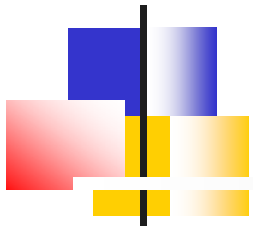


Programación Paralela y Computación de Altas Prestaciones Algoritmos Matriciales Básicos

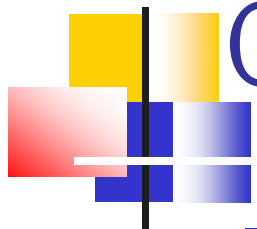


Javier Cuenca

Dpto. de Ingeniería y Tecnología de Computadores



Universidad de Murcia

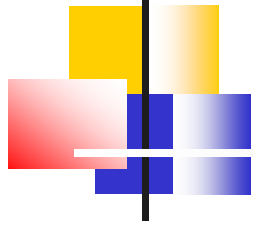


Contenido

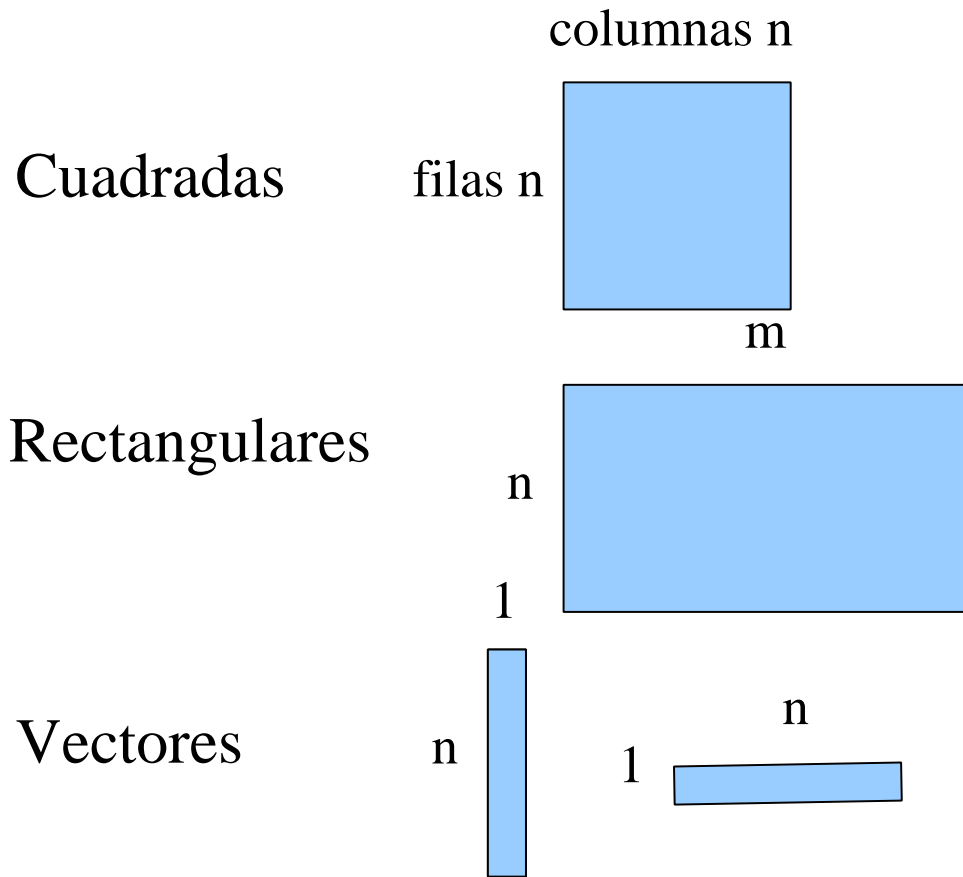
- Tipos de matrices
- Operaciones básicas con vectores
- Operaciones básicas con matrices
- Multiplicación de matrices
- Factorización LU
- Operaciones con matrices dispersas

