

Curso de Métodos Numéricos

DEMAT, Universidad de Guanajuato

Clase 7: Métodos directos de solución de ecuaciones lineales

- Factorización LU.
- Método de Doolittle.
- Cálculo de la inversa y del determinante.
- Lectura y escritura de archivos binarios.
- Factorización LU mediante la librería GSL.

MAT-251

Dr. Joaquín Peña Acevedo
CIMAT A.C.

e-mail: joaquin@ciimat.mx

Elim. Gaussiana mediante producto de matrices (I)

Sea \mathbf{A} la matriz

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix},$$

y definamos

$$\mathbf{L}_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -l_{21} & 1 & 0 & 0 \\ -l_{31} & 0 & 1 & 0 \\ -l_{41} & 0 & 0 & 1 \end{bmatrix}, \quad l_{i1} = \frac{a_{i1}}{a_{11}}, \quad i = 2, 3, 4.$$

Entonces

$$\mathbf{L}_1 \mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ 0 & a'_{22} & a'_{23} & a'_{24} \\ 0 & a'_{32} & a'_{33} & a'_{34} \\ 0 & a'_{42} & a'_{43} & a'_{44} \end{bmatrix}$$

Elim. Gaussiana mediante producto de matrices (II)

Si definimos

$$\mathbf{L}_2 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & -l_{32} & 1 & 0 \\ 0 & -l_{42} & 0 & 1 \end{bmatrix}, \quad l_{i2} = \frac{a'_{i2}}{a'_{22}}, \quad i = 3, 4.$$

entonces $\mathbf{L}_2 \mathbf{L}_1 \mathbf{A} =$

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ 0 & a'_{22} & a'_{23} & a'_{24} \\ 0 & 0 & a''_{33} & a''_{34} \\ 0 & 0 & a''_{43} & a''_{44} \end{bmatrix}$$

$$\mathbf{L}_3 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -l_{43} & 1 \end{bmatrix} \quad \Rightarrow \quad \mathbf{L}_3 \mathbf{L}_2 \mathbf{L}_1 \mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ 0 & a'_{22} & a'_{23} & a'_{24} \\ 0 & 0 & a''_{33} & a''_{34} \\ 0 & 0 & 0 & a'''_{44} \end{bmatrix}$$

con $l_{43} = a''_{43}/a''_{33}$.

Elim. Gaussiana mediante producto de matrices (III)

- A las matrices \mathbf{L}_i se les llama matrices triangulares inferiores elementales.
- El proceso de eliminación Gaussiana es una reducción a una forma triangular por medio de matrices triangulares inferiores elementales.

En general, tenemos que

$$\mathbf{L}_{n-1} \cdots \mathbf{L}_2 \mathbf{L}_1 \mathbf{A} = \mathbf{U}$$

donde \mathbf{U} es una matriz triangular superior.

Factorización LU

- Como los elementos de la diagonal de cada matriz \mathbf{L}_i es diferente de cero, \mathbf{L}_i es no singular.

- Como los elementos de la diagonal de cada matriz \mathbf{L}_i es diferente de cero, \mathbf{L}_i es no singular.
- El producto $\mathbf{L}_{n-1} \cdots \mathbf{L}_2 \mathbf{L}_1$ es no singular, de modo que

$$\mathbf{L}_{n-1} \cdots \mathbf{L}_2 \mathbf{L}_1 = \mathbf{L}^{-1}$$

Factorización LU

- Como los elementos de la diagonal de cada matriz \mathbf{L}_i es diferente de cero, \mathbf{L}_i es no singular.
- El producto $\mathbf{L}_{n-1} \cdots \mathbf{L}_2 \mathbf{L}_1$ es no singular, de modo que

$$\mathbf{L}_{n-1} \cdots \mathbf{L}_2 \mathbf{L}_1 = \mathbf{L}^{-1}$$

- Tenemos entonces que

$$\mathbf{L}^{-1} \mathbf{A} = \mathbf{U} \quad \Rightarrow \quad \mathbf{A} = \mathbf{L} \mathbf{U}$$

Factorización LU

- Como los elementos de la diagonal de cada matriz \mathbf{L}_i es diferente de cero, \mathbf{L}_i es no singular.
- El producto $\mathbf{L}_{n-1} \cdots \mathbf{L}_2 \mathbf{L}_1$ es no singular, de modo que

$$\mathbf{L}_{n-1} \cdots \mathbf{L}_2 \mathbf{L}_1 = \mathbf{L}^{-1}$$

- Tenemos entonces que

$$\mathbf{L}^{-1} \mathbf{A} = \mathbf{U} \quad \Rightarrow \quad \mathbf{A} = \mathbf{L} \mathbf{U}$$

