Ingeniería del software con MATLAB: programación en M, C y C++

Tomás Robles Valladares

Ramón Alcarria Garrido

Borja Bordel Sánchez

Diego Martín de Andrés

MATLAB aplicado a la ingeniería telemática

Departamento de Ingeniería de Sistemas Telemáticos (UPM)

PROGRAMA

- El lenguaje M
- Tipos de datos en M. Funciones y paquetes
- Programación concurrente en M
- Integración de subrutinas C y C++ con algoritmos en código M
- Generación de código C y C++ embebido a partir de código M

EL LENGUAJE M

- M es el lenguaje de programación con el que se codifican los algoritmos que se ejecutan en el entorno MATLAB
- Es muy anterior a la creación del propio MATLAB
  - Nace en 1970 como un recubrimiento de alto nivel sobre librerías FORTRAN de cálculo numérico
- No debe confundirse con M#
  - Para programación de sistemas Microsoft

EL LENGUAJE M

- Ejemplo de código escrito en M

```
% Bucle principal
while n < N
    % Incremento del contador
    n=n+1;

    %Integración
    [X, ~] = IntegraVanDerPolForzado_and_Jacobiano(mu, A_, omega, IC, TSpan);
    [rX,~]=size(X);

    % Extracción del sistema
    for i=1:d
        m1=d+1+(i-1)*d;
        m2=m1+d-1;
        A(:,i)=(X(rX,m1:m2))';
    end

    %Descomposición QR
    [Q,R]=qr(A);
    if T2>DiscardTime
        Q0=Q;
    else
        Q0=eye(d);
    end
```

EL LENGUAJE M

- Tradicionalmente M ha sido un lenguaje interpretado
  - Se habla de “scripts MATLAB”
- Desde la versión 8.5 de M (asociada a MATLAB R2015a) las rutinas se traducen de forma dinámica a código máquina nativo
  - Compilación *Just in time*, “en tiempo de ejecución” o “justo a tiempo” (voz latinoamericana)

EL LENGUAJE M

- Se trata de un lenguaje multiparadigma, pues admite:
  - Programación imperativa: Se basa en dar instrucciones al ordenador de como hacer las cosas en forma de algoritmos
  - Programación orientada a objetos: Está basada en el paradigma imperativo, pero encapsula elementos denominados “objetos” que incluyen tanto variables como funciones

EL LENGUAJE M

- Programación orientada a arrays: Está basada en el paradigma imperativo, en el que las operaciones habituales son las de tipo matricial, no escalar.
- Programación declarativa: Está basado en describir el problema declarando propiedades y reglas que deben cumplirse, en lugar de instrucciones
  - En concreto admite programación funcional, en el que las reglas y propiedades vienen en forma de predicado matemático

EL LENGUAJE M

- Además admite programación dinámica
  - El script se puede modificar en tiempo de ejecución
- M es además débilmente tipado…
  - No es preciso declarar el tipo de las variables empleadas

M

```
A = 0;
```

Java

```
int A = 0;
```

EL LENGUAJE M

- … y permite tipado dinámico
  - El tipo de un dato puede variar de forma dinámica

M

```
A = 0;
A = [0 0];
```

Java

```
int A = 0;
int [] A = [0 0];
```

Error en compilación

Illegal start of expression

Variable A is already defined

EL LENGUAJE M

- El paso de variables entre funciones sigue el modelo de “copia vaga” *(copy-on-write o lazy-copy)*
  - Las variables se pasan por referencia, pero si se modifica su valor se realiza una copia temporal (que es la que se actualiza)
  - Sólo se puede modificar el valor de una variable en el mismo nivel en el que se declara

EL LENGUAJE M

```
function [B] = myfunction (A)
        B = 2*A;
end
```

Sólo se consulta el valor de A en memoria

EL LENGUAJE M

```
function [B] = myfunction (A)
        A = A/2;
        B = A%2;
end
```

Se hace una copia local de A para esta función, que desaparece al terminar el bloque. La actualización de A no se conserva