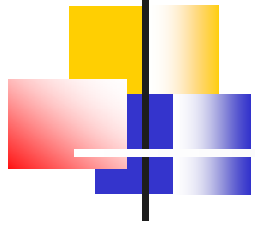
Códigos de los ejemplos en `luna.inf.um.es`:

`/home/javiercm/ejemplos_algmatbas`



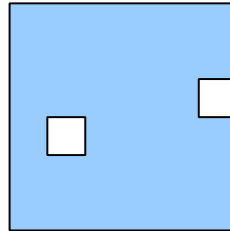
Tipos de matrices





Tipos de matrices

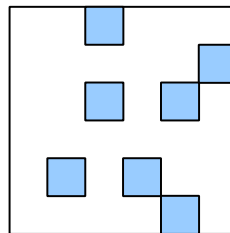
Densas



muchos elementos distintos de cero

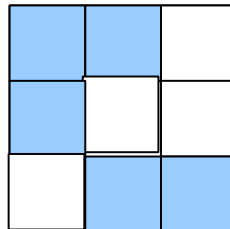
Dispersas

(escasas, vacías, ...)

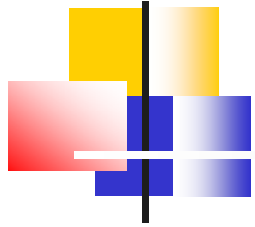


pocos elementos distintos de cero

Por bloques

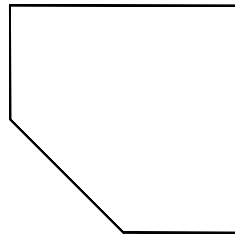


bloques nulos y bloques densos

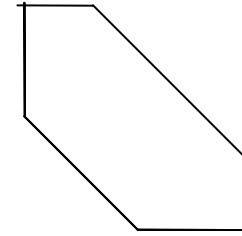


Tipos de matrices

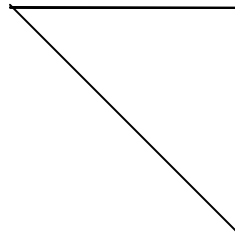
Trapezoidales



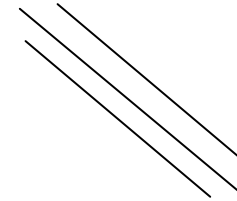
Banda



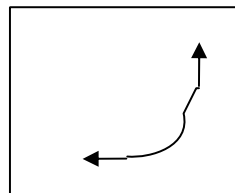
Triangulares



Tridiagonal

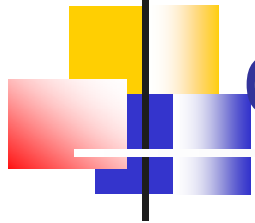


Simétrica



Cada elemento de la fila i columna j (a_{ij})
igual al de la columna i fila j (a_{ji})

Almacenamiento de matrices densas

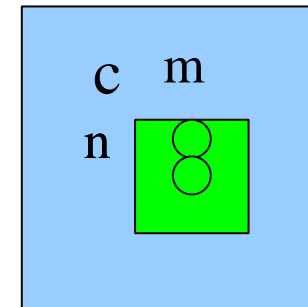


- En array bidimensional: `double a[n][m]`
- En array unidimensional: `double *b[n*m]`

- Fila i columna j :

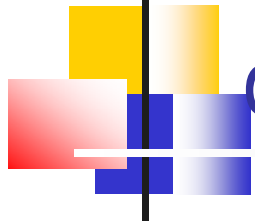
$a[i][j]$, $b[i*m+j]$

ld



- Cuando es submatriz de otra
 - "Leading dimension": posiciones de memoria entre dos elementos consecutivos de la misma columna
 - Fila i columna j : $c[i*ld+j]$

Almacenamiento de matrices dispersas



- Muchas maneras distintas

	0	1	2	3
0			1	
1	3			
2		2		4
3				

- Tres arrays:

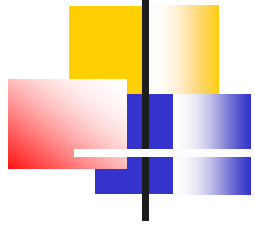
`datos[0,...,d-1]`, `filas[0,...,d-1]`, `columnas[0,...,d-1]`