- La matriz \mathbf{L} es triangular inferior,

$$\mathbf{L} = \begin{bmatrix} 1 & 0 & \cdots & 0 & 0 \\ l_{21} & 1 & \cdots & 0 & 0 \\ l_{31} & l_{32} & \cdots & 0 & 0 \\ \vdots & \vdots & \cdots & \vdots & \vdots \\ l_{n-1,1} & l_{n-1,2} & \cdots & 1 & 0 \\ l_{n1} & l_{n2} & \cdots & l_{n,n-1} & 1 \end{bmatrix}$$

Solución del sistema lineal de ecuaciones

De esta forma, el método de eliminación Gaussiana calcula la descomposición LU de la matriz.

Supongamos que \mathbf{A} tiene una factorización LU . Entonces para resolver

$$\mathbf{Ax} = \mathbf{b}$$

hacemos lo siguiente. Tenemos que

$$\mathbf{LUx} = \mathbf{b}$$

Definimos $\mathbf{y} = \mathbf{Ux}$ y entonces resolvemos

Solución del sistema lineal de ecuaciones

De esta forma, el método de eliminación Gaussiana calcula la descomposición LU de la matriz.

Supongamos que \mathbf{A} tiene una factorización LU . Entonces para resolver

$$\mathbf{Ax} = \mathbf{b}$$

hacemos lo siguiente. Tenemos que

$$\mathbf{LUx} = \mathbf{b}$$

Definimos $\mathbf{y} = \mathbf{Ux}$ y entonces resolvemos

$$\mathbf{Ly} = \mathbf{b} \quad (\text{usando sustitución hacia adelante})$$

$$\mathbf{Ux} = \mathbf{y} \quad (\text{usando sustitución hacia atrás})$$

Sobre la unicidad de la factorización LU

- Hay que notar que si \mathbf{D} es una matriz diagonal no singular, entonces

$$\mathbf{A} = \mathbf{L}\mathbf{U} = \mathbf{L}\mathbf{D}\mathbf{D}^{-1}\mathbf{U} = (\mathbf{L}\mathbf{D})(\mathbf{D}^{-1}\mathbf{U})$$

Sobre la unicidad de la factorización LU

- Hay que notar que si \mathbf{D} es una matriz diagonal no singular, entonces

$$\mathbf{A} = \mathbf{L}\mathbf{U} = \mathbf{L}\mathbf{D}\mathbf{D}^{-1}\mathbf{U} = (\mathbf{L}\mathbf{D})(\mathbf{D}^{-1}\mathbf{U})$$

$\mathbf{L}\mathbf{D}$ es triangular inferior y $\mathbf{D}^{-1}\mathbf{U}$ es triangular superior, por lo que podemos tener varias descomposiciones.

- Hay que notar que si \mathbf{D} es una matriz diagonal no singular, entonces

$$\mathbf{A} = \mathbf{L}\mathbf{U} = \mathbf{L}\mathbf{D}\mathbf{D}^{-1}\mathbf{U} = (\mathbf{L}\mathbf{D})(\mathbf{D}^{-1}\mathbf{U})$$

$\mathbf{L}\mathbf{D}$ es triangular inferior y $\mathbf{D}^{-1}\mathbf{U}$ es triangular superior, por lo que podemos tener varias descomposiciones.

- Hacemos el convenio de que por factorización LU nos referimos a la factorización en la que la matriz \mathbf{L} es triangular inferior y que los elementos de su diagonal son iguales a 1.

Sobre la unicidad de la factorización LU

- Hay que notar que si \mathbf{D} es una matriz diagonal no singular, entonces

$$\mathbf{A} = \mathbf{LU} = \mathbf{LDD}^{-1}\mathbf{U} = (\mathbf{LD})(\mathbf{D}^{-1}\mathbf{U})$$

\mathbf{LD} es triangular inferior y $\mathbf{D}^{-1}\mathbf{U}$ es triangular superior, por lo que podemos tener varias descomposiciones.

- Hacemos el convenio de que por factorización LU nos referimos a la factorización en la que la matriz \mathbf{L} es triangular inferior y que los elementos de su diagonal son iguales a 1.

