

Práctica 5. Desarrollo de aplicaciones MEX

Métodos Numéricos para la Computación

Grado en Ingeniería Informática
Escuela de Ingeniería Informática
Universidad de Las Palmas de Gran Canaria

Curso 2015/2016

Actividades de la Práctica

- Utilizar Visual Studio para crear proyectos que generan ficheros MEX que pueden ser utilizados como funciones propias del usuario en MATLAB.
- Posibilidad de extender las capacidades computacionales de MATLAB mediante computación de bajo nivel en C/C++
- Uso de librerías de altas prestaciones desde MATLAB, como por ejemplo BLAS y LAPACK, en aplicaciones propias del usuario.
- Extender las áreas de posibilidades de computación numérica.

Estructura de una función MATLAB

```
function [o1,o2,...,on]=mifuncion(i1,i2,...,im)
```



MATLAB es básicamente un lenguaje funcional, con paso de parámetros por valor y por tanto de ausencia de actuación entre el código de una función hacia atrás, es decir con las variables de las funciones de llamada. La función puede interpretarse desde el programa exterior como una “caja negra” de la cual solo se perciben las entradas y salidas.

Proceso de Desarrollo

1. Crear una función MATLAB **mifuncion.m** vacia, salvo un campo de inicial de comentario conteniendo la información que será presentado en pantalla con la invocación de **help mifuncion**.
2. Crear un fichero C/C++ que contiene la función interface siguiendo un protocolo definido en la documentación. Crear igualmente todas la funciones C/C++ computacionales de la aplicación.
3. Compilar el(los) fichero(s) C/C++ y enlazar generando un fichero que tiene la estructura de DLL (Librería de Enlace Dinámico) pero que se denominará de alguna de las formas siguientes:

mifuncion.mexw32 (Windows 32)

mifuncion.mexw64 (Windows 64)

mifuncion.mexa64 (Linux 64)

4. Instalar ambos ficheros **mifuncion.m** y **mifuncion.mexw32** (por ejemplo) en un directorio de acceso en el path de MATLAB

Dificultades del proceso

- El proceso puede ser altamente laborioso o sencillo dependiendo de si el conjunto de herramientas y configuraciones son compatibles.
- Uno de los aspectos que puede ser más crítico en el proceso de construcción es la compatibilidad del compilador utilizado con la versión concreta de MATLAB.
- También es muy importante en la facilidad/dificultad el conjunto de herramientas/entornos que se utilicen.
- Generalmente se disponen de plugins para entornos como Visual Studio, que facilitan enormemente el trabajo, pero dependen en gran medida de las releases relativa de ambos entornos.
- Otra opción es generar todo el proceso dentro de MATLAB mismo con el comando **mex** que utilizará un compilador externo.

Método 1. Interno a MATLAB

- El comando mex de MATLAB realiza el proceso completo de generación de la función, incluyendo compilación de los códigos fuentes y enlace de los módulos objetos y las librerías necesarias.
- Pero mex no es realmente un compilador, sino que necesita uno externo ajeno. Para definir el compilador externo debemos ejecutar
- `>> mex -setup`
- MATLAB trata de descubrir una lista de compiladores compatibles instalados en el ordenador. Nos proporciona la lista y nos pide que elijamos uno para ser usado.

Método 1. Interno a MATLAB

- En las instalaciones de MATLAB de 32 bits, Mathworks proporciona un compilador propio, denominado lcc, que se instala automáticamente. Esta política está discontinuada y en las versiones de 64 bits ya no se proporciona este compilador.
- Pero lcc es un compilador C no admite C++
- Determinadas utilidades permiten instalar ficheros de configuración que permita a MATLAB reconocer compiladores externos. **Requieren modificar los directorios de instalación de MATLAB, luego no son aplicables en los laboratorios.**
- <http://www.mathworks.com/matlabcentral/fileexchange/44408-matlab-mex-support-for-visual-studio-2013--and-mbuild->
- Los compiladores más comunes son el de Microsoft, gcc mediante Mingw e Intel en el entorno Windows y gcc en el entorno Linux.

