 xGGSVP 通过列选主元的 QR 分解计算。详细信息参见[8]。

两个三角矩阵的广义奇异值分解是由 xTGSJA 通过[7] 描述的类雅克比方法计算的。

4.1.7 参数列表与说明文档的约定

所有 LAPACK 程序的参数列表的设计与文档都遵守一个共同的协定。

LAPACK 所有驱动和计算程序的说明放在了第二部分。那些说明来自于代码部分的头部注释。但是为了节省空间，每个程序的实数和复数版本的说明放在了一起。

4.1.8 程序说明文档的约定

每个 LAPACK 程序的文档包括：

子程序或函数的声明，紧接着是参数类型和维数的声明。
程序功能的概要。
以参数列表中的顺序对每个参数的的描述
（可选）更详细的描述（只在代码部分，不在第二部分）
（可选）内部的参数（只在代码部分，不在第二部分）

4.1.9 参数次序

LAPACK 程序中参数按下列次序出现：

参数指定选项
问题维数
数组或者标量定义的输入参数；其中一些可能被返回结果回写
其它返回结果的数组或标量参数
Work 数组(和相关的维数)
诊断参数信息

4.1.9.1 参数描述

参数描述的形式用下面这个例子来说明：

N

(input) INTEGER

The number of columns of the matrix A. N $\geq$ 0.

A

(input/output) REAL array, dimension (LDA,N)
On entry, the m-by-n matrix to be factored.
On exit, the factors L and U from the factorization A = P*L*U; the unit diagonal elements of L are not stored.

下面是对每个参数的描述：

参数被分为：(input), (output), (input/output), (input or output) ,(workspace) 或 (workspace/output).
参数类型
（对数组来说）其维数
提供给参数的值的描述（如果是输入参数），或者返回给参数值的描述（如果是输出参数），或者二者都有（如果是输入/输出参数）。最后一种情况用“On entry”和 “On exit”来区别。
（如果是标量参数）提供的值所必须满足的强制条件（如：N≥0.）

4.1.9.2 参数选项

参数指定选项通常是 CHARACTER*1 型的。每个合法值的含意已经给出，如下面这个例子：

```
UPLO
        (input) CHARACTER*1
        = 'U': Upper triangle of A is stored;
        = 'L': Lower triangle of A is stored.
```

也可以用与之相对应的小写字母(有相同的含意),但是其它的值都是非法的(参见 2.3.1.9)

较长的字符串可以当作实参被传递，这样使得调用程序的可读性增强，但是只有第一个字母是有效的，这是标准 Fortran77 的用法。如下例：

```
CALL SPOTRS('upper', . . . )
```

4.1.9.3 问题的维数

对于问题的维数，传递 0 是允许的。在这种情况下，计算被忽略（或部分被忽略）。负维数被视为错误的。

4.1.9.4 数组参数

在参数列表中，每个二维数组变量后面紧跟主要维数，形式为 LD<array-name>.

如下例：

```
A
        (input/output) REAL/COMPLEX array, dimension (LDA,N)
```

...

LDA

(input) INTEGER

The leading dimension of the array A. LDA ≥/max(1,M).

除特别声明外，假设向量和矩阵分别是存储在一维和二维的数组中。如果一维数组 X[N]存储向量 x，那么 X[I]存储的是 x_i ，$i = 1, \ldots, n$.如果二维数组 A[LDA，N]存储的是 m*n 的矩阵 A，那么 A[i,j]存储的是 a_{ij}，$i = 1, \ldots, m$ ，$j = 1, \ldots, n$ (LDA 至少≥ m)。4.3.3 节将会对矩阵存给出更详细的说明。

注意，在程序中数组变量通常被声明成默认大小(第二个维数用*代替)。例如，REAL A(LDA, *)。尽管我们在文档中用(LDA,N)的形式给出维数，但是后面这种情况给出了更多的信息，因为它详细说明了最后一个维数所要求的最小值。但是为了克服标准 Fortran 77 中的一些局限，软件中使用了一个默认大小的数组声明。还特别允许程序能够在相关维数(在这种情况下为 N)为 0 时可以被调用。但是实数组的维数在调用程中最小为 1。(这个例子中为 LDA)。

4.1.9.5 工作数组

许多 LAPACK 程序要求传递一个或多个 work 数组当作参数。work 数组的名称通常为 WORK(有时也用 IWORK，RWORK，BWORK 来区别整型数组，实数型数组或布尔型数组)。有时，work 数组的第一个元素用来返回一些有用的信息。在这种情况下，变量被描述成(workspace/output)来代替(workspace).

许多的程序在实现块算法时要求有足够多的操作空间来存储矩阵的行或列。例如，操作空间的大小是 n*nb,nb 是块的大小。在这种情况下，work 数组的实际声明长度必须作为独立的变量 LWORK 来传递。在变量表中，LWORK 紧跟在 WORK 的后面。4.3.2 节将有更加深入的说明。

4.1.9.6 LWORK 查询

如果对提供给 LAPACK 程序的工作空间有疑问，用户可以选择 LWORK=-1，并且用返回在 WORK(1)中的值作为 LWORK 的正确值。设置 LWORK=-1 不会调用 XERBLA 提供的出错信息，并且定义成全局查询。

只有程序 xGEESX 和 xGGESX 是这个规则的例外。对这些程序，LWORK 的值是依赖于不变子空间的维数的(SDIM)，在程序入口处是不知道的。

4.1.9.7 出错处理和 INFO 参数的特征

所有的归档程序都有一个 INFO 参数来说明程序计算成功或失败，如下：

INFO = 0: 成功终结

INFO < 0: 一个或多个参数值非法——没有经过计算

INFO > 0: 计算过程中失败

所有驱动程序和辅助程序都会检查输入参数的类型，N，LDA 和选项参数都有特定类型值。如果第 i 个参数是非法类型，那么程序将置 INFO=-i,再调用错误处理程序 XERBLA。

标准的 XERBLA 会发出一个错误信息并且终止执行。因此，没有 LAPACK 程序能够在 INFO<0 的情况下返回到调用程序。但是如果用的是非标准的 XERBLA，那么就有可能会返回调用程序 。

4.1.10 确定块大小的块算法

实现块算法的 LAPACK 程序需要决定块大小。设计 LAPACK 的意图是尽可能对用户隐藏选择块的大小，但是同时能够让安装者在为特定的机器安装时能方便的进行设置。LAPACK 程序调用 ILAENV 辅助程序来获得最理想的块大小。返回值是通过 ILAENV 的参数传递的。我们所提供的 ILAENV 版本是在我们测试机上能达到最理想的情况，如果想要在你的机器中能达到最想的情况，那么必须调整 ILAENV 的设置。理论上，每台不同的机器都需要不同的 ILAENV 设置(见 2.4)。最理想的块大小同样也是依赖于程序，选项参数，以及问题规模的综合因素。

如果 ILAENV 返回块大小为 1，那么程序执行非块算法，调用 LEVE2 BLAS，而不调用 LEVEL3 BLAS.

一些 LAPACK 程序需要一个工作数组，它的大小同块大小成比例（见 4.3.1.7）。实际的工作数组作为参数 LWORK 被给出。参数 WORK 和 LWORK 典型描述如下：

WORK

(workspace) REAL array, dimension (LWORK)

On exit, if INFO = 0, then WORK(1) returns the optimal LWORK.

LWORK

(input) INTEGER

The dimension of the array WORK. LWORK $\geq$ max(1,N).

For optimal performance LWORK $\geq$ N*NB, where NB is the optimal block size returned by ILAENV.

程序由以下几步来决定块大小：

1. 通过调用 ILAENV 获得最理想的块大小。
2. 如果 LWORK 值表明分配了足够的工作空间，那么程序用最理想的块大小。
3. 否则，程序决定能够分配的最大块大小。
4. 如果新的块大小不小于最小值（由 ILAENV 返回），那么程序用新值。
5. 否则，程序调用非分块算法。

LWORK 的最小值返回在 WORK（1）.这个最小值在使用最理想的块大小时需要。

这样，程序就用提供的最大块，只要能够获得比非分块算法好的性能。WORK（1）不是 N,NB 的简单公式。

LWORK 为程序能返回正确的结果提供了最小值。如果提供的值小于了这个最小值，则表明没有足够的工作空间来实现分块算法，LWORK 的值被当作非法值，并且将它按照其它非法值一样处理。

如果对分配多小工作空间还有疑问，用户可以分配一个尽量大的值（如一个块 64），再检查出口处 WORK（1）的值。

4.1.11 矩阵存储机制

对于矩阵存储，LAPACK 提供了下列几种不同的存储机制。

1. 传统的二维数组存储方法；
2. 对称矩阵，厄米特矩阵，三角矩阵的压缩存储；
3. 对带状矩阵的带状存储；
4. 用三个或两个一维数组存储的三对角或双对角矩阵。

在下面的例子中，*号代表的是不必设置或不会被 LAPACK 程序访问到的数组元素。（不必设置的元素是那些不会被 LAPACK 程序读，写或访问到的元素）。

这些例子只说明了数组的相关部分，根据 FORTRAN77 传递数组参数的规则，数组参数还会有一些附加的行或列。

4.1.11.1 传统的存储方式

默认的矩阵存储机制是 4.3.1.6 小节中描述过的存储方式。一个矩阵 A 存放在一个二维数组 A 中。矩阵元素 a_{ij} 存储在数组元素 A(i,j)中。

如果一个矩阵是三角矩阵（参数 UPLO 来指定是上三角或下三角），只有三角矩阵的相关元素被访问，其它的元素不必设置（用*代替）。

例如：n=4.

UPLO	三角矩阵 A	矩阵 A 的存储
`U'	$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ & a_{22} & a_{23} & a_{24} \\ & & a_{33} & a_{34} \\ & & & a_{44} \end{pmatrix}$	$\begin{matrix} a_{11} & a_{12} & a_{13} & a_{14} \\ * & a_{22} & a_{23} & a_{24} \\ * & * & a_{33} & a_{34} \\ * & * & * & a_{44} \end{matrix}$
`L'	$\begin{pmatrix} a_{11} & & & \\ a_{21} & a_{22} & & \\ a_{31} & a_{32} & a_{33} & \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix}$	$\begin{matrix} a_{11} & * & * & * \\ a_{21} & a_{22} & * & * \\ a_{31} & a_{32} & a_{33} & * \\ a_{41} & a_{42} & a_{43} & a_{44} \end{matrix}$

相同的，如果是上 Hessenberg 矩阵，则第一个子对角线以下的元素也不必设置。程序解决对称矩阵或厄米特矩阵时允许用上三角矩阵或下三角矩阵存储相关的数组，其余的元素不必设置。

例如：n=4.

UPLO	厄米特 矩阵 ***A***	矩阵 A 的存储
`U'	$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ \bar{a}_{12} & a_{22} & a_{23} & a_{24} \\ \bar{a}_{13} & \bar{a}_{23} & a_{33} & a_{34} \\ \bar{a}_{14} & \bar{a}_{24} & \bar{a}_{34} & a_{44} \end{pmatrix}$	$\begin{matrix} a_{11} & a_{12} & a_{13} & a_{14} \\ * & a_{22} & a_{23} & a_{24} \\ * & * & a_{33} & a_{34} \\ * & * & * & a_{44} \end{matrix}$
`L'	$\begin{pmatrix} a_{11} & \bar{a}_{21} & \bar{a}_{31} & \bar{a}_{41} \\ a_{21} & a_{22} & \bar{a}_{32} & \bar{a}_{42} \\ a_{31} & a_{32} & a_{33} & \bar{a}_{43} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix}$	$\begin{matrix} a_{11} & * & * & * \\ a_{21} & a_{22} & * & * \\ a_{31} & a_{32} & a_{33} & * \\ a_{41} & a_{42} & a_{43} & a_{44} \end{matrix}$

4.1.11.2 压缩存储机制

如果相关的三角矩阵按列存储在一维数组中，那么对称矩阵，厄米特矩阵，三角矩阵可以更为紧密。在 LAPACK 中，用来压缩存储机制下的矩阵存储的数组以“P”结尾。因此：

if UPLO = `U', a_{ij} 存储在 AP(***i+j(j-1)/2***) ， $i \le j$;

if UPLO = `L', a_{ij} 存储在 AP(***i+(2n-j)(j-1)/2***) ， $j \le i$.

例如：

UPLO	三角矩阵 ***A***	数组 AP 中的压缩存储
`U'	$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ & a_{22} & a_{23} & a_{24} \\ & & a_{33} & a_{34} \\ & & & a_{44} \end{pmatrix}$	$a_{11}\ \underbrace{a_{12}\ a_{22}}\ \underbrace{a_{13}\ a_{23}\ a_{33}}\ \underbrace{a_{14}\ a_{24}\ a_{34}\ a_{44}}$
`L'	$\begin{pmatrix} a_{11} & & & \\ a_{21} & a_{22} & & \\ a_{31} & a_{32} & a_{33} & \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix}$	$\underbrace{a_{11}\ a_{21}\ a_{31}\ a_{41}}\ \underbrace{a_{22}\ a_{32}\ a_{42}}\ \underbrace{a_{33}\ a_{43}}\ a_{44}$

注意：对于复型和实型对称矩阵，上三角矩阵按列压缩存储等同于下三角矩阵按行压缩存储；下三角矩阵按列压缩存储等同于上三角矩阵按行压缩存储。对于复型厄米特矩阵，上三角矩阵按列同于下三角矩阵按行。下三角矩阵按列同于上三角矩阵按行。

4.1.11.3 带状存储机制

一个 m×n 的带形矩阵，kl 个下次对角线，ku 个上斜对角线，可以存储在一个 kl+ku 列 n 行的二维数组中。矩阵的列存储在数组相关 r 的列中，矩阵的斜对角线被存储在数组的行中。尽管 LAPACK 程序对于所有的 kl,ku 都正确，这种存储机制在实际中只有 kl,ku<<min(m,n) 时运用。带状存储的数组以“B”结尾。

为了精确 a_{ij} 存储在 AB(***ku*+1+*i*-*j*,*j***) ，$\max(1, j-ku) \le i \le \min(m, j+kl)$.

