Práctica 5
La película

Héctor Garbisu Arocha
Curso 2015/16
Métodos Numéricos para la Computación
Grado en Ingeniería Informática
Escuela de Ingeniería Informática
Universidad de Las Palmas de Gran Canaria

Índice

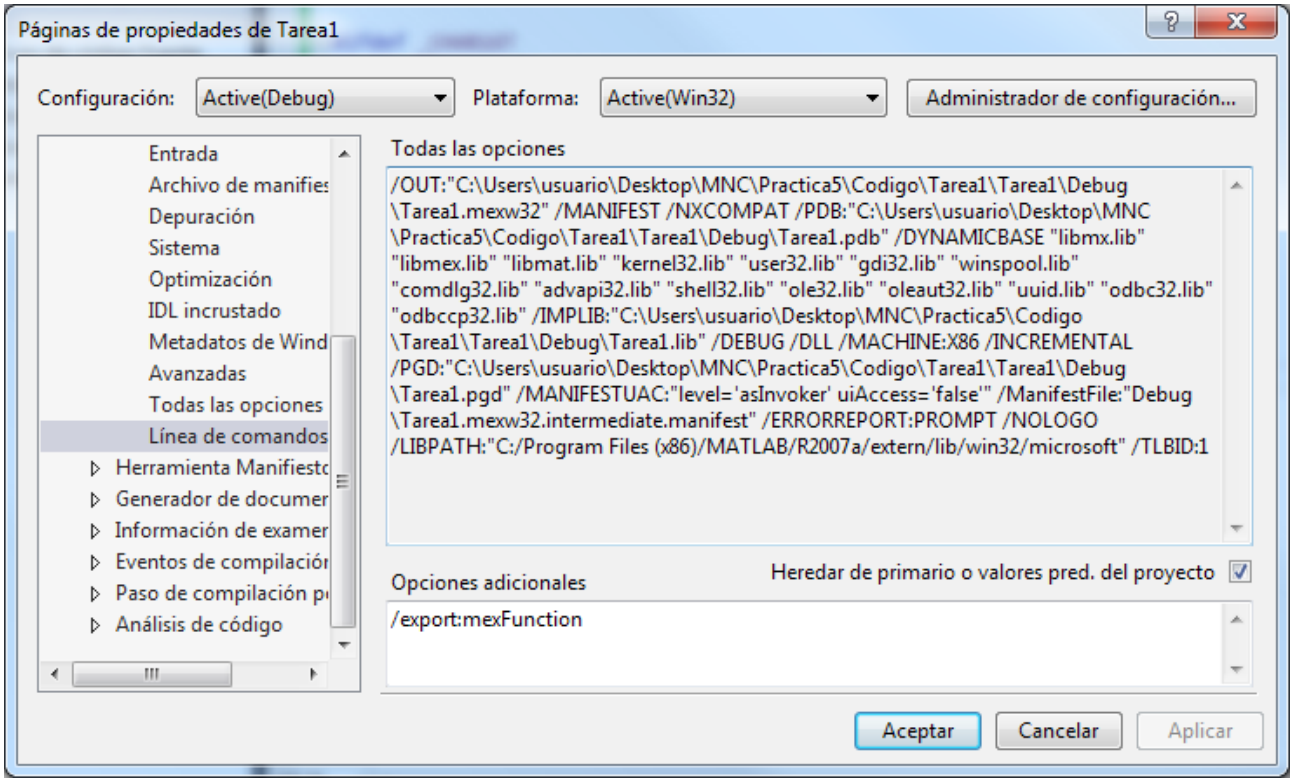
1. Aplicaciones MEX

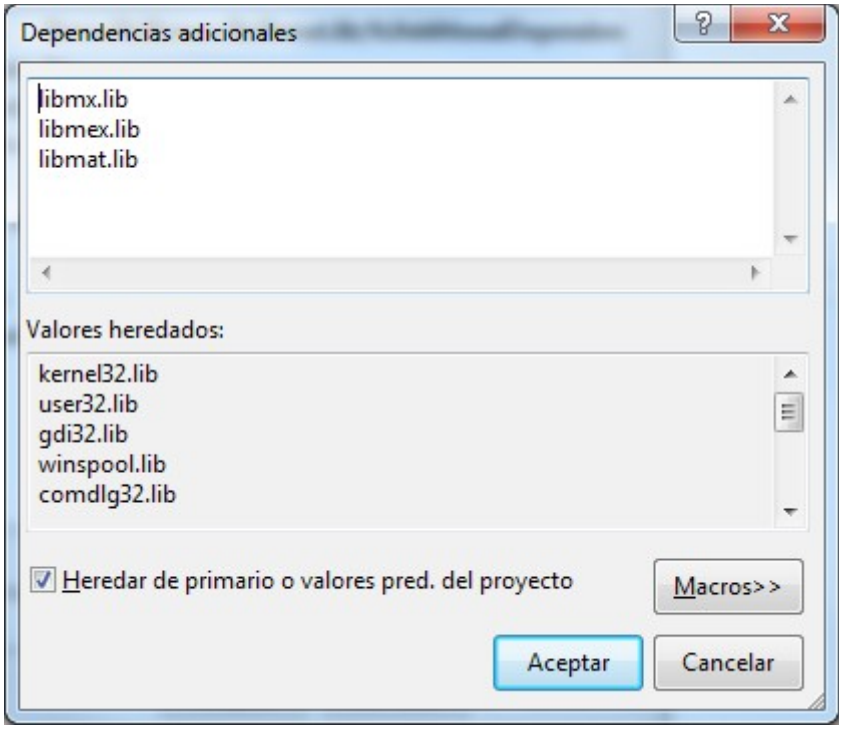
Las aplicaciones MEX nos permitien ejecutar programas compilados dentro de Matlab, pudiendo encapsular un proceso dentro de una librería de enlace dinámico. Las extensiones de estos ficheros para un SO Windows de 32 bits es .mexw32.