Proposición.

Si \mathbf{A} es no singular y tiene una factorización LU, entonces la factorización es única.

Existencia de la factorización LU

Aun cuando **A** sea no singular, la matriz podría no tener una factorización LU. Por ejemplo,

$$\mathbf{A} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

Existencia de la factorización LU

Aun cuando \mathbf{A} sea no singular, la matriz podría no tener una factorización LU. Por ejemplo,

$$\mathbf{A} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

Si la matriz \mathbf{A} es singular y tiene una factorización LU, ésta puede no ser única. Por ejemplo,

$$\mathbf{A} = \begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ \lambda & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 0 & 1 - \lambda \end{bmatrix}$$

es una factorización LU para cualquier λ .

Método de Doolittle (I)

Tenemos que si $\mathbf{A} = \mathbf{L}\mathbf{U}$, entonces

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nn} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ l_{21} & 1 & 0 & \cdots & 0 \\ l_{31} & l_{32} & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ l_{n1} & l_{n2} & l_{n3} & \cdots & 1 \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & u_{13} & \cdots & u_{1n} \\ 0 & u_{22} & u_{23} & \cdots & u_{2n} \\ 0 & 0 & u_{33} & \cdots & u_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & u_{nn} \end{bmatrix}$$

Para que se cumpla la igualdad se debe cumplir

$$a_{ij} = \begin{cases} \sum_{k=1}^{i-1} l_{ik} u_{kj} + u_{ij} & i \leq j \\ \sum_{k=1}^j l_{ik} u_{kj} & i > j \end{cases}$$

Hacemos el convenio de que si $i = 1$ la suma en el primer caso es cero.

Método de Doolittle (II)

Fijamos $j = 1$. Entonces para $i = 1$,

$$u_{11} = a_{11},$$

y para $i = 2, 3, \dots, n$ tenemos que

$$a_{i1} = l_{i1} u_{11} \longrightarrow l_{i1} = \frac{a_{i1}}{u_{11}}$$

Fijamos $j = 2$. Entonces

$$u_{12} = a_{12}, \quad u_{22} = a_{22} - l_{21} u_{12}$$

y para $i = 3, 4, \dots, n$,

$$a_{i2} = l_{i1} u_{12} + l_{i2} u_{22} \longrightarrow l_{i2} = \frac{a_{i2} - l_{i1} u_{12}}{u_{22}}$$

Podemos continuar de esta manera para $j = 3, \dots, n$. En general, se tiene que

Método de Doolittle (III)

Para $j = 1, 2, \dots, n$, hacemos los siguientes dos pasos

- Para $i = 1, 2, \dots, j$ calculamos

$$u_{ij} = a_{ij} - \sum_{k=1}^{i-1} l_{ik} u_{kj}.$$

- Para $i = j + 1, j + 2, \dots, n$, calculamos

$$l_{ij} = \frac{1}{u_{jj}} \left(a_{ij} - \sum_{k=1}^{j-1} l_{ik} u_{kj} \right)$$

A este procedimiento se le conoce como el *método de Doolittle*.

- Otras factorizaciones parecidas a LU se obtienen con el método de Crout, en el que la matriz triangular superior tiene 1's en la diagonal y la matriz triangular inferior no le impone alguna condición.

- Otras factorizaciones parecidas a LU se obtienen con el método de Crout, en el que la matriz triangular superior tiene 1's en la diagonal y la matriz triangular inferior no le impone alguna condición.
- En caso de que matriz **A** requiera pivoteo parcial, entonces podemos encontrar una matriz de permutación **P** tal que

$$\mathbf{PA} = \mathbf{LU}.$$

Observaciones sobre la factorización LU

- Para resolver el sistema $\mathbf{Ax} = \mathbf{b}$ hay que considerar realizar pivoteo parcial:

$$\mathbf{PAx} = \mathbf{Pb} \implies \mathbf{LUx} = \mathbf{Pb}$$

Observaciones sobre la factorización LU

- Para resolver el sistema $\mathbf{Ax} = \mathbf{b}$ hay que considerar realizar pivoteo parcial:

$$\mathbf{PAx} = \mathbf{Pb} \implies \mathbf{LUx} = \mathbf{Pb}$$

- Para pivoteo total, entonces

$$\mathbf{PAQ} = \mathbf{LU}.$$

donde \mathbf{P} y \mathbf{Q} son matrices de permutación.

