



中国科学院上海天文台

Shanghai Astronomical Observatory, Chinese Academy of Science



中国科学院大学

University of Chinese Academy of Sciences

# 参数估计方法

宋叶志  
2019年秋季

作业邮箱: [song.yz@foxmail.com](mailto:song.yz@foxmail.com)

课件地址: <http://202.127.29.4/astrodynamics/course.php>

# 线性超定方程

$$y_1 = H_1 \mathbf{x}_k + \epsilon_1; \quad w_1$$

$$y_2 = H_2 \mathbf{x}_k + \epsilon_2; \quad w_2$$

$$\vdots \quad \vdots \quad \vdots$$

$$y_\ell = H_\ell \mathbf{x}_k + \epsilon_\ell; \quad w_\ell$$

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_\ell \end{bmatrix}; \quad H = \begin{bmatrix} H_1 \\ H_2 \\ \vdots \\ H_\ell \end{bmatrix};$$

$$\boldsymbol{\epsilon} = \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_\ell \end{bmatrix}; \quad W = \begin{bmatrix} w_1 & 0 & \cdots & 0 \\ 0 & w_2 & \cdots & 0 \\ & & \ddots & \\ 0 & 0 & \cdots & w_\ell \end{bmatrix}$$

# 最优估值

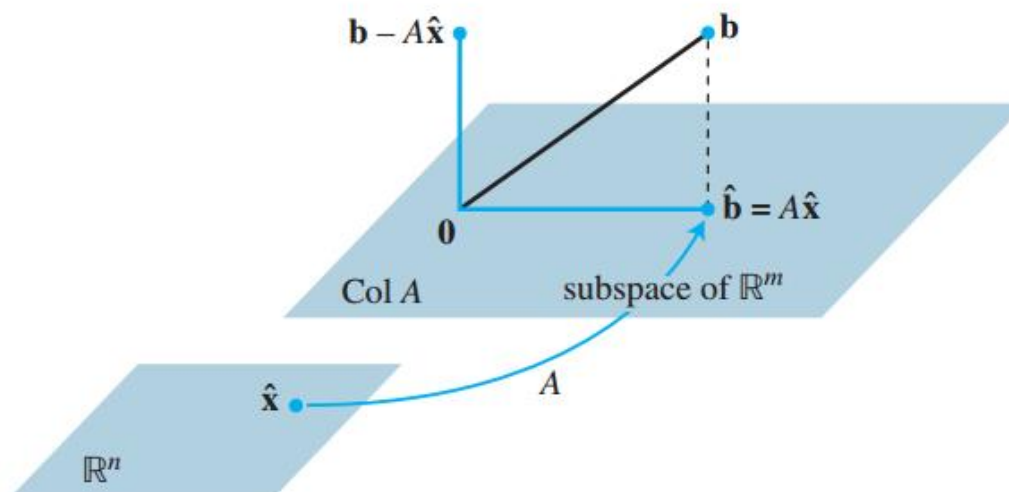
$$J(\mathbf{x}_k) = 1/2 \boldsymbol{\epsilon}^T W \boldsymbol{\epsilon} = \sum_{i=1}^{\ell} 1/2 \boldsymbol{\epsilon}_i^T w_i \boldsymbol{\epsilon}_i$$

$$J(\mathbf{x}_k) = 1/2 (\mathbf{y} - H \mathbf{x}_k)^T W (\mathbf{y} - H \mathbf{x}_k).$$

$$\frac{\partial J}{\partial \mathbf{x}_k} = 0 = -(\mathbf{y} - H \mathbf{x}_k)^T W H = -H^T W (\mathbf{y} - H \mathbf{x}_k).$$

$$(H^T W H) \mathbf{x}_k = H^T W \mathbf{y}. \quad P_k = (H^T W H)^{-1}.$$

# 几何解释



The least-squares solution  $\hat{\mathbf{x}}$  is in  $\mathbb{R}^n$ .

