Álgebra computacional en C con ALGLIN

Igor Fernández de Bustos

Grupo de Investigación ADM

Dpt. Ingeniería Mecánica

Universidad del País Vasco

ADM Diseño Mecánico grupo ehu

eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea

Ingeniaritza Mekanikoa Saila
Departamento de Ingeniería Mecánica

Sumario

1 Álgebra computacional en C con ALGLIN

Este texto tiene dos objetivos fundamentales. El primero es dar unas guías básicas de cómo se trabaja en lenguajes compilados con matrices densas y la razón de esta forma de funcionar. El segundo es explicar como las funciones básicas están implementadas en libalglin.

1.1 Un breve resumen de teoría de punteros.

Un puntero representa una dirección de memoria. La zona de código de programa, la pila y el montón están en la memoria, que se puede representar como una enorme cantidad de celdas a las cuales está asociada una dirección. Antiguamente, esta dirección correspondía exactamente a la posición de memoria física correspondiente en el computador, pero en el momento en el que surgen los sistemas operativos multitarea, resulta necesario establecer un intermediario que asocie a estas zonas de memoria reales una virtual. El programa trabaja con las direcciones virtuales y el sistema operativo se encarga de mantener una correspondencia entre estas direcciones virtuales y las reales. Así, un mismo programa ejecutado dos veces, puede estar en diferentes zonas de memoria, pero para el los números que identifican las celdas de memoria que utiliza son los mismos.

Tanto el código, como la pila, como el montón están en estas celdas de memoria, y por lo tanto mediante un puntero nos podemos referir tanto a una zona de un programa, como puede ser el principio de una función, como a una variable en la pila, o una en el montón. Esto implica que se pueden crear punteros a variable y a funciones. Mediante punteros se puede leer o modificar una variable, saltar a una función determinada, y otra serie de cosas.

Un array es, en realidad, un puntero a una zona de memoria reservada por el programador para una determinada tarea, que puede ser almacenar un vector o una matriz. Existen dos formas de reservar esta memoria. Una es mediante la declaración de un array de forma directa:

```
double a[40]; /*reserva en la pila espacio para 40 doubles de forma consecutiva. a es un puntero al primer elemento de ese espacio*/
```

Esto reserva un array de 40 elementos en coma flotante de doble precisión. La reserva se hace en la pila, por lo que es liberada automáticamente en cuanto se sale de la función en la que se declara. Esto es muy cómodo para matrices temporales, pero tienen que ser de tamaño pequeño porque la pila tiene un tamaño limitado.

La otra alternativa es la gestión dinámica de memoria. En este caso es importante recordar que toda memoria que se reserva debe ser liberada. Las funciones que se emplean para reservar memoria de forma dinámica son:

```
void *malloc(size_t t); /*reserva t celdas de memoria*/
void *free(void *p); /*libera la memoria apuntada por p*/
void *realloc(void *p, size_t t);
```

La primera función reserva un espacio en memoria de tamaño t este tamaño está especificado en función del tamaño de palabra del sistema, con lo que, por ejemplo, para reservar espacio para un array de 50 doubles y luego liberarlos (para esto se emplea free) tenemos que hacer:

```
double *a;/*declaramos un puntero a coma flotante de doble precisión*/
a=(double*)malloc(sizeof(double)*50); /*reservamos espacio para un array de 50 comas flotantes de doble precisión y hacemos que a apunte a ese espacio*/
free(a); /*liberamos el espacio*/
```

Es importante tener en cuenta que puede pasar que no exista espacio para reservar. En este caso malloc devuelve NULL. NULL es un puntero a nada. Quiere decir que el puntero no apunta a nada. Es importante comprobar que cuando se reserva memoria efectivamente el sistema ha podido reservarla y, si no se puede, manejar el problema.

Dos punteros pueden apuntar al mismo espacio de memoria, pero, obviamente, no se puede liberar la memoria más de una vez:

```
double *a,*b;
a=(double*)malloc(sizeof(double)*50);
b=a; /*tanto a como b apuntan al mismo espacio*/
free(a);
free(b); /*esto dará error, estamos liberando un espacio que ya estaba libre*/
```

Es importante no perder el puntero a una zona reservada porque luego habrá que liberarla. Por ejemplo:

```
double *a;
a=(double*)malloc(sizeof(double)*50);
a=(double*)malloc(sizeof(double)*50); /*reservamos otro espacio*/
free (a); /*liberamos el segundo espacio, pero el primero ya no lo podemos liberar*/
```

Esto generaría una pérdida de memoria (memory leak) aunque esto no suele provocar un fallo inmediato del programa, puede hacer que el programa crezca en memoria de forma indefinida hasta provocar el fallo del sistema.

La función realloc lo que hace es redimensionar un array, pero sirve también para reservar y liberar memoria. Por ejemplo:

```
double *a;
a=(double*) realloc(NULL, sizeof(double)*30); /*Si el primer puntero es null, es como si hicieramos malloc*/
a=(double *)realloc(a, sizeof(double)*50);/*redimensionamos el array para que entren 50 elementos*/
a=(double *)realloc(a,0); /*si llamamos a realloc con un tamaño 0, es como liberar la memoria y devuelve NULL*/
```

Si realloc en una reserva de memoria no nula devuelve NULL, es que no había memoria para la reserva. En este caso el puntero no se libera, así que deberíamos comprobar esto y manejar el error en este caso.

Se puede trabajar con un puntero como con un array y viceversa. En realidad no hay ninguna diferencia. Es decir, podemos hacer:

```
double *a;
a=(double*) realloc(NULL, sizeof(double)*30);
a[4]=5.0;
```

```
a[21]=3.0;
free(a);
```

Se puede crear un puntero a un elemento dentro de una zona reservada. Para ello se emplea el operador &:

```
double *a, *b;
a=(double*) realloc(NULL, sizeof(double)*30);
b=&(a[10]); /*b apunta al elemento 10 del array a*/
b[0]=3.0; /*es como hacer a[10]=3.0*/
free(a);/*libramos todo el array a, con lo que b ya no sirve para nada*/
```

Ojo. Hacer free(b) en lugar de free(a) en el programa anterior daría error.

2 Almacenamiento de matrices densas en C.

En cualquier lenguaje de programación una matriz densa se suele almacenar como una secuencia (array) de elementos ordenados por filas (RMO: row major order) o por columnas (CMO: column major order). En el caso de C la numeración de los arrays empieza en cero y, por ello, es mejor plantear los elementos como variando desde el 0 hasta el N-1. En C es mas habitual el empleo de RMO, pero una gran cantidad de bibliotecas de C se basan en la biblioteca BLAS (basic linear algebra subprograms), que originalmente se programó en FORTRAN, en el que era más habitual trabajar en CMO. Esto último tiene la teórica ventaja de que muchas matrices están formadas por vectores columna, aunque en la práctica esto representa una ventaja leve o muy leve (siempre se puede trabajar con las traspuestas de las matrices). Por ejemplo, la matriz:

$$\boldsymbol{A}=\begin{bmatrix}1. & 2. & 3.\\ 4. & 5. & 6.\end{bmatrix}$$

En RMO se almacenaría como un array cuyos elementos serían, de forma sucesiva: {1.,2.,3.,4.,5.,6.}, mientras que en CMO se almacenaría: {1.,4.,2.,5.,3.,6.}. Es evidente que la representación de $\boldsymbol{A}$ en formato CMO es igual a la representación de $\boldsymbol{A}^{\boldsymbol{T}}$ en RMO. Esto permite emplear las funciones en CMO para matrices RMO y viceversa de forma sencilla. Como regla general, no se debería jamás trasponer una matriz. Es un proceso costoso y la mayoría de las veces innecesario.

Obviamente, además del array con los elementos de la matriz, necesitamos el número de filas y el número de columnas de $\boldsymbol{A}$. Con esto, acceder a un elemento cualquiera de $\boldsymbol{A}$ es fácil. Sean fA el número de filas de $\boldsymbol{A}$ y cA el número de columnas de $\boldsymbol{A}$. El elemento (i,j) de $\boldsymbol{A}$ se puede puede obtener en C mediante la expresión: $A[i*cA+j]$.

Es importante tener en cuenta que cuando se trabaja en un ordenador, muchas veces el coste de acceso a datos es más importante que el coste en realizar las operaciones, por lo que dos puntos son importantes:

- Procurar disminuir el tiempo de acceso (siempre que se pueda hacer un trabajo más elaborado sobre un dato, suele ser mejor que hacerlo en varias pasadas). Por ejemplo. Supongamos que se quere sumar a un vector ($\boldsymbol{v}$) otro vector multiplicado ($\boldsymbol{u}$) por un escalar a. Un algoritmo

posible sería primero calcular $a\boldsymbol{u}$ y luego sumárselo a $\boldsymbol{v}$, pero es mucho más rápido hacer, elemento a elemento, $\boldsymbol{v}+a\boldsymbol{u}$.

- El acceso secuencial es mucho más rápido que el aleatorio. Es decir, recorrer un array elemento a elemento es en general mucho más rápido que ir saltando de elemento en elemento. Esto es especialmente importante hoy en día con procesadores con instrucciones SIMD (casi todos los actuales).

Hoy en día el coste de acceso es tan alto en comparación con el de operación que lo habitual es que todas las funciones incluyan escalado, ya que se hace prácticamente sin coste. Es decir, para sumar a un vector u un vector v no se crea una función de la forma:

```
vectorsumarv(double *u, unsigned long cu, double *v)
```

Sino que se hace:

```
vectorsumarav(double *u, unsigned long cu, double a, double *v)
```

Siendo a un factor de escala. Si solo se quiere sumar el vector, se hace a=1 y resuelto. Esto disminuye el número de funciones lo que simplifica la programación. Una razón adicional para hacer esto es que hoy en día existen muchos procesadores con la instrucción FMA (fused multiply add) que hace una suma escalada en una única instrucción.

2.1 SIMD y operaciones en matrices

Un asunto que es especialmente importante en operaciones matriciales son las instrucciones SIMD éstas instrucciones son capaces de hacer la misma operación sobre un grupo de variables. Por ejemplo, multiplicar a la vez varios números en coma flotante por un número. Es extremadamente importante pensar cuando se programan operaciones sobre matrices. Consideremos por ejemplo un caso muy sencillo. Si multiplicamos la matriz:

$$\boldsymbol{A}=\begin{bmatrix} 2. & 3. \\ 4. & 5. \\ 1. & 5. \\ 8. & 9. \end{bmatrix}$$

mediante el algoritmo:

```
int i, j;
for (i=0;i<fA;i++)
        for (j=0;j<cA;j++)
                A[i*ldA+j]*=a;
```

Curiosamente, si el procesador tiene instrucciones SIMD capaces de trabajar con más de dos coma flotantes a la vez, salvo que el compilador sea extremadamente inteligente, tardará más que si aplicamos la misma función a la matriz B:

$$\boldsymbol{A}=\begin{bmatrix} 2. & 4. & 1. & 8. \\ 3. & 5. & 5. & 9. \end{bmatrix}$$

Ya que en este segundo caso las operaciones se harán en paralelo sobre más números en coma flotante. La conclusión es que cuando se diseña un algoritmo, es conveniente trabajar sobre matrices con más columnas que filas si están almacenadas RMO y con más filas que columnas si están almacenadas CMO. Esto lleva a que a veces sea más interesante trabajar con las traspuestas de algunas matrices. Esto es especialmente importante en el caso de matrices de pequeñas dimensiones.