例如：当 $m = n = 5, kl = 2$ 和 $ku = 1$:

带状矩阵 ***A***	数组 AB 中的带状存储
$\begin{pmatrix} a_{11} & a_{12} & & & \\ a_{21} & a_{22} & a_{23} & & \\ a_{31} & a_{32} & a_{33} & a_{34} & \\ & a_{42} & a_{43} & a_{44} & a_{45} \\ & & a_{53} & a_{54} & a_{55} \end{pmatrix}$	$\begin{matrix} * & a_{12} & a_{23} & a_{34} & a_{45} \\ a_{11} & a_{22} & a_{33} & a_{44} & a_{55} \\ a_{21} & a_{32} & a_{43} & a_{54} & * \\ a_{31} & a_{42} & a_{53} & * & * \end{matrix}$

在 AB 数组中以*标记的元素不必设置，并且不会被 LAPACK 程序访问到。

注意：当一个带形矩阵被用来做 LU 因式分解时，所提供的空间必须能够存储下由 fill-in 产生的用来做结果交换的 kl 个上斜对角线。意思就是矩阵按上述机制存储，加上 kl+ku 个上斜对角线。

三角带形矩阵以相同的格式存储.当是上三角阵时，kl=0;下三角阵时，ku=0.对于有 kd 个上斜或下斜对角线的对称或厄米特带形矩阵，只有上三角或下三角需要存储。

如 果 UPLO = `U', a_{ij} 存储在 AB(***kd*+1+*i*-*j*,*j***) ，$\max(1, j-kd) \le i \le j$;

如果 UPLO = `L', a_{ij} 存储在 AB(**1+*i*-*j*,*j***) ，$j \le i \le \min(n, j+kd)$.

例如：n=5,kd=2:

UPLO	厄米特 带状矩阵 ***A***	数组 AB 中的带状存储
`U'	$\begin{pmatrix} a_{11} & a_{12} & a_{13} & & \\ \bar{a}_{12} & a_{22} & a_{23} & a_{24} & \\ \bar{a}_{13} & \bar{a}_{23} & a_{33} & a_{34} & a_{35} \\ & \bar{a}_{24} & \bar{a}_{34} & a_{44} & a_{45} \\ & & \bar{a}_{35} & \bar{a}_{45} & a_{55} \end{pmatrix}$	$\begin{matrix} * & * & a_{13} & a_{24} & a_{35} \\ * & a_{12} & a_{23} & a_{34} & a_{45} \\ a_{11} & a_{22} & a_{33} & a_{44} & a_{55} \end{matrix}$

`L'	$\begin{pmatrix} a_{11} & \bar{a}_{21} & \bar{a}_{31} & & \\ a_{21} & a_{22} & \bar{a}_{32} & \bar{a}_{42} & \\ a_{31} & a_{32} & a_{33} & \bar{a}_{43} & \bar{a}_{53} \\ & a_{42} & a_{43} & a_{44} & \bar{a}_{54} \\ & & a_{53} & a_{54} & a_{55} \end{pmatrix}$	$\begin{matrix} a_{11} & a_{22} & a_{33} & a_{44} & a_{55} \\ a_{21} & a_{32} & a_{43} & a_{54} & * \\ a_{31} & a_{42} & a_{53} & * & * \end{matrix}$

4.1.11.4 三对角和双对角矩阵

非对称的三对角矩阵存储在三个一维数组中，其中一个数组大小为 n，用来存储对角线元素，另外两个大小为 n-1，用来存储对角线两侧的元素。

对称的三对角矩阵存储在两个一维数组中，其中一个数组大小为 n，用来存储对角线元素。另外一个大小为 n-1，用来存储非对角线元素。

4.1.11.5 单位三角矩阵

一些 LAPACK 程序有处理单位三角矩的选项参数（单位三角矩阵的对角线元素为 1）。选项参数为 DIAG，如果 DIAG=“U”，则对角线元素不必存储，同时相应的数组元素不会被 LAPACK 程序访问。其它矩阵的存储机制没有改变。

4.1.11.6 复矩阵的实对角线元素

按定义，复厄米特矩阵拥有纯实数的对角线矩阵。一些被 LAPACK 程序计算的复三角矩阵需要有实对角线元素(如：cholesky 和 QR 因数分解)。

如果提供给 LAPACK 程序作为输入元素的矩阵，对角线元素的虚部不会被访问，但是设为 0。从 LAPACK 程序返回时，计算过后的虚部被设为 0。

4.1.12 正交或酉矩阵的表示

实正交矩阵或复酉矩阵（用 Q 标示）在 LAPACK 被描述成初等反射矩阵----也被描述成基本 Householder 矩阵（Hi）。

例如：

$$Q = H_1 H_2 \dots H_k.$$

大多数用户不需要关心这些细节，因为 LAPACK 程序提供了处理这种表示的机制：

程序名以 SORG- (real) 或 CUNG- (complex) 开始的可以产生全部或部份 $\boldsymbol{Q}$;

程序名以 SORM- (real) 或 CUNM- (complex) 开始的能够由 $\boldsymbol{Q}$ 或 $\boldsymbol{Q^H}$ 产生一个特定的矩阵。

一个基本 n 维参照矩阵 H 是下面形式的酉矩阵。

$$H = I - \tau v v^H$$

τ 是一个标量，v 是一个 n 维向量，$|\tau|^2||v||_2^2 = 2\mathrm{Re}(\tau)$；v 代表 Householder 向量。通常 v 有很多 0 元素。但是为了讨论，我们假设 H 没有这样特别的结构。

在实运算中，$1 \le \tau \le 2$，除了 $\tau = 0$ 隐含 H=I；

在复算法中，τ 可能是复数，并且满足 $1 \le \mathrm{Re}(\tau) \le 2$ 和 $|\tau - 1| \le 1$。复 H 矩阵不是厄米特矩阵，但是是酉矩阵。允许 τ 为复型的好处是，给定任意一个向量 x，H 可以利用 β 计算出 $H^H x = \beta(1,0,\ldots,0)^T$。这很有用。例如，从一个复厄米特转变为实对称三角矩阵或者从一个复矩形矩阵转变为实双对角矩阵。

4.1.13 注意事项

LAPACK 中需要知道如下几点：

1．双精度复型程序（名字以 Z 开头）使用一个 COMPLEX*16 的数据类型。这是作为 Fortran 77 标准的扩充，但在很多双精度计算很常用的机器上的 Fortran 编译器都支持它。下列相关扩充仍被使用：

- 内置函数 DCONJG，参数和结果的类型都是 COMPLEX*16；
- 内置函数 DBLE 和 DIMAG， COMPLEX*16 类型的参数和双精度类型的结果，分别返回实数和虚数部分；
- 内置函数 DCMPLX，双精度类型的参数和 COMPLEX*16 类型的结果；
- COMPLEX*16 常量，是由一对双精度常量放于圆括号内所组成。

一些编译器支持 DOUBLE COMPLEX 来取代 COMPLEX*16，内置函数 DREAL 代替 DBLE 来返回一个 COMPLEX*16 参数的实数部分。如果编译器不接受在 LAPACK 中使用的结构，则安装者不得不修改代码：例如，全局修改 COMPLEX*16 为 DOUBLE COMPLEX，或者有选择性的修改 DBLE 为 DREAL。

2．为了获得最佳性能，必须为每台机器、甚至为给定机器的每个组件都设置一个小的调谐参数集（例如，不同参数使不同数量的处理器性能达到最佳）。比如块大小，最小块大小，交叉点（在它下面使用一个未分块程序）等等这些数据，可以通过调用一个查询函数 ILAENV 来设置。LAPACK 中提供的 ILAENV 默认版本使用的值通常能达到让人满意的性能，但是那些对性能特别感兴趣的用户可能希望更改子程序或者替换为他们自己的版本。关于在某个特定环境中设置 ILAENV 的进一步细节将在 4.4.2 节中给出。

4.1.14 安装 ILAENV

与机器独立的参数（如块大小）可以通过调用查询函数来设置，对不同的机器设置不同的参数。环境查询函数的声明为：

INTEGER FUNCTION ILAENV(ISPEC, NAME, OPTS, N1, N2, N3, N4)

其中 ISPEC，N1，N2，N3 和 N4 是整型变量，NAME 和 OPTS 是 CHARACTER*(*)类型。NAME 指定了子程序的名字：OPTS 是用于选择子程序的一个字符串；N1-N4 表示问题的维数。ISPEC 表示被返回的参数；下面是 LAPACK 正在使用的数值：

ISPEC = 1: NB, 最优块大小
= 2: NBMIN,分块程序使用的最小块大小
= 3: NX,交叉点（在分块程序中，如果 N < NX，应使用未分块程序）
= 4: NS,移位数
= 6: NXSVD 是个阀值，根据这个阀值来优先使用 QR 分解规约到双对角形式。如果 M > NXSVD • N，那么执行 QR 分解。
= 8: MAXB, 分块多移位 QR 的交叉点。
= 9: SMLSIZ, 分治算法中的计算树底层子问题的最大数量。
= 10: NAN, IEEE NaN 不能被捕捉。
= 11: INFINITY, infinity 不能被捕捉。

三个块大小参数 NB，NBMIN，NX，在很多不同的子程序中被使用（见表 4.1）。NS 和 MAXB 在分块多多移位 QR 算法和 xHSEQR 中被使用。NXSVD 在驱动程序 xGELSS 和 xGESVD 中被使用。SMLSIZ 在分治程序 xBDSDC ，xGELSD, xGESDD, 和 xSTEDC 中被使用。参数 NAN 和 INFINITY 在驱动程序 xSTEVR 和 xSYEVR/xCHEEVR 中被使用来检查是否依照 IEEE-754 标准。如果检测出是依赖，那么这些驱动程序就会调用 xSTEGR。否则，就会选择一个比较缓慢的算法。

表 4.17:LAPACK 中块参数 NB, NBMIN 和 NX 的使用

real	complex	NB	NBMIN	NX
SGBTRF	CGBTRF	●		
SGEBRD	CGEBRD	●	●	●
SGEHRD	CGEHRD	●	●	●
SGELQF	CGELQF	●	●	●
SGEQLF	CGEQLF	●	●	●
SGEQRF	CGEQRF	●	●	●
SGERQF	CGERQF	●	●	●
SGETRF	CGETRF	●		

SGETRI	CGETRI	●	●	
SORGLQ	CUNGLQ	●	●	●
SORGQL	CUNGQL	●	●	●
SORGQR	CUNGQR	●	●	●
SORGRQ	CUNGRQ	●	●	●
SORMLQ	CUNMLQ	●	●	
SORMQL	CUNMQL	●	●	
SORMQR	CUNMQR	●	●	
SORMRQ	CUNMRQ	●	●	
SPBTRF	CPBTRF	●		
SPOTRF	CPOTRF	●		
SPOTRI	CPOTRI	●		
SSTEBZ		●		
SSYGST	CHEGST	●		
SSYTRD	CHETRD	●	●	●
SSYTRF	CHETRF	●	●	
	CSYTRF	●	●	
STRTRI	CTRTRI	●		

LAPACK 的测试和计时程序使用 ILAENV 的一个特殊版本，在这个版本中参数通过一个公共块接口来设置。这给使用不同大小的块进行实验提供了方便，即可以尝试运行代码的不同部分，以及比较不同参数值的相对性能。

LAPACK 中计时程序用来为所有在表 4.1 中的程序收集数据。要确定问题大小的范围就需要确定最佳块大小或者交叉点，这个范围是独立于机器的，但是初始可以使用由 LAPACK 测试和计时包提供的输入文件。因为子程序需要一个交叉点，最好是初始时找到一个最优块大小且交叉点设置为 0，然后找到使分块算法的性能比未分块算法好的那个点。找到的最好交叉点应该比未分块和分块方法的曲线交叉点的值稍微小一些。

例如，SGEQRF 在一台 CRAY-2 的单处理机上，对 $N = 176$ 和 $N = 192$ 之间的方阵，观察到 NB = 32 时块大小较好，这时的分块算法的性能超越了非分块算法。对交叉点从 64 到 192 进行试验，发现当 NX = 128 时是个很好的选择，尽管 NX 从 3*NB 变化到 5*NB，但其结

果都基本相似。这意味着对于 $N \leq 128$ 的矩阵都使用非分块算法，而对于 $N > 128$ 的矩阵则应该使用分块更新，直到剩余的子矩阵阶数少于 128。

通过用一些小的值试验块大小，发现它应该直接选择 NBMIN，这个最小的块大小相对于未分块算法得到了一个性能的改进。在一些机器上，最优块大小也许是 1（未分块算法得到最优性能）；在这个情况下，选择 NBMIN 太武断。假如工作空间不足，ILAENV 的原型版本设置 NBMIN 为 2，以便每次都完成模块化，尽管这会导致一个分块程序有很差的性能。

正交分解程序以及 SGEBRD 接受非方阵作为输入，确定最优参数会变得复杂。LAPACK 计时程序允许 M 和 N 独立地变化。我们已经确定矩阵形状一般对最优块大小没什么影响，但是交叉点更依赖于矩阵的形状。例如，如果在 *QR* 分解中 $M >> N$，在剩下的子矩阵中块更新可能总是比未分块更新更快，所以如果 $M \geq 2N$ 可以设置 NX = NB。

移位数的参数值用来调整块的多移位 QR 算法，从输入文件到特征值计时程序，这些参数值不同。特别地，xHSEQR 的性能对于正确选择块参数相当敏感。

4.2 辅助子程序索引

1. 以实型(单精度)程序名字(通常以 S－开始)的字母数字顺序列出了程序。(LAPACK 命名的方法详见 4.2.3。)
2. 将首先列出那些没有与之相对应的实型程序的复型程序；在 Level 1 或 Level 2 BLAS 有相对应的实型程序的将用斜体列出(例如,CROT)。
3. 这里没有列出双精度程序；它们的名字以 D－代替 S－,Z－代替 C－。只有几个程序没有使用这个简单的规则,它们是双精度版本的 ICMAX1,SCSUM1 和 CSRSCL,它们被命名为 IZMAX1,DZSUM1 和 ZDRSCL。
4. 列表中的一些程序具有独立数据类型的名字：ILAENV, LSAME, LSAMEN 和 XERBLA。
5. 这个索引仅仅给出了每一个程序用途的简单描述。如需详细信息请参考代码里面的开始注释部分,它们也是用驱动和计算程序的格式写出来的。

程序		描述
实型	复型	
	CLACGV	将一复型向量共扼
	CLACRM	实现矩阵相乘 $C = A * B$,A 是复型,B 是实型,C 是复型。
	CLACRT	实现变换 $\begin{pmatrix} c & s \\ -s & c \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$,c,s,x 和 y 均是复型。
	CLAESY	计算 2×2 复型对称矩阵的特征值和特征向量,并且检查特征向量的矩阵范数是否大于一个 threshold 值

