# Codeit矩阵运算库函数使用说明

矩阵运算函数的命名空间为codeit::model。定义了一种Size数据类型：

**typedef std::size\_t Size**

* **一维矩阵（向量，数组）**

(1)一维向量的定义是使用方式与C++一样。Codeit多提供了一种通过at函数访问一维数组下标的方式。

例如，

double x[4]={1, 2, 3, 4};

double n =x[at(1, 2)];

* **二维矩阵（数组）**

(1)二维矩阵的定义是使用方式与C++一样。Codeit一般需要用户将二维数组转成一维数组进行使用。Codeit提供了一种通过at函数访问二维数组下标的方式。

例如，

int xx [3][2] = {{8,5}, {7,9}, {6,3}};

int x [6]; //6=3\*2;

for (int i=0; i<3; i++)

for (int j=0; j<2; j++)

x[i\*2+j] = xx[i][j];

int n=x[at(1,1,2)];//访问xx[1][1];

* **Size at(Size i, Size ld)**

(1)返回i\*ld。

例如，

Size i=1, ld=3;

Size n =at(i, ld);

* **Size at(Size i, Size j, Size ld)**

(1)返回i\*ld+j。

例如，

Size i=1, j=2, ld=3;

Size n =at(i, j, ld);

* **T s\_sgn(T val, T zero\_check)**

(1)判断元素val的符号，公差为zero\_check。

例如，

double val;

double zero\_check=1.0e-3;

double sgn =s\_sgn(val, error\_check);

* **bool s\_is\_equal(Size n, const double \*v1, const double \*v2, double error)**

(1)判断一维数组v1和v2中的n个元素是否相等，公差为error。

例如，

double v1[10], v2[10];

double error=1.0e-3;

Size n=5;

bool is\_equal=s\_is\_equal(n, v1, v2, error);

* **void s\_rmx(double angle, double \*A)**

(1)生成绕x轴旋转angle(单位为rad)的旋转矩阵A(3\*3)。

例如，

double A[9];

double angle=1.57;

s\_rmx(angle, A);

* **void s\_rmy(double angle, double \*A)**

(1)生成绕y轴旋转angle(单位为rad)的旋转矩阵A(3\*3)。

例如，

double A[9];

double angle=1.57;

s\_rmy(angle, A);

* **void s\_rmz(double angle, double \*A)**

(1)生成绕z轴旋转angle(单位为rad)的旋转矩阵A(3\*3)。

例如，

double A[9];

double angle=1.57;

s\_rmz(angle, A);

* **bool s\_is\_equal(Size n, const double \*v1, V1Type v1\_t, const double \*v2, V2Type v2\_t, double error)**

(1)判断一维数组v1和v2中的n个元素是否相等，公差为error，其中v1中元素步长为v1\_t，v2中元素步长为v2\_t。当v1\_t=1且v2\_t=1时，该函数与上一个函数等效。

例如，

double v1[10], v2[10];

double error=1.0e-3;

Size n=3;

int v1\_t=2, v2\_t=4;

bool is\_equal=s\_is\_equal(n, v1, v1\_t, v2, v2\_t, error);

* **bool s\_is\_equal(Size m, Size n, const double \*m1, const double \*m2, double error)**

(1)判断二维数组v1(m\*n)和v2 (m\*n)中是否相等，公差为error。其中二维数组需转成一维数组的形式。

例如，

double v1[10], v2[10];

double error=1.0e-3;

Size m=2, n=5;

bool is\_equal=s\_is\_equal(m, n, v1, v2, error);

* **double s\_norm(Size n, double \*x,)**

(1)计算维数为n的一维向量x的模。

例如，

double x[9];

Size m=9;

double x\_norm=s\_norm (n, x);

* **void s\_eye(Size m, double \*A,)**

(1)生成维数为m的单位矩阵A，其中二维矩阵A会转换成一维矩阵A。

例如，

double A[9];