Suppose  $\hat{\mathbf{x}}$  satisfies  $A\hat{\mathbf{x}} = \hat{\mathbf{b}}$ . By the Orthogonal Decomposition Theorem in Section 6.3, the projection  $\hat{\mathbf{b}}$  has the property that  $\mathbf{b} - \hat{\mathbf{b}}$  is orthogonal to  $\text{Col } A$ , so  $\mathbf{b} - A\hat{\mathbf{x}}$  is orthogonal to each column of  $A$ . If  $\mathbf{a}_j$  is any column of  $A$ , then  $\mathbf{a}_j \cdot (\mathbf{b} - A\hat{\mathbf{x}}) = 0$ , and  $\mathbf{a}_j^T (\mathbf{b} - A\hat{\mathbf{x}}) = 0$ . Since each  $\mathbf{a}_j^T$  is a row of  $A^T$ ,

$$A^T (\mathbf{b} - A\hat{\mathbf{x}}) = \mathbf{0}$$

$$A^T \mathbf{b} - A^T A \hat{\mathbf{x}} = \mathbf{0}$$

$$A^T A \hat{\mathbf{x}} = A^T \mathbf{b}$$

These calculations show that each least-squares solution of  $A\mathbf{x} = \mathbf{b}$  satisfies the equation

$$A^T A \mathbf{x} = A^T \mathbf{b}$$

The matrix equation (3) represents a system of equations called the **normal equations** for  $A\mathbf{x} = \mathbf{b}$ . A solution of (3) is often denoted by  $\hat{\mathbf{x}}$ .

# 先验信息与序贯问题

$$\mathbf{y}_1 + \mathbf{v}_1 = \mathbf{A}_1 \mathbf{p}_1 \quad \text{with} \quad \mathbf{D}(\mathbf{y}_1) = \sigma_1^2 \mathbf{P}_1^{-1}$$

$$\mathbf{y}_2 + \mathbf{v}_2 = \mathbf{A}_2 \mathbf{p}_2 \quad \text{with} \quad \mathbf{D}(\mathbf{y}_2) = \sigma_2^2 \mathbf{P}_2^{-1}$$

$$\begin{bmatrix} \hat{\mathbf{p}}_1 \\ \hat{\mathbf{p}}_2 \end{bmatrix} + \begin{bmatrix} \mathbf{v}_{p_1} \\ \mathbf{v}_{p_2} \end{bmatrix} = \begin{bmatrix} \mathbf{I} \\ \mathbf{I} \end{bmatrix} \hat{\mathbf{p}}_c \quad \text{with} \quad \mathbf{D} \left( \begin{bmatrix} \hat{\mathbf{p}}_1 \\ \hat{\mathbf{p}}_2 \end{bmatrix} \right) = \sigma_c^2 \begin{bmatrix} \boldsymbol{\Sigma}_1 & \boldsymbol{\emptyset} \\ \boldsymbol{\emptyset} & \boldsymbol{\Sigma}_2 \end{bmatrix}.$$

$$\underbrace{\left[ \mathbf{A}_1^T \mathbf{P}_1 \mathbf{A}_1 \right]}_{\mathbf{N}_1} + \underbrace{\left[ \mathbf{A}_2^T \mathbf{P}_2 \mathbf{A}_2 \right]}_{\mathbf{N}_2} \hat{\mathbf{p}}_c = \underbrace{\left[ \mathbf{A}_1^T \mathbf{P}_1 \mathbf{y}_1 \right]}_{\mathbf{b}_1} + \underbrace{\left[ \mathbf{A}_2^T \mathbf{P}_2 \mathbf{y}_2 \right]}_{\mathbf{b}_2}$$

$$\Omega_c = \sum_{i=1}^m \mathbf{y}_i^T \mathbf{P}_i \mathbf{y}_i - \sum_{i=1}^m \mathbf{y}_i^T \mathbf{P}_i \mathbf{A}_i \hat{\mathbf{p}}_c$$

$$\hat{\sigma}_c^2 = \frac{1}{f_c} \left( \sum_{i=1}^m \mathbf{y}_i^T \mathbf{P}_i \mathbf{y}_i - \sum_{i=1}^m \mathbf{y}_i^T \mathbf{P}_i \mathbf{A}_i \hat{\mathbf{p}}_c \right)$$