	CROT	将一对复型向量应用实型 cosine 和复型 sine 的平面旋转
	CSPMV	计算 矩阵－向量 相乘 $y = \alpha Ax + \beta y$ ，α 和 β 是复型标量，x 和 y 是复型向量，A 是压缩存储的复型对称矩阵。
	CSPR	实现对称 秩-1 修正 $A = \alpha xx^T + A$，α 是复型标量，x 是复型向量，A 是压缩存储的复型对称矩阵。
	CSROT	将一对复型向量应用实型 cosine 和 sine 的平面旋转
	CSYMV	计算 矩阵－向量 相乘 $y = \alpha Ax + \beta y$ ，α 和 β 是复型标量，x 和 y 是复型向量，A 是复型对称矩阵。
	CSYR	实现对称 秩-1 修正 $A = \alpha xx^T + A$，α 是复型标量，x 是复型向量，A 是复型对称矩阵。
	ICMAX1	找到一个元素的索引它的实数部分具有最大的绝对值(类似 Level 1 BLAS ICAMAX，但是用的是实部分的最大绝对值)
ILAENV		环境查询函数，返回一个值可以用来调整算法性能。
LSAME		忽略大小写测试两个字符是否相等。
LSAMEN		忽略大小写测试两个字符串是否相等。
	SCSUM1	生成一个实向量的 1－范数(类似 Level 1 BLAS SCASUM，但是用的是实际的绝对值)
SGBTF2	CGBTF2	计算一般带状矩阵的 LU 分解，用的是行交换的部分选主元(非分块算法)
SGEBD2	CGEBD2	用 正交/酉 变换将一个一般矩形矩阵规约成双对角形式(非分块算法)。
SGEHD2	CGEHD2	用 正交/酉 相似变换将一个一般矩阵规约成上 Hessenberg 形式(非分块算法)。
SGELQ2	CGELQ2	计算一个一般矩形矩阵的 LQ 分解(非分块算法)。

Routine		Description
real	complex	

SGEQL2	CGEQL2	计算一个一般矩形矩阵的 QL 分解(非分块算法)。
SGEQR2	CGEQR2	计算一个一般矩形矩阵的 QR 分解(非分块算法)。
SGERQ2	CGERQ2	计算一个一般矩形矩阵的 QR 分解(非分块算法)。
SGESC2	CGESC2	解决线性方程 $A * X = scale * RHS$ 用通过 xGETC2 计算的全选主元的 LU 分解
SGETC2	CGETC2	对一般 n×n 矩阵通过全选主元计算 LU 分解
SGETF2	CGETF2	计算一般矩阵的 LU 分解,用行交换的部分选主元(非分块算法)。
SGTTS2	CGTTS2	对三对角矩阵 A 利用 SGTTRF/CGTTRF 计算的 LU 分解,解决一类 $A * X = B$ 或者 $A^H * X = B$ 的方程
SLABAD		如果指数域太大则返回下溢和上溢 threshold 值的方根
SLABRD	CLABRD	通过正交/酉变换将一个一般矩形矩阵的前 nb 行和列规约成实双对角形式,并且返回辅助矩阵,它们将会被 A 还没实行规约变换的部分用到
SLACON	CLACON	Estimates the 1-norm of a square matrix, using reverse communication for evaluating matrix-vector products. 估计方阵的 1－范数,用 reverse communication 计算矩阵－向量乘。
SLACPY	CLACPY	拷贝一个 二维数组的全部或一部分至另一个
SLADIV	CLADIV	在一个实际算法中实现复型相除,避免不必要的溢出。
SLAE2		计算一个 2×2 对称矩阵的特征值。
SLAEBZ		计算实对称三对角矩阵特征值的个数,小于或等于一个给定值,并且实现 SSTEBZ 程序所需要的其他任务。
SLAED0	CLAED0	被 xSTEDC 用到。用分治方法计算一个非规约对称三对角矩阵的所有特征值和相对应的特征向量。
SLAED1		被 SSTEDC 用到。计算一个被 秩－1 对称矩阵修正过的对角矩阵的修正特征系。 在原始矩阵是三对角矩阵的时候被用到。
SLAED2		被 SSTEDC 用到。合并特征值,压缩特征方程。在原始矩阵是三对角矩阵的时候用到。

SLAED3		被 SSTEDC 用到。找到特征方程的根并且修正特征向量。在原始矩阵是三对角矩阵的时候用到。
SLAED4		被 SSTEDC 用到。找到特征方程的单根

Routine		Description
real	complex	
SLAED5		被 SSTEDC 用到。解决 2×2 特征方程。
SLAED6		被 SSTEDC 用到。在特征方程的解法中计算一个 Newton 步。
SLAED7	CLAED7	被 SSTEDC 用到。计算一个被 秩－1 对称矩阵修正过的对角矩阵的修正系。在原始矩阵是稠密矩阵的时候被用到。
SLAED8	CLAED8	被 xSTEDC 用到。合并特征值,压缩特征方程。在原始矩阵是稠密矩阵的时候用到。
SLAED9		被 SSTEDC 用到。找到特征方程的根并且修正特征向量。在原始矩阵是稠密矩阵的时候用到。
SLAEDA		被 SSTEDC 用到。计算 Z 向量以确定对角矩阵的秩－1 修正。在原始矩阵是稠密矩阵的时候用到。
SLAEIN	CLAEIN	用反迭代方法计算一个上 Hessenberg 矩阵的指定左或右特征向量。
SLAEV2	CLAEV2	计算一个 2×2 对称/ 厄米特矩阵的特征值和特征向量。
SLAEXC		通过正交相似变换用 Schur canonical form 交换一个实上半对角矩阵相邻的对角块。
SLAG2		计算一个 2×2 一般特征值问题 $A - w * B$ 的特征值,如果需要避免上溢/下溢可以扩展。
SLAGS2		计算 2×2 正交矩阵 U,V 和 Q,并应用它们到矩阵 A 和 B 以使变换后的 A 和 B 的行是并行的。
SLAGTF		用行交换的部分选主元计算矩阵 $(T - \lambda I)$ 的 LU 分解,T 是一般的三对角矩阵, λ 是一个标量。
SLAGTM	CLAGTM	实现矩阵－矩阵相乘 $C = \alpha AB + \beta C$,A 是一个三对角矩阵,B 和 C 是矩形矩阵, α 和 β 是标量,可能为 0,1 或者-1。

SLAGTS		利用 SLAGTF 计算的 LU 分解,解决一类 $(T-\lambda I)x=y$ 或者 $(T-\lambda I)^T x=y$ 的方程系,T 是一个一般三对角矩阵, λ 是一个标量。
SLAGV2		计算一个实 2×2 矩阵的广义 Schur 分解,当 B 实上三对角时 pencil(A,B)
SLAHQR	CLAHQR	Computes the eigenvalues and Schur factorization of an upper Hessenberg matrix, using the double-shift/single-shift *QR* algorithm. 用 double-shift/single-shift *QR* 算法计算一个上 Hessenberg 矩阵的特征值和 Schur 分解。
SLAHRD	CLAHRD	通过正交/酉变换将一个一般矩形矩形矩阵的前 nb 列规约成第 K 个子对角块的下面元素均为 0,并且返回辅助矩阵,它们将会被 A 还没实行规约变换的部分用到
SLAIC1	CLAIC1	实现条件数估计的一步增加。

Routine		Description
real	complex	
SLALN2		解决一类形如 $(\gamma A-\lambda D)x=\sigma b$ 或 $(\gamma A^T-\lambda D)x=\sigma b$ 的 1×1 或 2×2 方程系,D 是对角方程, λ ,b 和 x 可能是复型, σ 是标量因子以防止上溢。
SLALS0	CLALS0	被 xGELSD 所使用,在采用分治奇异值分解方法求解的最小二乘问题时,将添加一行的对角矩阵的左或右奇异值向量矩阵的乘子应用回右端项矩阵 B。
SLALSA	CLALSA	被 xGELSD 用到。以压缩形式计算系数矩阵的 SVD 时计算最小二乘问题的一个中间步。
SLALSD	CLALSD	被 xGELSD 用到。用 A 的奇异值分解去解最小二乘问题,以确定 X 最小化 $A*X-B$ 每一列的 the Euclidean 范数。
SLAMCH		对于浮点数算法确定机器参数。

SLAMRG		将两个分别独立有序的条目以升序合并成一个集合并生成一个置换序列。
SLANGB	CLANGB	对于一个一般带状矩阵返回 1－范数,Frobenius－范数,无穷范数的值或者所有元素的最大绝对值。
SLANGE	CLANGE	对于一个一般矩形矩阵返回 1－范数,Frobenius－范数,无穷范数的值或者所有元素的最大绝对值。
SLANGT	CLANGT	对于一个一般三对角矩阵返回 1－范数,Frobenius－范数,无穷范数的值或者所有元素的最大绝对值。
SLANHS	CLANHS	对于一个上 Hessenberg 阵返回 1－范数,Frobenius－范数,无穷范数的值或者所有元素的最大绝对值。
SLANSB	CLANSB CLANHB	对于一个实对称/复对称/复厄米特带状矩阵返回 1－范数,Frobenius－范数,无穷范数的值或者所有元素的最大绝对值。
SLANSP	CLANSP CLANHP	对于一个以压缩格式存储的实对称/复对称/复厄米特矩阵返回 1－范数,Frobenius－范数,无穷范数的值或者所有元素的最大绝对值。
SLANST	CLANHT	对于一个对称/厄米特三对角矩阵返回 1－范数,Frobenius－范数,无穷范数的值或者所有元素的最大绝对值。
SLANSY	CLANSY CLANHE	对于一个实对称/复对称/复厄米特矩阵返回 1－范数,Frobenius－范数,无穷范数的值或者所有元素的最大绝对值。
SLANTB	CLANTB	对于一个三对角带状矩阵返回 1－范数,Frobenius－范数,无穷范数的值或者所有元素的最大绝对值。

Routine		Description
real	complex	
SLANTP	CLANTP	对于一个以压缩格式存储的三对角矩阵返回 1－范数,Frobenius－范数,无穷范数的值或者所有元素的最大绝对值。
SLANTR	CLANTR	对于一个三对角矩阵返回 1－范数,Frobenius－范数,无穷范数的值或者所有元素的最大绝对值。
SLANV2		对于一个 2×2 非对称矩阵以 Schur canonical form 计算 Schur 分解。
SLAPLL	CLAPLL	测试两个向量 X 和 Y 的线性依赖。

SLAPMT	CLAPMT	对于矩阵的列实行前向或后相置换。
SLAPY2		返回 $\sqrt{x^2+y^2}$，避免不必要的上溢或有害的下溢。
SLAPY3		返回 $\sqrt{x^2+y^2+z^2}$，避免不必要的上溢或有害的下溢。
SLAQGB	CLAQGB	用 SGBEQU/CGBEQU 计算得到的行和列扩展因子扩展一般带状矩阵。
SLAQGE	CLAQGE	用 SGBEQU/CGBEQU 计算得到的行和列扩展因子扩展一般矩形矩阵。
SLAQP2	CLAQP2	对块 *A*(*OFFSET*+1:*M*, 1:*N*)的列选主元计算 QR 分解。块 *A*(*OFFSET*+1:*M*, 1:*N*) 也相对的列选主元但不分解。
SLAQPS	CLAQPS	对于一个实 M×N 矩阵 A 的列选主元用 Level 3 BLAS 计算 QR 分解。
SLAQSB	CLAQSB	用 SGBEQU/CGBEQU 计算得到的扩展因子扩展对称/厄米特带状矩阵。
SLAQSP	CLAQSP	用 SGBEQU/CGBEQU 计算得到的扩展因子扩展以压缩形式存储的对称/ 厄米特矩阵。
SLAQSY	CLAQSY	用 SGBEQU/CGBEQU 计算得到的扩展因子扩展对称/厄米特带状矩阵。
SLAQTR		用实精度解决一类实半对角方程或是特殊形式的复半对角方程。
SLAR1V	CLAR1V	在行 B1 通过三对角矩阵 $LDL^T-\sigma I$ 的 BN 计算一个子矩阵的逆的第 r(扩展的)列。
SLAR2V	CLAR2V	用一个实 cosine 和 实/复 sines 平面旋转向量从两边对于一个 2×2 序列对称/ 厄米特矩阵实行变换。
SLARF	CLARF	对于一个一般矩形矩阵实行初等 reflector。
SLARFB	CLARFB	对于一个一般矩形矩阵实行块 reflector 或是它的转置/共扼转置。

SLARFG	CLARFG	生成一个初等 reflector(Householder 矩阵)。
SLARFT	CLARFT	形成一个块 reflector $H = I - V T V^H$ 的三角因子 T。

Routine		Description
real	complex	
SLARFX	CLARFX	对于一个一般矩形矩阵应用初等 reflector,如果 reflector 阶数小于等于 10 就循环展开。
SLARGV	CLARGV	生成一个实 cosine 和实/复 sines 的平面旋转向量。
SLARNV	CLARNV	从均匀分布或正态分布返回一个随机数的向量。
SLARRB		给定一个相对健壮的表示(RRR)$L D L^T$, SLARRB 限定对分以确定 $L D L^T$的特征值, W(IFIRST) 通过 W(ILAST),以便更精确。
SLARRE		给定一个三对角矩阵 T,SLARRE 将比较小的不在对角线上的元素置为0,对于每一个未规约的块 T_i,它找到一些数 σ_i,基 $T_i - \sigma_i I = L_i D_i L_i^T$ 表示和每一个 L_i D_i L_i^T的特征值。
SLARRF		找到一个相对健壮的表示 $LDL^T - \Sigma I = L(+)D(+)L(+)^T$ 以便至少 $L(+)$ $D(+)$ $L(+)^T$的一个特征值是相对独立的。
SLARRV	CLARRV	给定 L,D 和特征值 $L D L^T$,计算三对角矩阵 T=L $D L^T$ 的特征向量。
SLARTG	CLARTG	用是 cosine 和实/复 sine 生成一个平面旋转。
SLARTV	CLARTV	将实 cosine 和实/复 sine 的平面旋转向量应用到一对向量的元素操作。
SLARUV		返回一个从均匀(0,1)分布($n \leq 128$)生成的 n 个随机数的实向量。
SLARZ	CLARZ	应用一个初等 reflector(就像 xTZRZF 返回的)到一个一般矩阵。
SLARZB	CLARZB	应用一个块 reflector 或它的转置/共扼转置到一个一般矩阵。
SLARZT	CLARZT	生成一个块 reflector $H = I - V T V^H$的三角向量 T。

SLAS2		计算 2×2 三对角矩阵的奇异值。
SLASCL	CLASCL	用一个定义为 c_{to}/c_{from} 的标量去乘一个一般矩形矩阵。
SLASD0		被 SBDSDC 用到。通过分治方法计算实上双对角 n×m 矩阵的奇异值,其中矩阵为对角线 D 和不在对角线的 E,其中 $M = N + SQRE$。
SLASD1		被 SBDSDC 用到。计算实上双对角 n×m 矩阵的 SVD,其中 $N = NL + NR + 1$ 并且 $M = N + SQRE$。
SLASD2		被 SBDSDC 用到。将两组奇异值集合合并为一个有序序列,然后尽量压缩问题规模。
SLASD3		被 SBDSDC 用到。找到所有特征方程根的方根,就象在 D,Z 中定义的一样,然后用矩阵乘修正奇异向量。

