Computación de Altas Prestaciones

Seminario 7

1

Contenido

1. Uso de las librerías BLAS y LAPACK desde Matlab.
2. El fichero dgesv_mex.c.
3. Compilación de ficheros mex con funciones de BLAS y LAPACK.
4. Ejercicios.
5. Descomposiciones LU y de Cholesky.
6. Matrices dispersas.
7. Ordenación de matrices.

DSIC
DEPARTAMENTO DE SISTEMAS INFORMÁTICOS Y COMPUTACIÓN

2

1. Uso de las librerías BLAS y LAPACK desde Matlab

- Matlab dispone de las librerías BLAS y LAPACK (versión MKL de Intel para diferentes S.O.).
- Para resolver un sistema de ecuaciones, como el de la función membrana, escribiremos las llamadas a las funciones de LAPACK correspondientes:
  - DGESV: Resuelve un sistema de ecuaciones lineales.
  - DGETRF: Calcula la descomposición LU de una matriz.
  - DGETRS: Resuelve los sistemas triangulares de ecuaciones lineales a partir de una descomposición LU.

1. Uso de las librerías BLAS y LAPACK desde Matlab

- Paso de argumentos desde programas en C/C++ a funciones de Fortran (BLAS/LAPACK):

  http://es.mathworks.com/help/matlab/matlab_external/calling-lapack-and-blas-functions-from-mex-files.html#f44358

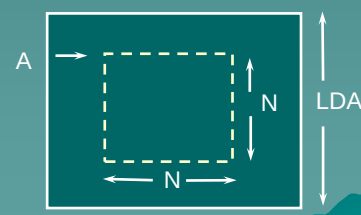
1. Uso de las librerías BLAS y LAPACK desde Matlab

- Entre los ejemplos disponibles en Matlab de programación con mex, está el fichero *matrixDivide.c*, al que hemos renombrado como *dgesv_mex.c*, que utiliza la función DGESV.
- En poliformat puedes encontrar el archivo *dgesv_mex.c*, que es similar al original de Matlab pero adaptado para 64 bits. También puedes encontrar los archivos *membrana.m* y *crea_matriz.m.*
- Para resolver el sistema de ecuaciones de la función *membrana*, invocaremos a la función *dgesv_mex*.

5

2. El fichero dgesv_mex.c

- Descripción de la función DGESV:
  - Calcula la solución del sistema AX=B, sobreescribiendo la matriz A y el vector B.
- Cabecera:
  - SUBROUTINE DGESV (N, NRHS, A, LDA, IPIV, B, LDB, INFO)
    - INTEGER INFO, LDA, LDB, N, NRHS
    - INTEGER IPIV(*)
    - DOUBLE PRECISION A(LDA, *), B(LDB, *)



6

2. El fichero dgesv_mex.c

- Vamos a revisar los detalles más relevantes del fichero *dgesv_mex.c*.
- La llamada desde Matlab, debería ser de la forma:

  >> u = dgesv_mex(A, b)

- Incorporamos los ficheros de cabecera:

```
#if !defined(_WIN32)
#define dgesv dgesv_
#endif

#include "mex.h"
#include "lapack.h"
```

7

2. El fichero dgesv_mex.c

- A continuación, obtenemos los punteros a los argumentos de entrada y controlamos los errores:

```
…
A = mxGetPr(prhs[0]); /* pointer to first input matrix */
B = mxGetPr(prhs[1]); /* pointer to second input matrix */
/* dimensions of input matrices */
m = mxGetM(prhs[0]);
n = mxGetN(prhs[0]);
p = mxGetN(prhs[1]);
…
```

8

2. El fichero dgesv_mex.c

- Muy importante:
  - Los argumentos de entrada de un mex (prhs) NO se pueden modificar (const mxArray *prhs[]).
  - Sin embargo, las funciones de BLAS y LAPACK modifican sus argumentos con mucha frecuencia. Por ejemplo, DGESV resuelve el sistema de ecuaciones sobreescribiendo la matriz de entrada con su descomposición LU y el vector parte derecha con la solución.
  - Como vimos, es necesario tratar de forma especial los argumentos de entrada que se vayan a modificar. Para ello:
    - Crearemos una variable auxiliar de la misma dimensión.
    - Copiaremos el contenido de la variable de entrada a la variable auxiliar.
    - Pasaremos dicha variable auxiliar a las funciones de BLAS o LAPACK.