`datos (1,3,2,4)`, `filas (0,1,2,2)`, `columnas (2,0,1,3)`

- **CRS**: un entero: n° filas y Tres arrays:

`NF`, `datos[0,...,d-1]`, `columnas[0,...,d-1]`, `comienzo fila[0,...,n-1]`

`NF=4`, `datos (1,3,2,4)`, `columnas (2,0,1,3)`, `com. filas (0,1,2,-1)`



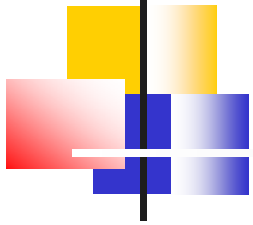
Operaciones básicas con vectores

```
void escalar_vector (double d, double *v, int n)
{
    int i;

    for (i=0; i<n; i++)
        v[i] *= d;
}
```

← n flops (operaciones en coma flotante)

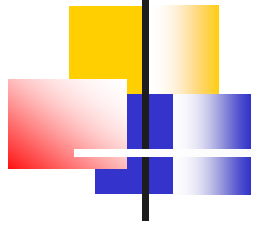
Operaciones básicas con vectores



```
double sumar_vector (double *v, int n)
{
    int i;
    double s=0.;

    for(i=0; i<n; i++)
        s+=v[i]; ← n flops

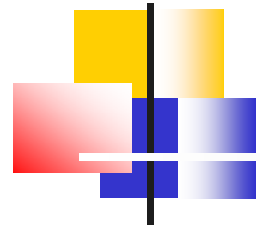
    return s;
}
```



Operaciones básicas con vectores

```
void sumar_vectores (double *v1, double *v2,  
                     double *vr, int n)  
{  
    int i;  
  
    for(i=0; i<n; i++)  
        vr[i]=v1[i]+v2[i];  
}
```

← n flops

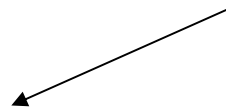


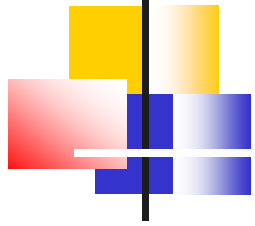
Operaciones básicas con vectores

```
double producto_escalar(double *v1, double
    *v2, int n)
{
    int i;
    double p=0.;

    for(i=0; i<n; i++)
        p+=v1[i]*v2[i];
    return p;
}
```

2n flops



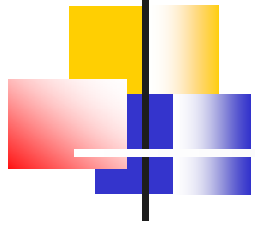


Operaciones básicas con matrices

```
void escalar_matriz (double d, double **a, int
    n, int m)
{
    int i, j;

    for(i=0; i<n; i++)
        for(j=0; j<m; j++)
            a[i][j] = a[i][j] * d;
}
```

nm, n^2 flops
←



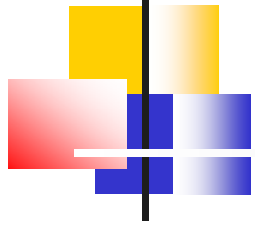
Operaciones básicas con matrices

Uso de "stride" de un vector

```
void producto_escalar_stride (double *v1,int str1,  
    double *v2,int str2,int n)  
{  
    int i;  
    double p=0.;  
    for(i=0;i<n;i++)  
        p+=v1[i*str1]*v2[i*str2];  
    return p;  
}
```