2.2 Trabajo con submatrices

Otro concepto importante en el trabajo con matrices en C es la dimensión fundamental de la matriz (leading dimension). Cuando se define una función para álgebra matricial se procura que ésta sea lo más flexible posible manteniendo la eficiencia computacional. Supongamos una función que calcule un producto de matrices. El prototipo de esta función podría ser:

```
matrizaAB(double *Res, double a, double *A, unsigned long fA, unsigned long cA, double *B, unsigned long cB);
```

sin embargo, si añadimos el parámetro ldA y el parámetro ldB, podemos hacer que la función sea válida no solo para multiplicar dos matrices de dimensión variable, sino que la podríamos emplear para multiplicar submatrices.

```
matrizAB(double *Res, unsigned long ldRes, double a, double *A, unsigned long fA, unsigned long cA, unsigned long lda, double *B, unsigned long cB, unsigned long ldB);
```

Para indexar un elemento (i, j) de la submatriz A, basta con hacer:

$$A[i*lda+j]$$

Donde i y j son los índices del elemento en la submatriz. Es importante indicar que A debe apuntar no a la matriz que contiene a la submatriz, sino al primer elemento de la submatriz. Como ejemplo, sea la matriz:

$$\boldsymbol{M}=\begin{bmatrix} 1. & 2. & 3. & 4. \\ 5. & 6. & 7. & 8. \\ 9. & 10. & 11. & 12. \end{bmatrix}$$

Si queremos trabajar con la submatriz A correspondiente a las dos últimas filas y columnas bastaría con apuntar al elemento (1,2): A=&(M[1*cM+2). El número de filas de A sería de 2, el número de columnas sería 2 y el ld de A sería el número de columnas de M: 4.

2.3 Matrices simétricas

Almacenar y trabajar con matrices simétricas como si fueran densas es un error, ya que por un lado el número de elementos distintos que hay que almacenar son algo más de la mitad y, por el otro, el coste computacional también se puede reducir de forma considerable para muchas operaciones. Aunque recientemente se han planteado nuevas formas de almacenar las matrices simétricas, de largo la forma habitual es la más evidente: se almacenan de la misma forma que las generales pero omitiendo los elementos duplicados. Por ejemplo la matriz:

$$\boldsymbol{A}=\begin{bmatrix} 8. & 3. & -2. \\ 3. & 5. & -1. \\ -2. & -1. & -4. \end{bmatrix}$$

Se almacenaría (en RMO) como {8., 3., -2., 5., -1., -4.}. El acceso a los elementos es un poco más complicado, ya que un elemento en la fila i,j (con j>=i) se accedería haciendo: A(i,j)=i*t-(i*(i+1))/2+j, siendo t el tamaño de la matriz. También se puede trabajar con una submatriz simétrica haciendo t=lda, aunque en este caso la dimensión fundamental (leading dimension) varía según la fila.

En el caso de las matrices simétricas es aún más importante revisar las operaciones para que los elementos sean accedidos de forma secuencial, ya que si así lo hacemos nos ahorramos emplear la expresión de A(i,j) que es bastante compleja. Es importante indicar que en compiladores modernos se puede meter la expresión de A(i,j) entera al acceder a los elementos y el optimizador de código del compilador normalmente ya se encarga de evaluar una única vez la expresión si el acceso se hace de forma secuencial. Por ejemplo, si hacemos:

```
#define INDICEARRAYDEFILACOLUMNA(t,f,c) (f)*(t)-((f)*((f)+1))/2+(c)
```

y luego hacemos, por ejemplo:

```
for (j=0;j<100;j++)
        acc+=INDICEARRAYDEFILACOLUMNA(200,3,j);
```

El compilador se encargará de que la expresión (2*200-(2*(2+1)/2)) se calcule una única vez. Esto es importante tener en cuenta porque es mucho más claro emplear este tipo de expresiones que desarrollar directamente un código optimizado. Ojo porque en casos más complicados no es tan sencillo.

Un buen ejemplo de como plantear una operación para optimizar el acceso secuencial está en la función vectorasAv.

3 La biblioteca alglin

La biblioteca alglin es una biblioteca que proporciona funciones para trabajar con matrices. Es similar a BLAS, pero tiene la ventaja de que su nomenclatura es más sencilla para castellanoparlantes. Para matrices de gran tamaño puede tener una menor eficiencia que BLAS, pero su objetivo no es tratar con matrices muy grandes, sino servir para problemas pequeños y como base para sistemas de matrices dispersas.

3.1 Conceptos generales

Un aspecto muy importante es que la biblioteca no gestiona memoria, es decir: los espacios en memoria empleados deben estar reservados por el programador. Tanto los de entrada como los de salida. En algún caso puede haber alguna función que gestione internamente reserva de memoria y por lo tanto devuelve códigos de error en caso de fallo de memoria.

Las funciones tienen como nombre una combinación de letras y números que están organizados en función del tipo de parámetro que emplean. Por ejemplo, la función matrizsumaraAB significa que

es una función que el resultado es una matriz. “sumar” significa que el resultado se suma a la matriz que hubiera en la zona de memoria indicada. aAB significa que lo que se suma es el producto de un escalar (a) por una matriz (A) por otra matriz (B). Los parámetros llevan el siguiente orden. Los primeros parámetros son el sitio donde se almacena (o modifica) el resultado. Aquí la definición es la mínima necesaria. Por ejemplo, en el caso de matrizsumaraAB para la matriz resultado solo se indicaría su leading dimension, ya que lo demás viene determinado por las filas y columnas de A y B. Los siguientes parámetros son, en orden, cada uno de los términos de la operación. Así, matrizsumaraAB tendría el siguiente prototipo:

```
void matrizsumaraAB(double *Res, unsigned long ldRes, double a, double *A, unsigned long fA, unsigned long cA, unsigned long ldA, double *B, unsigned long cB, unsigned long ldB)
```

Nótese que no se incluye el número de filas de B dado que debe ser igual al de columnas de A.

En el nombre de las funciones los escalares se denotan por a,b,c. Los vectores por v,u,w. Las matrices van con mayúsculas: A,B,C. Las traspuestas se generan añadiendo una T mayúscula después del elemento correspondiente, por ejemplo:

```
void matrizsumaraATB(double *Res, unsigned long ldRes, double a, double *A, unsigned long fA, unsigned long cA, unsigned long ldA, double *B, unsigned long cB, unsigned long ldB)
```

Por defecto las funciones están planteadas en doble precisión, con lo que los parámetros son double. Los números de elementos son unsigned long. En un futuro se planea hacer versión en simple precisión, con lo que se añadirá una f a los parámetros. Por ejemplo:

```
void fmatrizsumarfafAfB(float *Res, unsigned long ldRes, float a, float *A, unsigned long fA, unsigned long cA, unsigned long ldA, float *B, unsigned long cB, unsigned long ldB)
```

3.2 Primer nivel: vectores

Se consideran funciones de primer nivel aquellas que trabajan como mucho con vectores y, al menos, con un vector. Tenemos:

```
void vectoratexto(FILE *archivo, double *v, unsigned long cv, char cs);
```

Escribe el vector a texto

archivo: archivo donde escribirla

v: puntero al vector

cv: numero de elementos del vector

cs: número de cifras significativas

```
void vectoratextooctave(FILE *archivo, double *v, unsigned long cv, char cs);
```

Escribe el vector a texto (formato OCTAVE)

archivo: archivo donde escribirla

v: puntero al vector

cv: numero de elementos del vector

cs: número de cifras significativas

*double vectorvTu(double *v, unsigned long fv, double *u);*

Devuelve el producto de v traspuesto por u. u y v no se modifican.

v: puntero al primer elemento de v

fv: número de elementos de v

v: puntero al primer elemento de u

*void vectorav(double *u, double a, double *v, unsigned long fv);*

Calcula u=av. u puede coincidir con v. En este caso, obviamente v se modifica.

u: puntero al vector resultado

a: escalar

v: puntero al vector origen

fv: numero de elementos de v

*void vectorsumarav(double *u, double a, double *v, unsigned long fv);*

Calcula u=u+av. u puede coincidir con v. En este caso, obviamente v se modifica.

u: puntero al vector resultado

a: escalar

v: puntero al vector origen

fv: numero de elementos de v a sumar

*void vectoravxw(double *u, double a, double *v, double *w);*

Devuelve el producto vectorial de v por w multiplicado por a

u: puntero al vector donde se quiere almacenar avxA

a: factor de escala

v: vector. Obviamente de 3 elementos

w: vector. Obviamente de 3 elementos

*void vectorsumaravxw(double *u, double a, double *v, double *w);*

Suma el producto vectorial de v por w multiplicado por a

u: puntero al vector al que se quiere sumar avxA

a: factor de escala

v: vector. Obviamente de 3 elementos

w: vector. Obviamente de 3 elementos

3.3 Segundo nivel: matrices

Se consideran matrices de segundo nivel aquellas que operan como mucho con matrices y, como mínimo, con una matriz. Sin embargo aquí no se suelen incluir factorizaciones o resoluciones de sistemas de ecuaciones. Las funciones más importantes son:

*void matrizatexto(FILE *archivo, double *A, unsigned long fA, unsigned long cA, unsigned long ldA, char cs);*

Escribe la matriz a texto

archivo: archivo donde escribirla

A: puntero a la matriz

fA: numero de elementos de la matriz

cA: numero de columnas de la matriz

ldA: dimensión fundamental de la matriz

cs: número de cifras significativa

*void matrizatextooctave(FILE *archivo, double *A, unsigned long fA, unsigned long cA, unsigned long ldA, char cs);*

Escribe la matriz (formato OCTAVE) a texto

archivo: archivo donde escribirla

A: puntero a la matriz

fA: numero de elementos de la matriz

cA: numero de columnas de la matriz

ldA: dimensión fundamental de la matriz

cs: número de cifras significativas

*void vectoraAv(double *u, double a, double*A, unsigned long fA, unsigned long cA, unsigned long ldA, double *v);*

Devuelve el vector u=aAv.

u: puntero al vector resultado

a: escalar

A: puntero a la matriz

fA: numero de filas de la matriz

cA: numero de columnas de la matriz

ldA: separación entre el primer elemento de cada fila. Para poder aplicarlo a submatrices

v: puntero al vector

*void vectorsumaraAv(double *u, double a, double*A, unsigned long fa, unsigned long ca, unsigned long lda, double *v);*

Suma aAv al vector u.

u: puntero al vector a modificar

A: puntero a la matriz

a: escalar

fA: numero de filas de la matriz

cA: numero de columnas de la matriz

ldA: separación entre el primer elemento de cada fila. Para poder aplicarlo a submatrices

v: puntero al vector

*void vectoraAtv(double *u, double a, double*A, unsigned long fA, unsigned long cA, unsigned long ldA, double *v);*

Devuelve el vector u=aAtv

u: puntero al vector resultado

a: escalar

A: puntero a la matriz

fA: numero de filas de la matriz

cA: numero de columnas de la matriz

ldA: separación entre el primer elemento de cada fila. Para poder aplicarlo a submatrices

v: puntero al vector

*void vectorsumaraATv(double *u, double a, double*A, unsigned long fA, unsigned long cA, unsigned long ldA, double *v);*

Suma al vectoru el valor de a por A traspuesta por v. No hay reserva de espacios ni nada por el estilo

u: puntero al vector resultado

a: escalar

A: puntero a la matriz

fA: numero de filas de la matriz

cA: numero de columnas de la matriz

ldA: separación entre el primer elemento de cada fila. Para poder aplicarlo a submatrices

v: puntero al vector

*void matrizaA(double *C, unsigned long ldC, double a, double *A, unsigned long fA, unsigned long cA, unsigned long ldA);*