Size m=3;

s\_eye (m, A); //A的结果为{1,0,0,0,1,0,0,0,1}

* **void s\_swap\_v(Size n, double \*x, double \*y)**

(1)交换维数为n的一维向量x和y。

例如，

double x[9], y[9];

Size m=9;

s\_swap\_v (n, x);

* **void s\_fill(Size m, Size n, double value, double \*A)**

(1)初始化维数为m\*n的二维矩阵A，所以元素均为value。

例如，

double A[9];

Size m=3, n=3;

double value=1;

s\_swap\_v (m, n, value, A);

* **void s\_iv(Size n, double \*x)**

(1)对矩阵x中连续的n个元素取反。

例如，

double x[9];

Size n=3;

s\_iv (n, x);

* **void s\_vi(Size n, const double \*x, double \*y)**

(1)对矩阵x中连续的n个元素乘上-1赋值给y中对应的元素。

例如，

double x[9], y[9];

Size n=3;

s\_vi (n, x, y);

* **void s\_nv(Size n, double alpha, double \*x)**

(1)对矩阵x中连续的n个元素变为其自身的alpha倍。

例如，

double x[9];

Size n=3;

double alpha=0.6;

s\_nv (n, alpha, x);

* **void s\_vc(Size n, double alpha, const double \*x, double \*y)**

(1)对矩阵x中连续的n个元素乘上alpha倍，赋给矩阵y中对应的元素。

例如，

double x[9], y[9];

Size n=3;

double alpha=0.6;

s\_vc (n, alpha, x, y);

* **void s\_va(Size n, double alpha, const double\* x, double\* y)**

(1) 矩阵y中连续的n个元素元素，加上矩阵x中对应元素的alpha倍。

例如，

double x[9], y[9];

Size n=3;

double alpha=0.6;

s\_va (n, alpha, x, y);

* **void s\_vs(Size n, const double\* x, double\* y)**

(1) 矩阵y中连续的n个元素元素，减去矩阵x中对应元素。

例如，

double x[9], y[9];

Size n=3;

s\_vs (n, x, y);

* **double s\_vv(Size n, const double\* x, const double\* y)**

(1) 矩阵y中连续的n个元素元素，分别乘上矩阵x中对应元素，求和。

例如，

double x[9], y[9];

Size n=3;

double sum=s\_vv (n, x, y);

* **void s\_mm(Size m, Size n, Size k, const double\* A, const double\* B, double\* C)**

(1) 计算矩阵A(m\*k)与矩阵B(k\*n)的乘积，并赋值为矩阵C(m\*n)。

例如，

double A[12], B[12], C[9];

Size m=3, k=4, n=3;

s\_mm (m, n, k, A, B, C);

* **void s\_inv\_pm(const double \*pm\_in, double \*pm\_out)**

(1) 计算4\*4的旋转矩阵pm\_in的逆矩阵pm\_out。

例如，

double pm\_in[16], pm\_out[16];

s\_inv\_pm (pm\_in, pm\_out);

* **void s\_pm\_dot\_pm(const double \*pm1, const double \*pm2, double \*pm\_out)**

(1) 计算两个4\*4的旋转矩阵pm1与pm2的乘积，结果保存在pm\_out中。

例如，

double pm1[16], pm2[16], pm\_out[16];

s\_pm\_dot\_pm (pm1, pm2, pm\_out);

* **void s\_inv\_pm\_dot\_pm(const double \*pm1, const double \*pm2, double \*pm\_out)**

(1) 旋转矩阵pm1的逆乘上pm2，结果保存在pm\_out中。

例如，

double pm1[16], pm2[16], pm\_out[16];

s\_inv\_pm\_dot\_pm (pm1, pm2, pm\_out);

* **void s\_ pm\_dot\_inv\_pm(const double \*pm1, const double \*pm2, double \*pm\_out)**