# 对称正定矩阵Cholesky分解方法

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & & \ddots & \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} = \begin{bmatrix} l_{11} & & & \\ l_{21} & l_{22} & & \\ \cdots & & \ddots & \\ l_{n1} & l_{n2} & \cdots & l_{nn} \end{bmatrix} \begin{bmatrix} l_{11} & l_{21} & \cdots & l_{n1} \\ & l_{22} & \cdots & l_{n2} \\ & & \ddots & \\ & & & l_{nn} \end{bmatrix}$$

**STEP1**  $l_{11} = \sqrt{a_{11}}$

**STEP2**  $l_{i1} = \frac{a_{i1}}{l_{11}}, i = 2, N$

**STEP3** 对  $j = 2, N$ , 做 STEP4~STEP5。

**STEP4**  $l_{jj} = \sqrt{a_{jj} - \sum_{k=1}^{j-1} l_{jk}^2}$

**STEP5**  $l_{ij} = \frac{\left( a_{ij} - \sum_{k=1}^{j-1} l_{ik} l_{jk} \right)}{l_{jj}}, \quad i = j + 1, N$

# LDL分解

$$\mathbf{A} = \begin{bmatrix} 1 & & & \\ l_{21} & 1 & & \\ \vdots & \vdots & \ddots & \\ l_{n1} & l_{n2} & \cdots & 1 \end{bmatrix} \begin{bmatrix} d_1 & & & \\ & d_2 & & \\ & & \ddots & \\ & & & d_n \end{bmatrix} \begin{bmatrix} 1 & l_{21} & \cdots & l_{n1} \\ & 1 & \cdots & l_{n2} \\ & & \ddots & \vdots \\ & & & 1 \end{bmatrix}$$

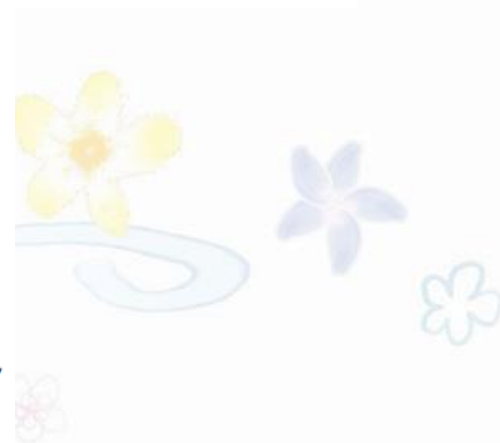
$$d_1 = a_{11}$$

$$g_{ij} = a_{ij} - \sum_{k=1}^{j-1} g_{ik} l_{jk} \quad (j = 1, \dots, i-1)$$

$$l_{ij} = \frac{g_{ij}}{d_j} \quad (j = 1, \dots, i-1)$$

$$d_i = a_{ii} - \sum_{k=1}^{i-1} g_{ik} l_{ik}$$

$i = 2, \dots, n$



# 由LDL分解计算协方差

```
subroutine covariance(L,D,P,N)
!-----
! Purpose : 2019-04-15 13:08 (Created)
! 根据LDL'分解计算协方差矩阵
!-----
! Input Parameters :
!   L   -----
!   D   ----  对角线元素
!   N   ----  矩阵维数
! Output Parameters :
!-----
! Author      : Song Yezhi      <song.yz@foxmail.com>
! Copyright (C) : Shanghai Astronomical Observatory, CAS
!               (All rights reserved, 2019)
!-----
implicit none
integer      :: N
real*8      :: L(N,N), D(N), P(N,N)
!-----
integer      :: i
real*8      :: invL(N,N), invLT(N,N)
!-----
call inv_dtri(L,invL,N)
invLT = transpose(invL)
do i =1,N
    invLT(:,i)=invLT(:,i)/D(i)
end do
P = matmul(invLT,invL)
end subroutine covariance
```