Sobre el almacenamiento de las matrices L y U

- Para la factorización de Doolittle tenemos que calcular n^2 valores

$\frac{n(n+1)}{2}$ valores para u_{ij} con $i \leq j$

$\frac{n(n+1)}{2} - n$ valores para l_{ij} con $i > j$

Sobre el almacenamiento de las matrices L y U

- Para la factorización de Doolittle tenemos que calcular n^2 valores
 $\frac{n(n+1)}{2}$ valores para u_{ij} con $i \leq j$
 $\frac{n(n+1)}{2} - n$ valores para l_{ij} con $i > j$
- Fijo j , en el algoritmo de Doolittle calculamos u_{ij} y l_{ij} , y no se vuelve a requerir los valores de a_{ij} .

Sobre el almacenamiento de las matrices L y U

- Para la factorización de Doolittle tenemos que calcular n^2 valores $\frac{n(n+1)}{2}$ valores para u_{ij} con $i \leq j$
 $\frac{n(n+1)}{2} - n$ valores para l_{ij} con $i > j$
- Fijo j , en el algoritmo de Doolittle calculamos u_{ij} y l_{ij} , y no se vuelve a requerir los valores de a_{ij} .
- Se pueden almacenar los valores u_{ij} y l_{ij} en la matriz del sistema \mathbf{A} .

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nn} \end{bmatrix} \longrightarrow \begin{bmatrix} u_{11} & u_{12} & u_{13} & \cdots & u_{1n} \\ l_{21} & u_{22} & u_{23} & \cdots & u_{2n} \\ l_{31} & l_{32} & u_{33} & \cdots & u_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ l_{n1} & l_{n2} & l_{n3} & \cdots & u_{nn} \end{bmatrix}$$

Sobre el almacenamiento de las matrices L y U

- Para la factorización de Doolittle tenemos que calcular n^2 valores $\frac{n(n+1)}{2}$ valores para u_{ij} con $i \leq j$
 $\frac{n(n+1)}{2} - n$ valores para l_{ij} con $i > j$
- Fijo j , en el algoritmo de Doolittle calculamos u_{ij} y l_{ij} , y no se vuelve a requerir los valores de a_{ij} .
- Se pueden almacenar los valores u_{ij} y l_{ij} en la matriz del sistema \mathbf{A} .
- Esto destruye la matriz \mathbf{A} .

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nn} \end{bmatrix} \longrightarrow \begin{bmatrix} u_{11} & u_{12} & u_{13} & \cdots & u_{1n} \\ l_{21} & u_{22} & u_{23} & \cdots & u_{2n} \\ l_{31} & l_{32} & u_{33} & \cdots & u_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ l_{n1} & l_{n2} & l_{n3} & \cdots & u_{nn} \end{bmatrix}$$

Almacenamiento en memoria de la matrices

Supongamos que tenemos una matriz $m \times n$,

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}$$

- Queremos reservar memoria para almacenar las entradas de la matriz, de manera que sea compatible con la forma en que la librería GSL almacena la información.
- También queremos que se pueda acceder mediante un arreglo bidimensional, para facilitar la forma en que accedan a las entradas de la matriz.

$$a_{ij} \quad \Leftrightarrow \quad a[i][j]$$

Cálculo de la inversa de la matriz

Supongamos que tenemos una matriz \mathbf{A} de tamaño n y su factorización LU:

$$\mathbf{A} = \mathbf{L}\mathbf{U}.$$

Sea \mathbf{x}_i el vector solución del sistema lineal

$$\mathbf{A}\mathbf{x}_i = \mathbf{L}\mathbf{U}\mathbf{x}_i = \mathbf{e}_i.$$

para $i = 1, \dots, n$, donde \mathbf{e}_i es el i 'ésimo vector canónico. Si $\mathbf{X} = [\mathbf{x}_1 \cdots \mathbf{x}_n]$, entonces

$$\mathbf{A}\mathbf{X} = [\mathbf{A}\mathbf{x}_1 \cdots \mathbf{A}\mathbf{x}_n] = [\mathbf{e}_1 \cdots \mathbf{e}_n] = \mathbf{I}.$$

Esto es, \mathbf{X} es la inversa de \mathbf{A} . Así, si tenemos la factorización LU de matriz, solo tenemos que resolver n sistemas de ecuaciones lineales para obtener las columnas de la matriz inversa.