Es interesante construir aplicaciones MEX construidas a partir de funciones eficientes, como las de BLAS o LAPACK. Hay que tener en cuenta que en Matlab todos los datos de salida se ofrecen como salida del subprograma; no hay variables de entrada/salida.

Para construir un fichero .mexw32 desde VisualStudio hay que configurar el proyecto con unas características específicas. Por suerte, la guía de configuración manual de que disponemos los alumnos da una serie de pasos muy fáciles de seguir.

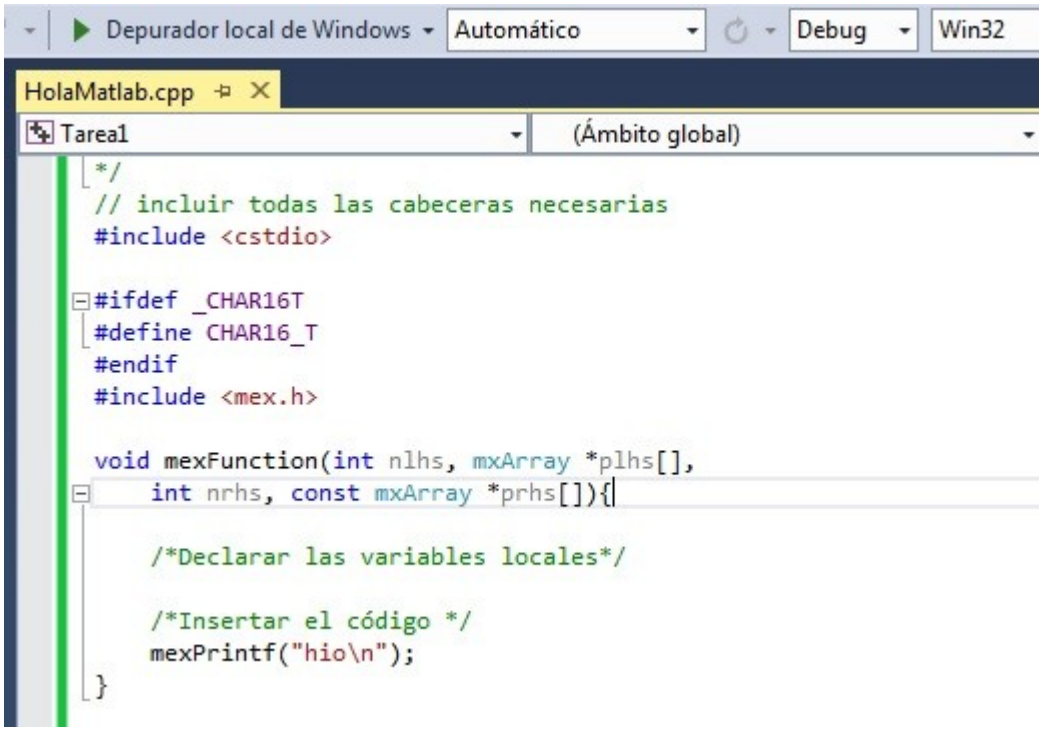
Estas son algunas de las cosas que hay que modificar dentro de las propiedades del proyecto.