Método 2. Externo a MATLAB

- Es el procedimiento más recomendable.
- En un entorno de desarrollo C/C++ (como Visual Studio) podemos generar el fichero `.mexw32`, por ejemplo. Pero deberemos seguir una serie de pasos para configurar el proceso de generación de la DLL.
- Las instrucciones pueden encontrarse en diversos enlaces, son laboriosas y muy dependientes de las versiones. Por ejemplo para Visual Studio:
- <http://www.codeproject.com/Articles/19565/Matlab-7-1-and-Visual-Studio-2005>
- http://coachk.cs.ucf.edu/GPGPU/Compiling_a_MEX_file_with_Visual_Studio2.htm

Resumen de configuración:

<http://stackoverflow.com/questions/21189832/build-a-mex-file-under-visual-studio-ultimate-2012/25449294#25449294>

I can confirm that the same steps as described in [this question](#) work for Visual Studio 2012 (and for Visual Studio 2013 too). Starting with an empty project the following settings in the project's Property Pages are necessary and sufficient to build a working .mexw64 file:

```
Configuration properties -> General:
  Set Target Extension to .mexw64
  Set Configuration Type to Dynamic Library (.dll)

Configuration properties -> VC++ Directories:
  Add $(MATLAB_ROOT)\extern\include; to Include Directories

Configuration properties -> Linker -> General:
  Add $(MATLAB_ROOT)\extern\lib\win64\microsoft; to Additional Library Directories

Configuration properties -> Linker -> Input:
  Add libmx.lib;libmex.lib;libmat.lib; to Additional Dependencies

Configuration properties -> Linker -> Command Line:
  Add /export:mexFunction to Additional Options
```

`$(MATLAB_ROOT)` is the path to Matlab's root folder, eg. C:\Program Files\MATLAB\R2014a. For a 32-bit build you'd need to change both occurrences of 64 to 32.

Si la plataforma de MATLAB es win64, activar la configuración x64 de Visual Studio

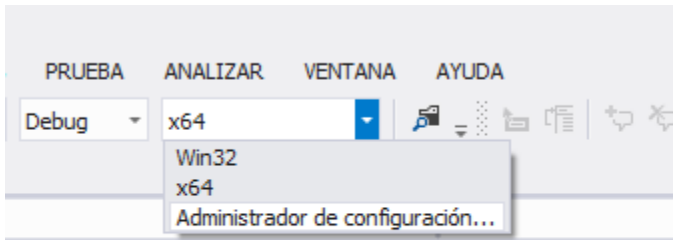
Activar versión x64 de Visual Studio

En principio Visual Studio muestra la configuración Win32. Debemos añadir la x64 .

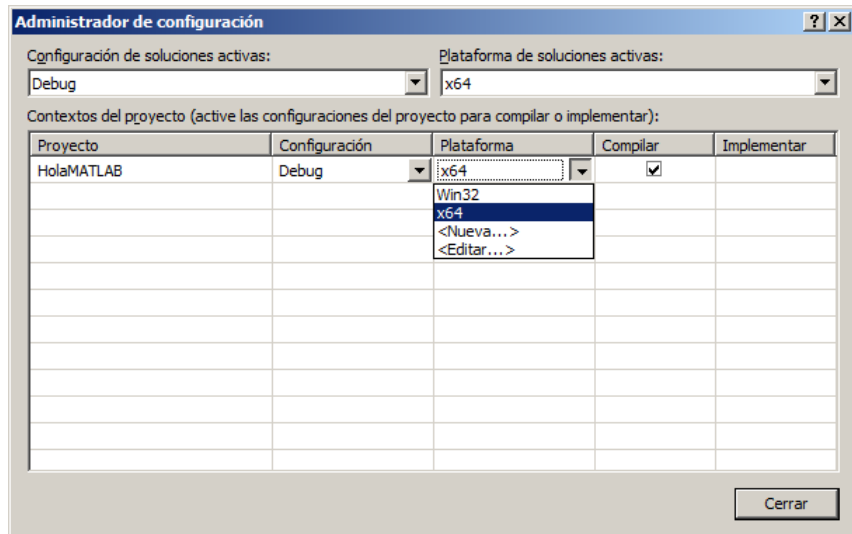
Elegir Administrado de configuración

Si no está x64, indicar Nueva en plataforma de soluciones

En nueva plataforma seleccionar x64



in MEX
computación



Includes y librerías

- Es necesario incluir en el código C/C++ la cabecera de mex, que es de la forma:
- `#include <mex.h>`
- También es necesario incluir los directorios para las cabeceras y las librerías situadas en:
- `(RAIZ MATLAB)/extern/include`
- `(RAIZ MATLAB)/extern/lib/win64/microsoft` o bien
- `(RAIZ MATLAB)/extern/lib/win32/microsoft`
- La RAIZ MATLAB suele incluir la Release que utilizamos, más o menos como:
- `C/Archivos de Programa/MATLAB/R2009a`