Cálculo del determinante de la matriz

Supongamos que tenemos una matriz \mathbf{A} de tamaño n y su factorización LU:

$$\mathbf{A} = \mathbf{L}\mathbf{U}.$$

Entonces por propiedades del determinante

$$\det \mathbf{A} = \det(\mathbf{L}\mathbf{U}) = \det \mathbf{L} \det \mathbf{U}.$$

Como las matrices \mathbf{L} y \mathbf{U} son triangulares, su determinante es igual al producto de los elementos de su diagonal. Además, para la factorización de Doolittle, se debe tener que $\det \mathbf{L} = 1$, por lo que

$$\det \mathbf{A} = \det \mathbf{U} = \prod_{i=1}^n u_{ii}.$$

- Para probar los algoritmos que operan con matrices y vectores conviene leer esta información de archivos en vez de tener que capturarla manualmente.
- Para no introducir más errores y usar menos espacio en disco, conviene usar archivos binarios en la que las entradas de vectores y matrices son del tipo double.
- Para que todos puedan leer la información de los archivos, hay que establecer un formato.

Almacenamiento en memoria de matrices

Supongamos que tenemos una matriz $m \times n$,

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}$$

- Queremos reservar memoria para almacenar las entradas de la matriz, de manera que sea compatible con la forma en que la librería GSL almacena la información.
- También queremos que se pueda acceder mediante un arreglo bidimensional, para facilitar la forma en que accedan a las entradas de la matriz.

$$a_{ij} \quad \Leftrightarrow \quad a[i][j]$$

Convención para almacenamiento de datos

Para matrices:

- El archivo debe comenzar con dos enteros, m y n . El primero, m , debe indicar el número de filas de la matriz, y el segundo, n , debe indicar el número de columnas.
- El resto del archivo debe contener mn valores, que corresponden a las entradas de la matriz, almacenadas por filas: los primeros n valores corresponden a la primer fila, los siguientes n valores corresponden a la segunda fila, etc.

Para vectores:

- El archivo debe comenzar con un entero, m , que indica la dimensión del vector.
- El resto del archivo debe contener m valores, que corresponden a las entradas del vector.

Creación de la matriz a partir de un archivo

Para conseguir lo anterior, el procedimiento sería el siguiente:

- ➊ Abrir el archivo y leer las dimensiones de la matriz.
- ➋ Creamos de manera dinámica un arreglo de apuntadores, tantos como filas tenga la matriz.
- ➌ Creamos un bloque de memoria de tamaño mn para almacenar todas las entradas.
- ➍ Hacemos que el primer apuntador apunte a esta posición de memoria.
- ➎ Hacemos que el resto de los apuntadores apunten a las posiciones del bloque de memoria reservada en donde empiezan cada fila de la matriz.
- ➏ Leemos los elementos de la matriz del archivo.

Los pasos 2–5 son realizados por la función *createMatrix()*, vista en la sección anterior. De modo que solo se explican el resto de los pasos.

Paso 1: Lectura de las dimensiones de la matriz

La función que lee los datos de un archivo binario es:

```
double **readMatrix(char *cfile, int *nr, int *nc) {
    double **mat;
    FILE *f1 = fopen(cfile, "rb");

    if(!f1) return(NULL);
    // Lectura del tamaño de la matriz
    fread(nr, sizeof(int), 1, f1);
    fread(nc, sizeof(int), 1, f1);
    // Reservamos memoria
    mat = createMatrix(*nr, *nc);
    // Lectura de los datos
    fread(mat[0], sizeof(double), (*nr)*(*nc), f1);
    fclose(f1);
    return(mat);
}
```

La opción "rb" hace que se abra el archivo de nombre cfile para lectura y especifica que el archivo es binario.

Paso 6: Lectura de los elementos de la matriz

```
double **readMatrix(char *cfile, int *nr, int *nc) {  
    double **mat;  
    FILE *f1 = fopen(cfile, "rb");  
  
    if(!f1) return(NULL);  
    // Lectura del tamaño de la matriz  
    fread(nr, sizeof(int), 1, f1);  
    fread(nc, sizeof(int), 1, f1);  
    // Reservamos memoria  
    mat = createMatrix(*nr, *nc);  
    // Lectura de los datos  
    fread(mat[0], sizeof(double), (*nr)*(*nc), f1);  
    fclose(f1);  
    return(mat);  
}
```

Se lee todos los elementos de la matriz y los almacena en el bloque de memoria al que hace referencia `mat[0]`.