# 快速Givens变换

Sum = 0

$U_{ii} = 1 \quad i = 1, \dots, n$

1. Do  $k = 1, \dots, m$

$$\delta_k = 1$$

2. Do  $i = 1, \dots, n$

If ( $h_{ki} = 0$ ) Go to 2

$$d'_i = d_i + \delta_k h_{ki}^2$$

$$\bar{C} = d_i / d'_i$$

$$\bar{S} = \delta_k h_{ki} / d'_i$$

$$y'_k = y_k - \bar{b}_i h_{ki}$$

$$\bar{b}_i = \bar{b}_i \bar{C} + y_k \bar{S}$$

$$y_k = y'_k$$

$$\delta_k = \delta_k \bar{C}$$

$$d_i = d'_i$$

3. Do  $j = i + 1, \dots, n$

$$h'_{kj} = h_{kj} - U_{ij} h_{ki}$$

$$U_{ij} = U_{ij} \bar{C} + h_{kj} \bar{S}$$

$$h_{kj} = h'_{kj}$$

$$\begin{array}{c} \underbrace{\quad n \quad} \quad \underbrace{\quad 1 \quad} \\ \left[ \begin{array}{cc} \bar{R} & \bar{\mathbf{b}} \\ H & \mathbf{y} \end{array} \right] \begin{array}{l} \}n \\ \}m \end{array} = \begin{array}{c} \underbrace{\quad n+1 \quad} \\ \left[ \begin{array}{c} \tilde{R} \\ \tilde{H} \end{array} \right] \begin{array}{l} \}n \\ \}m \end{array} \end{array}$$

Next  $j$

Next  $i$

$$e_k = \sqrt{\delta_k} y_k$$

$$\text{Sum} = \text{Sum} + e_k^2$$

Next  $k$

# Householder变换

如果给定向量  $\mathbf{x}, \mathbf{y}$ , 二者 2 范数相同, 即  $\|\mathbf{x}\|_2 = \|\mathbf{y}\|_2$ , 则可以找到正交变换  $\mathbf{H}$ , 使  $\mathbf{H}\mathbf{x} = \mathbf{y}$ 。而变换矩阵很容易给出, 即著名的 Householder 变换或称为镜像变换。取

$$\mathbf{u} = \frac{\mathbf{x} - \mathbf{y}}{\|\mathbf{x} - \mathbf{y}\|_2}$$

令

$$\mathbf{H} = \mathbf{I} - 2\mathbf{u}\mathbf{u}^T$$

即可。很容易证明  $\mathbf{H}$  为正交矩阵, 其变换保持向量 2 范数不变。

对于单个向量  $\mathbf{x}$ , 欲用镜像变换使之成为

$$\mathbf{H}\mathbf{x} = \mathbf{w} = \|\mathbf{x}\| \mathbf{e}_1, \mathbf{e}_1 = (1, 0, \dots)^T$$

可以令

$$\mathbf{u} = \mathbf{w} - \mathbf{x}$$

继而可以构造镜像变换矩阵

$$\mathbf{H} = \mathbf{I} - 2 \frac{\mathbf{u}\mathbf{u}^T}{\|\mathbf{u}\|_2^2}$$

$\|\mathbf{u}\|_2^2$  表示向量之 2 范数的平方。

# Householder变换

实施变换时为了让作为分母的  $\|\mathbf{u}\|_2^2$  尽可能的大，从而有利于数值稳定，取

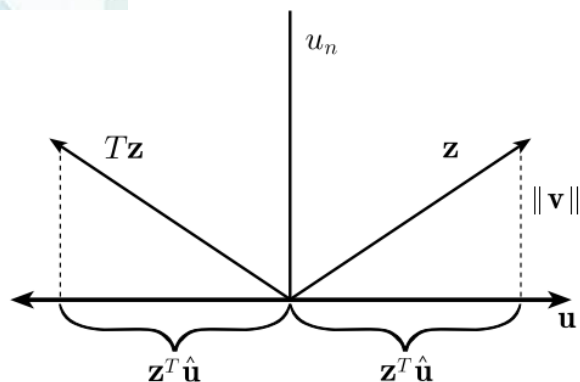
$$k = -\text{sgn}(x_1) \|\mathbf{x}\|_2, \text{sgn}(x_1) = \begin{cases} 1, & x_1 \geq 0 \\ -1, & x_1 < 0 \end{cases}$$

$$\mathbf{u} = (x_1 + \text{sgn}(x_1) \|\mathbf{x}\|_2, x_2, \dots, x_m)^T$$

意义很明了，及当  $\mathbf{x}$  的第一个分量大于等于 0 时候， $\mathbf{u}$  的第一个分量取  $x_1$  与  $\|\mathbf{x}\|_2$  的和，当  $\mathbf{x}$  的第一个分量小于 0 时候， $\mathbf{u}$  的第一个分量取  $x_1 - \|\mathbf{x}\|_2$ 。如果不这样做有可能会损失有效位数。