Ejemplos de ficheros MEX

- Se recomienda inspeccionar multitud de ejemplos de aplicaciones MEX situadas en la distribución de MATLAB:
- (RAIZ MATLAB)/extern/examples/mex

Estructura del fichero C/C++

El fichero C/C++ debe tener una estructura bien definida, con el nombre de la función idéntica en todo los casos “mexFunction” y argumentos que informan del número de entradas y salidas de la función, así como sendos punteros a arrays de esos argumentos:

lhs salidas
rhs entradas

```
/* The gateway function */
void mexFunction(int nlhs, mxArray *plhs[],
                  int nrhs, const mxArray *prhs[])
{
    /* variable declarations here */

    /* code here */
}
```

This table describes the input parameters for mexfunction.

Parameter	Description
nlhs	Number of output (left-side) arguments, or the size of the plhs array.
plhs	Array of output arguments.
nrhs	Number of input (right-side) arguments, or the size of the prhs array.
prhs	Array of input arguments.

mxArray es un tipo predefinido. Las entradas tienen el cualificador de “const”, pero las salidas deben ser construidas en la función.

Pasos dentro de mexFunction()

- 1. Verificar los números y tipos de parámetros de entrada y salida
- 2. Declarar las variables computacionales que se utilizaran.
- 3. Transferir los datos de entrada, presentes en un mxArray, a las variables computacionales.
- 3. Construir las rutinas computacionales de la aplicación en el mismo u otros ficheros fuentes.
- Construir las variables de salida en el mxArray y llenarlas de los contenidos obtenidos por las rutinas computacionales.

Verificar argumentos.

Ejemplos:

- Verificar que se llama con dos argumentos de entrada (rhs)

```
if(nrhs != 2) {  
    mexErrMsgIdAndTxt("MyToolbox:arrayProduct:nrhs",  
                      "Two inputs required.");  
}
```

- Verificar que se llama con un argumento de salida (lhs)

```
if(nlhs != 1) {  
    mexErrMsgIdAndTxt("MyToolbox:arrayProduct:nlhs",  
                      "One output required.");  
}
```

- Verificar que el primer argumento de entrada (prhs[0]) es un número en doble precisión pero ni un array ni complejo.

```
/* make sure the first input argument is scalar */  
if( !mxIsDouble(prhs[0]) ||  
    mxIsComplex(prhs[0]) ||  
    mxGetNumberOfElements(prhs[0]) != 1 ) {  
    mexErrMsgIdAndTxt("MyToolbox:arrayProduct:notScalar",  
                      "Input multiplier must be a scalar.");  
}
```

Verificar argumentos.

Ejemplos:

- Verificar que el segundo argumento de entrada (prhs[1]) es un vector fila $M=1$. M y N son las dimensiones:

```
/* check that number of rows in second input argument is 1 */  
if(mxGetM(prhs[1]) != 1) {  
    mexErrMsgIdAndTxt("MyToolbox:arrayProduct:notRowVector",  
        "Input must be a row vector.");  
}
```


Leer datos de la entrada.

- Los datos de entrada están en `prhs[i]`. Podemos extraerlos distinguiendo si son escalar o matriz. Si es escalar:
 - `Variable = mxGetScalar(prhs[i])`
- Si es matriz debemos conocer sus dimensiones y un puntero a los datos con:
 - `Nfilas = mxGetM(prhs[i])`
 - `Ncolumnas = mxGetN(prhs[i])`
 - `Puntero = mxGetPr(prhs[i])` → no debemos modificar este contenido , sino copiarlo a nuestro espacio de trabajo.
- NOTA: MATLAB codifica matrices con el criterio Fortran, es decir column-major.

Construir los datos de salida

- Es necesario crear las matrices de salida dentro de `mexFunction()`, recordemos que MATLAB no usa el paso por referencia. Los pasos son:
- 1. Crear una matriz de MATLAB para la salida `plhs[j]`, indicando `nfilas`, `ncolumnas` y tipo:

```
/* create the output matrix */  
plhs[0] = mxCreateDoubleMatrix(1,ncols,mxREAL);
```