Lo que resta de la función es la parte que cierra el archivo y devuelve el apuntador `mat`.

La función siguiente escribe las entradas de una matriz en el formato acordado:

```
int writeMatrix(double **mat, int nr, int nc, char *cfile) {  
    FILE *f1 = fopen(cfile, "wb");  
  
    if(!f1) return(1);  
    fwrite(&nr, sizeof(int), 1, f1);  
    fwrite(&nc, sizeof(int), 1, f1);  
    fwrite(mat[0], sizeof(double), nr*nc, f1);  
    fclose(f1);  
    return(0);  
}
```

Lectura de vectores

```
double *readVector(char *cfile, int *nr) {  
    double *vec;  
    FILE *f1 = fopen(cfile, "rb");  
  
    if(!f1) return(NULL);  
    fread(nr, sizeof(int), 1, f1);  
    vec = (double *) malloc( (*nr)*sizeof(double));  
    if(vec==NULL) return(NULL);  
    fread(vec, sizeof(double), *nr, f1);  
    fclose(f1);  
    return(vec);  
}
```

Para usarla:

```
int nr;  
double *vec = readVector(nombre_archivo, &nr);
```

Ejemplo

Hay que compilar el programa `lecturaBinarios.c`.

En la línea de comandos se especifica el nombre del archivo de la matriz y el nombre del archivo de un vector (en ese orden). Ejemplo:

```
./lecturaBinarios matA1.bin vecb1.bin
```

Como el programa imprime el contenido de los archivos en la pantalla, hay que probarlo con matrices pequeñas.

Ejemplo

Hay que compilar el programa `lecturaBinarios.c`.

En la línea de comandos se especifica el nombre del archivo de la matriz y el nombre del archivo de un vector (en ese orden). Ejemplo:

```
./lecturaBinarios matA1.bin vecb1.bin
```

Como el programa imprime el contenido de los archivos en la pantalla, hay que probarlo con matrices pequeñas.

Conversión de archivos de texto a binario

- Para probar los programas, se les proporciona datos (arreglos 1D y 2D) en archivos binarios para no perder precisión.
- Para generar estos archivos binarios a partir de un archivo de texto, puede usar como referencia el código:

`vecTxt2Bin.c` Convertir un archivo de texto que contiene un arreglo 1D a un archivo binario

`matTxt2Bin.c` Convertir un archivo de texto que contiene un arreglo 2D a un archivo binario

- El código está desarrollado en C.
- Tiene funciones para resolver todos los temas que están descritos en el temario y más.
- En el manual de referencia vienen la descripción de las funciones y ejemplos de como usarlas.
- Es importante notar que dependiendo de las funciones que se quieran usar hay que incluir ciertos archivos de encabezado.

`gsl_vector` es una estructura con 5 componentes.

```
typedef struct {  
    size_t size;  
    size_t stride;  
    double *data;  
    gsl_block *block;  
    int owner;  
} gsl_vector;
```

El rango válido de índices es de 0 a $\text{size} - 1$. El apuntador `data` da la posición del primer elemento del arreglo.

```
gsl_vector *gsl_vector_alloc(size_t n);  
void gsl_vector_free(gsl_vector *v);  
double gsl_vector_get(gsl_vector *v, size_t i);  
void gsl_vector_set(gsl_vector *v, size_t i, double x);
```

`gsl_matrix` es una estructura con 6 componentes.

```
typedef struct {  
    size_t size1;  
    size_t size2;  
    size_t tda;  
    double *data;  
    gsl_block *block;  
    int owner;  
} gsl_matrix;
```

El número de filas es *size1* y el de columnas es *size2*. El apuntador *data* da la posición del primer elemento del arreglo. Los datos están almacenadas por filas.

Para crear y liberar memoria, se usan las funciones:

```
gsl_matrix *gsl_matrix_alloc(size_t n1, size_t n2);  
void gsl_matrix_free(gsl_matrix *m);
```

- El código *luGsl.c* muestra como usar la librería GSL para resolver un sistema de ecuaciones lineales usando factorización LU.
- La función *gsl_linalg_LU_decomp* que calcula la factorización contempla el intercambio de fila, de modo que además de la factorización devuelve la permutación.
- El código *luGsl2.c* lee la información de los archivos con las funciones vistas en la clase anterior y muestra como pasar esta información a las estructuras de GSL sin tener que duplicar la memoria.