Establece en la matriz C la A multiplicada por un escalar

C: puntero a la matriz donde se quiere sumar

ldC: leading dimension de C

a: escalar

A: puntero a la matriz que se quiere sumar

fA: numero de filas de A

cA: numero de columnas de A

ldA: separacion entre las filas de A

*void matrizsumaraA(double *C, unsigned long ldC, double a, double *A, unsigned long fA, unsigned long cA, unsigned long ldA)*

Suma a una matriz C la A multiplicada por un escalar

C: puntero a la matriz donde se quiere sumar

ldC: leading dimension de C

a: escalar

A: puntero a la matriz que se quiere sumar

fA: número de filas de A

cA: numero de columnas de A

ldA: separacion entre las filas de A

*void matrizaAB(double *C, unsigned long ldC, double a, double *A,unsigned long fA, unsigned long cA, unsigned long ldA, double *B, unsigned long cB, unsigned long ldB);*

Multiplica la matriz A por la B y la devuelve en C, C no puede ser ni A ni B

C: puntero a la matriz donde se quiere almacenar AB

ldC: numero de elementos desde el primer elemento de una fila al primer elemento de la siguiente de c

a: factor de escala

A: puntero a la matriz A

fA: numero de filas de a

cA: numero de columnas de a

ldA: numero de elementos desde el primer elemento de una fila al primer elemento de la siguiente de a

B: puntero a la matriz B

cB: numero de columnas de b

ldB: numero de elementos desde el primer elemento de una fila al primer elemento de la siguiente de b

*void matrizsumaraAB(double *C, unsigned long ldC, double a, double *A,unsigned long fA, unsigned long cA, unsigned long ldA, double *B, unsigned long cB, unsigned long ldB);*

Multiplica la matriz A por la B y la suma a C, C no puede ser ni A ni B

C: puntero a la matriz donde se quiere sumar aAB

ldC: numero de elementos desde el primer elemento de una fila al primer elemento de la siguiente de c

a: factor de escala

A: puntero a la matriz A

fA: numero de filas de a

cA: numero de columnas de a

ldA: numero de elementos desde el primer elemento de una fila al primer elemento de la siguiente de a

B: puntero a la matriz B

cB: numero de columnas de b

ldB: numero de elementos desde el primer elemento de una fila al primer elemento de la siguiente de b

*void matrizaABT(double *C, unsigned long ldC, double a, double *A,unsigned long fA, unsigned long cA, unsigned long ldA, double *B, unsigned long fB, unsigned long ldB);*

Calcula la matriz A por la traspuesta de B multiplicada por a, C no puede ser ni A ni B

C: puntero a la matriz donde se quiere almacenar aABT

ldC: numero de elementos desde el primer elemento de una fila al primer elemento de la siguiente de c

a: escalar por el que se multiplica ABT antes de sumar a C

A: puntero a la matriz A

fA: numero de filas de a

cA: numero de columnas de a

ldA: numero de elementos desde el primer elemento de una fila al primer elemento de la siguiente de a

B: puntero a la matriz B

fB: numero de filas de b

ldB: numero de elementos desde el primer elemento de una fila al primer elemento de la siguiente de b

*void matrizsumaraABT(double *C, unsigned long ldc, double a, double *A,unsigned long fa, unsigned long ca, unsigned long lda, double *B, unsigned long fb, unsigned long ldb);*

Multiplica la matriz A por la traspuesta de B y la suma a C multiplicada por a, C no puede ser ni A ni B

C: puntero a la matriz donde se quiere almacenar AB

ldC: numero de elementos desde el primer elemento de una fila al primer elemento de la siguiente de c

a: escalar por el que se multiplica ABT antes de sumar a C

A: puntero a la matriz A

fA: numero de filas de a

cA: numero de columnas de a

ldA: numero de elementos desde el primer elemento de una fila al primer elemento de la siguiente de a

B: puntero a la matriz B

fB: numero de filas de B

ldB: leading dimension de B

*void matrizaATB(double *C, unsigned long ldC, double a, double *A,unsigned long fA, unsigned long cA, unsigned long ldA, double *B, unsigned long cB, unsigned long ldB);*

Calcula la matriz A traspuesta por B multiplicada por a, C no puede ser ni A ni B

C: puntero a la matriz donde se quiere almacenar aABT

ldC numero de elementos desde el primer elemento de una fila al primer elemento de la siguiente de C

a: escalar por el que se multiplica ABT antes de sumar a C

A: puntero a la matriz A

fA: numero de filas de a

cA: numero de columnas de a

ldA: numero de elementos desde el primer elemento de una fila al primer elemento de la siguiente de a

B: puntero a la matriz B

cB: numero de filas de b

ldB: numero de elementos desde el primer elemento de una fila al primer elemento de la siguiente de b

*void matrizsumaraATB(double *C, unsigned long ldC, double a, double *A,unsigned long fA, unsigned long cA, unsigned long ldA, double *B, unsigned long cB, unsigned long ldB);*

Calcula la matriz A traspuesta por B multiplicada por a, C no puede ser ni A ni B

C: puntero a la matriz donde se quiere almacenar aABT

ldC: numero de elementos desde el primer elemento de una fila al primer elemento de la siguiente de C

a: escalar por el que se multiplica ABT antes de sumar a C

A: puntero a la matriz A

fA: numero de filas de a

cA: numero de columnas de a

ldA: numero de elementos desde el primer elemento de una fila al primer elemento de la siguiente de a

B: puntero a la matriz B

cB: numero de filas de b

ldB: numero de elementos desde el primer elemento de una fila al primer elemento de la siguiente de b

*void matrizavxA(double *B, unsigned long ldB, double a, double *v, double *A, unsigned long ldA,double cA);*

Devuelve el producto vectorial de v por A multiplicado por a

B: puntero a la matriz donde se quiere almacenar avxA

ldB: leading dimension de B

a: factor de escala

v: vector. Obviamente de 3 elementos

A: puntero a la matriz

ldA: leading dimension de A

cA: numero de columnas de A (obviamente 3 filas)

*void matrizsumaravxI(double *A, unsigned long ldA, double a, double *v);*

Suma a la matriz A el producto vectorial de v por I y multiplicado por a, es equivalente a generar la matriz antisimétrica equivalente al producto vectorial y sumarla a la matriz regular multiplicada por un escalar

A: puntero a la matriz donde se quiere sumar avxI

ldA: leading dimension de A

a: factor de escala

v: vector, obviamente de 3 elementos

*void matrizsumaravxuxI(double *A, unsigned long ldA, double a, double *v, double *u);*

Suma a la matriz A el producto vectorial de v por u multiplicado vectorialmente por u I y multiplicado por a, es equivalente a generar la matriz equivalente a un producto vectorial doble

A: puntero a la matriz donde se quiere sumar avxuxI

ldA: leading dimension de A

a: factor de escala

v: vector, obviamente de 3 elementos

u: vector, obviamente de 3 elementos

3.4 Matrices simétricas

Para indicar que el resultado es una matriz simétrica, el nombre de la función empezará con simmatriz. Las matrices simétricas se indicarán con una s delante del nombre, (sA, sB,...).

*void simmatrizatexto(FILE *archivo, double *sA, unsigned long esA, unsigned long ldsA, char cs);*

Escribe la matriz a texto

archivo: archivo donde escribirla

sA: puntero a la matriz

esA: numero de elementos de la matriz

ldsA: dimensión fundamental de la matriz

cs: número de cifras significativas

*void simmatrizatextooctave(FILE *archivo, double *sA, unsigned long esA, unsigned long ldsA, char cs);*

Escribe la matriz (formato OCTAVE) a texto

archivo: archivo donde escribirla

sA: puntero a la matriz

esA: numero de elementos de la matriz

ldsA: dimensión fundamental de la matriz

cs: número de cifras significativas

*void vectorasAv(double *u, double a, double *sA, unsigned long esA, unsigned long ldsA, double *v);*

Multiplica el escalar a por la matriz simétrica A y por el vector v

u: puntero al vector u (resultado)

a: factor de escala

sA: puntero a la matriz simétrica

esA: número de elementos de la matriz simétrica

ldsA: dimensión fundamental de la matriz simétrica

v: puntero al vector v

*void vectorsumarasAv(double *u, double a, double *sA, unsigned long esA, unsigned long ldsA, double *v);*

Multiplica el escalar a por la matriz simétrica A y por el vector v y lo suma a u

u: puntero al vector u (resultado)

a: factor de escala

sA: puntero a la matriz simétrica

esA: número de elementos de la matriz simétrica

ldsA: dimensión fundamental de la matriz simétrica

v: puntero al vector v

*void simmatrizaATA(double *sC, unsigned long ldsC, double a, double *A, unsigned long fA, unsigned long cA, unsigned long ldA);*

Calcula el producto del escalar a por la matriz A traspuesta por la matriz A, que obviamente lleva a una matriz simetrica, C

sC: puntero a la matriz simetrica resultado

ldsC: dimension fundamental de C

a: escalar

fA: numero de filas de A

cA: numero de columnas de A

ldA: dimension fundamental de A

*void simmatrizaAAT(double *sC, unsigned long ldsC, double a, double *A, unsigned long fA, unsigned long cA, unsigned long ldA);*

Calcula el producto del escalar a por la matriz A por la matriz A traspuesta, que obviamente lleva a una matriz simetrica, C

sC: puntero a la matriz simetrica resultado

ldsC: dimension fundamental de C

a: escalar

fA: numero de filas de A

cA: numero de columnas de A

ldA: dimension fundamental de A

*void simmatrizsumaraATA(double *sC, unsigned long ldsC, double a, double *A, unsigned long fA, unsigned long cA, unsigned long ldA);*

Calcula el producto del escalar a por la matriz A traspuesta por la matriz A, y lo suma a la matriz simetrica, C

sC: puntero a la matriz simetrica resultado

ldsC: dimension fundamental de C

a: escalar

fA: numero de filas de A

cA: numero de columnas de A

ldA: dimension fundamental de A

*void simmatrizsumaraAAT(double *sC, unsigned long ldsC, double a, double *A, unsigned long fA, unsigned long cA, unsigned long ldA);*

Calcula el producto del escalar a por la matriz A por la matriz A traspuesta, y lo suma a la matriz simetrica, C

sC: puntero a la matriz simetrica resultado

ldsC: dimension fundamental de C

a: escalar

fA: numero de filas de A

cA: numero de columnas de A

ldA: dimension fundamental de A

3.5 Permutaciones

Habitualmente se consideran operaciones de segundo nivel, ya que implican una matriz de permutaciones. El problema de plantearlo así es que una matriz de permutaciones es una matriz dispersa con un comportamiento muy particular, por lo que en libmatrices se ha creado una expresión de matriz de permutación representada mediante un vector de permutaciones sucesivas. El vector representa una permutación de la fila 0 con una posterior, después de la 1 con una posterior, y así sucesivamente. Se facilitan también las operaciones con la matriz traspuesta para deshacer las operaciones, así como postmultiplicación para que afecte a columnas en lugar de a filas.