2n flops

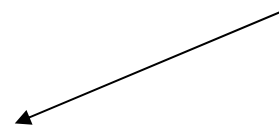




Operaciones básicas con matrices

```
void sumar_matrices (double *a,int fa,int ca,int lda,  
    double *b,int fb,int cb,int ldb, double *c,int  
    fc,int cc,int ldc)  
{  
    int i,j;  
  
    for(i=0;i<fa;i++)  
        for(j=0;j<ca;j++)  
            c[i*ldc+j]=a[i*lda+j]+b[i*ldb+j];  
}
```

n^2 flops



Operaciones básicas con matrices



```
void matriz_vector (double *m,int fm,int cm,int ldm,  
                    double *v,int fv,int strv,double *r,int  
                    fr,int strr)  
{ int i,j;  
  double s;  
  for(i=0;i<fm;i++)  
  {  
    s=0.;  
    for(j=0;j<cm;j++)  
      s+=m[i*ldm+j]*v[j*strv];  
  
    r[i*strr]=s;  
  }  
}
```

$2n^2$ flops

$2n$ flops

Operaciones básicas con matrices



```
void matriz_vector_pe (double *m,int fm,int cm,int  
    ldm,double *v,int fv,int strv,double *r,int fr,int strr)  
{  
    int i,j;  
    double s;  
  
    for(i=0;i<fm;i++)  
    {  
        r[i*strr]=producto_escalar_stride(&m[i*ldm],1,v,strv,cm);  
    }  
}
```

$2n^2$ flops

$2n$ flops


```

void producto_escalar_stride (double *v1,int str1,double *v2,int str2,int
    n)
{
    int i;
    double p=0.;
    for(i=0;i<n;i++)
        p+=v1[i*str1]*v2[i*str2];
    return p;
}

```

```

int i,j;
double s;

```

```

for(i=0;i<fm;i++)
{
    r[i*strr]=producto_escalar_stride(&m[i*ldm],1,v,strv,cm);
}
}

```

$2n^2$ flops

$2n$ flops

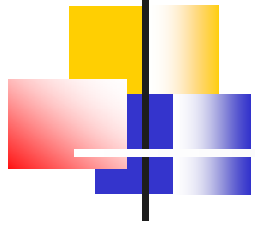
Operaciones básicas con matrices



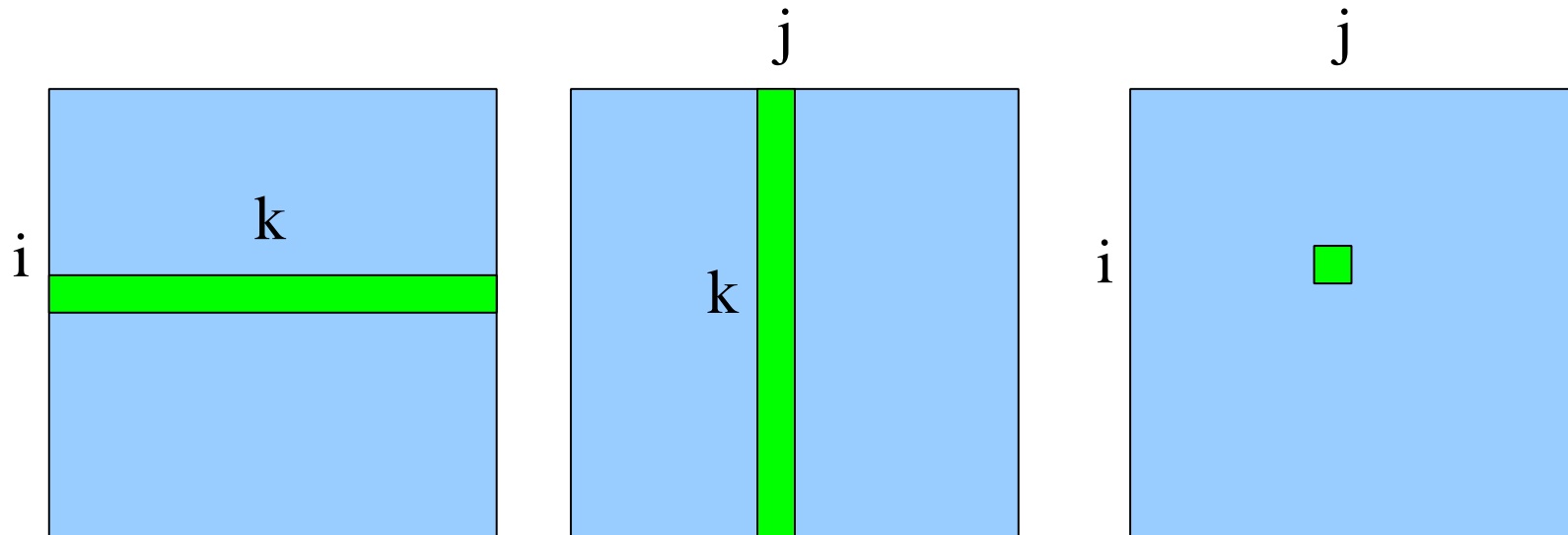
```
void trasponer_matriz (double *m,int n,int ld)
{
    int i,j;
    double t;
    for(i=0;i<n;i++)
        for(j=i+1;j<n;j++)
        {
            t=m[i*ld+j];
            m[i*ld+j]=m[j*ld+i];
            m[j*ld+i]=t;
        }
}
```