2. El fichero dgesv_mex.c

- La variable *A* apunta a la matriz de coeficientes del sistema y la variable *B* apunta al vector parte derecha. Creamos las variables auxiliares *LU* y *X*, a las cuales copiaremos el contenido de *A* y *B*, y las pasaremos a la función *DGESV*:

```
…
Awork = mxCreateDoubleMatrix(n, n, mxREAL);
LU = mxGetPr(Awork);
memcpy(LU, A, n * n * mxGetElementSize(prhs[0]));
plhs[0] = mxCreateDoubleMatrix(n, 1, mxREAL);
X = mxGetPr(plhs[0]);
memcpy(X, B, n * mxGetElementSize(prhs[1]));
…
```

2. El fichero dgesv_mex.c

- La función *DGESV* necesita que hayamos reservado espacio para un vector de enteros que guarde la pivotación. Este vector se obtiene al calcular la descomposición LU y se utiliza al resolver los sistemas triangulares (todo ello, dentro de dicha función):

```
…
dims[0] = n;
mxPivot = mxCreateNumericArray(1, dims,
mxINT64_CLASS, mxREAL);
iPivot = (mwSignedIndex*)mxGetData(mxPivot);
…
```

11

2. El fichero dgesv_mex.c

- Cabecera:
  - SUBROUTINE DGESV (N, NRHS, A, LDA, IPIV, B, LDB, INFO)
    - INTEGER INFO, LDA, LDB, N, NRHS
    - INTEGER IPIV(*)
    - DOUBLE PRECISION A(LDA, *), B(LDB, *)
- Invocación a la función DGESV:

```
…
dgesv(&n, &p, LU, &n, iPivot, X ,&n, &info);
…
```

12

2. El fichero dgesv_mex.c

- Al acabar la invocación a la función *DGESV*:
  - *X*, es decir plhs[0], apunta a la solución del sistema de ecuaciones lineales.
  - *LU* contiene la descomposición LU de la matriz original.
  - *iPivot* contiene el vector de pivotación utilizado.
- Como *LU* y *iPivot*, es decir *Awork* y *mxPivot* respectivamente, no se vuelven a utilizar, liberamos la memoria usada:

```
…
mxDestroyArray(Awork);
mxDestroyArray(mxPivot);
…
```

13

3. Compilación de ficheros mex con funciones de BLAS y LAPACK

- Las librerías BLAS y LAPACK se encuentran en las siguientes carpetas de Matlab (v.2022a):
  - En Windows (libmwblas.lib y libmwlapack.lib):

    C:\Program Files\MATLAB\R2022a\extern\lib\win64\microsoft
  - En Linux (libmwblas.so y libmwlapack.so):

    /opt/MATLAB/R2022a/bin/glnxa64

14

3. Compilación de ficheros mex con funciones de BLAS y LAPACK

- Se puede obtener la ruta completa, donde se encuentran las librerías, de este modo:
  - En Windows:

```
>> blaslib = fullfile(matlabroot, 'extern', 'lib',
     computer('arch'), 'microsoft', 'libmwblas.lib');
>> lapacklib = fullfile(matlabroot, 'extern', 'lib',
     computer('arch'), 'microsoft', 'libmwlapack.lib');
```

  - En Linux:

```
>> blaslib = fullfile(matlabroot, 'bin', computer('arch'),
     'libmwblas.so');
>> lapacklib = fullfile(matlabroot, 'bin', computer('arch'),
     'libmwlapack.so');
```


3. Compilación de ficheros mex con funciones de BLAS y LAPACK

- Para compilar (en Windows o Linux) escribiremos:

```
>> mex('dgesv_mex.c', '-largeArrayDims', lapacklib)
```

  La opción '-largeArrayDims' es necesaria para compilar con enteros de 64 bits.

- Otra posibilidad, más sencilla para compilar, es escribir simplemente:

```
>> mex dgesv_mex.c -largeArrayDims -lmwlapack
```

  En este caso, no es necesario haber obtenido previamente la ruta completa de la librería.