(1) 旋转矩阵pm1乘上pm2的逆，结果保存在pm\_out中。

例如，

double pm1[16], pm2[16], pm\_out[16];

s\_ pm\_dot\_inv\_pm (pm1, pm2, pm\_out);

* **void s\_ pm\_dot\_v3(const double \*pm, const double \*v3, double \*v3\_out)**

(1) 旋转矩阵pm乘上三维向量v3，结果保存在v3\_out中。

例如，

double pm [16], v3[3], v3\_out[16];

s\_ pm\_dot\_v3 (pm, v3, v3\_out);

* **void s\_ inv\_pm\_dot\_v3(const double \*pm, const double \*v3, double \*v3\_out)**

(1) 旋转矩阵pm的逆乘上三维向量v3，结果保存在v3\_out中。

例如，

double pm [16], v3[3], v3\_out[16];

s\_ inv\_pm\_dot\_v3 (pm, v3, v3\_out);

* **double s\_ sinx\_over\_x(double x)**

(1) 计算sin(x)/x。

例如，

double x=0.2;

double x1=s\_ sinx\_over\_x (x);

* **double s\_ one\_minus\_cosx\_over\_square\_x(double x)**

(1) 计算(1 - cos(x))/(x^2)。

例如，

double x=0.2;

double x1=s\_ one\_minus\_cosx\_over\_square\_x (x);

* **double s\_ cm3(const double \*a, double \*cm\_out)**

(1) 将三维向量a转换为3\*3的旋量矩阵。

cm\_out =[ 0 -z y

z 0 -x

-y x 0 ]

例如，

double a[3], cm\_out[9];

s\_ cm3 (a, cm\_out);

* **void s\_c3(const double \*a, const double \*b, double \*c\_out)**

(1) 计算三维向量a与b的叉乘，结果保存在c\_out。c\_out=a×b

例如，

double a[3], b[3], c\_out[3];

s\_ c3 (a, b, c\_out);

* **void s\_c3(double alpha, const double \*a, const double \*b, double \*c\_out)**

(1) 计算三维向量a与b的叉乘，再乘上alpha，结果保存在c\_out。c\_out=alpha\*a×b

例如，

double a[3], b[3], c\_out[3];

double alpha=2;

s\_ c3 (alpha, a, b, c\_out);

* **void s\_c3i(const double \*a, const double \*b, double \*c\_out)**

(1) 计算三维向量a与b的叉乘，再乘上-1，结果保存在c\_out。c\_out=-a×b

例如，

double a[3], b[3], c\_out[3];

s\_ c3i (a, b, c\_out);

* **void s\_c3a(const double \*a, const double \*b, double \*c\_out)**

(1) 计算三维向量a与b的叉乘，加到c\_out。c\_out=a×b+c\_out。

例如，

double a[3], b[3], c\_out[3];

s\_ c3a (a, b, c\_out);

* **void s\_c3a(double alpha, const double \*a, const double \*b, double \*c\_out)**

(1) 计算三维向量a与b的叉乘，乘上alpha，加到c\_out。c\_out=alpha\*a×b+c\_out。

例如，

double a[3], b[3], c\_out[3];

double alpha=1.0;

s\_ c3a (alpha, a, b, c\_out);

* **void s\_c3s(const double \*a, const double \*b, double \*c\_out)**

(1) 计算三维向量a与b的叉乘，减到c\_out。c\_out=a×b-c\_out。

例如，

double a[3], b[3], c\_out[3];

s\_ c3s (a, b, c\_out);

* **double\* s\_ra2rm(const double \*ra\_in, double \*rm\_out, Size rm\_ld = 3)**

(1)将三维轴角向量ra\_in转变为3\*3的旋转矩阵rm\_out，并返回rm\_out，rm\_ld的缺省值为3。

例如，

double ra\_in[3], rm\_out[9];

s\_ ra2rm (ra\_in, rm\_out);

* **double\* s\_rm2ra(const double \*rm\_in, double \*ra\_out, Size rm\_ld = 3)**