对于矩阵  $\mathbf{A}$  做 QR 分解，即连续使用 Householder 变换。如第一次使用正交变换后，使之成为如下形式：

$$\mathbf{H}_1 \mathbf{A} = \begin{bmatrix} * & * & * & \cdots & * \\ 0 & * & * & \cdots & * \\ 0 & * & * & \cdots & * \\ 0 & * & * & \cdots & * \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & * & * & \cdots & * \end{bmatrix}$$



# Householder变换

第二次变换使之成为如下形式：

$$H_2 H_1 A = \begin{bmatrix} * & * & * & \cdots & * \\ 0 & * & * & \cdots & * \\ 0 & 0 & * & \cdots & * \\ 0 & 0 & * & \cdots & * \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & * & \cdots & * \end{bmatrix}$$

如此循环到矩阵的最后一列，至此变换后的矩阵已经成为上三角矩阵。而变换矩阵即  $H = \dots H_2 H_1$ 。

由数值代数理论可知，如果  $R$  对角元素正负号都选正号或者负号，则 QR 分解是唯一的。

计算  $H$  时候，并不需要存储各次的变换结果，而是逐步矩阵累成而得，变换到矩阵的最后一列，新的矩阵已经是  $H$ 。上三角阵也不需要再执行  $R = H^T A$ ，当变换到最后一列时已经自动形成上三角阵，该矩阵即为  $R$ 。

# 修正的Gram-Schmidt 正交化方法

设  $\{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$  是  $p$  维向量空间  $W$  的任意一组基, 则子空间  $W$  的标准正交基  $\{\mathbf{u}_1, \mathbf{u}_2, \dots, \mathbf{u}_n\}$  可以通过 Gram-Schmidt 正交化构造, 这个方法是大家都很熟悉的, 即

$$\mathbf{p}_1 = \mathbf{x}_1, \mathbf{u}_1 = \frac{\mathbf{p}_1}{\|\mathbf{p}_1\|} = \frac{\mathbf{x}_1}{\|\mathbf{x}_1\|}$$

$$\mathbf{p}_k = \mathbf{x}_k - \sum_{i=1}^{k-1} (\mathbf{v}_i^H \mathbf{x}_k) \mathbf{u}_i, \mathbf{u}_k = \frac{\mathbf{p}_k}{\|\mathbf{p}_k\|}$$

对于超定的线性方程系数矩阵的 QR 分解, 可以通过 Gram-Schmidt 正交化方法来实现, 然而采用 Gram-Schmidt 正交化方法求解列正交矩阵  $Q$  时, 舍入误差较大, 这在求解最小二乘法时候, 有时会不稳定。针对 Gram-Schmidt 正交化的缺点, 下面给出修正的 Gram-Schmidt 正交化算法。

# 修正的Gram-Schmidt 正交化方法

对于  $n$  个向量  $\mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_n$  构造标准正交基  $\mathbf{q}_1, \mathbf{q}_2, \dots, \mathbf{q}_n$  方法如下:

$$R_{11} = \|\mathbf{a}_1\|$$

$$\mathbf{q}_1 = \frac{\mathbf{a}_1}{R_{11}}$$

对于  $k = 2, \dots, n$

$$R_{jk} = \mathbf{q}_j^H \mathbf{a}_k, j = 1, \dots, k-1$$

$$R_{kk} = \left\| \mathbf{a}_k - \sum_{j=1}^{k-1} \mathbf{q}_j R_{jk} \right\|$$

$$\mathbf{q}_k = \frac{\mathbf{a}_k - \sum_{j=1}^{k-1} \mathbf{q}_j R_{jk}}{R_{kk}}$$

# 扩展卡尔曼滤波

Given:  $P_{k-1}$ ,  $\hat{\mathbf{X}}_{k-1}$  and  $\mathbf{Y}_k$ ,  $R_k$ .