*void vectorPsv(unsigned long int *Ps, unsigned long ePs, double *v);*

Premultiplica el vector v por la matriz de permutaciones sucesivas Ps

Ps: puntero a la matriz de permutaciones sucesivas

ePs: numero de cambios en la matriz de permutación (obviamente es cuadrada)

v: vector a multiplicar

*void vectorPsTv(unsigned long int *Ps, unsigned long ePs, double *v);*

Premultiplica el vector v por la matriz de permutaciones sucesivas Ps traspuesta

Ps: puntero a la matriz de permutaciones sucesivas

ePs: numero de elementos de la matriz de permutación (obviamente es cuadrada)

v: vector a multiplicar

*void matrizPsA(unsigned long int *Ps, unsigned long int ePs, double *A, unsigned long int cA, unsigned long ldA);*

Premultiplica la matriz A por la matriz de permutaciones sucesivas

Ps: puntero a la matriz de permutaciones sucesivas

ePs: tamaño de la matriz de permutaciones

A: puntero a la matriz a multiplicar

cA: numero de columnas a girar

ldA: leading dimension de la matriz A

*void matrizPsTA(unsigned long int *Ps, unsigned long int ePs, double *A, unsigned long int cA, unsigned long ldA);*

Premultiplica la matriz A por la matriz de permutaciones sucesivas traspuesta

Ps: puntero a la matriz de permutaciones sucesivas

ePs: tamaño de la matriz de permutaciones

A: puntero a la matriz a multiplicar

cA: numero de columnas a girar

ldA: leading dimension de la matriz A

*void matrizAPs(double *A, unsigned long int fA, unsigned long ldA, unsigned long int *Ps, unsigned long ePs);*

Postmultiplica la matriz A por la matriz de permutaciones sucesivas Ps

A: puntero a la matriz a multiplicar

fA: numero de filas de A

ldA: leading dimension de la matriz A

Ps: puntero a la matriz de permutaciones sucesivas

ePs: tamaño de la matriz de permutaciones

*void matrizAPsT(double *A, unsigned long int fA, unsigned long ldA, unsigned long int *Ps, unsigned long ePs);*

Postmultiplica la matriz A por la matriz de permutaciones sucesivas Ps traspuesta

A: puntero a la matriz a multiplicar

fA: numero de filas de A

ldA: leading dimension de la matriz A

Ps: puntero a la matriz de permutaciones sucesivas

ePs: tamaño de la matriz de permutaciones

3.6 Búsqueda de pivotes

Son funciones para obtener un elemento que valga como pivote.

*double pivotajecompleto(double *A,unsigned long fA, unsigned long cA, unsigned long ldA, unsigned long *f, unsigned long *c);*

Busca el pivote más grande a lo bestia. Si. Quizá el nombre no es muy fino, pero es realista: es extremadamente lento. Es para pivotaje completo. Hay alternativas basadas en SIMD, y están en la librería, pero de momento son experimentales. Esta está bien para probar.

A: puntero a la matriz de datos

fA: número de filas de la (sub)matriz

cA: número de columnas de la (sub)matriz

ldA: leading dimension. en este caso número de columnas de la matriz original

f: parámetro de vuelta. fila donde está el máximo

c: parámetro de vuelta. columna donde está el máximo

devuelve: el valor absoluto del pivote

*double pivotajerook(double *A,unsigned long fA, unsigned long cA, unsigned long ldA, unsigned long *f, unsigned long *c, double tol);*

Busca el primer pivote dominante en fila y columna. Es lo que se conoce como pivotaje de Rook

A: puntero a la matriz de datos

fA: número de filas de la (sub)matriz

cA: número de columnas de la (sub)matriz

ldA: leading dimension. en este caso número de columnas de la matriz original

f: parámetro de vuelta. fila donde está el máximo

c: parámetro de vuelta. columna donde está el máximo

el: valor absoluto del pivote

*void permsuceaorden(unsigned long *Ps, unsigned long t, unsigned long *orden, unsigned long ePs);*

Pasa de un vector de permutaciones sucesivas a un vector de orden. Para ello aplica las permutaciones sucesivas sobre un vector que va desde 0 hasta N

Ps: puntero al vector de permutaciones sucesivas

t: dimensión de lo permutado

orden: puntero a un espacio de memoria que debe contener el orden de los elementos permutados. Debe tener el espacio reservado para ePs elementos y lo que tenía se pierde

ePs: numero de cambios en Ps

3.7 Factorización LDU

Estaría en lo que se conoce como funciones de nivel 3. Estas funciones están basadas en la factorización LDU presentada en []. La idea es que la factorización se realiza en la propia matriz (no se emplea espacio adicional) y tanto el subespacio nulo como el nulo izquierdo se almacenan en la propia estructura si es que se desea calcularlos.

*void matrizlduL11atexto(FILE *archivo, double *A, unsigned long ldA, unsigned long rnA, char cs);*

Saca a texto la submatriz L11

archivo: puntero al archivo donde se quiere volcar

A: matriz. Debe estar factorizada

ldA: leading dimension de A

rnA: rango de la matriz A

cs: numero de cifras significativas a escribir

*void matrizlduL11atextooctave(FILE *archivo, double *A, unsigned long ldA, unsigned long rnA, char cs);*

Saca a texto (formato OCTAVE) la submatriz L11

archivo: puntero al archivo donde se quiere volcar

A: matriz. Debe estar factorizada

ldA: leading dimension de A

rnA: rango de la matriz A

cs: numero de cifras significativas a escribir

*void matrizlduD1atexto(FILE *archivo, double *A, unsigned long ldA, unsigned long rnA, char cs);*

Saca a texto la submatriz D1

archivo: puntero al archivo donde se quiere volcar

A: matriz. Debe estar factorizada

ldA: leading dimension de A

rnA: rango de la matriz A

cs: numero de cifras significativas a escribir

*void matrizlduD1atextooctave(FILE *archivo, double *A, unsigned long ldA, unsigned long rnA, char cs);*

Saca a texto (formato OCTAVE) la submatriz D1

archivo: puntero al archivo donde se quiere volcar

A: matriz. Debe estar factorizada

ldA: leading dimension de A

rnA: rango de la matriz A

cs: numero de cifras significativas a escribir

*void matrizlduU11atexto(FILE *archivo, double *A, unsigned long ldA, unsigned long rnA, char cs);*

Saca a texto la submatriz U11

archivo: puntero al archivo donde se quiere volcar

A: matriz. Debe estar factorizada

ldA: leading dimension de A

rnA: rango de la matriz A

cs: numero de cifras significativas a escribir

*void matrizlduU11atextooctave(FILE *archivo, double *A, unsigned long ldA, unsigned long rnA, char cs);*

Saca a texto (formato OCTAVE) la submatriz U11

archivo: puntero al archivo donde se quiere volcar

A: matriz. Debe estar factorizada

ldA: leading dimension de A

rnA: rango de la matriz A

cs: numero de cifras significativas a escribir

*void matrizlduN1atexto(FILE *archivo, double *A, unsigned long cA, unsigned long ldA, unsigned long rnA, char cs);*

Saca a texto la submatriz N1 (parte del subespacio nulo)

archivo: puntero al archivo donde se quiere volcar

A: matriz. Debe estar factorizada y debe tener el subespacio nulo calculado

cA: numero de columnas de A

ldA: leading dimension de A

rnA: rango de la matriz A

cs: numero de cifras significativas a escribir

*void matrizlduN1atextooctave(FILE *archivo, double *A, unsigned long cA, unsigned long ldA, unsigned long rnA, char cs);*

Saca a texto (formato OCTAVE) la submatriz N1 (parte del subespacio nulo)

archivo: puntero al archivo donde se quiere volcar

A: matriz. Debe estar factorizada y debe tener el subespacio nulo calculado

cA: numero de columnas de A

ldA: leading dimension de A

rnA: rango de la matriz A

cs: numero de cifras significativas a escribir

*void matrizlduS1Tatexto(FILE *archivo, double *A, unsigned long fA, unsigned long ldA, unsigned long rnA, char cs);*

Saca a texto la submatriz S1T (parte del subespacio nulo izquierdo)

archivo: puntero al archivo donde se quiere volcar

A: matriz. Debe estar factorizada y debe tener el subespacio nulo izquierdo calculado

fA: numero de filas de A

ldA: leading dimension de A

rnA: rango de la matriz A

cs: numero de cifras significativas a escribir

*void matrizlduS1Tatextooctave(FILE *archivo, double *A, unsigned long fA, unsigned long ldA, unsigned long rnA, char cs);*

Saca a texto (formato OCTAVE) la submatriz S1T (parte del subespacio nulo izquierdo)

archivo: puntero al archivo donde se quiere volcar

A: matriz. Debe estar factorizada y debe tener el subespacio nulo izquierdo calculado

fA: numero de filas de A

ldA: leading dimension de A

rnA: rango de la matriz A

cs: numero de cifras significativas a escribir

*void ldumatrizfactptA(double *A, unsigned long fA, unsigned long cA, unsigned long ldA, unsigned long *rnA, unsigned long *pivf, unsigned long *pivc, double eps);*

Realiza la factorización ldu de la matriz A con pivotaje completo. la factorización se hace dentro de la propia matriz A

A: puntero a la matriz

fA: numero de filas de la matriz

cA: numero de columnas de la matriz

ldA: dimensión fundamental de A

rnA: en salida, rango de la matriz

pivf: puntero a los cambios de filas. Debe apuntar a un espacio reservado de al menos el rango de la matriz (en caso de no saber a priori, vale con el menor de fA y fC)

pivc: puntero a los cambios de columnas. Debe apuntar a un espacio reservado de al menos el rango de la matriz (en caso de no saber a priori, vale con el menor de fA y fC)

eps: valor a considerar como cero numérico

*void ldumatrizfactrookA(double *A, unsigned long fA, unsigned long cA, unsigned long ldA, unsigned long *rnA, unsigned long *pivf, unsigned long *pivc, double eps);*

Realiza la factorización ldu de la matriz A con pivotaje de Rook. la factorización se hace dentro de la propia matriz A

A: puntero a la matriz

fA: numero de filas de la matriz

cA: numero de columnas de la matriz

ldA: dimensión fundamental de A

rnA: en salida, rango de la matriz

pivf: puntero a los cambios de filas. Debe apuntar a un espacio reservado de al menos el rango de la matriz (en caso de no saber a priori, vale con el menor de fA y fC)

pivc: puntero a los cambios de columnas. Debe apuntar a un espacio reservado de al menos el rango de la matriz (en caso de no saber a priori, vale con el menor de fA y fC)

eps: valor a considerar como cero numérico

*void vectorlduL11Av(double *u, double *A, unsigned long ldA, unsigned long rnA, double*v);*

Multiplica la matriz L11 de la matriz factorizada A por el vector v

u: puntero al vector resultado. Puede coincidir con el origen

A: puntero a la matriz factorizada LDU

ldA: dimensión fundamental de la matriz factorizada

rnA: rango de la matriz

v: puntero al vector

*void matrizlduL11AB(double *C, unsigned long ldC, double *A, unsigned long int ldA, unsigned long int rnA, double *B, unsigned long cB, unsigned long ldB);*

Multiplica la matriz L11 de la matriz factorizada A por la matriz B

C: puntero a la matriz resultado. Puede ser igual a B

ldC: dimensión fundamental de C

A: puntero a los elementos de la matriz factorizada

ldA: dimensión fundamental de A

rnA: rango de la matriz factorizada

B: puntero a los elementos de la matriz a multiplicar y en salida resultado

cB: número de columnas de B

ldB: dimensión fundamental de B

*void vectorlduinvL11Av(double *u, double *A, unsigned long ldA, unsigned long rnA, double*v);*

Multiplica la inversa de la matriz L11 de la matriz factorizada A por el vector v

u: puntero al vector resultado. Puede coincidir con el origen

A: puntero a la matriz factorizada LDU

ldA: dimensión fundamental de la matriz factorizada

rnA: rango de la matriz

v: puntero al vector

*void matrizlduinvL11AB(double *C, unsigned long ldC, double *A, unsigned long int ldA, unsigned long int rnA, double *B, unsigned long cB, unsigned long ldB);*

Multiplica la inversa de la matriz L11 de la matriz factorizada A por la matriz B