Routine		Description
real	complex	
SLASD4		被 SBDSDC 用到。计算一个正定对称秩－1 修正为正定对角矩阵的第 I 个修正特征值的方根。
SLASD5		被 SBDSDC 用到。计算一个 2×2 对角矩阵的正定对称秩－1 修正第 I 个修正特征值的方根。
SLASD6		被 SBDSDC 用到。计算一个修正上双对角矩阵的 SVD,这个矩阵是通过合并两个小的并且添加一行得到的。
SLASD7		被 SBDSDC 用到。将两组奇异值集合合并为一个有序序列,然后尽量压缩问题规模。
SLASD8		被 SBDSDC 用到。找到特征方程根的方根,并且存储每一个 D 中的元素到两个最近标杆的距离。(*DSIGMA* 中的元素)
SLASD9		被 SBDSDC 用到。找到特征方程根的方根,并且存储每一个 D 中的元素到两个最近标杆的距离。(*DSIGMA* 中的元素)
SLASDA		被 SBDSDC 用到。计算实上双对角 n×m 矩阵的 SVD,矩阵为对角线 D 和不在对角线的 E,其中 $M = N + SQRE$。
SLASDQ		被 SBDSDC 用到。计算实(上或下)双对角 n×m 矩阵的 SVD,其中矩阵是对角线 D 和非对角线 E,如果需要的话将变换累加。
SLASDT		被 SBDSDC 用到。通过双对角分治策略生成一个子问

		题的树。
SLASET	CLASET	将一个矩阵非对角线元素初始化为α,对角线元素初试化为β。
SLASQ1		被 SBDSQR 用到。对一个对角线为 D 和非对角线为 E 的矩 n×n 矩阵计算奇异值。
SLASQ2		被 SBDSQR 和 SSTEGR 用到。计算与 qd 数组 Z 相关的相对精确的对称正定三对角矩阵的所有特征值。
SLASQ3		被 SBDSQR 用到。检查压缩,计算 shift (TAU)并且调用 dqds。
SLASQ4		被 SBDSQR 用到。用前一个变换所得到的 d 值计算一个与最小特征值接近的 TAU。
SLASQ5		被 SBDSQR 和 SSTEG 用到。用乒乓形式计算一个 dqds 变换形式。
SLASQ6		被 SBDSQR 和 SSTEG 用到。用乒乓形式计算一个 dqds 变换形式。
SLASR	CLASR	对一个一般矩形矩阵实行一系列平面旋转变换。

Routine		Description
real	complex	
SLASRT		用快速排序将数字升序或降序排列。在数组规模小于等于 20 的时候回复用插入排序。
SLASSQ	CLASSQ	修正以扩展形式描述的方根和。
SLASV2		计算一个 2×2 三角矩阵的奇异值分解。
SLASWP	CLASWP	在一个一般矩形矩阵上实行一系列行选主元。
SLASY2		解决 Sylvester 矩阵方程$AX \pm XB = \sigma C$,A 和 B 是一阶或二阶或者已经被转置过,σ是一个标量因子。
SLASYF	CLASYF CLAHEF	用对角旋转方法计算一个实对称/复对称/复厄米特不定矩阵的部分分解。
SLATBS	CLATBS	解决一系列三角带状方程$Ax = \sigma b$, $A^T x = \sigma b$, 或者 $A^H x = \sigma b$, σ实一个扩展标量因子以防止上溢。
SLATDF	CLATDF	使用 SGETC2 计算出的 n×n 矩阵的 LU 分解,并且计算

		对 reciprocal Dif-estimate 的贡献。
SLATPS	CLATPS	解决一系列三角方程 $Ax = \sigma b$， $A^T x = \sigma b$，或者 $A^H x = \sigma b$，A 是以压缩格式存储的，σ 实一个扩展标量因子以防止上溢。
SLATRD	CLATRD	通过正交/酉相似变换将一个对称/ 厄米特矩阵 A 的前 nb 行和列规约成实三对角形式，并且返回辅助矩阵，它们将会被 A 还没实行规约变换的部分用到。
SLATRS	CLATRS	解决一类三角方程 $Ax = \sigma b$， $A^T x = \sigma b$，或 $A^H x = \sigma b$，σ 是防止上溢的扩展因子。
SLATRZ	CLATRZ	用正交/酉变换将一个梯形矩阵 Factors。
SLAUU2	CLAUU2	计算 UU^H or $L^H L$ 的乘积，U 和 L 是上或下三角矩阵(非分块算法)。
SLAUUM	CLAUUM	计算 UU^H or $L^H L$ 的乘积，U 和 L 是上或下三角矩阵。
SORG2L	CUNG2L	用 SGEQLF/CGEQLF 确定的 QL 分解生成正交/酉矩阵 Q 的全部或者部分(非分块算法)。
SORG2R	CUNG2R	用 SGEQRF/CGEQRF 确定的 QR 分解生成正交/酉矩阵 Q 的全部或者部分(非分块算法)。
SORGL2	CUNGL2	用 SGELQF/CGELQF 确定的 LQ 分解生成正交/酉矩阵 Q 的全部或者部分(非分块算法)。

Routine		Description
real	complex	
SORGR2	CUNGR2	用 SGERQF/CGERQF 确定的 RQ 分解生成正交/酉矩阵 Q 的全部或者部分(非分块算法)。
SORM2L	CUNM2L	用 SGEQLF/CGEQLF 确定的 QL 分解生成正交/酉矩阵与一个一般矩阵相乘(非分块算法)。
SORM2R	CUNM2R	用 SGEQRF/CGEQRF 确定的 QR 分解生成正交/酉矩阵与一个一般矩阵相乘(非分块算法)。
SORML2	CUNML2	用 SGELQF/CGELQF 确定的 LQ 分解生成正交/酉矩阵与一个一般矩阵相乘(非分块算法)。
SORMR2	CUNMR2	用 SGERQF/CGERQF 确定的 RQ 分解生成正交/酉矩阵与一个一般矩阵相乘(非分块算法)。
SORMR3	CUNMR3	用 STZRZF/CTZRZF 确定的 RZ 分解生成正交/酉矩阵与一个一般矩阵相乘(非分块算法)。
SPBTF2	CPBTF2	计算一个对称/ 厄米特 正定带状矩阵的 Cholesky 分解(非分块算法)。

SPOTF2	CPOTF2	计算一个对称/ 厄米特 正定矩阵的 Cholesky 分解(非分块算法)。
SPTTS2	CPTTS2	用 SPTTRF/CPTTRF 得到的 A 的 $L*D*L^H$ 分解解决一类三对角方程 $A*X=B$。
SRSCL	CSRSCL	用一个实标量的倒数去乘一个向量。
SSYGS2	CHEGS2	将一个对称/厄米特 正定一般化的特征问题 $Ax=\lambda Bx$, $ABx=\lambda x$, 或 $BAx=\lambda x$, 规约到标准形式,B 是被 SPOTRF/CPOTRF 分解过的(非分块算法)。
SSYTD2	CHETD2	用正交/酉相似变换将一个对称/厄米特矩阵规约成实对称三对角形式(非分块算法)。
SSYTF2	CSYTF2 CHETF2	用对角旋转方法计算一个实对称/复对称/复厄米特不定矩阵的分解(非分块算法)。
STGEX2	CTGEX2	用正交/酉同等变换交换一个上(半)三角矩阵的 1×1 或 2×2 的相邻对角块(*A*11, *B*11)和(*A*22, *B*22)。
STGSY2	CTGSY2	解决一般 Sylvester 方程(非分块算法)。
STRTI2	CTRTI2	计算一个三角矩阵的逆(非分块算法)。
XERBLA		当输入有无效值时,被 LAPACK 程序调用的错误处理程序。

5 FFT 模块

5.1 傅立叶变换

5.1.1 FFT 的数学变换

正向的傅立叶变换把一组时域中的数据变换为频域中的数据。逆向傅立叶变换则执行相反的操作：把频域中的数据变换为时域中的数据。

这一部分我们回顾一下各种傅立叶变换的数学定义。

5.1.1.1 一维 FFT 的数学定义

一维离散傅立叶变换的数学表达式如下

$$H(k)=\sum_{m=0}^{n-1}h(m)e^{-i2\pi km/n} \qquad (5.1\text{-}1)$$

其中 $k=0,1,2,\ldots,n-1$；$H(k)$ 和 $h(m)$ 分别表示时域和频域中的均匀分布数据点上的离散函数；n 是数据长度，$i=\sqrt{-1}$。

一维逆变换如下定义：

$$h(m)=\frac{1}{n}\sum_{k=0}^{n-1}H(k)e^{i2\pi mk/n} \qquad (5.1\text{-}2)$$

其中 $m=0,1,2,\ldots,n-1$。

5.1.1.2 二维 FFT 的数学定义

关于二维离散傅立叶变换的最简单的解释是把一个两维有限序列看成双周期序列的一个周期。记 $H(j,k)$ 为 $h(m_x,m_y)$ 的离散傅立叶变换，则二维离散傅立叶变换的数学表达式如下：

$$H(j,k)=\sum_{m_x=0}^{n_x-1}\sum_{m_y=0}^{n_y-1}h(m_x,m_y)e^{-i2\pi jm_x/n_x}e^{-i2\pi km_y/n_y} \qquad (5.1\text{-}3)$$

其中：$j=0,1,2,\ldots,n_x-1$，$k=0,1,2,\ldots,n_y-1$，$i=\sqrt{-1}$。

逆变换如下定义：

$$h(m_x,m_y)=\frac{1}{n_xn_y}\sum_{j=0}^{n_x-1}\sum_{k=0}^{n_y-1}H(j,k)e^{i2\pi jm_x/n_x}e^{i2\pi km_y/n_y} \qquad (5.1\text{-}4)$$

由（1.1-4）方程给出的二维离散傅立叶变换也可以写成如下形式：

$$H(j,k)=\sum_{m_x=0}^{n_x-1}\sum_{m_y=0}^{n_y-1}h(m_x,m_y)e^{-i2\pi km_y/n_y}e^{-i2\pi jm_x/n_x} \qquad (5.1\text{-}5)$$

我们把花括弧里面的那部分用 $G(m_x,k)$ 来表示，它是一个二维序列。从而我们可以如下表示 $H(j,k)$：

$$G(m_x,k)=\sum_{m_y=0}^{n_y-1}h(m_x,m_y)e^{-i2\pi km_y/n_y} \qquad (5.1\text{-}6)$$

$$H(j,k)=\sum_{m_x=0}^{n_x-1}G(m_x,k)e^{-i2\pi jm_x/n_x} \qquad (5.1\text{-}7)$$

G 的每一行是 h 的相应一行的一维离散傅立叶变换，H 每一列是 G 的相应一列的一维离散傅立叶变换。

5.1.1.3 三维 FFT 的数学定义

对于三维情况，正向离散傅立叶变换如下定义

（5.1-8）

其中 $j=0,1,2,\ldots,n_x-1$，$k=0,1,2,\ldots,n_y-1$，$l=0,1,2,\ldots,n_z-1$，$i=\sqrt{-1}$。

三维逆向变换如下定义：

（5.1-9）

其中 $n=0,1,2,\ldots,n_x-1$，$n=0,1,2,\ldots,n_y-1$，$n=0,1,2,\ldots,n_z-1$

5.2 功能声明

xMath 库的 FFT 模块，主要提供给用户一组快速进行离散傅里叶变换（DFT）的接口，用户可以调用他们进行所需要的信号处理的变换。

xMath FFT 所支持的功能	
变换维度	1D，2D，3D
变换类型	复数到复数（C2C）
变换规模	任意大于 1 的正整数（2 的幂，非 2 的幂）
带跨步的输入、输出	1D，2D，3D 计算都支持
批量计算	仅 1D 计算支持

5.2.1 例程接口说明和使用

5.2.1.1 FFT 接口说明

下面以三维 FFT 变换说明例程的各个参数的含义、例程的返回值以及使用。

首先需要包含头文件“swfft.h”。xMath 提供了一个 DFT 的描述符句柄，通过建立不同的描述符句柄，可以实现不同的 DFT 计算。C 语言中关于描述符句柄的定义如下：

```
#define DFTI_DESCRIPTOR_HANDLE DFTI_DESCRIPTOR *
```

xMath FFT 中定义了单精度复数类型 swfftfComplex 和双精度复数类型 swfftdComplex，以便于 FFT 计算的处理。另外提供对界申请内存地址的接口，在很多情况下，保证内存地址 128 bytes 对界能最大化接口的性能。

```
void *swfftMalloc(long bytes, long align);
void swfftFree(void *ptr);
```

总的来说，在 xMath 中，FFT 的函数大至可分为 4 部分：

5.2.1.2 描述符操作

在这部分中，有几个函数用来建立并操作描述符：

(1) **DftiCreateDescriptor**：用来建立一个新的描述符

```
status = DftiCreateDescriptor( &desc_handle, precision, forward_domain,
dimension, lengths );
```

(2) **DftiCommitDescriptor**：提交一个已有的描述符

```
status = DftiCommitDescriptor( desc_handle );
```

(3) **DftiCopyDescriptor**：将一个已有的描述符复制到一个新的描述符中

```
status = DftiCopyDescriptor( desc_handle_original, &desc_handle_copy );
```

(4) **DftiFreeDescriptor**：释放一个描述符

```
status = DftiFreeDescriptor( &desc_handle );
```

5.2.1.3 FFT 计算

(1) **DftiComputeForward**：进行一个 DFT 正变换

```
status = DftiComputeForward( desc_handle, x_inout );
status = DftiComputeForward( desc_handle, x_in, x_out );
```

(2) **DftiComputeBackward**：进行一个 DFT 逆变换

```
status = DftiComputeBackward( desc_handle, x_inout );
status = DftiComputeBackward( desc_handle, x_in, x_out );
```

5.2.1.4 描述符配置

(1) DftiSetValue：对描述符中的一个或多个参数进行设置

```
status = DftiSetValue( desc_handle, config_param, config_val );
```

(2) DftiGetValue：读取现有描述符中的参数设置

```
status = DftiGetValue( desc_handle, config_param, &config_val );
```

5.2.1.5 状态检查

用来检查前 3 部分操作中的每一步的完成状态。

预定义错误类型	
Named Constants	Named Constants
DFTI_NO_ERROR	无错误
DFTI_MEMORY_ERROR	内存相关的错误
DFTI_INVALID_CONFIGURATION	非法的配置参数
DFTI_INCONSISTENT_CONFIGURATION	配置参数不一致
DFTI_BAD_DESCRIPTOR	描述符错误