(1) Integrate from  $t_{k-1}$  to  $t_k$ ,

$$\begin{aligned}\dot{\mathbf{X}}^* &= F(\mathbf{X}^*, t), & \mathbf{X}^*(t_{k-1}) &= \hat{\mathbf{X}}_{k-1} \\ \dot{\Phi}(t, t_{k-1}) &= A(t)\Phi(t, t_{k-1}), & \Phi(t_{k-1}, t_{k-1}) &= I.\end{aligned}$$

(2) Compute

$$\begin{aligned}\bar{P}_k &= \Phi(t_k, t_{k-1})P_{k-1}\Phi^T(t_k, t_{k-1}) \\ \mathbf{y}_k &= \mathbf{Y}_k - G(\mathbf{X}_k^*, t_k) \\ \tilde{H}_k &= \partial G(\mathbf{X}_k^*, t_k)/\partial \mathbf{X}_k.\end{aligned}$$

(3) Compute

$$\begin{aligned}K_k &= \bar{P}_k \tilde{H}_k^T [\tilde{H}_k \bar{P}_k \tilde{H}_k^T + R_k]^{-1} \\ \hat{\mathbf{X}}_k &= \mathbf{X}_k^* + K_k \mathbf{y}_k \\ P_k &= [I - K_k \tilde{H}_k] \bar{P}_k.\end{aligned}$$

(4) Replace  $k$  with  $k + 1$  and return to (1).

# cmatlib矩阵类库定义 (C++)

```
class VEC
/*-----
Versions and Changes :
v1.0----2015-7-18
vector class
下标索引算符为()
-----
Public Paras:
int size1 ----- rows of matrix
int size2 ----- cols of matrix
Functions :
VEC (M) ---- creat a vector with lenght of M
operator() --- access the component
-----
Author      : 宋叶志 <song.yz@foxmail.com>
Copyright(C) : Shanghai Astronomical Observatory, CAS
              (All rights reserved)          2015
-----*/
{
private:
double *x ;
int M1;
public:
// Constructors
VEC (int M)
//creat a vector
{
M1=M;
x= new double [M] ;
for(int i=0;i<M;i++)
x[i]=0.0;
}
// Destructor
~ VEC ()
{ delete [] x; }

double& operator() (int i)
// Component access
{ return x[i]; }
double operator () (int i) const { return x[i]; };
int size() const { return M1; };
};
```

```
void output(int width,int precision);
//output to the screen
void setv(double value);
//set the elements of the vector to a double value
//+,-,*,/ for the same index of two vectors
//which means that the size of the vectors must be the same
friend VEC operator + (const VEC& V1, const VEC& V2); //V1+V2
friend VEC operator - (const VEC& V1, const VEC& V2); //V1-V2
friend VEC operator * (const VEC& V1, const VEC& V2); //V1.*V2,
friend VEC operator / (const VEC& V1, const VEC& V2); //V1./V2,
//V1./V2
//+,-,*,/ for the vector and a scala
// the operation works on every elements of the vector
friend VEC operator + (const VEC& V1, double a); //V1+a
friend VEC operator - (const VEC& V1, double a); //V1-a
friend VEC operator * (const VEC& V1, double a); //V1*a
friend VEC operator / (const VEC& V1, double a); //V1/a
};
```