$3n(n-1)/2$ flops

$3(n-i-1)$ flops



Multiplicación de matrices





Multiplicación de matrices

```
void matriz_matriz (double **a,int fa,int ca,double  
    **b,int fb,int cb,double **c,int fc,int cc)
```

```
{ int i,j,k; double s;
```

```
    for(i=0;i<fa;i++)
```

$2n^3$ flops

```
        for(j=0;j<cb;j++)
```

$2n^2$ flops

```
        {
```

```
            s=0.;
```

```
            for(k=0;k<ca;k++)
```

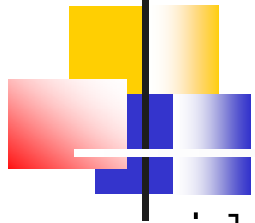
$2n$ flops

```
                s+=a[i][k]*b[k][j];
```

```
            c[i][j]=s;
```

```
        }
```

```
}
```



Multiplicación de matrices

```
void matriz_matriz_ld (double *a,int fa,int ca,int  
    lda, double *b,int fb,int cb,int ldb,double *c,int  
    fc,int cc,int ldc)
```

```
{ int i,j,k;
```

```
double s;
```

```
for(i=0;i<fa;i++)
```

$2n^3$ flops

```
    for(j=0;j<cb;j++)
```

```
    {
```

```
        s=0.;
```

```
        for(k=0;k<ca;k++)
```

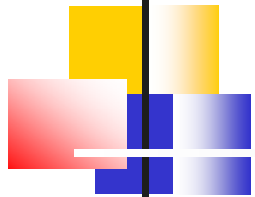
```
            s+=a[i*lda+k]*b[k*ldb+j];
```

```
        c[i*ldc+j]=s;
```

```
    }  
}
```

$2n^2$ flops

$2n$ flops

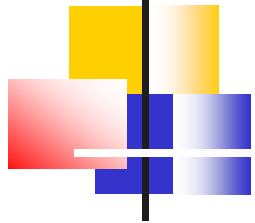


Multiplicación de matrices

```
void matriz_matriz_tras (double *a,int fa,int ca,int lda,double *b,int
    fb,int cb,int ldb,double *c,int fc,int cc,int ldc)
{ int i,j,k; double s; double *bt,*da,*db;

    bt=(double *) malloc(sizeof(double)*cb*fb);
    trasponer_matriz_esp(b,fb,cb,ldb,bt,cb,fb,fb);

    for(i=0;i<fa;i++) ← 2n³ flops
        for(j=0;j<cb;j++) ← 2n² flops
        {
            s=0.;    da=&a[i*lda]; db=&bt[j*fb];
            for(k=0;k<ca;k++,da++,db++) ← 2n flops
                s+=da[0]*db[0];
            c[i*ldc+j]=s;
        }
    free(bt);
}
```