Crea un vector fila, 1 fila, de numero reales para la primera salida

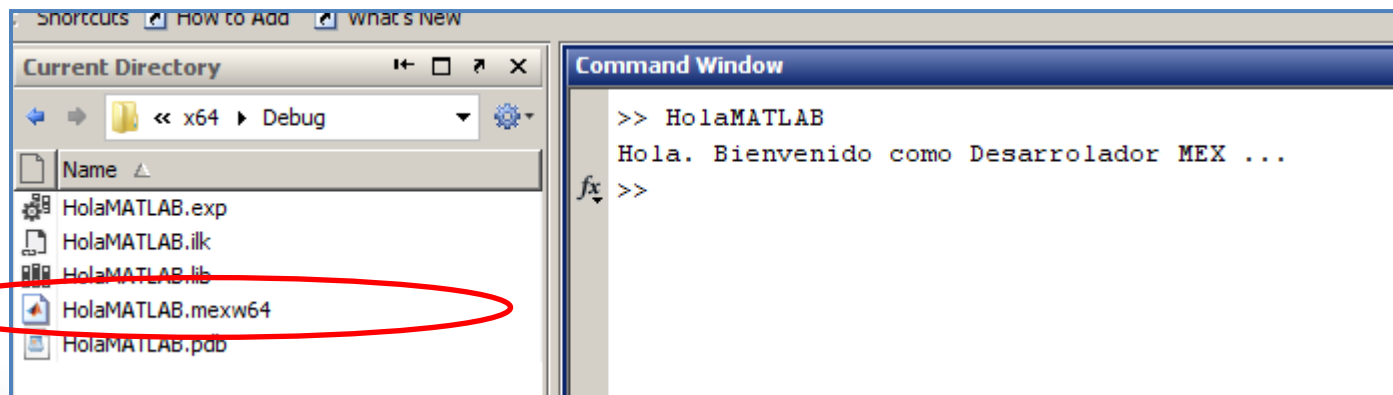
- 2. Obtener el puntero a los datos de la matriz

```
/* get a pointer to the real data in the output matrix */  
outMatrix = mxGetPr(plhs[0]);
```

- 3. Rellenar la memoria de la matriz a partir de los resultados de las rutinas computacionales.

Tarea 1: HolaMATLAB

- Generar un comando MATLAB denominado **HolaMATLAB** que nos presente un texto de bienvenida como desarrolladores de aplicaciones MEX.
- Utilizaremos el Método2, basado en el entorno desarrollo de Visual Studio. En ambos casos generaría el fichero HolaMATLAB.mexw32 dado que en el Laboratorio 1-2 tenemos instalada MATLAB 2007a de 32 bits.