C: puntero a la matriz resultado. Puede ser igual a B

ldC: dimensión fundamental de C

A: puntero a los elementos de la matriz factorizada

ldA: dimensión fundamental de A

rnA: rango de la matriz factorizada

B: puntero a los elementos de la matriz a multiplicar y en salida resultado

cB: número de columnas de B

ldB: dimensión fundamental de B

*void vectorlduL11TAv(double *u, double *A, unsigned long ldA, unsigned long rnA, double*v);*

Multiplica la matriz L11 de la matriz factorizada A traspuesta por el vector v

u: puntero al vector resultado. Puede coincidir con el origen

A: puntero a la matriz factorizada LDU

ldA: dimensión fundamental de la matriz factorizada

rnA: rango de la matriz

v: puntero al vector

*void matrizlduL11TAB(double *C, unsigned long ldC, double *A, unsigned long int ldA, unsigned long int rnA, double *B, unsigned long cB, unsigned long ldB);*

Multiplica la matriz L11 de la matriz factorizada A por la matriz B

C: puntero a la matriz resultado. Puede ser igual a B

ldC: dimensión fundamental de C

A: puntero a los elementos de la matriz factorizada

ldA: dimensión fundamental de A

rnA: rango de la matriz factorizada

B: puntero a los elementos de la matriz a multiplicar y en salida resultado

cB: número de columnas de B

ldB: dimensión fundamental de B

*void vectorlduinvL11TAv(double *u, double *A, unsigned long ldA, unsigned long rnA, double*v);*

Multiplica la inversa de la matriz L11 de la matriz factorizada A traspuesta por el vector v

u: puntero al vector resultado. Puede coincidir con el origen

A: puntero a la matriz factorizada LDU

ldA: dimensión fundamental de la matriz factorizada

rnA: rango de la matriz

v: puntero al vector

*void matrizlduinvL11TAB(double *C, unsigned long ldC, double *A, unsigned long int ldA, unsigned long int rnA, double *B, unsigned long cB, unsigned long ldB);*

Multiplica la matriz L11 de la matriz factorizada A por la matriz B

C: puntero a la matriz resultado. Puede ser igual a B

ldC: dimensión fundamental de C

A: puntero a los elementos de la matriz factorizada

ldA: dimension fundamental de A

rnA: rango de la matriz factorizada

B: puntero a los elementos de la matriz a multiplicar y en salida resultado

cB: número de columnas de B

ldB: dimensión fundamental de B

*void vectorlduD1Av(double *u, double *A, unsigned long ldA, unsigned long rnA, double*v);*

Multiplica la matriz D11 de la matriz factorizada A por el vector v

u: puntero al vector resultado. Puede coincidir con el origen

A: puntero a la matriz factorizada LDU

ldA: dimensión fundamental de la matriz factorizada

rnA: rango de la matriz

v: puntero al vector

*void matrizlduD1AB(double *C, unsigned long ldC, double *A, unsigned long int ldA, unsigned long int rnA, double *B, unsigned long cB, unsigned long ldB);*

Multiplica la matriz D1 de la matriz factorizada A por la matriz B

C: puntero a la matriz resultado. Puede ser igual a B

ldC: dimensión fundamental de C

A: puntero a los elementos de la matriz factorizada

ldA: dimension fundamental de A

rnA: rango de la matriz factorizada

B: puntero a los elementos de la matriz a multiplicar y en salida resultado

cB: número de columnas de B

ldB: dimension fundamental de B

*void vectorlduinvD1Av(double *u, double *A, unsigned long ldA, unsigned long rnA, double*v);*

Multiplica la inversa de la matriz D1 de la matriz factorizada A por el vector v

u: puntero al vector resultado. Puede coincidir con el origen

A: puntero a la matriz factorizada LDU

ldA: dimensión fundamental de la matriz factorizada

rnA: rango de la matriz

v: puntero al vector

*void matrizlduinvD1AB(double *C, unsigned long ldC, double *A, unsigned long int ldA, unsigned long int rnA, double *B, unsigned long cB, unsigned long ldB);*

Multiplica la inversa de la matriz D1 de la matriz factorizada A por la matriz B

C: puntero a la matriz resultado. Puede ser igual a B

ldC: dimensión fundamental de C

A: puntero a los elementos de la matriz factorizada

ldA: dimensión fundamental de A

rnA: rango de la matriz factorizada

B: puntero a los elementos de la matriz a multiplicar y en salida resultado

cB: número de columnas de B

ldB: dimensión fundamental de B

*void vectorlduU11Av(double *u, double *A, unsigned long ldA, unsigned long rnA, double*v);*

Multiplica la matriz U11 de la matriz factorizada A por el vector v

u: puntero al vector resultado. Puede coincidir con el origen

A: puntero a la matriz factorizada LDU

ldA: dimensión fundamental de la matriz factorizada

rnA: rango de la matriz

v: puntero al vector

*void matrizlduU11AB(double *C, unsigned long ldC, double *A, unsigned long int ldA, unsigned long int rnA, double *B, unsigned long cB, unsigned long ldB);*

Multiplica la matriz U11 de la matriz factorizada A por la matriz B

C: puntero a la matriz resultado. Puede ser igual a B

ldC: dimensión fundamental de C

A: puntero a los elementos de la matriz factorizada

ldA: dimension fundamental de A

rnA: rango de la matriz factorizada

B: puntero a los elementos de la matriz a multiplicar y en salida resultado

cB: número de columnas de B

ldB: dimension fundamental de B

*void vectorlduinvU11Av(double *u, double *A, unsigned long ldA, unsigned long rnA, double*v);*

Multiplica la inversa de la matriz U11 de la matriz factorizada A por el vector v

u: puntero al vector resultado. Puede coincidir con el origen

A: puntero a la matriz factorizada LDU

ldA: dimensión fundamental de la matriz factorizada

rnA: rango de la matriz

v: puntero al vector

*void matrizlduinvU11AB(double *C, unsigned long ldC, double *A, unsigned long int ldA, unsigned long int rnA, double *B, unsigned long cB, unsigned long ldB);*

Multiplica la inversa de la matriz U11 de la matriz factorizada A por la matriz B

C: puntero a la matriz resultado. Puede ser igual a B

ldC: dimensión fundamental de C

A: puntero a los elementos de la matriz factorizada

ldA: dimension fundamental de A

rnA: rango de la matriz factorizada

B: puntero a los elementos de la matriz a multiplicar y en salida resultado

cB: número de columnas de B

ldB: dimension fundamental de B

*void vectorlduU11TAv(double *u, double *A, unsigned long ldA, unsigned long rnA, double*v);*

Multiplica la matriz U11 de la matriz factorizada A traspuesta por el vector v

u: puntero al vector resultado. Puede coincidir con el origen

A: puntero a la matriz factorizada LDU

ldA: dimensión fundamental de la matriz factorizada

rnA: rango de la matriz

v: puntero al vector

*void matrizlduU11TAB(double *C, unsigned long ldC, double *A, unsigned long int ldA, unsigned long int rnA, double *B, unsigned long cB, unsigned long ldB);*

Multiplica la matriz U11 de la matriz factorizada A traspuesta por la matriz B

C: puntero a la matriz resultado. Puede ser igual a B

ldC: dimensión fundamental de C

A: puntero a los elementos de la matriz factorizada

ldA: dimension fundamental de A

rnA: rango de la matriz factorizada

B: puntero a los elementos de la matriz a multiplicar y en salida resultado

cB: número de columnas de B

ldB: dimension fundamental de B

*void vectorlduinvU11TAv(double *u, double *A, unsigned long ldA, unsigned long rnA, double*v);*

Multiplica la inversa de la matriz U11 de la matriz factorizada A traspuesta por el vector v

u: puntero al vector resultado. Puede coincidir con el origen

A: puntero a la matriz factorizada LDU

ldA: dimensión fundamental de la matriz factorizada

rnA: rango de la matriz

v: puntero al vector

*void matrizlduinvU11TAB(double *C, unsigned long ldC, double *A, unsigned long int ldA, unsigned long int rnA, double *B, unsigned long cB, unsigned long ldB);*

Multiplica la inversa de la matriz U11 de la matriz factorizada A traspuesta por la matriz B

C: puntero a la matriz resultado. Puede ser igual a B

ldC: dimensión fundamental de C

A: puntero a los elementos de la matriz factorizada

ldA: dimension fundamental de A

rnA: rango de la matriz factorizada

B: puntero a los elementos de la matriz a multiplicar y en salida resultado

cB: número de columnas de B

ldB: dimension fundamental de B

*void ldumatrizobtenersubespacionulo(double *A, unsigned long int cA, unsigned long int ldA, unsigned long int rnA);*

Calcula la parte superior del subespacio nulo de la matriz factorizada (U11^-1 U12)

y lo almacena en la parte U_12 de la propia matriz, de manera que la matriz

|-U12|

| I |

Es el subespacio nulo de la matriz factorizada.

cA: número de columnas de A

ldA: dimensión fundamental de A

rnA: rango de la matriz factorizada

*void ldumatrizobtenersubespacionuloizquierdo(double *A, unsigned long int fA, unsigned long int ldA, unsigned long int rnA);*

Calcula la parte superior del subespacio nulo izquierdo de la matriz factorizada (L21 L11^-1)

y lo almacena en la parte L_21 de la propia matriz, de manera que la matriz

|-L21 || I |

Es el subespacio nulo izquierdo de la matriz factorizada.

fA: número de filas de A

ldA: dimensión fundamental de A

rnA: rango de la matriz factorizada

*void vectoraldusubS1Av(double *u, double a, double *A, unsigned long int fA, unsigned long int ldA, unsigned long int rnA, double *v);*

Establece en el vector destino, u, el escalar a por la matriz S1 en la matriz A factorizada ldu con subespacio nulo izquierdo calculado por el vector v

u: vector donde se quiere obtener el resultado.

a: escalar por el que se quiere multiplicar la operación al sumar

A: matriz factorizada ldu y con la parte del subespacio nulo izquierdo calculado

fA: numero de filas de la matriz

ldA: dimensión fundamental de A

rnA: rango de la matriz

v: vector

*void vectorsumaraldusubS1Av(double *u, double a, double *A, unsigned long int fA, unsigned long int ldA, unsigned long int rnA, double *v);*

Suma al vector destino, u, el escalar a por la matriz S1 en la matriz A factorizada ldu con subespacio nulo izquierdo calculado por el vector v

u: vector donde se quiere obtener el resultado.