EL LENGUAJE M

- ¿Cómo se puede actualizar una variable en una subrutina?
  - Sólo es recomendable si es imprescindible
    - P. ej. Por problemas de disponibilidad de memoria para mantener copias locales
  - Se deben aplicar las llamadas “funciones in-place”
  - Lo veremos más adelante

EL LENGUAJE M

- La alternativa es devolver una nueva variable con el valor actualizado

```
function [B, C] = myfunction (A)
        C = A/2;
        B = C%2;
end
```

TIPOS DE DATOS EN M

- En M los datos se clasifican en dos grandes grupos
  - Básicos
  - Arrays
- Al margen de estos grupos, existen otros tipos
  - Tablas
  - Arrays categóricos
  - Manejadores de funciones
  - Objetos

TIPOS DE DATOS EN M

- Los tipos básicos se caracterizan porque todo el espacio reservado se ocupa por el valor de la variable
- Hay 12 tipos básicos
  - int8 : Entero con signo de 8 bits (-2^7 a 2^7)
  - uint8 : Entero sin signo de 8 bits (0 a 2^8)
  - int16, uint16, int32, uint32, int64, uint64
  - logical: Ocupan 8 bits. Codifican el valor 0 ó 1

TIPOS DE DATOS EN M

- char: ocupa 16 bits. Representa un carácter Unicode
- single: Datos numéricos en precisión simple (número decimales)
- double: Datos numéricos en doble precisión (número decimales)

TIPOS DE DATOS EN M

- Los arrays ocupan en memoria el espacio correspondiente a la suma de las variables básicas que los componen, más una cabecera
- Hay 3 tipos de array
  - Vectores y matrices: Arrays n-dimensionales de tipos básicos. Añaden una cabecera de 112 bytes

```
Vector = [0 0];
Matriz = [0 0 ; 0 0];
```

TIPOS DE DATOS EN M

- Arrays de celdas: Arrays de celdas indexadas, donde cada celda contiene un vector o matriz de dimensiones cualesquiera. Añaden una cabecera de 112 bytes (adicional a la que acompaña a cada vector o matriz).

```
Celdas = {[0 0]};
```

TIPOS DE DATOS EN M

- Estructuras: Conjuntos de variables que pueden ser referenciadas mediante un nombre. Añaden una cabecera de 112 bytes general, 64 bytes por cada campo (para almacenar el nombre), y la cabecera (si la tiene) asociada a cada una de las variables almacenadas

```
Estructura = struct('Vector', [0 0]);
```

TIPOS DE DATOS EN M

- Tablas: Conjunto de conjuntos de datos. Se pueden mezclar diferentes tipos (M mantiene metadatos para soportarlo). Permite indexación flexible, ordenación automática, fusión de conjuntos, etc.

	1	2	3	4	5	6	7	8	9	10	11
1	0.8147	0.1622	0.6443	0.0596	0.4229	0.5822	0.8507	0.5590	0.6837	0.9879	0.6312
2	0.9058	0.7943	0.3786	0.6820	0.0942	0.5407	0.5606	0.8541	0.1321	0.1704	0.3551
3	0.1270	0.3112	0.8116	0.0424	0.5985	0.8699	0.9296	0.3479	0.7227	0.2578	0.9970
4	0.9134	0.5285	0.5328	0.0714	0.4709	0.2648	0.6967	0.4460	0.1104	0.3968	0.2242
5	0.6324	0.1656	0.3507	0.5216	0.6959	0.3181	0.5828	0.0542	0.1175	0.0740	0.6525
6	0.0975	0.6020	0.9390	0.0967	0.6999	0.1192	0.8154	0.1771	0.6407	0.6841	0.6050
7	0.2785	0.2630	0.8759	0.8181	0.6385	0.9398	0.8790	0.6628	0.3288	0.4024	0.3872
8	0.5469	0.6541	0.5502	0.8175	0.0336	0.6456	0.9889	0.3308	0.6538	0.9828	0.1422
9	0.9575	0.6892	0.6225	0.7224	0.0688	0.4795	5.2238e-04	0.8985	0.7491	0.4022	0.0251
10	0.9649	0.7482	0.5870	0.1499	0.3196	0.6393	0.8654	0.1182	0.5832	0.6207	0.4211
11	0.1576	0.4505	0.2077	0.6596	0.5309	0.5447	0.6126	0.9884	0.7400	0.1544	0.1841
12	0.9706	0.0838	0.3012	0.5186	0.6544	0.6473	0.9900	0.5400	0.2348	0.3813	0.7258
13	0.9572	0.2290	0.4709	0.9730	0.4076	0.5439	0.5277	0.7069	0.7350	0.1611	0.3704
14	0.4854	0.9133	0.2305	0.6490	0.8200	0.7210	0.4795	0.9995	0.9706	0.7581	0.8416
15	0.8003	0.1524	0.8443	0.8003	0.7184	0.5225	0.8013	0.2878	0.8669	0.8711	0.7342
16	0.1419	0.8258	0.1948	0.4538	0.9686	0.9937	0.2278	0.4145	0.0862	0.3508	0.5710
17	0.4218	0.5383	0.2259	0.4324	0.5313	0.2187	0.4981	0.4648	0.3664	0.6855	0.1769
18	0.9157	0.9961	0.1707	0.8253	0.3251	0.1058	0.9009	0.7640	0.3692	0.2941	0.9574
19	0.7922	0.0782	0.2277	0.0835	0.1056	0.1097	0.5747	0.8182	0.6850	0.5306	0.2653