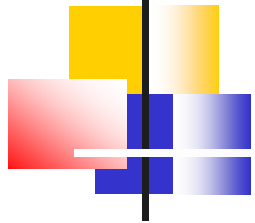
Multiplicación de matrices

```
void matriz_matriz_pe(double *a,int fa,int ca,int lda,  
    double *b,int fb,int cb,int ldb,double *c,int fc,int  
    cc,int ldc)  
{  
    int i,j;  
  
    for(i=0;i<fa;i++)  
        for(j=0;j<cb;j++)  
            c[i*ldc+j]=producto_escalar_stride(&a[i*lda],1,  
                &b[j],ldb,ca);  
}
```

$2n^3$ flops

$2n^2$ flops

$2n$ flops



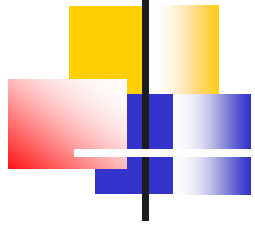
Multiplicación de matrices

```
void matriz_matriz_pe(double *a,int fa,int ca,int lda,  
    double *b,int fb,int cb,int ldb,double *c,int fc,int  
    cc,int ldc)  
{  
    int i,j;  
    for(i=0;i<fa;i++)  
        for(j=0;j<cb;j++)  
            c[i*ldc+j]=producto_escalar_stride(&a[i*lda],1,  
                &b[j],ldb,ca);  
}
```

$2n^3$ flops

$2n^2$ flops

$2n$ flops

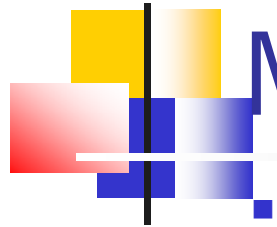


Multiplicación de matrices

```
void matriz_matriz_mv(double *a,int fa,int ca,int  
    lda,double *b,int fb,int cb,int ldb,double *c,int  
    fc,int cc,int ldc)  
{  
    int i;  
  
    for(i=0;i<cb;i++)  
        matriz_vector_pe(a,fa,ca,lda,&b[i],fb,ldb,&c[i],fc,ldc);  
}
```

$2n^3$ flops

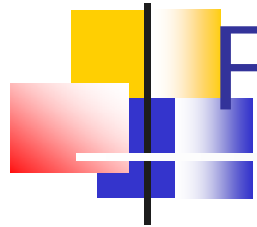
$2n^2$ flops



Multiplicación de matrices

Ejemplo de tiempo ejecución en un PC (segundos):

Método\tamaño	1000	1200	1400
bidimensional	12.26	20.15	32.37
leading dimension	12.70	21.95	36.41
leading+punteros	12.29	22.84	34.90
traspuesta	12.71	20.88	36.29
producto escalar	12.92	21.75	35.17
matriz-vector	12.19	21.47	36.92



Factorización LU

$$\begin{array}{|c|c|} \hline A_{00} & A_{01} \\ \hline A_{10} & A_{11} \\ \hline \end{array} = \begin{array}{|c|c|} \hline L_{00} & 0 \\ \hline L_{10} & L_{11} \\ \hline \end{array} \times \begin{array}{|c|c|} \hline U_{00} & U_{01} \\ \hline 0 & U_{11} \\ \hline \end{array}$$

$$U_{ii} = 1$$

$$\text{Paso 1: } L_{00} U_{00} = A_{00}$$

$$\rightarrow L_{00} = A_{00}$$

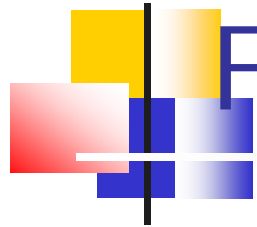
$$\text{Paso 2: } L_{00} U_{0i} = A_{0i}$$