5.2.2 Descriptor（描述符）详解

一个描述符可能会有如下几种不同的类别。

5.2.2.1 DFT 变换数据精度

DFTI_PRECISION 描述了 FFT 变换的精度。
DFTI_SINGLE：单精度
DFTI_DOUBLE: 双精度

5.2.2.2 FFT 正变换的 Domain

DFTI_FORWARD_DOMAIN 指明了正变换域，用来指示 FFT 所采用的变换域数据类型：

Forward Domain	Implied Backward Domain
DFTI_COMPLEX	DFTI_COMPLEX

注：当前库只支持复数到复数变换（C2C）的计算。

5.2.2.3 变换维度和长度

FFT 变换维度是用一个非负整数来表示，类型为 DFTI_CONFIG_VALUE。对于一维变换来说，变换长度是指输入数的个数。而对于多维变换来说，每一维的变换长度是对应一个数组的每一个下标。**DFTI_DIMENSION** 和 **DFTI_LENGTHS** 分别用来描述变换维度和长度，并且是不具有默认值的。相应取值规则总结如下：

DFTI_DIMENSION	DFTI_LENGTHS
1	Integer scalar（如 64）
>1	Integer array（如(64,32)，表示 64×32 的 2 为变换）

应用程序中保证 DFTI_DIMENSION 和 DFTI_LENGTHS 的一致性。

5.2.2.4 数据缩放

可以通过指定 **DFTI_FORWARD_SCALE** 与 **DFTI_BACKWARD_SCALE** 的值来决定正变换和逆变换后的 scale 系数。一个常用的方法是，对于 N 个数的一维 FFT，将 DFTI_BACKWARD_SCALE 设置为 1/N,这使得一维正变换和逆变换后的结果是输入本身。

5.2.2.5 FFT 结果的存放位置

默认的 FFT 结果将覆盖原有的输入值，即 **DFTI_PLACEMENT**=DFTI_INPLACE。用户可以指定 DFTI_PLACEMENT=DFTI_NOT_INPLACE，这样可以将结果保存到另外的内存区域中。

5.2.2.6 带跨步的输入输出

FFT接口提供更复杂的多维数组数据布局。对于d维数据集X，规模为$N_d\times\ldots\times N_2\times N_1$，其中最低维$N_1$为连续存储的维度，其具体的元素由一个d维的坐标（$k_d,\ldots,k_2,k_1$）来指定。若由Addr(a)代表元素a的地址，那么元素$X(k_d,\ldots,k_2,k_1)$的地址计算公式为

$$\begin{aligned}Addr(X(k_d,\cdots,k_2,k_1)) &= Addr(X(0,\cdots,0,0)) + offset\\ &= Addr(X(0,\cdots,0,0)) + s_0 + k_1*s_1+k_2*s_2+\ldots+k_d*s_d\end{aligned}$$

其中 s_0 代表第一个元素的起始偏移，$s_1,s_2,\cdots,s_d$ 代表各个维度上的跨步。由配置参数 **DFTI_INPUT_STRIDES** 和 **DFTI_OUTPUT_STRIDES** 来指定这些值，即 DFTI_INPUT_STRIDES 和 DFTI_OUTPUT_STRIDES 参数所对应值为一个长度为 d+1 的 1 为数组，取值为（$s_0,s_d,\ldots,s_2,s_1$）。跨步以变换域指定的基本数据类型为单位，即一个 d 维的规模为 $N_d\times\ldots\times N_2\times N_1$ 的变换，

默认该参数值为（0,N_{d-1}×…×N_2×N_1,…,N_1,1）。对于 inplace 的 FFT 计算，输出和输入采用同样的跨步参数，即由 DFTI_OUTPUT_STRIDES 指定的参数将会忽略不计。

当前库只支持正数的跨步，由应用程序保证参数的合法性。

5.2.2.7 批量计算

xMath FFT 库中提供 1D FFT 的批量计算，即通过配置相应参数，调用一次计算函数即可完成多个相同配置的 FFT 计算，该功能主要由 DFTI_NUMBER_OF_TRANSFORMS、DFTI_INPUT_DISTANCE 和 DFTI_OUTPUT_DISTANCE 这三个参数控制。它们所代表的含义分别为：

DFTI_NUMBER_OF_TRANSFORMS，一次进行计算的变换的个数；

DFTI_INPUT_DISTANCE，两个变换的输入序列第一个元素之间的距离，以基本数据单元为单位计数；

DFTI_OUTPUT_DISTANCE，两个变换的输出序列第一个元素之间的距离，以基本数据单元为单位计数。

该 3 个参数只能接收正整数值。当设置了参数 DFTI_NUMBER_OF_THRANSFORMS 而没有设置另外两个参数时，那两个参数将默认设置为输入序列的长度，即 lengths 参数个元素之积。对于 inplace 的 FFT 计算，输入和输出采用同样的距离参数，即由 DFTI_OUTPUT_DISTANCE 所指定的参数将会忽略不计。

5.2.3 快速傅立叶变换三步法步骤

5.2.3.1 快速傅立叶变换的初始化步骤

(1)创建一个描述符，然后提交：

```
status = DftiCreateDescriptor( &desc_handle, precision, forward_domain,
             dimension, lengths );
status = DftiCommitDescriptor( desc_handle );
```

5.2.3.2 快速傅立叶变换的变换步骤

(1) DftiComputeForward：进行一个 DFT 正变换

```
status = DftiComputeForward( desc_handle, x_inout );
status = DftiComputeForward( desc_handle, x_in, x_out );
```

(2) DftiComputeBackward：进行一个 DFT 逆变换

```
status = DftiComputeBackward( desc_handle, x_inout );
status = DftiComputeBackward( desc_handle, x_in, x_out );
```

5.2.3.3 快速傅立叶变换的退出步骤

(1) DftiFreeDescriptor：释放一个描述符

```
status = DftiFreeDescriptor( &desc_handle );
```

5.3 Descriptor(描述符)配置总结

信号处理模块比较独立，每种变换自成一个模块不相互依赖。下面具体给出各部分的所有的宏的预定义类型和模块功能。

5.3.1 DFT 配置的可设置参数

对于所有的 descriptor 的可设置参数总结如下：

可设置参数		
常量名称	值类型	描述
最常用的设置，没有默认值，必须显式设置		
DFTI_FORWARD_DOMAIN	Named constant	用于正变换的 Domain，只能由 DftiCreateDescriptor 设置
DFTI_PRECISION	Named constant	计算精度，只能由 DftiCreateDescriptor 设置
DFTI_DIMENSION	Integer scalar	计算维度，只能由 DftiCreateDescriptor 设置
DFTI_LENGTHS	Integer scalar/array	每个维度的长度，只能由 DftiCreateDescriptor 设置
常用设置，包含数据缩放系数和结果存放位置		
DFTI_FORWARD_SCALE	Floating-point scalar	正变换的缩放系数
DFTI_BACKWARD_SCALE	Floating-point scalar	逆变换的缩放系数
DFTI_PLACEMENT	Named constant	计算结果的存放
对于 stride 的设置		
DFTI_INPUT_STRIDES	Integer array	DFTI_INPUT_STRIDES
DFTI_OUTPUT_STRIDES	Integer array	DFTI_OUTPUT_STRIDES

可设置参数		
常量名称	值类型	描述
批量计算（1D FFT）		
DFTI_NUMBER_OF_TRANSFORMS	Integer scalar	变换次数
DFTI_INPUT_DISTANCE	Integer scalar	多次变换，第一个元素之间的距离
DFTI_OUTPUT_DISTANCE	Integer scalar	多次变换，第一个元素之间的距离

5.3.2 常量名称的取值

常量名称的取值	
常量名称	描述
DFTI_SINGLE	单精度浮点数
DFTI_DOUBLE	双精度浮点数
DFTI_COMPLEX	复数变换
DFTI_INPLACE	输出覆盖输入
DFTI_NOT_INPLACE	输出不覆盖输入
DFTI_COMMITTED	Descriptor 已提交
DFTI_UNCOMMITTED	Descriptor 未提交
DFTI_MAX_MESSAGE_LENGTH	Message 最大长度

5.3.3 离散可配置参数的设置

离散可配置参数的设置	
常量名称	取值范围
DFTI_PRECISION	DFTI_SINGLE 或 DFTI_DOUBLE
DFTI_FORWARD_DOMAIN	DFTI_COMPLEX

离散可配置参数的设置	
常量名称	取值范围
DFTI_PLACEMENT	DFTI_INPLACE (默认), 或 DFTI_NOT_INPLACE

5.3.4 可设置参数的默认值

可设置参数的默认值	
常量名称	默认值
DFTI_NUMBER_OF_TRANSFORMS	1
DFTI_FORWARD_SCALE	1.0
DFTI_BACKWARD_SCALE	1.0
DFTI_PLACEMENT	DFTI_INPLACE
DFTI_COMPLEX_STORAGE	DFTI_COMPLEX_COMPLEX
DFTI_INPUT_DISTANCE	0
DFTI_OUTPUT_DISTANCE	0

5.3.5 xMath 傅立叶变换函数总结

下表 xMath 傅立叶变换函数总结

函数名	操作
操作符构造函数	
DftiCreateDescriptor	新建并初始化一个描述符
DftiCommitDescriptor	提交一个描述符
DftiCopyDescriptor	复制一个已有的描述符
DftiFreeDescriptor	删除一个描述符
FFT 计算函数	
DftiComputeForward	计算正向 FFT
DftiComputeBackward	计算逆向 FFT
操作符配置函数	
DftiSetValue	向特定配置中的某个参数赋值
DftiGetValue	从特定配置中获取一个值
状态检查函数	
DftiErrorClass	查看是否有一个错误

DftiErrorMessage	生成一个错误信息

5.3.6 例子参考

以下分别给出了一个进行一维、二维 FFT 计算测试例子。

1> 开头均要包含以下头文件：

```
#include <stdio.h>
#include "swfft.h"
```

2> 一维变换测试函数

```
void test_DFTI_1D()
{
    swfftdComplex *input,*output;
    DFTI_DESCRIPTOR_HANDLE hDesc;
    DFTI_CONFIG_VALUE N = 64, is=2, os=3;
    DFTI_CONFIG_VALUE input_strides[2], output_strides[2];
    long status;

    /* Create DFT Descriptor
    * 1D FFT, size N1, ( Double type, Complex data)
    */
    status = DftiCreateDescriptor(&hDesc, DFTI_DOUBLE, DFTI_COMPLEX, 1, N);
    if(status) {
        printf("%s",DftiErrorMessage(status));
        exit(1);
    }

    /* Set Optional Descriptor Parameters */
    input_strides[0] = 0;       // displacement
    input_strides[1] = is;      // s1
    status = DftiSetValue(hDesc, DFTI_INPUT_STRIDES, input_strides);
    output_strides[0] = 0;     // displacement
    output_strides[1] = os;    // s1
    status |= DftiSetValue(hDesc, DFTI_OUTPUT_STRIDES, output_strides);
    sttaus |= DftiSetValue(hDesc, DFTI_PLACEMENT, DFTI_NOT_INPLACE);
    if(status){
        printf("%s",DftiErrorMessage(status));
        exit(1);
    }

    /* Allocate aligned memory space */
    input = (swfftdComplex *) swfftMalloc(N*sizeof(swfftdComplex), 128);
    output = (swfftdComplex *) swfftMalloc(N*sizeof(swfftdComplex), 128);
```

```
    /* Prepare the Descriptor for Execution */
    status = DftiCommitDescriptor(hDesc);
    if(status){
        printf("%s",DftiErrorMessage(status));
        exit(1);
    }

    /* FFT */
    status = DftiComputeForward(hDesc, input, output);
    if(status){
        printf("%s",DftiErrorMessage(status));
        exit(1);
    }

    /* Free the Descriptor */
    status = DftiFreeDescriptor(&hDesc);
    if(status){
        printf("%s",DftiErrorMessage(status));
        exit(1);
    }

    swfftFree(input);
    swfftFree(output);
}
```

3> 二维变换

```
void test_DFTI_2D()
{
    swfftdComplex *input, *output;
    DFTI_DESCRIPTOR_HANDLE hDesc;
    DFTI_CONFIG_VALUE N1, N2, lengths[2];
    DFTI_CONFIG_VALUE is[2] = {3, 2}, os[2] = {2, 3};
    DFTI_CONFIG_VALUE input_strides[3], output_strides[3];
    long status, bytes;

    /* Create    DFT    Descriptor
    * 2D FFT, size N2xN1, (Double type, Complex data)
    */
    lengths[0] = N2;
    lengths[1] = N1;
    status = DftiCreateDescriptor(&hDesc, DFTI_DOUBLE, DFTI_COMPLEX, 2, lengths);
    if(status) {
        printf("%s",DftiErrorMessage(status));
        exit(1);
    }
```

```
/* Set Optional  Descriptor  Parameters */
input_strides[2] = is[1];
input_strides[1] = is[1]*lengths[1];
input_strides[0] = 0;
status = DftiSetValue(hDesc, DFTI_INPUT_STRIDES, input_strides);
output_strides[2] = os[1];
output_strides[1] = os[1]*lengths[1];
output_strides[0] = 0;
status |= DftiSetValue(hDesc, DFTI_OUTPUT_STRIDES, output_strides);
status |= DftiSetValue(hDesc, DFTI_PLACEMENT, DFTI_NOT_INPLACE);
if(status){
      printf("%s",DftiErrorMessage(status));
      exit(1);
}

/* Allocate aligned memory space */
bytes = lengths[0] * is[0] * lengths[1] * is[1] * sizeof(swfftdComplex);
input = (swfftdComplex *) swfftMalloc(bytes, 128);
bytes = lengths[0] * os[0] * lengths[1] * os[1] * sizeof(swfftdComplex);
output = (swfftdComplex *) swfftMalloc(bytes, 128);

/* Prepare  the  Descriptor  for  Execution */
status =  DftiCommitDescriptor(hDesc);
if(status){
      printf("%s",DftiErrorMessage(status));
      exit(1);
}

/* FFT */
status = DftiComputeForward(hDesc, input, output);
if(status){
      printf("%s",DftiErrorMessage(status));
      exit(1);
}


/* Free the Descriptor */
status = DftiFreeDescriptor(&hDesc);
if(status){
      printf("%s",DftiErrorMessage(status));
      exit(1);
}
```

```
    swfftFree(input);
    swfftFree(output);
}
```

6 迭代解法器模块

科学领域和工程领域的许多应用都需要求解线性方程组。数学上,这类问题通常被描述为矩阵——向量方程,Ax=b,其中 A 是 n*n 矩阵,x 和 b 是 n 元列向量。矩阵 A 通常称作系数矩阵,而向量 x 和 b 则分别称作解向量和右端项。

在许多实际应用中,A 中的大多数元素是零,这样的矩阵称作稀疏矩阵。相应的,零元很少的矩阵则称作稠密矩阵。如果 A 是稀疏矩阵而且能利用它的稀疏性,则无论在存储空间上还是时间上都可以大大提高方程 Ax=b 的求解效率。在不牺牲正确性的前提下,越是能够充分利用稀疏性的算法就越优秀。