TIPOS DE DATOS EN M

- Arrays categóricos: Vectores o matrices en los que cada posición toma un valor no numérico de entre un conjunto finito de valores posibles
  - P.ej. {“Bueno”, “Malo”, “Pésimo”}
- Es mucho más eficiente que un array tradicional de cadenas de caracteres
- Sobre este tipo de datos pueden realizarse operaciones lógicas como si fueran arrays numéricos

TIPOS DE DATOS EN M

- Manejadores de funciones: Punteros a funciones
- Objetos: Se distinguen dos subtipos
  - Objetos de usuario: Creados a partir de clases creadas en M
  - Objetos Java: Creados a partir de clases Java

FUNCIONES Y PAQUETES

- Hay cinco tipos de funciones en M
  - Principales
  - Locales
  - Anidadas
  - Privadas
  - Anónimas
- Cuando se realiza programación con objetos, además, se pueden emplear métodos

FUNCIONES Y PAQUETES

- Las funciones principales deben estar escritas en un fichero nombrado igual que la función

```
function [sal1, sal2,…, saln] = myfunction(en1, en2,…, enn)
        …
end
```

FUNCIONES Y PAQUETES

- Si no se desea explicitar las variables de entrada y/o salida se pueden emplear palabras reservadas en su lugar
  - *varargout* para indicar cualquier tipo y número de variables de salida
  - *varargin* para indicar cualquier tipo y número de variables de entrada
  - *nargout* para indicar ninguna variable de salida
  - *nargin* para indicar ninguna variable de entrada

FUNCIONES Y PAQUETES

- Las funciones locales son funciones que se codifican en el mismo fichero que una función principal, pero fuera de la misma. Son sólo visibles por las funciones escritas en ese mismo fichero

```
function [sal1, sal2,…, saln] = myfunction(en1, en2,…, enn)
        …
end
function [sal1, sal2,…, saln] = local(en1, en2,…, enn)
        …
end
```

FUNCIONES Y PAQUETES

- Las funciones anidadas son funciones que se codifican dentro de otras funciones (habitualmente justo antes del cierre). Sólo son vistas por la función “padre”. Pueden hacer uso de todas las variables del “padre”

```
function [sal1, sal2,…, saln] = myfunction(en1, en2,…, enn)
        function [sal1, sal2,…, saln] = anidada(en1, en2,…, enn)
                …
        end
end
```

FUNCIONES Y PAQUETES

- Las funciones privadas son funciones habituales cuyos ficheros se almacenan en una carpeta llamada “private”. Estas funciones sólo pueden ser vistas desde los ficheros situados en el directorio inmediatamente superior.