A: escalar por el que se quiere multiplicar la operación al sumar

A: matriz factorizada ldu y con la parte del subespacio nulo izquierdo calculado

fA: numero de filas de la matriz

ldA: dimensión fundamental de A

rnA: rango de la matriz

v: vector

*void matrizsumaraldusubS1AB(double *C, unsigned long ldC, double a, double *A, unsigned long int fA, unsigned long int ldA, unsigned long int rnA, double *B, double cB, unsigned long ldB);*

Suma a la matriz destino, C la matriz S1 por la matriz B y todo ello multiplicado por el escalar a

C: matriz donde se quiere sumar la operación. Debe tener espacio reservado

ldC: dimension fundamental de C

a: escalar por el que se quiere multiplicar la operación al sumar

A: matriz factorizada ldu y con la parte del subespacio nulo izquierdo calculado

fA: numero de filas de la matriz

ldA: dimension fundamental de A

rnA: rango de la matriz

B: matriz a multiplicar

cB: numero de columnas de la matriz

ldB: dimension fundamental de la matriz B

*void vectorsumaraldusubS1ATv(double *u, double a, double *A, unsigned long int fA, unsigned long int ldA, unsigned long int rnA, double *v);*

Suma al vector destino, u, el escalar a por la matriz S1 traspuesta en la matriz A factorizada ldu con subespacio nulo izquierdo calculado por el vector v

u: vector donde se quiere obtener el resultado.

a: escalar por el que se quiere multiplicar la operacion al sumar

A: matriz factorizada ldu y con la parte del subespacio nulo izquierdo calculado

fA: numero de filas de la matriz

ldA: dimension fundamental de A

rnA: rango de la matriz

v: vector

*void matrizsumaraldusubS1ATB(double *C, unsigned long ldC, double a, double *A, unsigned long int fA, unsigned long int ldA, unsigned long int rnA, double *B, double cB, unsigned long ldB);*

Suma a la matriz destino, C la matriz S1 traspuesta por la matriz B y todo ello multiplicado por el escalar a

C: matriz donde se quiere sumar la operación. Debe tener espacio reservado

ldC: dimension fundamental de C

a: escalar por el que se quiere multiplicar la operacion al sumar

A: matriz factorizada ldu y con la parte del subespacio nulo izquierdo calculado

fA: numero de filas de la matriz

ldA: dimension fundamental de A

rnA: rango de la matriz

B: matriz a multiplicar

cB: numero de columnas de la matriz

ldB: dimension fundamental de la matriz B

*void vectoraldusubN1Av(double *u, double a, double *A, unsigned long int cA, unsigned long int ldA, unsigned long int rnA, double *v);*

Establece en el vector destino, u, el escalar a por la matriz N1 en la matriz A factorizada ldu con subespacio nulo izquierdo calculado por el vector v

u: vector donde se quiere obtener el resultado.

a: escalar por el que se quiere multiplicar la operacion al sumar

A: matriz factorizada ldu y con la parte del subespacio nulo izquierdo calculado

cA: numero de filas de la matriz

ldA: dimension fundamental de A

rnA: rango de la matriz

v: vector

*void vectorsumaraldusubN1Av(double *u, double a, double *A, unsigned long int cA, unsigned long int ldA, unsigned long int rnA, double *v);*

Suma al vector destino, u, el escalar a por la matriz N1 en la matriz A factorizada ldu con subespacio nulo izquierdo calculado por el vector v

u: vector donde se quiere obtener el resultado.

a: escalar por el que se quiere multiplicar la operacion al sumar

A: matriz factorizada ldu y con la parte del subespacio nulo izquierdo calculado

cA: numero de filas de la matriz

ldA: dimension fundamental de A

rnA: rango de la matriz

v: vector

*void matrizaldusubN1AB(double *C, unsigned long ldC, double a, double *A, unsigned long int cA, unsigned long int ldA, unsigned long int rnA, double *B, double cB, unsigned long ldB);*

Establece en la matriz destino, C la matriz N1 por la matriz B y todo ello multiplicado por el escalar a

C: matriz donde se quiere realizar la operación. Debe tener espacio reservado

ldC: dimension fundamental de C

a: escalar por el que se quiere multiplicar la operacion al sumar

A: matriz factorizada ldu y con la parte del subespacio nulo izquierdo calculado

cA: numero de columnas de la matriz

ldA: dimension fundamental de A

rnA: rango de la matriz

B: matriz a multiplicar

cB: numero de columnas de la matriz

ldB: dimension fundamental de la matriz B

*void matrizsumaraldusubN1AB(double *C, unsigned long ldC, double a, double *A, unsigned long int cA, unsigned long int ldA, unsigned long int rnA, double *B, double cB, unsigned long ldB);*

Suma a la matriz destino, C la matriz N1 por la matriz B y todo ello multiplicado por el escalar a

C: matriz donde se quiere sumar la operación. Debe tener espacio reservado

ldC: dimension fundamental de C

a: escalar por el que se quiere multiplicar la operacion al sumar

A: matriz factorizada ldu y con la parte del subespacio nulo izquierdo calculado

cA: numero de columnas de la matriz

ldA: dimension fundamental de A

rnA: rango de la matriz

B: matriz a multiplicar

cB: numero de columnas de la matriz

ldB: dimension fundamental de la matriz B

*void matrizaAldusubN1B(double *C, unsigned long ldC, double a, double *A, unsigned long fA, unsigned long int cA, unsigned long int ldA, double *B, unsigned long cB, unsigned long ldB);*

Establece en a la matriz destino, C, el escalar a por la matriz A por la matriz N1 de B factorizada ldu y con subespacio nulo calculado

C: matriz donde se quiere obtener el resultado. Debe tener espacio reservado

ldC: dimension fundamental de C

a: escalar por el que se quiere multiplicar la operacion al sumar

A: matriz a multiplicar

fA: número de filas de la matriz A

cA: numero de columnas de la matriz, obviamente igual al rango de B

ldA: dimension fundamental de A

B: matriz cuyo subespacio nulo se quiere postmultiplicar. Debe estar factorizada y con los subespacios nulos calculados

cB: número de columnas de B

ldB: dimension fundamental de la matriz B

*void matrizsumaraAldusubN1B(double *C, unsigned long ldC, double a, double *A, unsigned long fA, unsigned long int cA, unsigned long int ldA, double *B, unsigned long cB, unsigned long ldB);*

Suma a la matriz destino, C, el escalar a por la matriz A por la matriz N1 de B factorizada ldu y con subespacio nulo calculado

C: matriz donde se quiere sumar la operación. Debe tener espacio reservado

ldC: dimension fundamental de C

a: escalar por el que se quiere multiplicar la operacion al sumar

A: matriz a multiplicar

fA: número de filas de la matriz A

cA: numero de columnas de la matriz, obviamente igual al rango de B

ldA: dimension fundamental de A

B: matriz cuyo subespacio nulo se quiere postmultiplicar. Debe estar factorizada y con los subespacios nulos calculados

cB: número de columnas de B

ldB: dimension fundamental de la matriz B

*void vectoraldusubN1ATv(double *u, double a, double *A, unsigned long int cA, unsigned long int ldA, unsigned long int rnA, double *v);*

Establece en el vector destino, u, el escalar a por la matriz N1 en la matriz A traspuesta factorizada ldu con subespacio nulo izquierdo calculado por el vector v

u: vector donde se quiere obtener el resultado.

a: escalar por el que se quiere multiplicar la operacion al sumar

A: matriz factorizada ldu y con la parte del subespacio nulo izquierdo calculado

cA: numero de filas de la matriz

ldA: dimension fundamental de A

rnA: rango de la matriz

v: vector

*void matrizaldusubN1ATB(double *C, unsigned long ldC, double a, double *A, unsigned long int cA, unsigned long int ldA, unsigned long int rnA, double *B, double cB, unsigned long ldB)*

Establece en la matriz destino, C, el escalar a por la matriz N1 en la matriz A traspuesta factorizada ldu con subespacio nulo izquierdo calculado por la matriz B

C: matriz donde se quiere sumar la operación. Debe tener espacio reservado

ldC: dimension fundamental de C

a: escalar por el que se quiere multiplicar la operacion al sumar

A: matriz factorizada ldu y con la parte del subespacio nulo izquierdo calculado

cA: numero de columnas de la matriz

ldA: dimension fundamental de A

rnA: rango de la matriz

B: matriz a multiplicar

cB: numero de columnas de la matriz

ldB: dimension fundamental de la matriz B

```
void vectorsumaraldusubN1ATv(double *u, double a, double *A, unsigned long int cA, unsigned long int ldA, unsigned long int rnA, double *v);
```

Suma al vector destino, u, el escalar a por la matriz N1 en la matriz A traspuesta factorizada ldu con subespacio nulo izquierdo calculado por el vector v

u: vector donde se quiere obtener el resultado.

a: escalar por el que se quiere multiplicar la operacion al sumar

A: matriz factorizada ldu y con la parte del subespacio nulo izquierdo calculado

cA: numero de filas de la matriz

ldA: dimension fundamental de A

rnA: rango de la matriz

v: vector

```
void matrizsumaraldusubN1ATB(double *C, unsigned long ldC, double a, double *A, unsigned long int cA, unsigned long int ldA, unsigned long int rnA, double *B, double cB, unsigned long ldB);
```

Suma a la matriz destino, C la matriz N1 traspuesta por la matriz B y todo ello multiplicado por el escalar a

C: matriz donde se quiere sumar la operación. Debe tener espacio reservado

ldC: dimension fundamental de C

a: escalar por el que se quiere multiplicar la operacion al sumar

A: matriz factorizada ldu y con la parte del subespacio nulo izquierdo calculado

cA: numero de columnas de la matriz

ldA: dimension fundamental de A

rnA: rango de la matriz

B: matriz a multiplicar

cB: numero de columnas de la matriz

ldB: dimension fundamental de la matriz B

*void matrizaAldusubNB(double *C, unsigned long ldC, double a, double *A, unsigned long fA, unsigned long cA, unsigned long ldA, double *B, unsigned long ldB, unsigned long rnB);*

Establece en C el producto de A por el subespacio nulo de B

C: matriz resultado

ldC: dimension fundamental de C

a: escalar

A: matriz factorizada ldu debe terner los subespacios calculados

fA: número de filas de A

cA: número de columnas de A

ldA: dimensión fundamental de A

B: matriz a multiplicar (fB=cA, cB=fB-rnB=cA-rnB)

ldB: dimensión fundamental de B

rnB: rango de B

*void matrizreduccionAldusubNB(double *A, unsigned long fA, unsigned long cA, unsigned long ldA, double *B, unsigned long ldB, unsigned long rnB, unsigned long *pivcB)*

Establece en &A([rnB]) el producto de A por el subespacio nulo de B (ojo que el ldA se mantiene). El resto de A queda inutilizado.

A: matriz a reducir

fA: número de filas de A

cA: número de columnas de A

ldA: dimensión fundamental de A

B: matriz factorizada ldu. Debe tener los subespacios calculados

ldB: dimensión fundamental de B

rnB: rango de B

pivcB: cambios de columnas en B en la factorización

*void vectorreduccionldusubNTAv(double *A, unsigned long cA, unsigned long ldA, unsigned long rnA, unsigned long *pivcA, double *v)*

Establece en &v([rnA]) el producto del subespacio nulo de A traspuesto por v. El resto de v queda inutilizado

A: matriz factorizada ldu. Debe tener el subespacio nulo calculado

fA: número de filas de A

cA: número de columnas de A

ldA: dimensión fundamental de A

rnA: rango de A

pivcA: cambios de columnas en A en la factorización

v: vector a reducir

*void matrizreduccionldusubNTAB(double *A, unsigned long cA, unsigned long ldA, unsigned long rnA, unsigned long *pivcA, double *B, unsigned long cB, unsigned long ldB);*

Establece en &B([rnA*ldB]) el producto del subespacio nulo de A traspuesto por B. El ldB se mantiene, ojo. El resto de B queda inutilizado.