- Podemos añadir '-v' a la lista de argumentos anteriores para el modo "verbose" (descriptivo).

4. Ejercicio 1

- Compilar el fichero *dgesv_mex.c.*
- A modo de ejemplo, probar su funcionamiento en la resolución del siguiente sistema Au=b:

$$A = \begin{pmatrix} 1 & -1 & 2 \\ 2 & 1 & 0 \\ 3 & 1 & 1 \end{pmatrix}, b = \begin{pmatrix} 8 \\ -1 \\ 2 \end{pmatrix}$$

```
>> u = dgesv_mex(A, b)
```

$$u = \begin{pmatrix} 1 \\ -3 \\ 2 \end{pmatrix}$$

17

4. Ejercicio 1

- Incorporar a la función *membrana* (opcion=6) la resolución del sistema de ecuaciones *Au=b* mediante la función *dgesv_mex.* Como hemos visto, la llamada debería ser de la forma:

```
u = dgesv_mex(A, b)
```

- Comparar los tiempos de ejecución obtenidos con los del seminario anterior, con n=30.

opcion	1	2	3	4	5	6
Tiempo (segs.)	2.46	85.15	22.34	0.53	0.04	

18

4. Ejercicio 2

- No obstante, en el seminario anterior observamos que era conveniente resolver por separado:
  - La descomposición LU, que sólo realizábamos una vez.
  - Los sistemas de ecuaciones triangulares, que había que resolver para cada paso de tiempo.

19

4. Ejercicio 2

- Implementar un fichero mex llamado *dgetrf_mex.c* que calcule la descomposición LU de una matriz, usando la función DGETRF.
- Probar su funcionamiento calculando la descomposición LU de la siguiente matriz:

$$A = \begin{pmatrix} 1 & -1 & 2 \\ 2 & 1 & 0 \\ 3 & 1 & 1 \end{pmatrix}$$

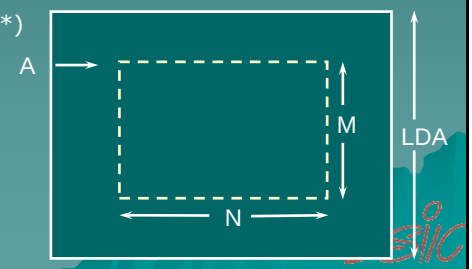
```
>> [LU, iPivot] = dgetrf_mex(A)
```

$$LU = \begin{pmatrix} 3 & 1 & 1 \\ 0.3333 & -1.3333 & 1.6667 \\ 0.6667 & -0.25 & -0.25 \end{pmatrix} \qquad iPivot = \begin{pmatrix} 3 \\ 3 \\ 3 \end{pmatrix}$$

20

4. Ejercicio 2

- Descripción de la función DGETRF:
  - Calcula la descomposición LU de una matriz A de tamaño MxN con pivotación parcial y con sobreescritura de A.
- Cabecera:
  - SUBROUTINE DGETRF (M, N, A, LDA, IPIV, INFO)
    - INTEGER INFO, LDA, M, N
    - INTEGER IPIV(*)
    - DOUBLE PRECISION A(LDA, *)



4. Ejercicio 2

- Incorporar la llamada a la función *dgetrf_mex* desde fuera del bucle de la función *membrana* (opcion=7), de la forma:

  [LU, iPivot]=dgetrf_mex(A)

  donde *A* es la matriz de coeficientes, *LU* es la matriz que contiene la descomposición *LU* de *A* e *iPivot* es el vector de enteros que contiene los intercambios de filas.
- Conviene tener en cuenta que la función DGETRF sobreescribe la matriz *A* con las matrices *L* y *U*.