FUNCIONES Y PAQUETES

- Las funciones anónimas son funciones que no quedan almacenadas en un fichero. Se define una expresión matemática que se mapea en memoria volátil y de la que se obtiene un puntero para referenciarla
- Una función puede cambiar de forma dinámica

```
myfunctionAnonima = @(en1) en1^2;
```

FUNCIONES Y PAQUETES

- Todas las funciones pueden ser escritas siguiendo el modelo “in-place”
- Se emplea para modificar de forma permanente una de las variables de entrada sin que haya copia vaga
- Basta colocar como variable de salida la variable de entrada que se desea modificar

FUNCIONES Y PAQUETES

```
function [en1] = myfunction(en1)
        en1 = … ;
end
```

- No es recomendable salvo que se esté seguro de su necesidad
- Puede dar lugar a errores si se emplean nombres de variables comunes (x, y, etc.)

FUNCIONES Y PAQUETES

- En M los paquetes se crean tomando como referencia el sistema de ficheros
- No es necesario incluir ninguna sentencia en los scripts, clases o funciones para explicitar el paquete al que pertenecen
  - Un fichero pertenece al paquete que se llama como el directorio que lo contiene
- En M un mismo paquete puede contener clases, funciones, scripts…

PROGRAMACIÓN CONCURRENTE EN M

- Se distinguen dos tipos de concurrencia en M
  - Implícita: Aquella que se soporta en mecanismos propios del lenguaje.
    - P. ej. Operaciones especiales que se ejecutan en varias hebras de trabajo paralelas
  - Explícita: Aquella que se soporta en mecanismos del entorno de ejecución.
    - Para lograr concurrencia explícita hay que añadir sentencias especiales destinadas a modificar el comportamiento habitual del motor de ejecución

PROGRAMACIÓN CONCURRENTE EN M

- El modelo de concurrencia en MATLAB sigue el paradigma MIMD
  - Múltiples instrucciones, múltiples datos
  - Cada procesador ejecuta código de forma asíncrona e independiente
- Admite también SIMD, pero su uso es complejo
- Aunque sólo es una recomendación, la concurrencia en MATLAB solo es eficiente en arquitecturas multinúcleo

PROGRAMACIÓN CONCURRENTE EN M

- El entorno donde se arranca la ejecución (llamado cliente) orquesta la distribución y ejecución el código en un conjunto de procesos independientes llamados (trabajadores o *workers*)
- Cada trabajador ejecuta un “trozo” de código que se llama “tarea”

PROGRAMACIÓN CONCURRENTE EN M

- La programación concurrente debe emplearse, básicamente, cuando enfrentemos alguno de estos problemas
  - Bucles de gran número de iteraciones
  - Bucles con iteraciones muy pesadas y largas
  - Scripts donde el código pueda separarse en varias tareas independientes de forma natural

PROGRAMACIÓN CONCURRENTE EN M

  - Cuando se evalúen conjuntos de datos muy grandes
  - Cuando queramos que una ejecución no bloquee el entorno MATLAB
    - Ejecución en segundo plano
- En los tres primeros casos, se empleará concurrencia implícita. En los dos últimos explícita

PROGRAMACIÓN CONCURRENTE EN M

- La utilización más extendida de la concurrencia implícita en M son los bucles *for* paralelos

```
parfor i=1:n
        …
end
```

PROGRAMACIÓN CONCURRENTE EN M

- Basta cambiar la palabra reservada *for* por *parfor* para que el motor de ejecución organice de forma automática el paralelismo
- Se crea una *piscina* de tareas (siendo cada tarea una iteración), de donde una serie de *trabajadores* van extrayendo el código que van a ejecutar

PROGRAMACIÓN CONCURRENTE EN M

- No hay ninguna garantía de que el orden de iteración se respete
  - Es decir, la iteración $i = 200$, puede ejecutarse antes que $i = 1$
- Por tanto, es imprescindible que las iteraciones sean independientes
- Al finalizar la ejecución los resultados serán visibles en el cliente como si de un bucle *for* tradicional se tratarse

PROGRAMACIÓN CONCURRENTE EN M

- La concurrencia explícita sólo estará disponible si se dispone del *Parallel Processing Toolbox* instalado
- Aunque permite organizar sistemas concurrentes a gran escala y de forma compleja, aquí sólo vamos a revisar un uso inicial
  - Pero suficiente para la mayoría de los casos