A: matriz factorizada ldu. Debe tener el subespacio nulo calculado

cA: número de columnas de A

ldA: dimensión fundamental de A

rnA: rango de A

pivcA: cambios de columnas en A en la factorización

B: matriz a reducir

cB: columnas de B

ldB: dimensión fundamental de B

*void simmatrizaldusubS1TS1A(double *sC, unsigned long ldsC, double a, double *A, unsigned long int fA, unsigned long int ldA, unsigned long int rnA);*

Crea en la matriz simétrica destino, C, el escalar a por la matriz S1 traspuesta por la matriz S1

sC: puntero a la matriz simétrica

ldsC: dimensión fundamental de la matriz simétrica

a: escalar

A: puntero a la matriz factorizada ldu con el subespacio nulo calculado

fA: número de filas de la matriz A

ldA: dimensión fundamental de la matriz A

rnA: rango de la matriz A

*void simmatrizsumaraldusubS1TS1A(double *sC, unsigned long ldsC, double a, double *A, unsigned long int fA, unsigned long int ldA, unsigned long int rnA);*

Suma a la matriz simétrica destino, C, el escalar a por la matriz S1 traspuesta por la matriz S1

sC: puntero a la matriz simétrica

ldsC: dimensión fundamental de la matriz simétrica

a: escalar

A: puntero a la matriz factorizada ldu con el subespacio nulo calculado

fA: número de filas de la matriz A

ldA: dimensión fundamental de la matriz A

rnA: rango de la matriz A

*void simmatrizaldusubS1S1TA(double *sC, unsigned long ldsC, double a, double *A, unsigned long int fA, unsigned long int ldA, unsigned long int rnA);*

Crea en la matriz simétrica destino, C, el escalar a por la matriz S1 traspuesta por la matriz S1

sC: puntero a la matriz simétrica

ldsC: dimensión fundamental de la matriz simétrica

a: escalar

A: puntero a la matriz factorizada ldu con el subespacio nulo calculado

fA: número de filas de la matriz A

ldA: dimensión fundamental de la matriz A

rnA: rango de la matriz A

*void simmatrizsumaraldusubS1S1TA(double *sC, unsigned long ldsC, double a, double *A, unsigned long int fA, unsigned long int ldA, unsigned long int rnA);*

Suma a la matriz simétrica destino, C, el escalar a por la matriz S1 traspuesta por la matriz S1

sC: puntero a la matriz simétrica

ldsC: dimensión fundamental de la matriz simétrica

a: escalar

A: puntero a la matriz factorizada ldu con el subespacio nulo calculado

fA: número de filas de la matriz A

ldA: dimensión fundamental de la matriz A

rnA: rango de la matriz A

*void simmatrizaldusubN1TN1A(double *sC, unsigned long ldsC, double a, double *A, unsigned long int cA, unsigned long int ldA, unsigned long int rnA);*

Crea en la matriz simétrica destino, C, el escalar a por la matriz N1 traspuesta por la matriz N1

sC: puntero a la matriz simétrica

ldsC: dimensión fundamental de la matriz simétrica

a: escalar

A: puntero a la matriz factorizada ldu con el subespacio nulo calculado

cA: número de columnas de la matriz A

ldA: dimensión fundamental de la matriz A

rnA: rango de la matriz A

*void simmatrizsumaraldusubN1TN1A(double *sC, unsigned long ldsC, double a, double *A, unsigned long int cA, unsigned long int ldA, unsigned long int rnA);*

Suma a la matriz simétrica destino, C, el escalar a por la matriz N1 traspuesta por la matriz N1

sC: puntero a la matriz simétrica

ldsC: dimensión fundamental de la matriz simétrica

a: escalar

A: puntero a la matriz factorizada ldu con el subespacio nulo calculado

cA: número de columnas de la matriz A

ldA: dimensión fundamental de la matriz A

rnA: rango de la matriz A

*void simmatrizaldusubN1N1TA(double *sC, unsigned long ldsC, double a, double *A, unsigned long int cA, unsigned long int ldA, unsigned long int rnA);*

Crea en la matriz simétrica destino, C, el escalar a por la matriz N1 traspuesta por la matriz N1

sC: puntero a la matriz simétrica

ldsC: dimensión fundamental de la matriz simétrica

a: escalar

A: puntero a la matriz factorizada ldu con el subespacio nulo calculado

cA: número de columnas de la matriz A

ldA: dimensión fundamental de la matriz A

rnA: rango de la matriz A

*void simmatrizsumaraldusubN1N1TA(double *sC, unsigned long ldsC, double a, double *A, unsigned long int cA, unsigned long int ldA, unsigned long int rnA);*

Suma a la matriz simétrica destino, C, el escalar a por la matriz N1 traspuesta por la matriz N1

sC: puntero a la matriz simétrica

ldsC: dimensión fundamental de la matriz simétrica

a: escalar

A: puntero a la matriz factorizada ldu con el subespacio nulo calculado

cA: número de columnas de la matriz A

ldA: dimensión fundamental de la matriz A

rnA: rango de la matriz A

*void simmatrizaldusubNTNA(double *sC, unsigned long ldsC, double a, double *A, unsigned long int cA, unsigned long int ldA, unsigned long int rnA);*

Construye en la matriz simétrica destino, C el escalar a por la matriz N traspuesta por la matriz N

sC: puntero a la matriz simétrica

ldsC: dimensión fundamental de la matriz simétrica

a: escalar

A: puntero a la matriz factorizada ldu con el subespacio nulo calculado

cA: número de columnas de la matriz A

ldA: dimensión fundamental de la matriz A

rnA: rango de la matriz A

*void simmatrizaldusubSTSA(double *sC, unsigned long ldsC, double a, double *A, unsigned long int fA, unsigned long int ldA, unsigned long int rnA);*

Construye en la matriz simétrica destino, C el escalar a por la matriz S traspuesta por la matriz S

sC: puntero a la matriz simétrica

ldsC: dimensión fundamental de la matriz simétrica

a: escalar

A: puntero a la matriz factorizada ldu con el subespacio nulo calculado

fA: número de filas de la matriz A

ldA: dimensión fundamental de la matriz A

rnA: rango de la matriz A

*int ldumatrizcalcularsimetricaspinv(double **w,double *A, unsigned long fA, unsigned long cA, unsigned long ldA, unsigned long rnA);*

Reserva espacio para las matrices simétricas necesarias para calcular soluciones de mínimos cuadrados y mínima norma, calcula estas matrices y las factoriza

w: parámetro de vuelta. En retorno, puntero al espacio reservado. Se debe librar con free cuando se haya acabado con él

A: puntero a la matriz factorizada ldu con el subespacio nulo calculado

fA: número de filas de la matriz A

ldA: dimensión fundamental de la matriz A

rnA: rango de la matriz A

devuelve: 0 si todo correcto, 1 si no memoria

*void vectorpinvlduAv(double *u, double *A, unsigned long fA, unsigned long cA, unsigned long ldA, unsigned long rnA, unsigned long int *pivf, unsigned long int *pivc, double *w, double *v);*

Resuelve el sistema Au=v, dando solución de norma mínima de mínimos cuadrados, a partir de una matriz ya factorizada y con subespacios calculados.

u: vector solución. puede ser igual a v

A: matriz. Debe estar ya factorizada

fA: número de filas de la matriz

cA: número de columnas de la matriz

ldA: dimensión fundamental de la matriz A

rnA: rango de la matriz A

pivf: matriz de cambio de filas de A (dimensión rnA)

pivC: matriz de cambio de columnas de A (dimensión rnA)

w: puntero a las matrices simétricas

v: vector de independientes

*void matrizpinvlduAB(double *C,unsigned long ldC, double *A, unsigned long fA, unsigned long cA, unsigned long ldA, unsigned long rnA, unsigned long int *pivf, unsigned long int *pivc, double *w, double *B, unsigned long cB, unsigned long ldB);*

Resuelve el sistema AC=B, dando solución de norma mínima de mínimos cuadrados, a partir de una matriz ya factorizada y con subespacios calculados. Requiere el cálculo de las matrices simétricas realizado con ldumatrizcalcularsimetricaspinv

C: vector solución. Puede ser igual a B

ldC: leading dimension de C.

A: matriz. Debe estar ya factorizada

fA: número de filas de la matriz

cA: número de columnas de la matriz

ldA: dimensión fundamental de la matriz A

rnA: rango de la matriz A

pivf: matriz de cambio de filas de A (dimensión rnA)

pivC: matriz de cambio de columnas de A (dimensión rnA)

w: puntero a las matrices simétricas realizado con ldumatrizcalcularsimetricaspinv

B: matriz de independientes

cB: número de columnas de B

ldB: leading dimension de B

*void vectorpinvlduATv(double *u, double *A, unsigned long fA, unsigned long cA, unsigned long ldA, unsigned long rnA, unsigned long int *pivf, unsigned long int *pivc, double *w, double *v);*

Resuelve el sistema ATu=v, dando solución de norma mínima de mínimos cuadrados, a partir de una matriz ya factorizada y con subespacios calculados. Requiere el cálculo de las matrices simétricas

u: vector solución. puede ser igual a v

A: matriz. Debe estar ya factorizada

fA: número de filas de la matriz

cA: número de columnas de la matriz

ldA: dimensión fundamental de la matriz A

rnA: rango de la matriz A

pivf: matriz de cambio de filas de A (dimensión rnA)

pivC: matriz de cambio de columnas de A (dimensión rnA)

w: puntero a las matrices simétricas

v: vector de independientes

*void vectorreduccionldupinvsubNAv(double *A, unsigned long fA, unsigned long cA, unsigned long ldA, unsigned long rnA, unsigned long int *pivc, double *w, double *v)*

Calcula la pseudoinversa del subespacio nulo de A por el vector v y lo almacena en &(V[rnA]). Esto es equivalente a proyectar el vector v en el subespacio. La matriz A debe estar factorizada por ldu con los subespacios calculados y con las matrices simetricas para calculos de minimos cuadrados de minima norma calculadas.

A: puntero a la matriz factorizada

fA: numero de filas de la matriz

cA: numero de columnas de la matriz

ldA: dimensión fundamental de A

rnA: en salida, rango de la matriz

pivc: puntero a los cambios de columnas. Debe apuntar a un espacio reservado de al menos el rango de la matriz (en caso de no saber a priori, vale con el menor de fA y fC)

w: puntero a espacio de trabajo. Es para almacenar las matrices simetricas. Debe tener reservado al menos dim*(dim+1)/2+dim2*(dim2+1)/2, siendo dim1=min(rnA,cA-rnA) y dim2=min(rnA,fA-

rnA). En salida tiene las matrices simétricas necesarias para la solución de mínimos cuadrados de mínima norma

v: vector que se quiere reducir

*void ldumatrizpinvptA(double *A, unsigned long fA, unsigned long cA, unsigned long ldA, unsigned long *rnA, unsigned long *pivf, unsigned long *pivc, double *w, double eps);*

Realiza la factorización ldu de la matriz A con pivotaje completo, cálculo de subespacios y de matrices simétricas (factorizadas) para el cálculo de solución de mínimos cuadrados de mínima norma. Todo lo posible se hace dentro de la matriz A.

A: puntero a la matriz

fA: numero de filas de la matriz

cA: numero de columnas de la matriz

ldA: dimensión fundamental de A

rnA: en salida, rango de la matriz

pivf: puntero a los cambios de filas. Debe apuntar a un espacio reservado de al menos el rango de la matriz (en caso de no saber a priori, vale con el menor de fA y fC)

pivc: puntero a los cambios de columnas. Debe apuntar a un espacio reservado de al menos el rango de la matriz (en caso de no saber a priori, vale con el menor de fA y fC)

w: puntero a espacio de trabajo. Es para almacenar las matrices simetricas. Debe tener reservado al menos dim*(dim+1)/2+dim2*(dim2+1)/2 siendo dim1=min(rnA,cA-rnA) y dim2=min(rnA,fA-rnA). En salida tiene las matrices simétricas necesarias para la solución de mínimos cuadrados de mínima norma

w: puntero a espacio de trabajo. Es para almacenar las matrices simetricas. Debe tener reservado al menos dim*(dim+1)/2+dim2*(dim2+1)/2 siendo dim1=min(rnA,cA-rnA) y dim2=min(rnA,fA-rnA). En salida tiene las matrices simétricas necesarias para la solución de mínimos cuadrados de mínima norma

*void ldumatrizpinvrookA(double *A, unsigned long fA, unsigned long cA, unsigned long ldA, unsigned long *rnA, unsigned long *pivf, unsigned long *pivc, double *w, double eps);*

Realiza la factorización ldu de la matriz A con pivotaje de Rook, cálculo de subespacios y de matrices simétricas (factorizadas) para el cálculo de solución de mínimos cuadrados de mínima norma. Todo lo posible se hace dentro de la matriz A.