2. Tarea 1 HolaMATLAB

Como en este primer ejercicio de toma de contacto no vamos a hacer nada con datos de entrada, no hace falta organizar ni duplicar variables. Simplemente se saca por consola un saludo.
El proyecto está en modo de depuración porque de momento no hace falta que esté de otra forma. Eso hará que los ficheros de salida aparezcan en la carpeta Debug. No es un problema.



Al compilar y construir el proyecto tenemos en la carpeta de destino (~/Tarea1\Tarea1\Debug) varios ficheros: Tarea1.exp ,Tarea1.ilk ,Tarea1.lib ,Tarea1.mexw32 y Tarea1.pdb.
Con ésto, si en Matlab cambiamos la ruta del directorio actual a donde están estos ficheros, podremos llamar al programa Tarea1 y ver cómo se ejecuta como si fuera un script de Matlab.

```
>> Tarea1
hio
>>
```

Al añadir un fichero Tarea1.m, el comando help Tarea1 mostrará los comentarios de dicho fichero como texto de ayuda en Matlab.

```
1   %Tarea1.m
2   %fichero de ayuda
3   %Héctor Garbisu MNC 2015
```

```
>> help Tarea1
 Tarea1.m
 fichero de ayuda
 Héctor Garbisu MNC 2015
>>
```

3. Tarea 2. Aplicación myla

Para hacer una aplicación MEX que funcione con MKL además de los anteriores hay que hacer otras modificaciones en las propiedades del proyecto que también están recogidas en el pdf de configuración manual para MEX.
Además, hay que recordar que el proyecto debe estar en modo Release, así que si antes estaba en modo Debug para hacer HolaMATLAB, los cambios hechos para esa tarea debemos ponerlos también para el modo Release.

Este segundo ejercicio será un pequeño programa que devuelva el determinante y la matriz inversa a una dada.
En este caso el programa es un poco más elaborado así que habrá que tener en cuenta lo siguiente: Inclusión de librerías necesarias, control sobre el tipo y número de parámetros de entrada, copia de parámetros de interés en variables locales y cómo mostrar el resultado.