PROGRAMACIÓN CONCURRENTE EN M

- Para solicitar que un determinado script se ejecute por un trabajador distinto del cliente se emplea el comando *batch*

```
trabajador = batch (‘myScript’)
```

- La función *batch* devuelve el ID del trabajador al que se le ha asignado la tarea

PROGRAMACIÓN CONCURRENTE EN M

- Si se desea que el script se distribuya entre más de un trabajador, basta indicarlo al invocar el comando

```
trabajador = batch (‘myScript’, ‘pool’, 3);
```

- En este caso 3 trabajadores ejecutarán el script

PROGRAMACIÓN CONCURRENTE EN M

- Cuando se arrancan varios trabajadores con el comando *batch* siempre se arranca uno adicional que coordina la ejecución
  - *batch* es no bloqueante
- Al contrario que en la concurrencia implícita, cuando *batch* termina no devuelve los resultados de ejecución, es preciso recuperarlos

PROGRAMACIÓN CONCURRENTE EN M

```
load (trabajador, ‘nombre_variable’);
```

- Si se desea, se puede bloquear el cliente hasta que termine de ejecutar sus tareas un determinado trabajador

```
wait (trabajador);
```

PROGRAMACIÓN CONCURRENTE EN M

- Finalmente, es preciso vaciar la zona de memoria ocupada por los trabajadores, una vez se hayan recuperado todos los resultados

```
delete (trabajador);
```

INTEGRACIÓN DE SUBRUTINAS C Y C++ CON ALGORITMOS EN CÓDIGO M

- En algunas ocasiones se desea realizar acciones que no pueden ser abordadas desde un lenguaje muy de alto nivel como M
  - Por ejemplo, clonar un proceso
- Otras veces, se desea explotar la eficiencia computacional de los lenguajes a más bajo nivel, o aprovechar un aspecto ventajoso de otro lenguaje

INTEGRACIÓN DE SUBRUTINAS C Y C++ CON ALGORITMOS EN CÓDIGO M

- También es posible querer emplear librerías escritas en otros lenguajes dentro de un algoritmo escrito en M
  - Por ejemplo el cliente SSH PuTTY (escrito en C)
- MATLAB incluye mecanismos para integrar código escrito en C, C++ y Java, principalmente
  - .NET y Phyton también se admiten, pero de momento su uso en MATLAB es muy marginal

INTEGRACIÓN DE SUBRUTINAS C Y C++ CON ALGORITMOS EN CÓDIGO M

- La integración de código Java es muy sencilla, ya que el motor de ejecución MATLAB integra una máquina virtual de Java
- Basta incluir los archivos compilados y comprimidos en un JAR en el *classpath* de MATLAB para poder crear objetos y emplear sus métodos

```
Objeto = ConstructorObjeto(param1, param2,…,paramn);
```

INTEGRACIÓN DE SUBRUTINAS C Y C++ CON ALGORITMOS EN CÓDIGO M

- La integración de código C y C++ es mucho más usual…
  - Al contrario que Java, C y C++ poseen funcionalidades de bajo nivel no cubiertas en M
- … y compleja
  - El fichero compilado no es independiente de la plataforma donde se ejecuta
  - Pueden necesitarse permisos especiales

INTEGRACIÓN DE SUBRUTINAS C Y C++ CON ALGORITMOS EN CÓDIGO M

- Al contrario que con Java, M no permite llamadas directas a rutinas C/C++
- Las rutinas deben contener una serie de elementos y funciones, y compilarse utilizando una herramienta especial de MATLAB
  - Compilador MEX
- Se obtendrá una librería contra la que ya se podrán realizar llamadas

INTEGRACIÓN DE SUBRUTINAS C Y C++ CON ALGORITMOS EN CÓDIGO M

- En primer lugar, en el fichero principal del programa C/C++ deben incluirse las siguientes líneas

```
#include “mex.h”
#include “matrix.h”
```