A: puntero a la matriz

fA: numero de filas de la matriz

cA: numero de columnas de la matriz

ldA: dimensión fundamental de A

rnA: en salida, rango de la matriz

pivf: puntero a los cambios de filas. Debe apuntar a un espacio reservado de al menos el rango de la matriz (en caso de no saber a priori, vale con el menor de fA y fC)

pivc: puntero a los cambios de columnas. Debe apuntar a un espacio reservado de al menos el rango de la matriz (en caso de no saber a priori, vale con el menor de fA y fC)

w: puntero a espacio de trabajo. Es para almacenar las matrices simetricas. Debe tener reservado al menos dim*(dim+1)/2+dim2*(dim2+1)/2 siendo dim1=min(rnA,cA-rnA) y dim2=min(rnA,fA-rnA). En salida tiene las matrices simétricas necesarias para la solución de mínimos cuadrados de mínima norma

eps: valor a considerar como cero numérico

3.8 Factorización LDL

Obviamente, se aplica a matrices simétricas.

*void ldlmatrizetapareduccion(double *sA, unsigned long esA, unsigned long ldsA);*

Hace la etapa de reducción de la factorización.

sA: puntero a la matriz simétrica

esA: numero de elementos de la matriz

ldsA: dimensión fundamental de la matriz

*void ldlmatrizfact(double *sA, unsigned long esA, unsigned long ldsA);*

Hace la factorización ldl sin pivotamiento. Interesante para sistemas positivos definidos.

sA: puntero a la matriz simátrica

esA: numero de elementos de la matriz simétrica

ldsA: dimensión fundamental de la matriz simétrica

*void vectorldlDsAv(double *u, double *sA, unsigned long esA, unsigned long ldsA, double *v);*

Multiplica la matriz diagonal D en la matriz sA factorizada ldl por el vector v y lo almacena en u.

u: puntero al vector resultado. puede ser igual a v

sA: puntero a la matriz simétrica

esA: número de elementos de la matriz

ldsA: dimensión fundamental de A

v: puntero al vector v

*void vectorldlinvDsAv(double *u, double *sA, unsigned long esA, unsigned long ldsA, double *v);*

Multiplica la inversa de la matriz diagonal D en la matriz sA factorizada ldl por el vector v y lo almacena en u.

u: puntero al vector resultado. puede ser igual a v

sA: puntero a la matriz simétrica

esA: número de elementos de la matriz

ldsA: dimensión fundamental de A

v: puntero al vector v

*void matrizldlDsAB(double *C, unsigned long ldC, double *sA, unsigned long esA, unsigned long ldsA, double *B, unsigned long cB, unsigned long ldB);*

Multiplica la matriz B por la matriz diagonal D en la matriz sA factorizada ldl y lo almacena en C.

C: puntero a la matriz resultado. puede ser igual a B

ldC: dimensión fundamental de la matriz C

sA: puntero a la matriz simétrica

esA: número de elementos de la matriz

ldsA: dimensión fundamental de A

B: puntero a la matriz B

*void matrizldlinvDsAB(double *C, unsigned long ldC, double *sA, unsigned long esA, unsigned long ldsA, double *B, unsigned long cB, unsigned long ldB);*

Multiplica la matriz B por el inverso de la matriz diagonal D en la matriz sA factorizada ldl y lo almacena en C.

C: puntero a la matriz resultado. puede ser igual a B

ldC: dimensión fundamental de la matriz C

sA: puntero a la matriz simétrica

esA: número de elementos de la matriz

ldsA: dimensión fundamental de A

B: puntero a la matriz B

*void vectorldlLsAv(double *u, double *sA, unsigned long esA, unsigned long ldsA, double *v);*

Multiplica la matriz L en la matriz sA factorizada ldl por el vector v y lo almacena en u.

u: puntero al vector resultado. puede ser igual a v

sA: puntero a la matriz simétrica

esA: número de elementos de la matriz

ldsA: dimensión fundamental de A

v: puntero al vector

*void vectorldlinvLsAv(double *u, double *sA, unsigned long esA, unsigned long ldsA, double *v);*

Multiplica la inversa de la matriz L en la matriz sA factorizada ldl por el vector v y lo almacena en u.

u: puntero al vector resultado. puede ser igual a v

sA: puntero a la matriz simétrica

esA: número de elementos de la matriz

ldsA: dimensión fundamental de A

v: puntero al vector v

*void matrizldlLsAB(double *C, unsigned long ldC, double *sA, unsigned long esA, unsigned long ldsA, double *B, unsigned long cB, unsigned long ldB);*

Multiplica la matriz B por la matriz L en la matriz sA factorizada ldl y lo almacena en C.

C: puntero a la matriz resultado. puede ser igual a B

ldC: dimensión fundamental de la matriz C

sA: puntero a la matriz simétrica

esA: número de elementos de la matriz

ldsA: dimensión fundamental de A

B: puntero a la matriz B

*void matrizldlinvLsAB(double *C, unsigned long ldC, double *sA, unsigned long esA, unsigned long ldsA, double *B, unsigned long cB, unsigned long ldB);*

Multiplica la matriz B por el inverso de la matriz L en la matriz sA factorizada ldl y lo almacena en C.

C: puntero a la matriz resultado. puede ser igual a B

ldC: dimensión fundamental de la matriz C

sA: puntero a la matriz simétrica

esA: número de elementos de la matriz

ldsA: dimensión fundamental de A

B: puntero a la matriz B

*void vectorldlLTsAv(double *u, double *sA, unsigned long esA, unsigned long ldsA, double *v);*

Multiplica la matriz L en la matriz sA factorizada ldl traspuesta por el vector v y lo almacena en u.

u: puntero al vector resultado. puede ser igual a v

sA: puntero a la matriz simétrica

esA: número de elementos de la matriz

ldsA: dimensión fundamental de A

v: puntero al vector v

*void vectorldlinvLTsAv(double *u, double *sA, unsigned long esA, unsigned long ldsA, double *v);*

Multiplica la inversa de la matriz L en la matriz sA factorizada ldl traspuesta por el vector v y lo almacena en u.

u: puntero al vector resultado. puede ser igual a v

sA: puntero a la matriz simétrica

esA: número de elementos de la matriz

ldsA: dimensión fundamental de A

v: puntero al vector v

*void matrizldlLTsAB(double *C, unsigned long ldC, double *sA, unsigned long esA, unsigned long ldsA, double *B, unsigned long cB, unsigned long ldB);*

Multiplica la matriz B por la matriz L traspuesta en la matriz sA factorizada ldl y lo almacena en C.

C: puntero a la matriz resultado. puede ser igual a B

ldC: dimensión fundamental de la matriz C

sA: puntero a la matriz simétrica

esA: número de elementos de la matriz

ldsA: dimensión fundamental de A

B: puntero a la matriz B

cB: número de columnas de B

ldB: dimension fundamental de B

*void matrizldlinvLTsAB(double *C, unsigned long ldC, double *sA, unsigned long esA, unsigned long ldsA, double *B, unsigned long cB, unsigned long ldB);*

Multiplica la matriz B por el inverso de la matriz L traspuesta en la matriz sA factorizada ldl y lo almacena en C.

C: puntero a la matriz resultado. puede ser igual a B

ldC: dimensión fundamental de la matriz C

sA: puntero a la matriz simétrica

esA: número de elementos de la matriz

ldsA: dimensión fundamental de A

B: puntero a la matriz B

*void vectorldlinvAv(double *u, double *sA, unsigned long esA, unsigned long ldsA, double *v);*

Calcula en u la solución de sA^{-1}v, a partir de la matriz A factorizada ldl

u: vector solución. puede ser igual a v

sA: matriz simétrica factorizada

esA: número de elementos de la matriz

ldsA: dimensión fundamental de sA

v: vector de independientes.

*void matrizldlinvAB(double *C, unsigned long ldC, double *sA, unsigned long esA, unsigned long ldsA, double *B, unsigned long cB, unsigned long ldB);*

Calcula en C la solución de sA^{-1}B, a partir de la matriz A factorizada ldl

C: puntero a la matriz resultado. puede ser igual a B

ldC: dimensión fundamental de la matriz C

sA: puntero a la matriz simétrica

esA: número de elementos de la matriz

ldsA: dimensión fundamental de A

B: puntero a la matriz B

cB: número de columnas de B

ldB: dimension fundamental de B

*void ldlgenfactpt(double *sA, unsigned long esA, unsigned long ldsA, unsigned long *rnsA, unsigned long int *piv, double tol);*

Realiza una factorización completa de la matriz simétrica por el método de rotación (algoritmo recursivo) con pivotaje completo

sA: puntero a la matriz simétrica

esA: dimensión de la matriz (numero de filas/columnas)

ldsA: dimensión fundamental de la matriz

rnsA: en salida, rango de la matriz A

piv: array de pivotes. Tiene por dimension la dimensión de la submatriz multiplicada por dos. Son cambios sucesivos, es decir, la fila/columna a cambiar se permuta por la que esta piv veces a la derecha/abajo. Si estamos en la columna i y el pivote es j se cambia la i por la j+i. Si el elemento de mayor valor absoluto está en la diagonal principal, piv[0]=i, piv[1]=0, si no, piv[0]=i, piv[1]=j, donde i es la fila del pivote ij en la que aparece el elemento en la diagonal principal de mayor valor absoluto.

tol: tolerancia para definir el cero numérico que determina el fin de la factorización

*void matrizldlgeninvL11sAB(double *sA, unsigned long int ldsA, unsigned long rsA, double *B, unsigned long cB, unsigned long int ldB, unsigned long int *piv);*

Resolución del problema E=L^-1 E mediante pivotamiento girado completo, con E matriz (incluye giros y cambios de filas/columnas)

sA: puntero a la matriz simétrica factorizada con ldlgen

ldsA: dimensión fundamental de la matriz

rsA: rango de la matriz simétrica

B: matriz a multiplicar

cB: numero de columnas de B

ldB: Dimension pricipal de B

piv: puntero a los pivotes

*void matrizldlgeninvL11TsAv(double *sA, unsigned long int ldsA, unsigned long int rsA, double *v, unsigned long int *piv);*

Resolución del problema v=L11t^-1 v

sA: matriz factorizada de forma LDL generalizada

ldsA: dimension ppal de A

rsA: rango de la matriz (y tamaño del vector)

v: vector

piv: puntero a los pivotes

*void matrizldlgenpinvDsAB(double *C, unsigned long ldC, double *sA, unsigned long esA, unsigned long ldsA, unsigned long rsA,double *B, unsigned long cB, unsigned long ldB);*

Multiplica la pseudoinversa de la matriz diagonal D en la matriz sA factorizada ldl generalizada por la matriz B y lo almacena en C. C puede ser la misma matriz que B.

C: matriz resultado

ldC: dimension principal de C

sA: puntero a la matriz simétrica factorizada con ldlgen

ldsA: dimensión fundamental de la matriz

rsA: rango de la matriz simétrica

B: matriz a multiplicar

cB: numero de columnas de B

ldB: Dimension pricipal de B

*void vectorldlgenpinvDsAv(double *u, double *sA, unsigned long esA, unsigned long ldsA, unsigned long rsA, double *v);*

Multiplica la pseudoinversa de la matriz diagonal D en la matriz sA factorizada ldl generalizada por el vector v y lo almacena en el vector u. u puede ser el mismo que v

u: puntero al vector resultado. puede ser igual a v

sA: puntero a la matriz simétrica

esA: dimensión de la matriz (numero de filas/columnas)

ldsA: dimensión fundamental de A

rsA: rango de la matriz simétrica

v: puntero al vector v