充分利用了矩阵 A 的稀疏性的解法通常被称为稀疏解法器。稀疏解法器通常又被分作两类：直接稀疏解法器和迭代稀疏解法器。当稀疏线性方程组的稀疏矩阵不规则时,直接法在求解过程中带来大量非零元素,增加了计算量和存储量；同时直接法不易并行,不能满足求解大规模问题的需要,因此通常使用迭代法。

对于使用迭代法的求解器,采用的基本技术是预条件子与 Krylov 子空间方法相结合使用。对原方程组的求解转化为如下的求解过程：

1. 根据系数矩阵和求解需要,寻找预条件子 Q;
2. 代入选择的解法器进行求解。

对应于求解的第一个阶段,有许多种生成预条件子的方式。针对不同的稀疏矩阵和不同的迭代方法,以及预条件子的生成方式,预条件子又可以分为左预条件子和右预条件子。如果对称矩阵 A 同时也是正定的,那么可以看出 A 可以被分解为$L\ L^T$,其中L 是一个下三角矩阵。这种形式的分解称作 Cholesky 分解。在预优过程中采用不完全的 Cholesky 分解生成预条件子可以节省存储空间。如果 A 是非对称的矩阵,那么对 A 可以进行 LU 分解：A=LU,L 是单位下三角矩阵,U 是上三角矩阵,将 L 严格下三角部分和 U 存储在和 A 同阶的结构中,可以节省存储空间。

对应于求解的第二个阶段,可以选择适当的 Krylov 子空间方法,代入相应的解法器进行迭代求解。

6.1 xMath（众核版）迭代解法器特点

xMath（众核版）数学库中实现了两种 Krylov 子空间迭代法：cg（Conjugate Gradient Method）， fgmres（Flexible Generalized Minimal RESidual Solver）。具体的算法说明可以参见参考文献[3]，[4]。xMath（众核版）数学库还实现了两种预条件子：ILU0、ILUT，是基于稀疏存储格式 CSR 实现的，目前这两种预条件子还是串行实现。

迭代解法器接口需要的输入如下：矩阵 A,右端项 b,还有一个关于解的初始猜测,然而,对于稀疏矩阵的大量存储方法而言,不可能提供一个接口使所有存储格式适用。xMath（众核版）采用 RCI 机制来解决这个问题,即由用户提供矩阵向量乘、预条件子产生等函数,在解法器需要的时候调用并将结果传给解法器。

所有迭代解法器中涉及到的系数矩阵或者是其推导出的矩阵,如预条件子,只应用在下列三种操作中：

1. 预条件的创建；
2. 系数矩阵与向量的乘积；
3. 预条件子的应用。

ILU0 预条件子是基于著名的 LU 分解,将一个矩阵分解为一个下三角矩阵和一个上三角矩阵的乘积。通常这个分解过程中会产生一些非零元填充。ILU0 预条件子保持原始矩阵的数据结构。ILUT 预条件子保留了一些非零元填充,当它满足以下两个条件时：

1. 填充数值大于给定的误差限和当前矩阵行范数的乘积；
2. 填充位置在给定的结果预条件矩阵带宽内。

ILU0 和 ILUT 预条件子可以用到任何非退化矩阵中,它们可单独使用或者与 FGMRES 预条件子一起使用。这两个预条件子不能与 CG 法同时使用,因为即使原始矩阵是对称的,他们也会产生非对称的预条件矩阵。为了应用这两个预条件子,需要调用 dcsrtrsv 必须得使用两次。

提示：尽管 ILU0 和 ILUT 可以应用到任何非退化矩阵,但是在某些情况下,算法也不能成功收敛。预条件函数只有在实际计算中采用验证是否有效。对某些系统和初始猜测,使用预条件子可能会增加迭代次数,甚至破坏收敛性。用户负责使用合适的预条件子。

6.2 迭代解法器函数列表

表 6-1 迭代解法器函数汇总

函数	描述
Dcsrgemv,ddiagemv,dellgemv	三种存储格式稀疏矩阵向量乘函数
Dcg_init,dfgmres_init	初始化求解器
Dcsrilu0,dcsrilut	生成预条件子
Dcg_check,dfgmres_check	检查用户定义参数的一致性和合法性
Dcg,dfgmres	计算近似解向量
Dcg_get,dfgmres_get	返回当前迭代步数

下面指明了本模块被调用的典型次序。其中 dcg_get,dfgmrs_get 可以在任何位置调用。高级用户可以按照需要改变此顺序。

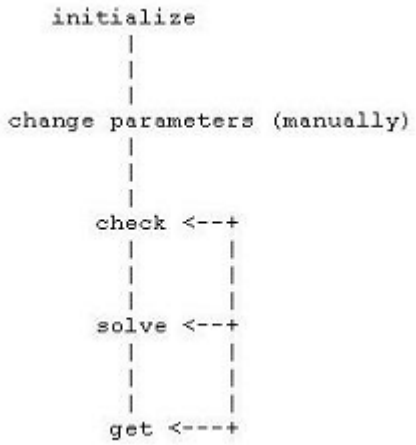

图 6-1 函数调用流程图

如果要使用预条件子,则需在 check 之前生成预条件子,本模块中包含了预条件子应用函数。

6.3迭代解法器调用伪代码

6.3.1 CG 函数调用伪代码

```
产生矩阵 A
产生预条件子 C(可选)
    Call dcg_init(n,x,b,RCI_request,ipar,dpar,tmp)
    根据需要调整 ipar,dpar 中参数
    Call dcg_check(n,x,b, RCI_request,ipar,dpar,tmp)
1   call dcg(n,x,b, RCI_request,ipar,dpar,tmp)
    If (RCI_request .eq. 1) then
    计算矩阵 A 乘以向量 tmp(1:n,1),并将结果存入 tmp(1:n,2)
    此运算可以调用 sparse blas 2 级函数来完成
C   继续执行 CG 法迭代
     Goto 1
    Endif
    If (RCI_request .eq. 2) then
    执行收敛检测
      If (测试未通过) then
C      继续执行 CG 法迭代
       Goto 1
       Else
C      终止 CG 法迭代
        Goto 2
       Endif
```

```
    Endif
    If (RCI_request .eq. 2) then (若使用预条件子,则有此句,否则没有)
    计算矩阵 C^-1 乘以向量 tmp(1:n,3),并将结果存入 tmp(1:n,4)
C   继续执行 CG 法迭代
      Goto 1
    Endif
2   call dcg_get(n,x,b, RCI_request,ipar,dpar,tmp,itercount)
    Itercount 的值即为当前迭代次数
    向量 x 存的是计算近似解
```

6.3.2 FGMRES 函数调用伪代码

```
    产生矩阵 A
    产生预条件子 C(可选)
    Call dfgmres_init(n,x,b,RCI_request,ipar,dpar,tmp)
    根据需要调整 ipar,dpar 中参数
    Call dfgmres_check(n,x,b, RCI_request,ipar,dpar,tmp)
1   call dfgmres(n,x,b, RCI_request,ipar,dpar,tmp)
      If (RCI_request .eq. 1) then
      计算矩阵 A 乘以向量 tmp(ipar(22)),并将结果存入 tmp(ipar(23))
      此运算可以调用 sparse blas 2 级函数来完成
C     继续执行 FGMRES 法迭代
       Goto 1
      Endif
      If (RCI_request .eq. 2) then
      执行收敛检测
        If (测试未通过) then
C        继续执行 FGMRES 法迭代
         Goto 1
        Else
C        终止 FGMRES 法迭代
         Goto 2
        Endif
      Endif
      If (RCI_request .eq. 3) then (若使用预条件子,则有此句,否则没有)
        计算矩阵 C^-1 乘以向量 tmp(ipar(22)),并将结果存入 tmp(ipar(23))
C       继续执行 FGMRES 法迭代
        Goto 1
      Endif
2     call dfgmres_get(n,x,b, RCI_request,ipar,dpar,tmp,itercount)
      Itercount 的值即为当前迭代次数
      向量 x 存的是计算近似解
```

6.4 迭代解法器参数介绍

本文档中提到的数据类型指 FORTRAN 标准类型,INTEGER,DOUBLE PRECISION。

6.4.1 CG 参数介绍

n: 整型变量，在 dcg_init 中设置问题规模。
x: 实型变量，大小为 n。该参数存储解的当前近似。在第一次调用 dcg 之前，它包含解的初始估计。输出时该参数返回解向量。
b: 实型变量，右端向量，大小为 n。
RCI_request: 整型变量，通知函数工作的结果及下一步应该调用函数的提示。值为负数，表明函数出错或者出现警告。值为 0，表明任务成功完成，值为正数，代表用户必须执行特定的功能，具体说明如下：
RCI_request＝1: 用矩阵乘以 tmp(:,1)，将结果放到 tmp(:,2),并将控制转回 dcg 函数。
RCI_request＝2: 执行收敛标准检测，如果失败，将控制转回 dcg，否则，解已找到，将其存入 x 数组。
RCI_request＝3: 对 tmp(:,3) 应用预条件子，将结果放入 tmp(:,4),返回控制到 dcg 函数。
ipar: 整型数组，大小为 128。包含 cg 计算所需要的整型参数集。
ipar(1): 指定问题规模。dcg_init 指定 ipar(1)=n,其它函数均使用 ipar(1),该参数无默认值。
ipar(2): 指定 cg 函数所产生的错误和警告输出类型。默认值为 6，表示所有错误和警告都输出到屏幕上，否则，错误和警告信息将分别输出在新创建的文件 dcg_errors.txt 和 dcg_check_warnings.txt。注意如果 ipar(6)和 ipar(7)均设置为 0，这些信息根本不产生。
ipar(3): 包含 RCI CG 计算的当前状态，初始值为 1。
ipar(4): 存储当前迭代次数，初始值为 0。
ipar(5): 指定最大迭代次数，初始值为 min(150,n)。
ipar(6): 如果该值不等于 0，则错误信息输出跟 ipar(2)一致，否则，不输出任何错误信息，但是 RCI_request 返回一个负数。默认值为 1。
ipar(7): 如果该值不等于 0，则警告信息输出跟 ipar(2)一致，否则，不输出任何警告信息，但是 RCI_request 返回一个负数。默认值为 1。
ipar(8): 如果该值不等于 0,则 dcg 函数执行最大迭代次数检测,即 ipar(4)≤ ipar(5),如果不满足该条件，则函数停止并赋给 RCI_request 相应的值。如果该值等于 0，则 cg 函数不执行这个检测。默认值为 1。
ipar(9): 如果该值不等于 0，则 dcg 函数执行残量标准检测，即  。如果不满足该条件，则函数停止并赋给 RCI_request 相应的值。。如果该值等于 0，则 cg 函数不执行这个检测。默认值为 1。
ipar(10): 如果该值不等于 0，则执行用户指定的收敛检测，并将输出参数 RCI_request 设置为 2。如果该值为 0，用户不执行用户指定的收敛检测，默认值为 1。

注： ipar(8)-ipar(10) 至少有一个设置为 1。
ipar(11)： 如果该值等于 0，则执行不带预条件的共轭梯度法，否则，执行预条件版本，并且需要用户通过设置参数 RCI_request＝3 指定预条件步。默认值为 0。
ipar(11：128)： 保留参数，目前解法器中没用到。
dpar： 实型数组，大小为 128。 包含 cg 计算所需要的实型参数集。
dpar(1)： 指定相对误差限。默认值为 1.0D-6。
dpar(2)： 指定绝对误差限。默认值为 0.0D-0。
dpar(3)： 指定初始残量的平方范数。初始值为 0。
dpar(4)： 存储参数 dpar(1)*dpar(3)+dpar(2)，初始值为 0。
dpar(5)： 指定当前残量的平方范数。初始值为 0。
dpar(6)： 指定上一迭代步残量的平方范数。初始值为 0。
dpar(7)： 包含 CG 方法的 alpha 参数。初始值为 0。
dpar(8)： 包含 CG 方法的 beta 参数，等于 dpar(5)/dpar(6) ，初始值为 0。
dpar(9:128)：当前 RCI CG 函数的保留参数。
tmp： 实型数组，大小为 (n,4)。RCI CG 中所需的临时工作空间。
tmp(:,1)： 存储当前搜索方向。初始值为 0.0。
tmp(:,2)： 存储系数矩阵乘以当前搜索方向的结果。初始值为 0.0。
tmp(:,3)： 存储当前残量。初始值为 0。
tmp(:,4)： 存储当前残量乘以预条件子的逆的结果。初始值为 0.0。

6.4.2 FGMRES 参数介绍

n： 整型变量,在 dfgmres 中设置问题规模。
x： 实型变量,大小为 n。该参数存储解的当前近似。在第一次调用 dfgmres 之前,它包含解的初始估计。
b： 实型变量,右端向量,大小为 n。
RCI_request： 整型变量,通知函数工作的结果及下一步应该调用函数的提示。值为负数,表明函数出错或者出现警告。值为 0,表明任务成功完成,值为正数,代表用户必须执行特定的功能,具体说明如下：
RCI_request＝1：用矩阵乘以 tmp(ipar(22))，将结果放到 tmp(ipar(23)),并将控制转回 dfgmres 函数。
RCI_request＝2：执行收敛标准检测,如果失败,将控制转回 dfgmres,否则,通过调用 dfgmres_get 函数来更新解。
RCI_request＝3：对 tmp(ipar(22)) 应用预条件子,将结果放入 tmp(ipar(23)),返回控制到 dfgmres 函数。
RCI_request＝4：检查带舍入误差的当前正交向量的范数是否不为 0,如果不为 0,返回 dfgmres 函数,否则通过调用 dfgmres_get 函数来完成解的过程。
ipar(128)： 整型数组,包含 dfgmres 计算所需要的整型参数集。
ipar(1)： 指定问题规模。dfgmres_init 指定 ipar(1)=n,其它函数均使用 ipar(1),该参数无默认值。
ipar(2)： 指定 RCI FGMRES 函数所产生的错误和警告输出类型。默认值为 6,表示所有错误和警告都输出到屏幕上,否则,错误和警告信息将输出在新创建的文件 MKL_RCI_FGMRES_Log.txt。注意如果 ipar(6)和 ipar(7)均设置为 0,这些信息根本不产