- Estos ficheros los agregará el compilador MEX

INTEGRACIÓN DE SUBRUTINAS C Y C++ CON ALGORITMOS EN CÓDIGO M

- La función principal de entrada *main()* típica de los programas C….

```
int main(int argc, char *argv[])
```

- …debe ser sustituida por la siguiente función

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

INTEGRACIÓN DE SUBRUTINAS C Y C++ CON ALGORITMOS EN CÓDIGO M

- **int nlhs:** Indica el número de argumentos de salida de la función
- **mxArray *plhs[]**: Array de punteros que “apuntan” a las variables de salida de la función
- **int nrhs:** Indica el número de argumentos de entrada de la función
- **mxArray *prhs[]:** Array de punteros que “apuntan” a las variables de entrada de la función

INTEGRACIÓN DE SUBRUTINAS C Y C++ CON ALGORITMOS EN CÓDIGO M

- El compilador MEX hace uso de los compiladores externos apropiados a la arquitectura hardware subyacente, por lo que es precisa una fase de configuración previa a su uso.
- Para ello basta ejecutar

```
mex -setup
```

INTEGRACIÓN DE SUBRUTINAS C Y C++ CON ALGORITMOS EN CÓDIGO M

- Una vez configurado el compilador MEX basta invocarlo seguido del la ruta del fichero para compilar

```
mex C:/ruta/archivo.c
```

- La extensión del fichero compilado depende del equipo de trabajo, pero en general varía entre *.mex*, *.mexw64* y *.dll*

INTEGRACIÓN DE SUBRUTINAS C Y C++ CON ALGORITMOS EN CÓDIGO M

- Agregando la librería obtenida al *path* de MATLAB se podrá invocar la nueva rutina como si se tratase de una función nativa de M
- MUY IMPORTANTE: El nombre que MATLAB asocia a esta nueva función es el del fichero que contiene el código

GENERACIÓN DE CÓDIGO C Y C++ EMBEBIDO A PARTIR DE CÓDIGO M

- En algunos casos, se desea crear algoritmos que en M son muy sencillos, pero que se pretenden ejecutar en plataformas hardware como software embebido
  - Arduino, PICs genéricos, etc.
- En la inmensa mayoría de estas plataformas, el código debe ser C o C++

GENERACIÓN DE CÓDIGO C Y C++ EMBEBIDO A PARTIR DE CÓDIGO M

- Codificar dichos algoritmos directamente en C o C++ puede ser altamente costoso.
- Ejemplo: producto de matrices

M

```
R = A*B;
```

C

```
for (i = 0 ; i < num_fil1 ; i++ ) {
    for (k = 0 ; k < num_col2 ; k++ ) {
        temporal = 0 ;
        for (j = 0 ; j < numcol1 ; j++ ) {
            temporal += A[i][j] * B[j][k];
            R[i][k] = temporal ;
        }
    }
}
```

GENERACIÓN DE CÓDIGO C Y C++ EMBEBIDO A PARTIR DE CÓDIGO M

- Desde el año 2011 junto con MATLAB se distribuye el componente *MATLAB Coder*
- *MATLAB Coder* genera automáticamente código C y C++ **legible y portable** directamente desde algoritmos escritos en M
- *MATLAB Coder* se complementa con *Embedded Coder* cuando se desean implementaciones muy optimizadas y eficientes

GENERACIÓN DE CÓDIGO C Y C++ EMBEBIDO A PARTIR DE CÓDIGO M

- *Embedded Coder*, sin embargo, puede complicar la legibilidad del código
- En la mayoría de los casos bastará con emplear *MATLAB Coder*
- Aunque es menos común, el mismo procedimiento descrito para traducciones a C/C++ se puede aplicar a Java, .NET o (desde 2015) Phyton