(1)将3\*3的旋转矩阵rm\_in转变为三维轴角向量ra\_out，并返回ra\_out，rm\_ld的缺省值为3。

例如，

double rm\_in[9], ra\_out[3];

s\_ rm2ra (rm\_in, ra\_out);

* **double\* s\_re2rm(const double \*re\_in, double \*rm\_out, const char \*eu\_type\_in , Size rm\_ld = 3)**

(1)将三维欧拉角向量re\_in转变为3\*3的旋转矩阵rm\_out转变为，并返回rm\_out，欧拉角类型eu\_type\_in类似“313”（ZXZ），rm\_ld的缺省值为3。

例如，

double re\_in[3], rm\_out[9];

s\_ re2rm (re\_in, ra\_out, “313”);

* **double\* s\_rm2re(const double \*rm\_in, double \*re\_out, const char \*eu\_type\_in, Size rm\_ld = 3)**

(1)将3\*3的旋转矩阵rm\_in转变为三维欧拉角向量ra\_out，并返回ra\_out，欧拉角类型eu\_type\_in类似“313”（ZXZ），rm\_ld的缺省值为3。

例如，

double rm\_in[9], re\_out[3];

s\_ rm2re (rm\_in, re\_out, “313”);

* **double\* s\_rq2rm(const double \*rq\_in, double \*rm\_out, Size rm\_ld = 3)**

(1)将四维四元数向量rq\_in转变为3\*3的旋转矩阵rm\_out，并返回rm\_out，rm\_ld的缺省值为3。

例如，

double rq\_in[4], rm\_out[9];

s\_ rq2rm (rq\_in, rm\_out);

* **double\* s\_rm2rq(const double \*rm\_in, double \*rq\_out, Size rm\_ld = 3)**

(1)将3\*3的旋转矩阵rm\_in转变为四维四元数向量rq\_out，并返回rq\_out，rm\_ld的缺省值为3。

例如，

double rm\_in[9], rq\_out[4];

s\_ rm2rq (rm\_in, rq\_out);

* **double\* s\_pp2pm(const double \*pp\_in, double \*pm\_out)**

(1) 将三维向量pp\_out中的位置信息赋值给4\*4的旋转矩阵pm\_out中的位置,并返回pm\_out，。

例如，

double pp\_in[3], pm\_out[16];

s\_ pp2pm (pp\_in, pm\_out);

* **double\* s\_pm2pp(const double \*pm\_in, double \*pp\_out)**

(1)从4\*4的旋转矩阵中提取三维位置信息存放在pp\_out中，并返回pp\_out。

例如，

double pm\_in[16], pp\_out[3];

s\_ pm2pp (pm\_in, pp\_out);

* **double\* s\_ra2pm(const double \*ra\_in, double \*pm\_out)**

(1) 将三维轴角向量ra\_in中的姿态信息赋值给4\*4的旋转矩阵pm\_out中的位置,并返回pm\_out，。

例如，

double ra\_in[3], pm\_out[16];

s\_ ra2pm (ra\_in, pm\_out);

* **double\* s\_pm2ra(const double \*pm\_in, double \*ra\_out)**

(1)从4\*4的旋转矩阵中提取三维轴角信息存放在ra\_out中，并返回ra\_out。

例如，

double pm\_in[16], ra\_out[3];

s\_ pm2ra (pm\_in, ra\_out);

* **double\* s\_re2pm(const double \*re\_in, double \*pm\_out**,**const char \*eu\_type\_in)**

(1) 将三维欧拉角向量re\_in中的姿态信息赋值给4\*4的旋转矩阵pm\_out中的位置,并返回pm\_out，欧拉角类型eu\_type\_in类似“313”（ZXZ）。

例如，

double re \_in[3], pm\_out[16];

s\_ re2pm (ra\_in, pm\_out, “313”);

* **double\* s\_pm2re(const double \*pm\_in, double \*re\_out**,**const char \*eu\_type\_in)**