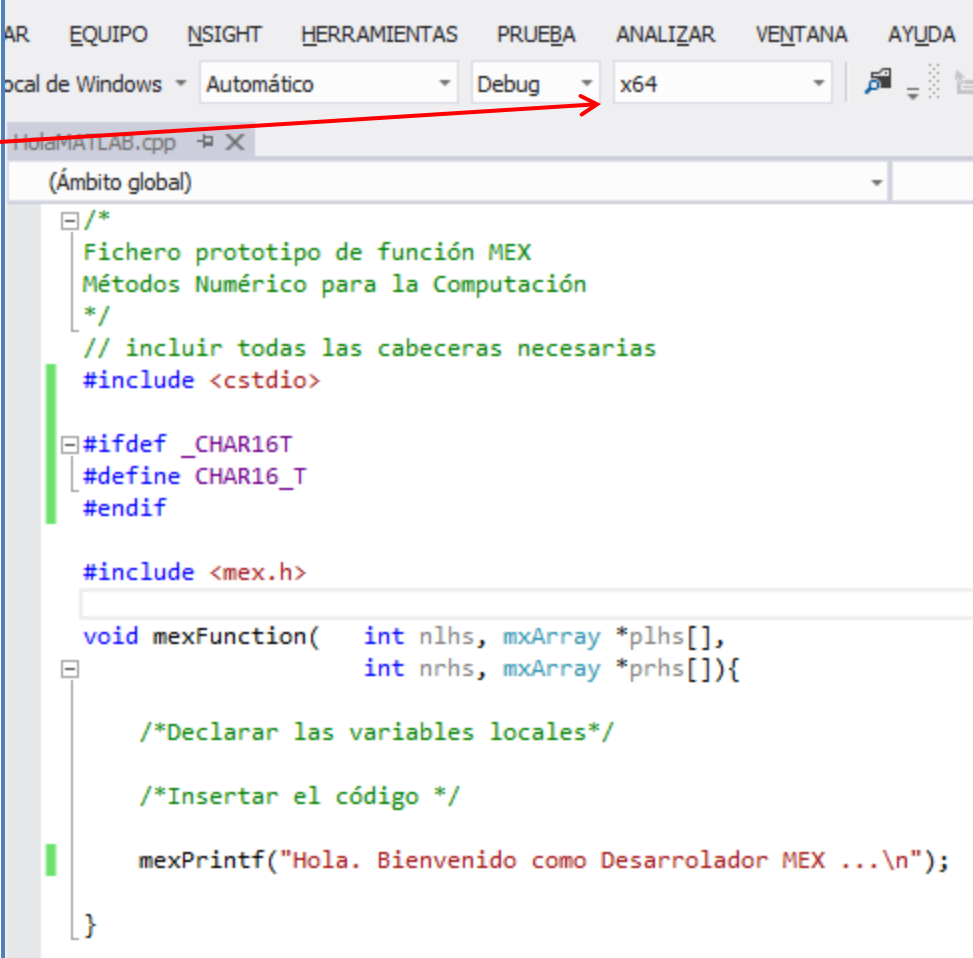


Ejemplo en un entorno de 64 bits

Pasos

- Crear un proyecto C/C++ general, vacío que se denomine HolaMATLAB (por ejemplo) y añadir un nuevo elemento, por ejemplo el fichero HolaMATLAB.cpp.
- Copiar un prototipo vacío de fichero MEX suministrado en el material de prácticas.
- Incorporar el contenido sugerido.
- Para el Método 2, seguir los pasos de la página 9 de este Documento, o bien resumido en un fichero adjuntado.
- Prestar atención a los directorios donde se genera la solución. Normalmente aparece en los comentarios de las fases de compilación, enlace y generación del resultado.

Si MATLAB es una release de 64 bits, debe activarse x64 en lugar de Win32



```
AR EQUIPO NSIGHT HERRAMIENTAS PRUEBA ANALIZAR VENTANA AYUDA
Local de Windows Automático Debug x64
HolaMATLAB.cpp
(Ámbito global)
/*
Fichero prototipo de función MEX
Métodos Numérico para la Computación
*/
// incluir todas las cabeceras necesarias
#include <stdio>

#ifdef _CHAR16T
#define CHAR16_T
#endif

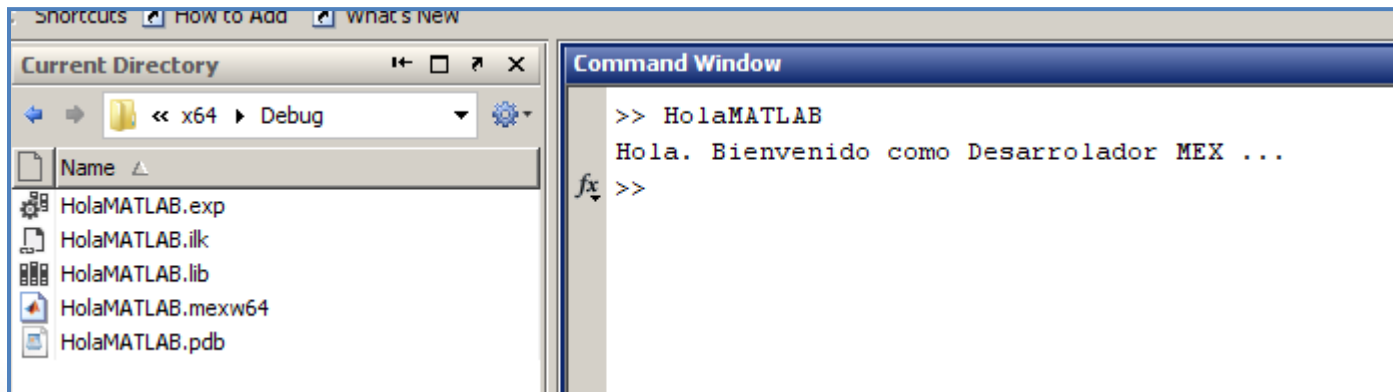
#include <mex.h>

void mexFunction( int nlhs, mxArray *plhs[],
                  int nrhs, mxArray *prhs[]){

    /*Declarar las variables locales*/

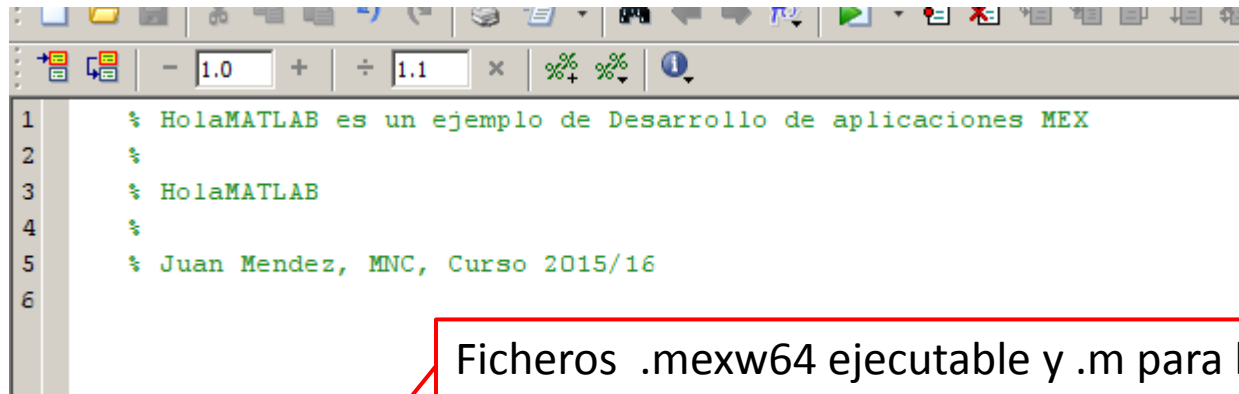
    /*Insertar el código */

    mexPrintf("Hola. Bienvenido como Desarrollador MEX ...\n");
}
```