# cmatlib矩阵类库定义

```
class MAT
/*-----
Versions and Changes :
v1.0----2015-7-18
matrix class
在数组基础上构造,下标算符重载()
-----
Public Paras:
int size1 ----- rows of matrix
int size2 ----- cols of matrix
Functions :
MAT(M,N) ---- creat a vector with the size of (M,N)
operator() --- access the component
-----
Author : 宋叶志 <song.yz@foxmail.com>
Copyright(C) : Shanghai Astronomical Observatory, CAS
(All rights reserved) 2015
-----*/
{
private:
double ** A;
int M1,N1;
public:
// Constructors
MAT(int M,int N)
{
M1=M;
N1=N;

int i,j;

A= new double *[M];
for(i=0;i<M;i++)
A[i]=new double [N] ;

// set the initial value to zero
for (i=0;i<M;i++)
for(j=0;j<N;j++)
A[i][j]=0.0;
}
```

```
// Destructor
~MAT()
{
int i;
for(i=0;i<M1;i++)
delete[] A[i];
delete[] A;
}

//Component access
double operator() (int i,int j) const { return A[i][j]; };
double & operator() (int i,int j) { return A[i][j]; };

int size1() const { return M1; };
int size2() const { return N1; };

void output(int width,int precision);
//output to the screen
void setv(double value);
//set the elements of the matrix to a double value

//+,-,*,/ for the same index of two matrixs
//which means that the size of the matrixs must be the same
friend MAT operator + (const MAT& A1, const MAT& A2); //A1+A2
friend MAT operator - (const MAT& A1, const MAT& A2); //A1-A2
friend MAT operator * (const MAT& A1, const MAT& A2); //A1.*A2
friend MAT operator / (const MAT& A1, const MAT& A2); //A1./A2

//+,-,*,/ for the matrix and a scala
// the operation works on every elements of the matrix
friend MAT operator + (const MAT& A1, double a); //A1+a
friend MAT operator - (const MAT& A1, double a); //A1-a
friend MAT operator * (const MAT& A1, double a); //A1*a
friend MAT operator / (const MAT& A1, double a); //A1/a
};
```

# cmatlib向量运算与矩阵分解

```
//-----基本向量运算
void veccopy(VEC &b,const VEC &a);
//按值复制一个向量
double vecdot(const VEC &v1,const VEC &v2);
//向量内积
double norm(const VEC &v);
//向量2范数
//-----基本矩阵运算
void matcopy(MAT &B,const MAT &A);
//矩阵复制
void transpose(MAT &AT,const MAT &A);
//矩阵转置
void matmul(VEC &b,const MAT &A,const VEC &x);
// 矩阵乘以向量
void matmul(MAT &C,const MAT &A,const MAT &B);
//矩阵乘以矩阵

//-----矩阵分解与求逆
void LDL(const MAT &A,MAT &L,VEC &D);
// LDL 分解
void MGS(const MAT &A, MAT &Q, MAT &R);
//修正的Gram-Schmidt正交化方法
void householder(const MAT &A,MAT &Q,MAT &R);
//hoseholder正交变换
void invlowtri(const MAT &R,MAT &S);
//inverse of lower triangula matrix
void invuptri(const MAT &U,MAT &R);
//inverse of upper triangula matrix
void invmat(const MAT &A,MAT &invA);
//对称正定矩阵逆矩阵
void inv(const MAT &A,MAT &iA);
//一般矩阵的逆矩阵 采用MGS分解方法
```

# cmatlib求解线性代数方程

```
//-----线性代数方程
void LS_LDL(VEC &x,const MAT &A,const VEC &b);
//基于不开平方的cholesky分解计算最小二乘问题, A为对称正定矩阵
//void LS_hous(VEC &x,const MAT &A,const VEC &b);
//基于Householder变换求解最小二乘问题或适定问题
void LS_MGS( VEC &x,const MAT &A,const VEC &b);
//通过修正的Gram-Schmidt正交化求解最小二乘问题或适定问题
void LS_hous( VEC &x,const MAT &A,const VEC &b);
//least square solution or general linear equation
void uptri(const MAT &A,const VEC &b,VEC &x);
//上三角矩阵方程计算
void lowtri(const MAT &A, const VEC &b, VEC &x);
//下三角矩阵方程计算
void downtri(const MAT &A, const VEC &b, VEC &x);
//-----插值算法
void lagrange(const VEC &x,const VEC &y,const VEC &xx,VEC &yy);
//lagrange interp
void rot_mat(MAT &Rmat,double angle,char axID);
//rotation matrix
double robustweight(double v,double sigma);
```

下载地址: <http://202.127.29.4/astrodynamics/course.php>