生。

ipar(3)： 包含 RCI CG 计算的当前状态,初始值为 1。

ipar(4)： 存储当前迭代次数,初始值为 0。

ipar(5)： 指定最大迭代次数,初始值为 min(150,n)。

ipar(6)： 如果该值不等于 0,则错误信息输出跟 ipar(2)一致,否则,不输出任何错误信息,但是 RCI_request 返回一个负数。默认值为 1。

ipar(7)： 如果该值不等于 0,则警告信息输出跟 ipar(2)一致,否则,不输出任何警告信息,但是 RCI_request 返回一个负数。默认值为 1。

ipar(8)： 如果该值不等于 0,则 dfgmres 函数执行最大迭代次数检测,即 ipar(4)≤ ipar(5), 如果不满足该条件,则函数停止并赋给 RCI_request 相应的值。如果该值等于 0,则 cg 函数不执行这个检测。默认值为 1。

ipar(9)： 如果该值不等于 0, 则 dfgmres 函数执行残量标准检测,即  。 如果不满足该条件,则函数停止并赋给 RCI_request 相应的值。。如果该值等于 0,则 dfgmres 函数不执行这个检测。默认值为 1。

ipar(10)： 如果该值不等于 0,则执行用户指定的收敛检测,并将输出参数 RCI_request 设置为 2。如果该值为 0,用户不执行用户指定的收敛检测,默认值为 1。

注： ipar(8)-ipar(10) 至少有一个设置为 1。

ipar(11)： 如果该值等于 0,则执行不带预条件的 FGMRES,否则,执行预条件版本,并且需要用户通过设置参数 RCI_request＝3 指定预条件步。默认值为 0。

ipar(12)： 如果该值不等于 0,dfgmres 函数执行当前产生向量 0 范数的自动检测,即 dpar(7)≤dpar(8),dpar(8)存的是收敛误差限。否则,函数需要用户执行该检测,这是通过设置 RCI_request=4 来完成的。默认值为 0。

ipar(13)： 如果该值为 0,dfgmres_get 根据计算结果将解更新到 x 向量中,如果该值为正数,则将解更新到右端向量 b 中。如果该值为负,则只返回当前迭代次数,不更新解。默认值为 0。

ipar(14)： 包含重启前的迭代步数。初始值为 0。

ipar(15)： 指定 non-restarted FGMRES 迭代的步数。为了执行 FGMRES 的重启版本,在重启前,用户必须指定 ipar(15)。默认值为 min{150, n},即默认使用非重启版的 FGMRES。

ipar(16)： 服务变量,指定旋转 Hessenberg 矩阵在 TMP 数组中的起始位置。

ipar(17)： 服务变量,指定旋转 cosines 函数在 TMP 数组中的起始位置。

ipar(18)： 服务变量,指定旋转 sines 函数在 TMP 数组中的起始位置。

ipar(19)： 服务变量,指定旋转残量向量在 TMP 数组中的起始位置。

ipar(20)： 服务变量,指定最小平方解向量在 TMP 数组中的起始位置。

ipar(21)： 服务变量,指定预条件向量集在 TMP 数组中的起始位置。在 TMP 数组中从 ipar(21)开始的位置只供预条件 FGMRES 方法使用。

ipar(22)： 服务变量,指定运算反馈即 RCI_request 使用的第一个向量在 TMP 数组中的起始位置。

ipar(23)： 服务变量,指定运算反馈即 RCI_request 使用的第二个向量在 TMP 数组中的起始位置。

ipar(24:128)： 保留变量,暂时未使用。

dpar(128)： 实型数组,包含 dfgmres 计算所需要的实型参数集。

dpar(1)： 指定相对误差限。默认值为 1.0D-6。
dpar(2)： 指定绝对误差限。默认值为 0.0D-0。
dpar(3)： 指定初始残量的欧拉范数。初始值为 0。
dpar(4)： 存储参数 dpar(1)*dpar(3)+dpar(2),初始值为 0。
dpar(5)： 指定当前残量的欧拉范数。初始值为 0。
dpar(6)： 指定上一迭代步残量的欧拉范数。初始值为 0。
dpar(7)： 包含产生向量的范数。初始值为 0。
dpar(8)： 包含当前产生向量 0 范数的误差限,初始值为 1.0D-12。
dpar(9:128)：当前 RCI FGMRES 函数的保留参数。
tmp：实型数组,大小为((2n*ipar(15)+4*n+(ipar(15)+9)*ipar(15)/2+4)),提供 RCI FGMRES 函数所需的双精度临时空间。
tmp(1:ipar(16)-1)： 包含 FGMRES 方法产生的序列。初始值为 0,对于该空间的前 n 个值。
tmp(ipar (16) :ipar (17) -1)： 存储由 FGMRES 方法产生的旋转 Hessenberg 矩阵,该矩阵用 packed 格式存储,无初始值。
tmp(ipar (17) :ipar (18) -1)： 存储由 FGMRES 方法产生的旋转 cosines 向量,无初始值。
tmp(ipar (18) :ipar (19) -1)： 存储由 FGMRES 方法产生的旋转 sines 向量,无初始值。
tmp(ipar (19) :ipar (20) -1)： 存储由 FGMRES 方法产生的旋转残量向量,无初始值。
tmp(ipar (20) :ipar (21) -1)： 存储由 FGMRES 方法产生的最小平方问题的解向量,无初始值。
tmp(ipar(21):)： 存储由 FGMRES 方法产生的预条件向量集,对于不用预条件的 FGMRES 方法,该空间没有使用。无初始值。

6.5 迭代解法器函数接口介绍

6.5.1 dcg

6.5.1.1 dcg_init

功能：初始化 CG 求解器
语法：dcg_init(n, x, b, RCI_request, ipar, dpar, tmp)
描述：函数 dcg_init 初始化解法器,其设置的参数值被解法器 CG 使用,高级用户可以略过这步直接在相应函数设置参数。
提示：在传递给解法器函数这些参数后,用户只有在保证参数满足正确性和一致性条件时才能修改。通过调用 dcg_check 可以做正确性和一致性的基本检查,但是不保证此方法能正确运行。
输入参数：

n： 整型变量。存储问题规模。向量 x 和 b 的长度。

x： n 长的实型数组,包含解的初始猜测,通常为 0 或者 b。

b: n 长的实型数组,包含右端向量。

输出参数：

RCI_request: 整型变量。通知函数功能结果。

ipar: 整型数组,长度为 128。

dpar: 实型数组,长度为 128。

tmp: 实型数组,长度为(n,4)。

返回值：

RCI_request= 0: 函数实现功能,正常返回。

RCI_request= -10000: 函数出错。

6.5.1.2 dcg_check

功能：检查 CG 法用户定义数据的一致性和正确性

语法：dcg_check(n, x, b, RCI_request, ipar, dpar, tmp)

描述：函数 dcg_check 检查传递给解法器 cg 的参数的正确性和一致性。然而此函数并不能保证算法给出正确的结果,它只是减少算法参数出错的机率。高级用户如果确定指定给解法器的参数正确就可以略过此步。

提示：在传递给解法器函数这些参数后,用户只有在保证参数满足正确性和一致性条件时才能修改。通过调用 dcg_check 可以做正确性和一致性的基本检查,但是不保证此方法能正确运行。

输入参数：

n: 整型变量。存储问题规模。向量 x 和 b 的长度。

x: n 长的实型数组,包含解的初始猜测,通常为 0 或者 b。

b: n 长的实型数组,包含右端向量。

输出参数：

RCI_request: 整型变量。通知函数功能结果。

ipar: 整型数组,长度为 128。

dpar: 实型数组,长度为 128。

tmp: 实型数组,长度为(n,4)。

返回值：

RCI_request= 0: 函数正常返回。

RCI_request= -1100: 函数终止,出现错误。

RCI_request= -1001: 函数返回一些警告信息。

RCI_request= -1010: 为了保持一致,函数修改一些参数。

RCI_request= -1011: 函数返回一些警告信息并修改了一些参数。

6.5.1.3 dcg

功能：CG 求解器,计算近似解向量

语法：dcg (n, x, b, RCI_request, ipar, dpar, tmp)

描述：函数 dcg 采用 CG 法求解近似解向量。调用前向量 x 包含解的初始猜测。参数 RCI_request 通知用户任务完成或者解法器需要某些计算结果。

输入参数：

n: 整型变量。存储问题规模。向量 x 和 b 的长度。

x: n 长的实型数组,包含解的初始猜测,通常为 0 或者 b。
b: n 长的实型数组,包含右端向量。
tmp: 实型数组,长度为(n,4)。

输出参数:

RCI_request: 整型变量。通知函数功能结果。
x: n 长的实型数组,包含求得得估计解。
ipar: 整型数组,长度为 128。
dpar: 实型数组,长度为 128。
tmp: 实型数组,长度为(n,4)。

返回值:

RCI_request=0: 函数正常返回,解已找到并存在 x 向量中。这个是针对采用默认收敛标准得情况,对用户自定义的收敛标准,参见 RCI_request= 2。
RCI_request=-1: 当最大迭代次数已达到但是相对收敛误差限没有满足时返回此信息。
RCI_request=-2: 计算中出现除 0 的错误,可能是矩阵非正定。
RCI_request=- 10: 残量范数无效。可能是 dpar(6)在函数外部被改变,或者没有调用 dcg_check 函数。
RCI_request=-11: 函数进入无限循环,可能是 ipar(8),ipar(9),ipar(10) 在函数外部被改变。
RCI_request= 1: 需要执行矩阵乘以 tmp(1:n,1),并将结果放入 tmp(1:n,2),返回控制到 dcg 中。
RCI_request= 2: 需要用户执行收敛检测,如果失败,将返回 dcg 函数,否则解已找到,将其存入 x 向量。
RCI_request= 3: 需要用户对 tmp(:, 3)应用预条件子,将结果存入 tmp(:, 4),返回 dcg。

6.5.1.4 dcg_get

功能:提取当前迭代步数
语法:dcg_get (n, x, b, RCI_request, ipar, dpar, tmp, itercount)
描述:函数 dcg_get 提取求解过程中的当前迭代步数。

输入参数:

n: 整型变量。存储问题规模。向量 x 和 b 的长度。
x: n 长的实型数组,包含解的初始猜测,通常为 0 或者 b。
b: n 长的实型数组,包含右端向量。
RCI_request: 整型变量,该参数未使用。
ipar: 整型数组,长度为 128。
dpar: 实型数组,长度为 128。
tmp: 实型数组,长度为(n,4)。

输出参数:

itercount: 返回当前迭代次数。

返回值:

该函数无返回值。

6.5.2 dfgmres

6.5.2.1 dfgmres_init

功能：初始化 FGMRES 求解器
语法：dfgmres_init(n, x, b, RCI_request, ipar, dpar, tmp)
描述：函数 dfgmres_init 初始化解法器,其设置的参数值被解法器 FGMRES 使用,高级用户可以略过这步直接在相应函数设置参数。
提示：在传递给解法器函数这些参数后,用户只有在保证参数满足正确性和一致性条件时才能修改。通过调用 dfgmres_check 可以做正确性和一致性的基本检查,但是不保证此方法能正确运行。
输入参数：

n: 整型变量。存储问题规模。向量 x 和 b 的长度。

x: n 长的实型数组,包含解的初始猜测,通常为 0 或者 b。

b: n 长的实型数组,包含右端向量。
输出参数：

RCI_request: 整型变量。通知函数功能结果。

ipar: 整型数组,长度为 128。

dpar: 实型数组,长度为 128。

tmp: 实型数组,长度为(n,4)。
返回值：
RCI_request= 0: 函数实现功能,正常返回。
RCI_request= -10000: 函数出错。

6.5.2.2 dfgmres_check

功能：检查 FGMRES 法用户定义数据的一致性和正确性
语法：dfgmres_check(n, x, b, RCI_request, ipar, dpar, tmp)
描述：函数 dfgmres_check 检查传递给解法器 dfgmres 的参数的正确性和一致性。然而此函数并不能保证算法给出正确的结果,它只是减少算法参数出错的机率。高级用户如果确定指定给解法器的参数正确就可以略过此步。
提示：在传递给解法器函数这些参数后,用户只有在保证参数满足正确性和一致性条件时才能修改。通过调用 dfgmres_check 可以做正确性和一致性的基本检查,但是不保证此方法能正确运行。
输入参数：
n: 整型变量。存储问题规模。向量 x 和 b 的长度。
x: n 长的实型数组,包含解的初始猜测,通常为 0 或者 b。
b: n 长的实型数组,包含右端向量。
输出参数：
RCI_request: 整型变量。通知函数功能结果。

ipar: 整型数组,长度为 128。
dpar: 实型数组,长度为 128。
tmp: 实型数组,长度为(n,4)。

返回值：

RCI_request= 0: 函数正常返回。
RCI_request= -1100: 函数终止,出现错误。
RCI_request= -1001: 函数返回一些警告信息。
RCI_request= -1010: 为了保持一致,函数修改一些参数。
RCI_request= -1011: 函数返回一些警告信息并修改了一些参数。

6.5.2.3 dfgmres

功能：FGMRES 求解器,计算近似解向量

语法：dfgmres (n, x, b, RCI_request, ipar, dpar, tmp)

描述：函数 dfgmres 采用 FGMRES 法求解近似解向量。调用前向量 x 包含解的初始猜测。为了更新当前近似解到解向量,必须调用 dfgmres_get 函数。在调用 dfgmres_get 函数后,只有当 ipar(13)不等于 0(默认值)时 FGMRES 迭代才能够继续进行。如果 ipar(13)是正数,则最新解重写到右端项 b 中,此时 FGMRES 的重启版本不能运行。当 ipar(13)为正数时,如果右端项必须被保留,则应该在第一次调用 dfgmres_get 时将其存入其它位置。参数 RCI_request 通知用户任务完成或者解法器需要某些计算结果。假定调用 dfgmres_init 函数之前,所有向量长度已经被定义。

输入参数：

n: 整型变量。存储问题规模。向量 x 和 b 的长度。
x: n 长的实型数组,包含解的初始猜测,通常为 0 或者 b。
b: n 长的实型数组,包含右端向量。
tmp: 实型数组,长度为(n,4)。

输出参数：

RCI_request: 整型变量。通知函数功能结果。
ipar: 整型数组,长度为 128。
dpar: 实型数组,长度为 128。
tmp: 实型数组,长度为(n,4)。

返回值：

RCI_request=0: 函数正常返回,解已找到并存在 x 向量中。这个是针对采用默认收敛标准得情况,对用户自定义的收敛标准,参见 RCI_request= 2,4。
RCI_request=-1: 当最大迭代次数已达到但是相对收敛误差限没有满足时返回此信息。
RCI_request=-10: 计算中出现除 0 的错误,通常发生在矩阵退化的情况。也有可能是 dpar 被错误改变,或者当解找到时方法没有停止。
RCI_request=-11: 函数进入无限循环,可能是 ipar(8),ipar(9),ipar(10) 在函数外部被改变。或者没有调用 dfgmres_check 函数。
RCI_request=-12: 函数中某些参数出错,可能是 ipar,dpar 在函数外部被错误改变。
RCI_request= 1: 需要执行矩阵乘以 tmp(ipar(22)),并将结果放入 tmp(ipar(23)),返回控制到 dfgmres 中。
RCI_request= 2: 需要用户执行收敛检测,如果失败,将返回 dfgmres 函数,否则解已找到,调用 dfgmres_get 更新解。