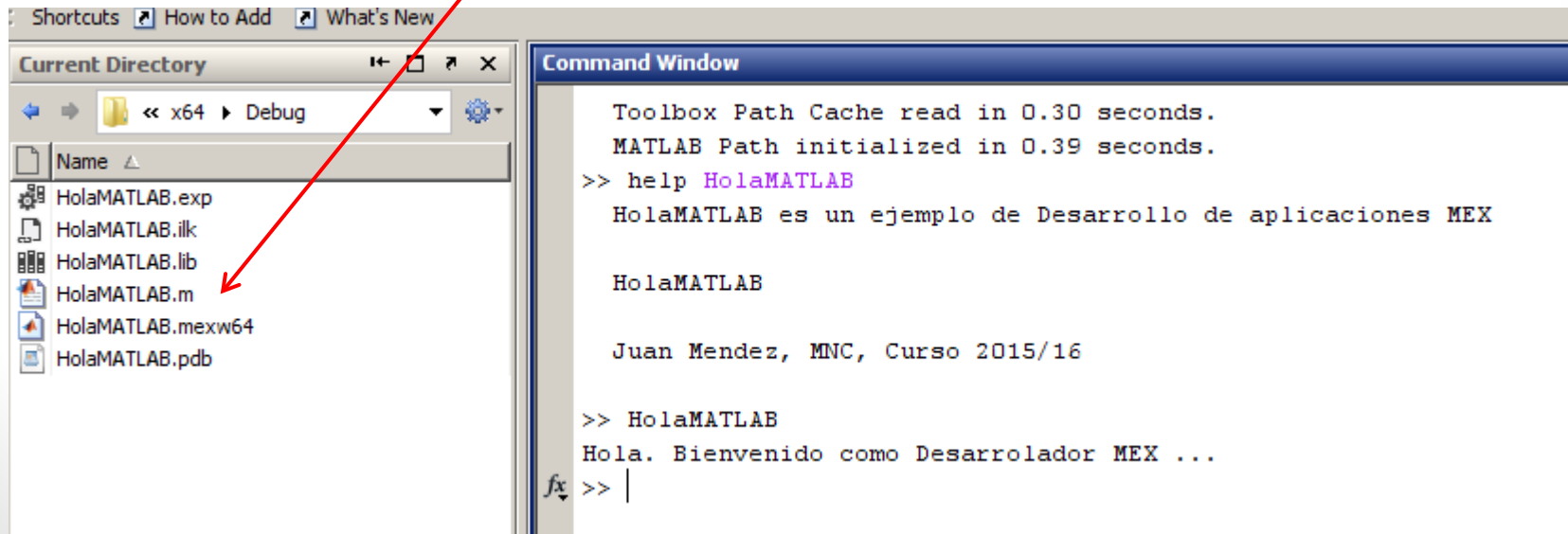
Fichero para help

- Crear un fichero de comentarios HolaMATLAB.m en el mismo directorio. Solo interviene cuando se solicite help



```
1 % HolaMATLAB es un ejemplo de Desarrollo de aplicaciones MEX
2 %
3 % HolaMATLAB
4 %
5 % Juan Mendez, MNC, Curso 2015/16
6
```

