LSQR求解如下方程中的x向量：

1 非对称线性方程组 – A\*x = b

2 线性最小二乘解 – A\*x = b，在最小二乘意义下

3 阻尼最小二乘解 – ( A )\*x = ( b )

( damp\*I ) = ( 0 )，在最小二乘意义下

A是m行n列的矩阵，x是长度为n的向量，b是长度为m的向量，damp是阻尼标量。

A可以是大型的稀疏矩阵。

注意

下列标记在参数解释中可能提到：

Abar = ( A ), bbar = ( b )

( damp\*I ) ( 0 )

r = b – A\*x, rbar = bbar – Abar\*x

rnorm = sqrt ( norm(r)\*\*2 + damp\*\*2 \* norm(x)\*\*2 )

= norm (rbar)

rel\_prec = 计算机的浮点型数据有效位数

LSQR在最小二乘意义下解x使rnorm最小

void lsqr (lsqr\_input \*input, lsqr\_output \*output, lsqr\_work \*work, lsqr\_func \*fuc);

typedef struct LSQR\_INPUTS {

long num\_rows;

long num\_cols;

double damp\_val;

double rel\_mat\_err;

double rel\_rhs\_err;

double cond\_lim;

long max\_iter;

FILE \*lsqr\_fp\_out;

dvec \*rhs\_vec;

dvec \*sol\_vec;

} lsqr\_input;

num\_rows: 矩阵A的行数；

num\_cols: 矩阵A的列数；

damp\_val: 阻尼系数，范围一般在0 – sqrt(‘rel\_prec’)\*norm(A)；

rel\_mat\_err: 矩阵A中元素的精度，参数给零的话认为A中元素的精度和计算机位数精度相同；

rel\_rhs\_err: 方程右端向量b的精度，参数给零的话认为b中元素的精度和计算机位数精度相同；

cond\_lim: 矩阵Abar条件数的上限，若Abar的条件数超过该参数，迭代终止；

max\_iter: 最大迭代次数，建议值4\*n；

lsqr\_fp\_out: 打印的东西输入至此文件；

rhs\_vec: 存储右端向量b，长度为num\_rows，每次迭代的b覆盖上次的b；

sol\_vec: 解向量x的初始值，长度为num\_cols，每次迭代更新的x覆盖上次的x。

typedef struct LSQR\_OUTPUTS {

long term\_flag;

long num\_iters;

double frob\_mat\_norm;

double mat\_cond\_num;

double resid\_norm;

double mat\_resid\_norm;

double sol\_norm;

dvec \*sol\_vec;

dvec \*std\_err\_vec;

} lsqr\_output;

Term\_flag: 标记不同的程序结束方式：

1. x = x0 是精确解，没有迭代；
2. 方程 A\*x = b 的解是兼容的，达到rel\_mat\_err 和 rel\_rhs\_err 的精度；
3. 方程 A\*x = b 的解可能不兼容，在最小二乘的意义下得到了高精度的解，达到 rel\_mat\_err 的精度；
4. 估计 Abar 的条件数超过了给定的上限 cond\_lim ，方程是病态的，子函数 APROD 中存在误差；
5. 方程 A\*x = b 的解可能兼容，A\*x – b 的范数尽量小在合理的范围内尽量小；
6. 方程 A\*x = b 的解可能不兼容，在合理的范围内得到精度尽量高的最小二乘解；
7. Abar的条件数可能特别大以至于不能继续迭代，以达到给定的精度。

在子函数 APROD 中有误差。

Num\_iters: 迭代次数；

Frob\_mat\_norm: Abar 的 Frobenius 范数估计，如果 damp\_val 很小，并且 A 的列向量的长度均为1， frob\_mat\_norm 应该增加至接近sqrt(n)。frob\_mat\_norm根本上的差异值可能是指子函数 APROD 中的误差（可能是1和2之间的一个易变的值）；

Mat\_cond\_num: 估计得到的矩阵 Abar 的条件数，如果该数很大的话，指向子函数 APROD 中的错误；

Resid\_norm: 最终当目标函数极小是， rbar 的范数的估计值；

Mat\_resid\_norm: 最终的Abar’\*rbar 的范数的估计值；

Sol\_norm: 最终解x的范数的估计值；

Sol\_vec: 储存”re.urns”计算x的结果，长度为num\_cols；

Std\_err\_vec: 计算结果x的标注误差向量，长度为 num\_cols ，其计算方法为

Resid\_norm \* sqrt (sigma(i,i) / T)

Sigma(i,i)是矩阵 Abar’ \* Abar 的第i个对角值，如果m <= n ，T = 1，如果m > n，T = m – n，如果 damp\_val != 0 ，T = m。

typedef struct LSQR\_WORK {

dvec \*bidiag\_wrk\_vec;

dvec \*srch\_dir\_vec;

} lsqr\_work;

Bidiag\_wrk\_vec: 该向量是一次迭代中Lanczos双对角化的工作空间，长度为num\_cols；

Srch\_dir\_vec: 一次迭代中存储搜索方向的向量，长度为num\_cols；

typedef struct LSQR\_FUNC {

void (\*mat\_vec\_prod) (long, dvec \*, dvec \*, void \*);

} lsqr\_func;

Mat\_vec\_prod:

定义：

lsqr\_input \*test\_in;

lsqr\_output \*test\_out;

lsqr\_work \*test\_work;

lsqr\_func \*test\_func;

prod\_data \*test\_prod;

分配内存：

alloc\_lsqr\_mem( &test\_in, &test\_out, &test\_work, &test\_func, num\_rows, num\_cols );

输入：

test\_func->mat\_vec\_prod = test\_mult;

test\_in->lsqr\_fp\_out = stdout;

dvec\_copy( act\_rhs\_vec, test\_in->rhs\_vec );

for( col\_indx = 0; col\_indx < num\_cols; col\_indx++ )

test\_in->sol\_vec->elements[col\_indx] = 0.0;

test\_in->num\_rows = num\_rows;

test\_in->num\_cols = num\_cols;

test\_in->damp\_val = damp\_param;

test\_in->rel\_mat\_err = 1.0e-15; /\* Previously 1.0e-10; \*/

test\_in->rel\_rhs\_err = 1.0e-15; /\* Previously 1.0e-10; \*/

test\_in->cond\_lim = 10.0 \* act\_mat\_cond\_num;

test\_in->max\_iter = num\_rows + num\_cols + 50;