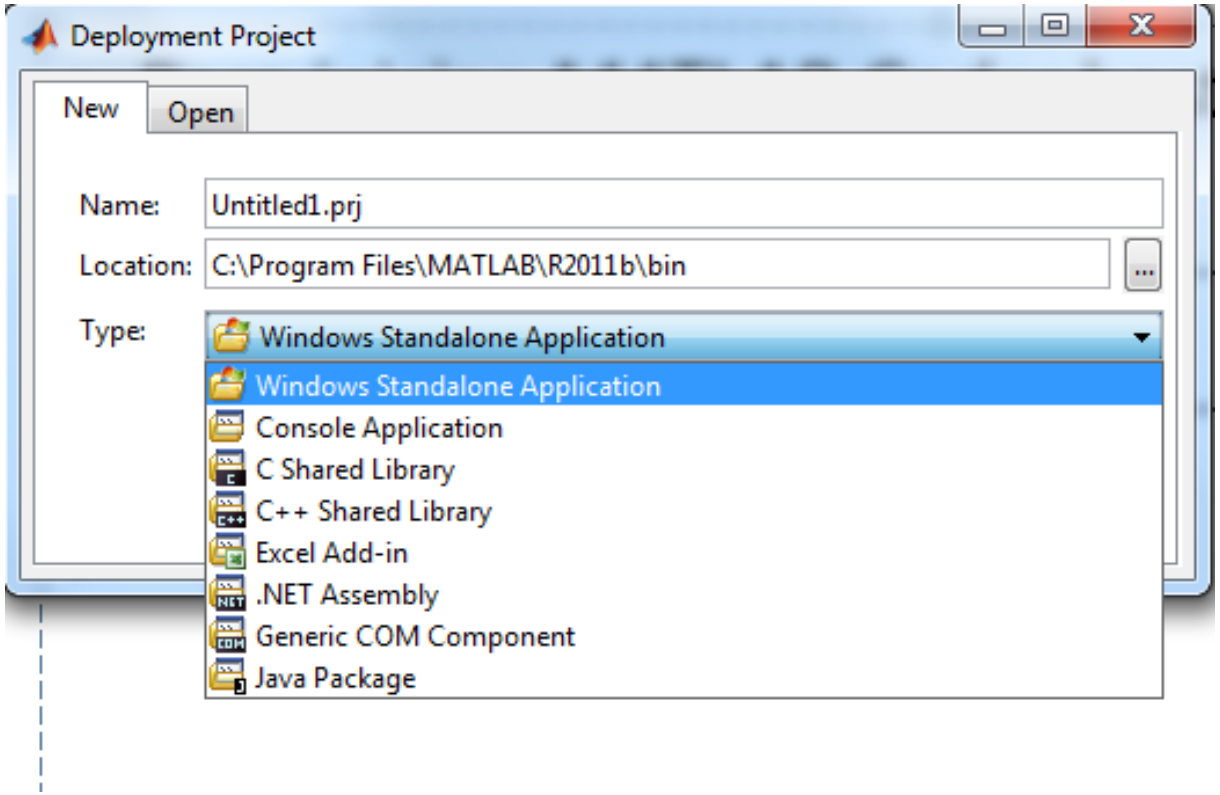
GENERACIÓN DE CÓDIGO C Y C++ EMBEBIDO A PARTIR DE CÓDIGO M

- Para iniciar *MATLAB Coder* basta lanzar el siguiente comando

```
deploytool
```

- Se abrirá una nueva ventana para seleccionar las características del código de salida

GENERACIÓN DE CÓDIGO C Y C++ EMBEBIDO A PARTIR DE CÓDIGO M



GENERACIÓN DE CÓDIGO C Y C++ EMBEBIDO A PARTIR DE CÓDIGO M

- Se escoge el nombre de la librearía y el tipo de salida
  - C shared library
  - C++ shared library
- Al aceptar se abrirá un nuevo marco en la interfaz de MATLAB donde se deberá seleccionar el archivo principal y todas las funciones de las que haga uso

GENERACIÓN DE CÓDIGO C Y C++ EMBEBIDO A PARTIR DE CÓDIGO M

- Seleccionados los ficheros se presiona el icono de *Build*
- *MATLAB Coder* dará como salida dos ficheros
  - Un proyecto *MATLAB Coder* (extensión .prj)
  - La librería en el lenguaje seleccionado

GENERACIÓN DE CÓDIGO C Y C++ EMBEBIDO A PARTIR DE CÓDIGO M