Ficheros .mexw64 ejecutable y .m para help



```
Toolbox Path Cache read in 0.30 seconds.
MATLAB Path initialized in 0.39 seconds.
>> help HolaMATLAB
HolaMATLAB es un ejemplo de Desarrollo de aplicaciones MEX

HolaMATLAB

Juan Mendez, MNC, Curso 2015/16

>> HolaMATLAB
Hola. Bienvenido como Desarrollador MEX ...
fx >> |
```

Tarea 2. Aplicación myla

- Generaremos nuestra propia aplicación de computo del determinante y la matriz inversa de una matriz utilizando LAPACK de MKL desde MATLAB.
- myla es nuestro comando de “My Linear Algebra”
- $D = \text{myla}[A]; \rightarrow$ calcula el determinante de la matriz A
- $[D, B] = \text{myla}[A]; \rightarrow$ calcula determinante e inversa B de A
- El soporte de MKL en Visual Studio es perfecto para aplicaciones .exe, pero para desarrollo MEX no está bien configurado. Por ello deberemos realizar la configuración manualmente definiendo librerías y directorios.

Integración de MKL en Visual Studio para desarrollo MEX

Cierta dificultad para crear software que involucra productos de tres concepciones diferentes: Intel+Microsoft+Mathworks

Crearemos un proyecto MEX y lo configuraremos como se recomienda en la página 9 o en el documento adjuntado de resumen.

Además incluiremos el soporte de uso de MKL de forma manual. Se debe incluir:

1. Directorio de include de mkl.h
2. Directorio de las librerías de MKL
3. Librerías de MKL

La raíz de MKL se debe buscar en el navegador de carpetas, puede ser algo así como:

RAIZ MKL = C:/Archivos de Programas (x86)/Intel/Composer XE 2015/mkl

Configuración manual

- En el soporte de MKL de Visual Studio indicaremos No, como está por defecto.
- En los directorios de inclusión añadir además:
(RAIZ MKL)/include
- En los directorios de librerías adicionales añadir:
(RAIZ MKL)/lib/ia32 o bien
(RAIZ MKL)/lib/intel64
- Añadir las librerías adicionales:
mkl_core.lib
mkl_intel_c.lib
mkl_sequential.lib

```
>> help myla
myla es "My Linear Algebra" un programa de cálculo de operaciones
básicas de algebra lineal como el determinante y la matriz inversa.

D = myla(A)
[D,B] = myla(A)

donde A es una matriz cuadrada, D es su determinante y B la matriz
inversa
>> A = [1 2; 3 4]

A =

     1     2
     3     4

>> det(A)

ans =

    -2

>> inv(A)

ans =

   -2.0000    1.0000
    1.5000   -0.5000

>> [D,B] = myla(A)

D =

    -2






B =

   -2.0000    1.0000
    1.5000   -0.5000

>> |
```

Ejemplo de sesión que realiza el cálculo del determinante y matriz inversa en MATLAB ordinario y con nuestra nueva función MEX myla.

En el directorio donde trabajamos está presente la versión .m, de ayuda, que es identificada como un M-file, y la versión .mexw32, de computo, que es identificada como MEX-file

All Files ▲	Type
 myla.exp	EXP File
 myla.lib	LIB File
 myla.m	M-file
 myla.mexw32	MEX-file
 myla.pdb	PDB File

```

>> A = [1 2 3; 4 5 6; 7 8 9]

A =

     1     2     3
     4     5     6
     7     8     9

>> det(A)

ans =

     0

>> inv(A)
Warning: Matrix is close to singular or badly scaled.
Results may be inaccurate. RCOND = 1.541976e-018.

ans =

 1.0e+016 *

 -0.4504    0.9007   -0.4504
  0.9007   -1.8014    0.9007
 -0.4504    0.9007   -0.4504

>> [D,B]=myla(A)
Warning: Matriz singular o casi singular

D =

 6.6613e-016

B =

 1.0e+016 *

 -0.4504    0.9007   -0.4504
  0.9007   -1.8014    0.9007
 -0.4504    0.9007   -0.4504

>> |

```

Diferencia e igualdad de comportamiento en el caso de una matriz singular. En el fichero MEX hemos incluido un Warning de aviso al usuario.