RCI_request= 3: 需要用户对 tmp(ipar(22))应用预条件子,将结果存入 tmp(ipar(23)),返回 dfgmres。
RCI_request= 4: 需要用户检测当前产生向量的范数,如果带上舍入误差不为 0,则返回 dfgmres。否则解已找到,调用 dfgmres_get 更新解。

6.5.2.4 dfgmres_get

功能：提取当前迭代步数,更新解。
语法：dfgmres_get (n, x, b, RCI_request, ipar, dpar, tmp, itercount)
描述：函数 dfgmres_get 提取求解过程中的当前迭代步数。
输入参数：
n: 整型变量。存储问题规模。向量 x 和 b 的长度。
ipar: 整型数组,长度为 128。
dpar: 实型数组,长度为 128。
tmp: 实型数组,长度为(n,4)。
输出参数：
x: n 长的实型数组,如果 ipar(13)=0,返回解的计算值,否则该值未改变。
b: n 长的实型数组,如果 ipar(13)>0,返回解的计算值,否则该值未改变。
RCI_request: 整型变量,返回函数执行结果。
itercount: 返回当前迭代次数。
返回值：
RCI_request= 0: 函数完成功能正常返回。
RCI_request= -12: 参数出错,可能是 ipar,dpar 在函数外部被错误修改。
RCI_request= -10000: 函数失败。

6.5.3 预条件子

6.5.3.1 dcsrilu0

功能： 产生 ILU0 预条件子
语法：dcsrilu0(n, a, ia, ja, bilu0, ipar, dpar, ierr)
描述：采用 CSR 格式,且基于不完全 LU 分解的 ILU0 预条件子
输入参数：
n: 整型变量,矩阵规模。
a: 双精度数组,包含矩阵的非零元。
ia: 整型数组,存储 a 中元素每行的起始下标。
ja: 整型数组,存储 a 中元素的列下标。
ipar: 整型数组,长度为 128。
ipar(31): 指定如果计算中出现 0 对角元的处理方式。如果该值设为 0,则计算终止,返回错误信息,否则,对角元被设定为某一个特定的值,继续计算。
dpar: 实型数组,长度为 128。
dpar(31): 指定计算中跟对角元相比较的小值。如果对角元值比它小,则设置到 dpar(32),否则计算终止,默认值为 1.0D-16。

dpar(32)： 如果对角元小于 dpar(31),则存储对角元值,默认值为 1.0D-10。

输出参数：

bilu0: 实型数组。存储预条件矩阵的非零元,矩阵以 csr 格式存储,ia,ja 跟原始矩阵相同。

ierr: 整型变量。错误标志。

返回值：

ierr=0：函数正常完成任务

ierr=-101：函数被中断,发生如下错误：在 CSR 格式中至少一个对角元被忽略

ierr=-102：函数被中断,因为矩阵包含零对角元

ierr=-103：函数被中断,因为矩阵包含太多值很小的对角元,在将来除的运算中可以会发生溢出

ierr=-104：函数被中断,因为内存工作空间不够

ierr=-105：函数被中断,因为输入矩阵维数小于或者等于 0

ierr=-106：函数被中断,因为列下标矩阵不是按升序排列

6.5.3.2 dcsrilut

功能： 产生 ILUT 预条件子

语法：dcsrilut(n, a, ia, ja, bilut, ibilut, jbilut, tol, maxfil, ipar, dpar, ierr)

描述：采用 CSR 格式,且基于不完全 LU 分解的 ILUT 预条件子

输入参数：

n: 整型变量,矩阵规模。

a: 双精度数组,包含矩阵的非零元。

ia: 整型数组,存储 a 中元素每行的起始下标。

ja: 整型数组,存储 a 中元素的列下标。

tol: 双精度变量,预条件矩阵填充的阀值。

maxfil: 整型变量,每行最大填充数,预条件矩阵带宽的一半,预条件矩阵每行非零元不能超过(2*maxfil+1)。

ipar: 整型数组,长度为 128。

ipar(31): 指定如果计算中出现 0 对角元的处理方式。如果该值设为 0,则计算终止,返回错误信息,否则,对角元被设定为某一个特定的值,继续计算。

dpar: 实型数组,长度为 128。

dpar(31): 如果对角元的值小于 tol 乘以矩阵行范数,并且根据 ipar(31)的值,矩阵没有停止时,需要重置对角元。该参数指定需要乘以矩阵行范数的数值。并将乘积结果赋给对角元。

输出参数：

bilut: 实型数组。存储预条件矩阵的非零元,矩阵以 csr 格式存储,ia,ja 跟原始矩阵相同。矩阵维数为(2*maxfil+1)*n-maxfil*(maxfil+1)+1。

ibilut: 整型数组,维数为 n+1,存储预条件矩阵每行的起始下标。

jbilut: 整型数组,维数与 bilut 相同,存储预条件矩阵每个非零元的列下标。

ierr: 整型变量。错误标志。

返回值：

ierr=0：函数正常完成任务。

ierr=-101：函数被中断,发生如下错误：某些矩阵行的非零元数等于或者小于 0。

Ierr=-102：函数被中断,因为计算的对角元小于阀值与当前行范数的乘积且因 ipar(31)=0 而不能被替换。
ierr=-103：函数被中断,发生如下错误：ia(I+1)小于或者等于 ia(I)。
ierr=-104：函数被中断,因为内存工作空间不够。
ierr=-105：函数被中断,因为 maxfil 的值小于 0。
ierr=-106：函数被中断,因为输入矩阵维数小于或者等于 0
ierr=-107：函数被中断,发生如下错误：矩阵 ja 的某个值小于 0,或者大于 n。
ierr=101：maxfil 的值大于或者等于 n,通过将 maxfil 的值设置为(n-1)后,计算继续进行。
ierr=102：tol 的值小于 0,将 tol 值设置为-tol 后,计算继续进行。
ierr=103：tol 绝对值大于 dpar(31),可能会导致不稳定的结果。
ierr=104：dpar(31)等于 0,可能会导致计算失败。

7 性能调优指导

对使用了__thread_local 关键字在 LDM 中保存数据的程序，xMath 提供 LDM 现场保护功能（默认关闭）。用户能够通过设置环境变量 xMath_PRESERVE_LDM=1 或 WEXML_PRESERVE_LDM=1 来开启该功能。但因为这项功能的实现需要在内存和 LDM 之间来回传输多余数据，因此会对 xMath 库的性能产生不利影响，因此建议若客户程序中没有使用__thread_local 关键字，则可确保该功能一直处于关闭状态。另外，xMath 库中函数的实际性能受输入的影响较大，在接下来的各个小节中，我们将说明 xMath 中各个模块在何种输入条件下能够达到相对较优的性能。

7.1 BLAS 模块

7.1.1 众核版

BLAS 模块包含 Level 1、Level 2 和 Level 3 三个级别的所有函数，三级函数性能均显著受到计算规模的影响，一般情况下，计算规模越大，性能越优。其中，Level 1 和 Level 2 级函数具有访存受限的特点,我们在此两级函数中进行了线程数自动调优的机制帮助用户充分利用访存带宽，使其在一般情况下也有较优性能，且不需要用户进行其它设置。另一方面，Level 3 级函数具有计算密集型的特点，我们通过计算与访存重叠的方式进行了优化，用户在调用时矩阵首地址为 32 Bytes 对齐，且矩阵维度 M 维为 128 的倍数，N 维为 256 的倍数，K 维为 512 的倍数时性能较优。

7.1.2 片上多核版

片上多核版仅包含主核函数，最大线程数为 4，用户调用时需通过标准的 OpenMP 环境变量(OMP_NUM_THREADS)进行设置。多核版 BLAS 显著受到线程数以及计算规模的影响，一般情况下，线程数越大，以及矩阵规模越大，性能越大；另外由于此版本函数仅包含主核函数，矩阵地址为 32 Bytes 字节带来的性能提升可能不大。

7.2 LAPACK 模块

7.2.1 众核版

LAPACK 模块的性能显著地受到计算规模的影响，因此在矩阵规模为 2048（经验值）之下时，并不建议使用众核版 LAPACK。另外，矩阵首地址是否为 32 Byte 对齐也对性能的高低有一定影响，因此建议用户在产生矩阵时，使用类似 memalign 函数，产生首地址为 32 Byte 整数倍的存储空间。在众核 LAPACK 库中，用户不需要自行调整矩阵的分块大小，对每个函数的最优分块规模已经被开发人员手工调整到较优，并固化在函数 ilaenv.f 文件中。用户也不需要手动配置每个 LAPACK 函数使用的从核核数，LAPACK 库有线程数自动适应功能，确保在一般情况下能产生较优的性能。

7.2.2 片上多核版

与众核情况类似，用户在矩阵规模为 1024（经验值）之下时，不应将线程数量设置过多。多核 LAPACK 的线程使用数量可以使用标准的 OpenMP 环境变量（OMP_NUM_THREADS）进行设置。另外矩阵首地址是否为 32 Byte 对多核 LAPACK 的性能并无显著影响。用户也不需要设置分块大小，各个函数的最优分块性能业已由开发人员人工配置并固化，确保在一般情况下的较优性能。

7.3 FFT 模块

7.3.1 众核版

xMath 众核版 FFT 使用访存计算重叠的双缓存技术，性能与输入数据规模相关。对于一维 FFT，数据规模小于 512 时，计算量较少，与主核上运行，大于 512 时，于从核上调用 8 个或 64 个主核运行，数据规模越大，性能越高，例如数据规模为 131072 时，性能较优。对于多维 FFT 计算，行数较大时，性能较优。同时，性能与数据首地址 32Byte 对齐息息相关，建议用户使用类似 memalign 函数，以产生首地址为 32Byte 整数倍的存储空间。

7.3.2 片上多核版

xMath 多核版 FFT 性能同样与输入数据规模相关，数据规模较大时，性能较高，例如，数据规模为 131072 时，性能较优。建议用户于数据规模较小时（1024），不宜将线程数量设置过多。

7.4迭代解法器模块

7.4.1 众核版

稀疏迭代解法器模块的性能与输入矩阵的结构紧密相关。由于实际应用中产生的稀疏矩阵的结构种类繁多，所以，很难产生一种使所有结构都达到最高性能的稀疏求解器。XMath 稀疏解法器对处理以下特征的稀疏矩阵，能达到很高的性能：

1.稀疏矩阵的非零元素排布比较紧密；

2.稀疏矩阵的各个行的非零元的个数差别不大；

在稀疏迭代解法器库中，用户不需要手动配置使用的从核核数，解法器库中有线程数自动适应功能，确保在一般情况下能产生较优的性能。

7.4.2 片上多核版

同众核版。

8 参考文献

1. G. W. STEWART, On the sensitivity of the eigenvalue problem Ax=λBx, SIAM J. Num. Anal., 9 (1972), pp. 669-686.
2. J. H. WILKINSON, Kronecker's canonical form and the QZ algorithm, Lin. Alg. Appl., 28 (1979), pp. 285-303.
3. J. W. DEMMEL AND B. KÅGSTR¨OM, Computing stable eigendecompositions of matrix pencils, Lin. Alg. Appl., 88/89 (1987), pp. 139-186.
4. G. W. STEWART, Error and perturbation bounds for subspaces associated with certain eigenvalue problems, SIAM Review, 15 (1973), pp. 727-764.
5. G. GOLUB AND C. F. VAN LOAN, Matrix Computations, Johns Hopkins University Press, Baltimore, MD, third ed., 1996.
6. C. PAIGE, Computing the generalized singular value decomposition, SIAM J. Sci. Stat., 7 (1986), pp. 1126-1146.
7. Z. BAI AND J. W. DEMMEL, Computing the generalized singular value decomposition, SIAM J. Sci. Comp., 14 (1993), pp. 1464-1486.
8. Z. BAI, , AND H. ZHA, A new preprocessing algorithm for the computation of the generalized singular value decomposition, SIAM J. Sci. Comp., 14 (1993), pp. 1007-1012.
9. E. ANDERSON, Z. BAI, AND J. DONGARRA, Generalized QR factorization and its applications, Linear Algebra and Its Applications, 162-164 (1992), pp. 243-271.
10. A. GREENBAUM AND J. J. DONGARRA, Experiments with QL/QR methods for the symmetric tridiagonal eigenproblem, Computer Science Dept. Technical Report CS-89-92, University of Tennessee, Knoxville,TN, 1989.
11. Z. BAI, J. W. DEMMEL, AND A. MCKENNEY, On computing condition numbers for the nonsymmetric eigenproblem, ACM Trans. Math. Softw., 19 (1993), pp. 202-223.

12. L. KAUFMAN, Banded eigenvalue solvers on vector machines, ACM Trans. Math. Softw., 10 (1984), pp. 73-86.
13. V. FERNANDO AND B. PARLETT, Accurate singular values and differential qd algorithms, Numerisch Math., 67 (1994), pp. 191-229.
14. M. GU AND S. EISENSTAT, A divide-and-conquer algorithm for the bidiagonal SVD, SIAM J. Mat. Anal. Appl., 16 (1995), pp. 79-92.
15. R. C. WARD, Balancing the generalized eigenvalue problem, SIAM J. Sci. Stat. Comput., 2 (1981), pp. 141-152.
16. B. KÅGSTRÖM, A direct method for reordering eigenvalues in the generalized real Schur form of a regular matrix pair (a,b), in Linear Algebra for Large Scale and Real-Time Applications, Kluwer Academic Publishers, 1993, pp. 195-218.
17. B. KÅGSTR¨OM AND P. POROMAA, Computing eigenspaces with specified eigenvalues of a regular matrix pair (A,B) and condition estimation: Theory, algorithms and software, Tech. Rep. UMINF 94.04, Department of Computing Science, Umeå University, 1994.
18. R. LEHOUCQ, The computation of elementary unitary matrices, Computer Science Dept. Technical Report CS-94-233, University of Tennessee, Knoxville, TN, 1994.
19. Intel(R) Math Kernel Library Reference Manual。
20. http://www.intel.com/software/products/CLEXML/docs/WebHelp/whnjs.html
21. Sparsekit: a basic tool kit for sparse matrix computations, Youcef Saad, Technical Report, Computer Science Department, University of Minnesota, June 1994.
22. Direct methods for sparse linear systems, Timothy A. Davis,, 2006-09-15.
23. Templates for the solution of Linear System: Building Blocks for Iterative method.
24. D.M.Young. Iterative Solution of Large Linear Systems. New York, Academic Press, Inc., 1971.
25. Y. Saad. Iterative Methods for Sparse Linear Systems. 2nd edition,SIAM, Philadelphia, PA, 2003.
26. Matrix Market http://math.nist.gov/MatrixMarket/.