Inclusión de dependencias

Control de prámetros de entrada: los datos deben suficientes y consistentes con el problema (debe ser una matriz, etc.)

```
#include <cstdio>
#include <cstring>
#include <cmath>
#include <mkl.h>
#ifdef _CHAR16T
#define CHAR16_T
#endif
#include <mex.h>
void mexFunction(int nlhs, mxArray *plhs[],
    int nrhs, const mxArray *prhs[]){
    double *A, *B, determinante;
    int *pivot, info, Nfilas, Ncolumnas;
    if (nrhs != 1){ // nº args diferente de 1
        mexErrMsgTxt("Error. myla, Debe tener un arg de entrada");
    }
    if (!mxIsNumeric(prhs[0])){
        mexErrMsgTxt("Error. El argumento de entrada debe ser una matriz");
    }
    Nfilas = mxGetM(prhs[0]);
    Ncolumnas = mxGetN(prhs[0]);
    if (Nfilas != Ncolumnas){
        mexErrMsgTxt("Error. La matriz debe ser cuadrada");
    }
    if (Nfilas == 0){
        mexErrMsgTxt("Error. La matriz debe no ser vacía");
    }
    if (nlhs > 2){
        mexErrMsgTxt("Error. Debe haber uno o dos args de salida");
    }
```

Se hace el proceso duro de los datos.
En este caso se usan dos funciones que vimos en la práctica anterior, además de dos de sus propiedades.

```
36      A = mxGetPr(prhs[0]);
37      B = (double *)mkl_malloc(Nfilas*Ncolumnas*sizeof(double), 64);
38      memcpy(B, A, Nfilas*Ncolumnas*sizeof(double));
39      pivot = (int *)mkl_malloc(Nfilas*sizeof(int), 32);
40      //procesos computacionales
41      info = LAPACKE_dgetrf(LAPACK_COL_MAJOR, Nfilas, Ncolumnas, B, Ncolumnas, pivot);
42      determinante = 1.0;
43      for (int i = 0; i < Nfilas; i++){
44          if (pivot[i] != (i+1)){
45              determinante *= -B[i*Ncolumnas + i];
46          }
47          else{
48              determinante *= B[i*Ncolumnas + i];
49          }
50      }
```

Por último se saca el determinante y la inversa al ponerlos en el objeto mxArray *plsh[]
También se hace una última comprobación sobre los parámetros de entrada para optimizar. Sólo se calcula la inversa de la matriz si el número de parámetros de salida es 2.

```
52      plhs[0] = mxCreateDoubleScalar(determinante);
53      if (nlhs == 2){
54          if (fabs(determinante) < 1.0e-8){
55              mexWarnMsgTxt("Matriz singular o casi singular");
56          }
57          LAPACKE_dgetri(LAPACK_COL_MAJOR, Nfilas, B, Ncolumnas, pivot);
58          plhs[1] = mxCreateDoubleMatrix(Nfilas, Ncolumnas, mxREAL);
59          double *C = mxGetPr(plhs[1]);
60          memcpy(C, B, Nfilas*Ncolumnas*sizeof(double));
61      }
62      mkl_free(pivot);
63      mkl_free(B);
64  }
```

Para comprobar si nuestro programa funciona, vamos a comparar el resultado con el dado por las funciones de MATLAB.
La matriz A es una matriz generada por magic(5)

```
>> A, det(A), inv(A)

A =                                 ans =       ans =

    17    24     1     8    15          5070000   -0.0049    0.0512   -0.0354    0.0012    0.0034
    23     5     7    14    16                     0.0431   -0.0373   -0.0046    0.0127    0.0015
     4     6    13    20    22                    -0.0303    0.0031    0.0031    0.0031    0.0364
    10    12    19    21     3                     0.0047   -0.0065    0.0108    0.0435   -0.0370
    11    18    25     2     9                     0.0028    0.0050    0.0415   -0.0450    0.0111
```

Ahora ejecutamos Tarea1 y vemos qué devuelve.

```
>> [det, inv] = Tarea1(A)
hios

det =

  5.0700e+006


inv =

   -0.0049    0.0512   -0.0354    0.0012    0.0034
    0.0431   -0.0373   -0.0046    0.0127    0.0015
   -0.0303    0.0031    0.0031    0.0031    0.0364
    0.0047   -0.0065    0.0108    0.0435   -0.0370
    0.0028    0.0050    0.0415   -0.0450    0.0111
```

Salvo una pequeña variación en la notación de la salida del determinante, todo está como debería.