Cabeceras y declaración de variables computacionales

```
(Ambito global) mexF
/*
Fichero prototipo de función MEX
Métodos Numéricos para la Computación
*/
// incluir todas las cabeceras necesarias
#include <stdio>
#include <cstring>
#include <cmath>

#include <mk1.h>

#ifdef _CHAR16T
#define CHAR16_T
#endif
#include <mex.h>

void mexFunction( int nlhs, mxArray *plhs[],
                 int nrhs, const mxArray *prhs[]){

    /*Declarar las variables locales*/
    double *A, *B, determinante;
    int *pivot, info, Nfilas, Ncolumnas;

    /*Insertar el código */
    // Verificación de condiciones
    if (nrhs != 1){
```

Verificación de Argumentos

```
/*Insertar el código */  
// Verificación de condiciones  
if (nrhs != 1){  
    mexErrMsgTxt("Error: myla, Debe tener un argumento de entrada");  
}  
if (!mxIsNumeric(prhs[0])){  
    mexErrMsgTxt("Error: myla: El argumento de entrada debe ser una matriz");  
}  
Nfilas = mxGetM(prhs[0]);  
Ncolumnas = mxGetN(prhs[0]);  
if (Nfilas != Ncolumnas){  
    mexErrMsgTxt("Error: myla: La matriz de entrada debe ser cuadrada");  
}  
if (Nfilas == 0){  
    mexErrMsgTxt("Error: myla: La matriz de entrada no debe ser vacia");  
}  
  
if (nlhs > 2){  
    mexErrMsgTxt("Error: myla, Debe tener uno o dos argumento de salida");  
}  
  
// copia de las variables de entrada en el espacio computacional
```

Copia, proceso y generación de resultados

```
// copia de las variables de entrada en el espacio computacional
A = mxGetPr(prhs[0]); // El espacio de A pertenece a la función de llamada, no tocar
B = (double *)mkl_malloc(Nfilas*Ncolumnas*sizeof(double), 64);
memcpy(B, A, Nfilas*Ncolumnas*sizeof(double)); // copia, implementación del paso por valor

pivot = (int*)mkl_malloc(Nfilas*sizeof(int), 32);

// procesos computacionales
info = LAPACKE_dgetrf(LAPACK_COL_MAJOR, Nfilas, Ncolumnas, B, Ncolumnas, pivot);
determinante = 1.0;
for (int i = 0; i < Nfilas; i++){
    if (pivot[i] != (i + 1)){
        determinante *= -B[i*Ncolumnas + i];
    }
    else {
        determinante *= B[i*Ncolumnas + i];
    }
}

// crear los resultados de salida
plhs[0] = mxCreateDoubleScalar(determinante);
if (nlhs == 2){
    if (fabs(determinante) < 1.0e-8){
        mexWarnMsgTxt("Matriz singular o casi singular");
    }
    LAPACKE_dgetri(LAPACK_COL_MAJOR, Nfilas, B, Ncolumnas, pivot);
    plhs[1] = mxCreateDoubleMatrix(Nfilas, Ncolumnas, mxREAL);
    double *C = mxGetPr(plhs[1]);
    memcpy(C, B, Nfilas*Ncolumnas*sizeof(double));
}
mkl_free(pivot);
mkl_free(B);
}
```

Nótese que C es un puntero al almacenamiento creado por mxCreate..., y servirá para retornar los resultados

Que debe entregar el alumno?

- Cada alumno entregará en el Campus Virtual una memoria en PDF o Word en la que estará contenida una descripción del trabajo realizado, incluyendo descripción, el listado MATLAB o C de la actividad realizada y la captura de pantalla de las gráficas o imágenes generadas. Para autentificar las imágenes cuando sea posible el alumno incluirá su nombre en cada imagen mediante la función title()).
- En principio la tarea quedará abierta para su entrega hasta cierta fecha que se indicará.
- Se puede trabajar en grupo en el Laboratorio, pero la memoria elaborada y entregado será individual.

Bibliografía

MKL Reference Manual: <https://software.intel.com/en-us/mkl-reference-manual-for-c>

C++ random: http://www.cplusplus.com/reference/random/normal_distribution/

MEX File Creation API: <http://es.mathworks.com/help/matlab/call-mex-files-1.html>

Guía de funciones mex: <http://es.mathworks.com/help/matlab/mex-library.html>
<http://es.mathworks.com/help/matlab/cc-mx-matrix-library.html>

Ejemplos mex:

http://es.mathworks.com/help/matlab/matlab_external/table-of-mex-file-source-code-files.html

http://es.mathworks.com/help/matlab/matlab_external/compiling-mex-files-with-the-microsoft-visual-c-code.h

<http://www.codeproject.com/Articles/19565/Matlab-7-1-and-Visual-Studio-2005>

http://coachk.cs.ucf.edu/GPGPU/Compiling_a_MEX_file_with_Visual_Studio2.htm