(1)从4\*4的旋转矩阵中提取三维欧拉角信息存放在re\_out中，并返回re\_out，欧拉角类型eu\_type\_in类似“313”（ZXZ）。

例如，

double pm\_in[16], ra\_out[3];

s\_ pm2ra (pm\_in, ra\_out, “313”);

* **double\* s\_rq2pm(const double \*rq\_in, double \*pm\_out)**

(1) 将四维四元数向量rq\_in中的姿态信息赋值给4\*4的旋转矩阵pm\_out中的位置,并返回pm\_out。

例如，

double rq\_in[4], pm\_out[16];

s\_ rq2pm (rq\_in, pm\_out);

* **double\* s\_pm2rq(const double \*pm\_in, double \*rq\_out)**

(1)从4\*4的旋转矩阵中提取四维四元数信息存放在rq\_out中，并返回rq\_out。

例如，

double pm\_in[16], rq\_out[4];

s\_ pm2rq (pm\_in, rq\_out);

* **double\* s\_rm2pm(const double \*rm\_in, double \*pm\_out)**

(1) 将3\*3旋转矩阵rm\_in赋值给4\*4的旋转矩阵pm\_out中的位置,并返回pm\_out。

例如，

double rm\_in[9], pm\_out[16];

s\_ rm2pm (rm\_in, pm\_out);

* **double\* s\_pm2rm(const double \*pm\_in, double \*rm\_out)**

(1)从4\*4的旋转矩阵中提取3\*3的信息存放在rm\_out中，并返回rm\_out。

例如，

double pm\_in[16], rm\_out[4];

s\_ pm2rm (pm\_in, rm\_out);

* **double\* s\_pa2pm(const double \*pa\_in, double \*pm\_out)**

(1) 将六维位置与轴角向量pa\_in转变为4\*4的旋转矩阵pm\_out中的位置,并返回pm\_out。

例如，

double pa \_in[6], pm\_out[16];

s\_ pa2pm (pa\_in, pm\_out);

* **double\* s\_pm2pa(const double \*pm\_in, double \*pa\_out)**

(1)将4\*4的旋转矩阵转变为六维位置加轴角信息存放在pa\_out中，并返回pa\_out。

例如，

double pm\_in[16], pa\_out[6];

s\_ pm2pa (pm\_in, pa\_out);

* **double\* s\_pe2pm(const double \*pe\_in, double \*pm\_out, const char \*eu\_type\_in)**

(1) 将六维位置与欧拉角向量pe\_in转变为4\*4的旋转矩阵pm\_out中的位置,并返回pm\_out, 欧拉角类型eu\_type\_in类似“313”（ZXZ）。

例如，

double pe \_in[6], pm\_out[16];

s\_ pe2pm (pe\_in, pm\_out, “313”);

* **double\* s\_pm2pe(const double \*pm\_in, double \*pe\_out, const char \*eu\_type\_in)**

(1)将4\*4的旋转矩阵转变为六维位置加欧拉角信息存放在pe\_out中，并返回pe\_out, 欧拉角类型eu\_type\_in类似“313”（ZXZ）。

例如，

double pm\_in[16], pe\_out[6];

s\_ pm2pe (pm\_in, pe\_out, “313”);

* **double\* s\_pq2pm(const double \*pq\_in, double \*pm\_out)**

(1) 将七维位置与四元数向量pq\_in转变为4\*4的旋转矩阵pm\_out中的位置,并返回pm\_out。

例如，

double pq \_in[7], pm\_out[16];

s\_ pq2pm (pq\_in, pm\_out);

* **double\* s\_pm2pq(const double \*pm\_in, double \*pq\_out)**

(1)将4\*4的旋转矩阵转变为七维位置加四元数信息存放在pq\_out中，并返回pq\_out。

例如，

double pm\_in[16], pq\_out[7];

s\_ pm2pq (pm\_in, pq\_out);