4. Ejercicio 3

- Implementar un fichero mex llamado *dgetrs_mex.c* para resolver los dos sistemas triangulares conjuntamente, usando la función DGETRS.
- Probar su funcionamiento a partir de los datos de las comprobaciones anteriores:

$$LU = \begin{pmatrix} 3 & 1 & 1 \\ 0.3333 & -1.3333 & 1.6667 \\ 0.6667 & -0.25 & -0.25 \end{pmatrix} \quad iPivot = \begin{pmatrix} 3 \\ 3 \\ 3 \end{pmatrix} \quad b = \begin{pmatrix} 8 \\ -1 \\ 2 \end{pmatrix}$$

```
>> u = dgetrs_mex(LU, b, iPivot)
```

$$u = \begin{pmatrix} 1 \\ -3 \\ 2 \end{pmatrix}$$

4. Ejercicio 3

- Descripción de la función DGETRS:
  - Resuelve un sistema de ecuaciones lineales AX=B o $A^TX=B$, siendo A de tamaño NxN, mediante la descomposición LU obtenida por la función DGETRF. Sobreescribe B con X.
- Cabecera:
  - SUBROUTINE DGETRS (TRANS, N, NRHS, A, LDA, IPIV, B, LDB, INFO)
    - CHARACTER TRANS
    - INTEGER INFO, LDA, LDB, N, NRHS
    - INTEGER IPIV(*)
    - DOUBLE PRECISION A(LDA, *), B(LDB, *)
- Aclaración. La variable TRANS la declararemos e inicializaremos de este modo:

```
char TRANS = 'N';
```

4. Ejercicio 3

- Incorporar la llamada a la función del modo siguiente, desde dentro del bucle de la función *membrana* (opcion=7):

  u = dgetrs_mex(LU, b, iPivot)

- Recordar que la función DGETRS sobreescribe el vector parte derecha *b* con el vector solución *u*.
- Tomar tiempos en la ejecución de la función *membrana* con n=30.

opcion	1	2	3	4	5	6	7
Tiempo (segs.)	2.46	85.15	22.34	0.53	0.04		

5. Descomposiciones LU y de Cholesky

- Alternativamente, en Matlab también podemos separar la descomposición de la matriz y la resolución de los sistemas triangulares mediante la LU.
- Ejercicio 4: Incorpora la llamada a la descomposición LU de Matlab, fuera del bucle (*opcion*=8):

```
[L,U]=lu(A);
```

- Añade la resolución de los sistemas de ecuaciones triangulares, dentro del bucle (*opcion*=8):

```
y=L\b;
u=U\y;
```

5. Descomposiciones LU y de Cholesky

- También podríamos resolver los sistemas de ecuaciones mediante la descomposición de Cholesky, en vez de emplear la LU.
- Ejercicio 5: Incorpora la llamada a la descomposición de Cholesky de Matlab, fuera del bucle (*opcion*=9):

```
L=chol(A,'lower');
U=L';
```

- La resolución de los sistemas de ecuaciones triangulares, dentro del bucle (*opcion*=9) será idéntica a la del ejercicio anterior:

```
y=L\b;
u=U\y;
```

27

6. Matrices dispersas

- No obstante, la matriz de coeficientes del problema de la membrana es claramente dispersa. Matlab dispone de la función *sparse* que convierte una matriz a formato disperso:

```
A=sparse(A);
```

- Ejercicio 6: Convierte la matriz A a una matriz dispersa antes de llevar a cabo la descomposición de Cholesky, fuera del bucle (*opcion*=10):

```
A=sparse(A); L=chol(A,'lower'); U=L';
```

- La resolución de los sistemas de ecuaciones triangulares será igual a la de los ejercicios anteriores.

28

7. Ordenación de matrices

- Ejercicio 7: Entre otros, Matlab incluye los métodos de ordenación de Cuthill-McKee Inverso Simétrico (*symrcm*) y Mínimo Grado Aproximado Simétrico (*symamd*). Incorpóralos a la función membrana con el valor 11 y 12 en la variable *opcion*.

```
% opcion=11
A=sparse(A);
p=symrcm(A);
R=A(p,p);
L=chol(R,'lower');
U=L';
```

```
br=b(p);
y=L\br;
ur=U\y;
u(p)=ur;
```

```
% opcion=12
A=sparse(A);
p=symamd(A);
R=A(p,p);
L=chol(R,'lower');
U=L';
```

```
br=b(p);
y=L\br;
ur=U\y;
u(p)=ur;
```

Antes del bucle

Dentro del bucle

8. Toma de tiempos

- Completa la siguiente tabla, a partir de los tiempos de ejecución de la función membrana, empleando *n*=100 y valores de 7 a 12 en la variable *opcion*:

opcion	Tiempo (segs.)
7	
8	
9	
10	
11	
12	