$$\rightarrow U_{0i} = A_{0i} / L_{00}$$

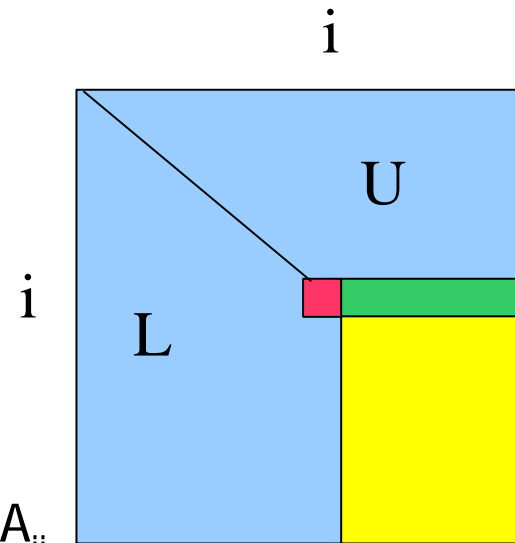
$$L_{i0} = A_{i0}$$

$$\text{Paso 3: } A_{ij} = L_{i0} U_{0j} + \dots$$

$$\rightarrow A'_{ij} = A_{ij} - L_{i0} U_{0j}$$



Factorización LU



$$U_{ii}=1$$

Paso 1: $L_{ii} U_{ii} = A_{ii}$

$$\rightarrow L_{ii} = A_{ii}$$

Paso 2: $L_{ij} U_{ij} = A_{ij}$

$$\rightarrow U_{ij} = A_{ij} / L_{ii}$$

$$L_{ji} = A_{ji}$$

Paso 3: $A_{jk} = L_{ji} U_{ik} + \dots \rightarrow A'_{jk} = A_{jk} - L_{ji} U_{ik}$



Factorización LU

```
void lu (double *a,int fa,int ca,int lda)
{ int i,j,k;

  for(i=0;i<fa;i++)
  {
    for(j=i+1;j<ca;j++)    //Paso 2
      a[i*lda+j]/=a[i*lda+i];

    for(j=i+1;j<fa;j++)    //Paso 3
      for(k=i+1;k<ca;k++)
        a[j*lda+k]-=a[j*lda+i]*a[i*lda+k];
  }
}
```



Factorización LU

- Para resolver sistema $Ax=b$

En la forma: $LUx=b$

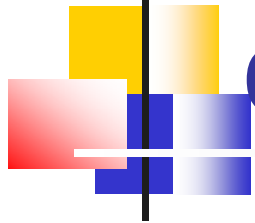
Resolver: $Ly=b$

(Sistema triangular inferior: sustitución progresiva)

Seguido de: $Ux=y$

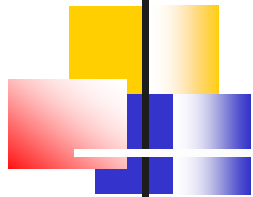
(Sistema triangular superior: sustitución regresiva)

Operaciones con matrices dispersas



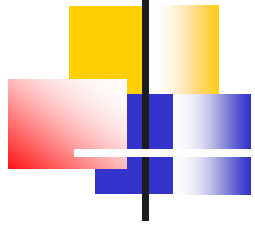
- Producto matriz-vector
 - Por cada elemento de la matriz
 - Buscamos si hay elemento de la misma columna en el vector y acumulamos sobre la suma correspondiente a su fila

datos	3 1 2 4	datos	1 3	resultado
fila	0 1 2 2	fila	0 2	datos 9 1 6
columna	2 0 2 3			fila 0 1 2



Operaciones con matrices dispersas

```
void matriz_vector_disperso(double *dm,int *fm,int *cm,
    int ndm,double *dv,int *fv,int ndv,double *dr,int *fr,int ndr)
{ int i,j,fact; double s;
  for(i=0;i<ndr;i++)
    fr[i]=i;
  i=0;
  while(i<ndm)
  {
    fact=fm[i]; j=0; s=0.;
    while(i<ndm && fm[i]==fact) // recorrer la fila fact
    {
      while(j<ndv && fv[j]<cm[i]) j++; // buscar en vector
      if(j<ndv && fv[j]==cm[i]) // si tenemos fil(vec) == col(mat)
        s+=dm[i]*dv[j];          // operamos
      j++;
    }
    dr[fact]=s;
  }
}
```

Trabajo alumnos.

- Conectarse a `luna.inf.um.es`
- Copiar a tu directorio los ejemplos que están en:
`/home/javiercm/ejemplos_algmatbas`
- Probar los programas de las sesiones y corregir errores