Ingeniería del software con MATLAB: gestión de versiones, pruebas unitarias y distribución de código

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

- Control de versiones en MATLAB
- Marcos de pruebas unitarias para MATLAB
- Distribución de código y aplicaciones MATLAB

CONTROL DE VERSIONES EN MATLAB

- Debido a su origen como programa de cálculo numérico matricial, los usuarios de MATLAB nunca han precisado de herramientas específicas para la gestión de versiones
- Con la expansión de MATLAB a partir de 2010 hacia otras áreas (incluyendo la ingeniería telemática), los nuevos usuarios comenzaron a demandar estas utilidades

CONTROL DE VERSIONES EN MATLAB

- Desde 2011 MATLAB permite su integración con diversos sistemas de control de versiones, aunque de forma nativa sólo lo hace con dos
  - Git
  - Subversion
- Otros sistemas de control de versiones precisan de la instalación previa de módulos de conexión, cuyo funcionamiento no tiene soporte de MathWorks

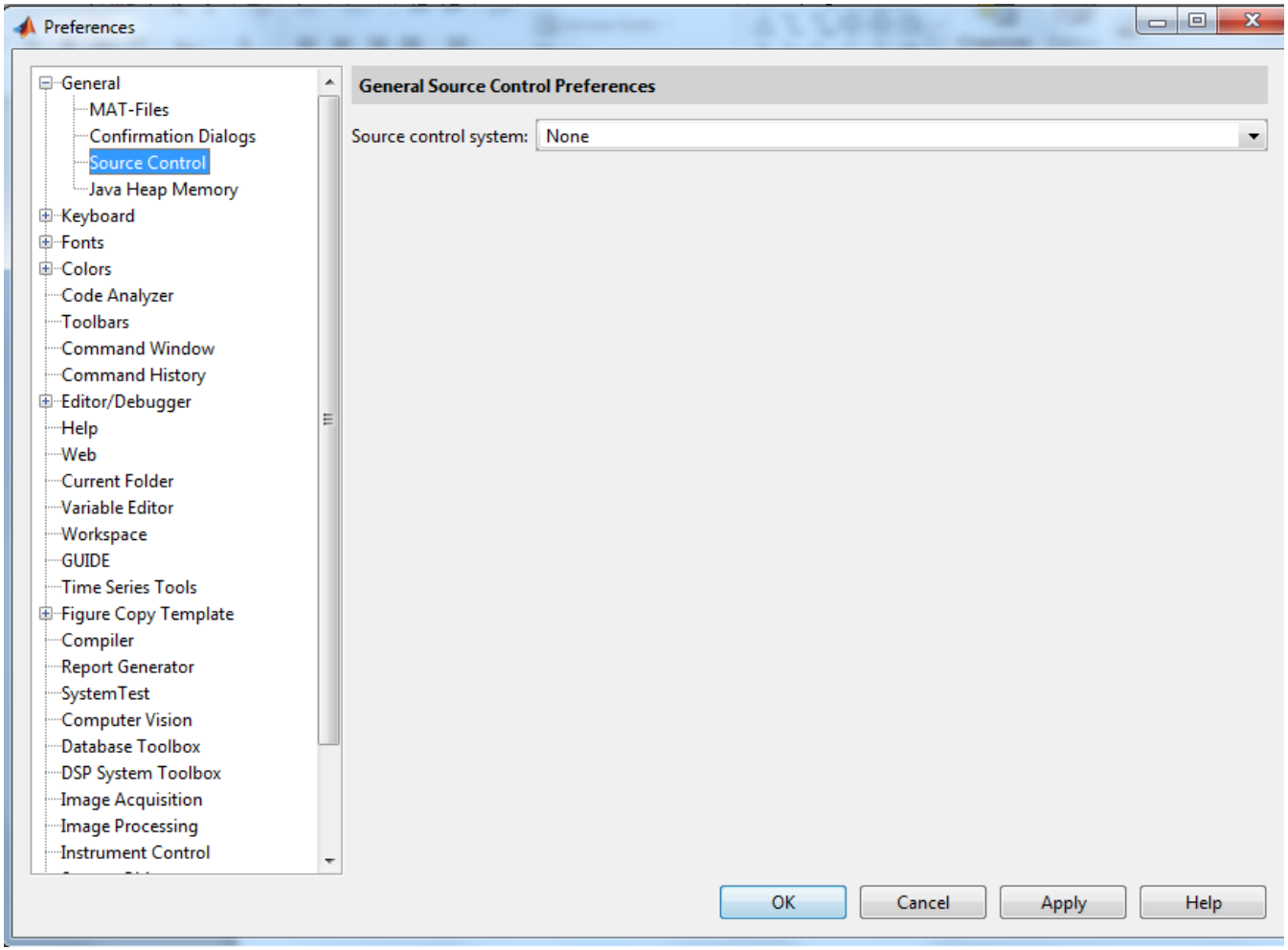
CONTROL DE VERSIONES EN MATLAB

- Para integrar un sistema de control de versiones con MATLAB primero debe habilitarse esta opción
- Dependiendo de la versión de MATLAB esto se encuentra en
  - File > Preferences > General > Source Control
  - Home > Environment > Preferences > MATLAB > General > Source Control

CONTROL DE VERSIONES EN MATLAB

- En el menú desplegable se debe seleccionar
  - *Enable MathWorks source control integration*, para habilitar el control de versiones (Git o Subversion)
  - *None*, para desabilitar el control de versiones
- Si se deshabilita el control de versiones tras haberlo tenido activo, MATLAB deja de mostrar la información relativa a las versiones del código, pero no elimina los repositorios creados, ni locales ni remotos

CONTROL DE VERSIONES EN MATLAB

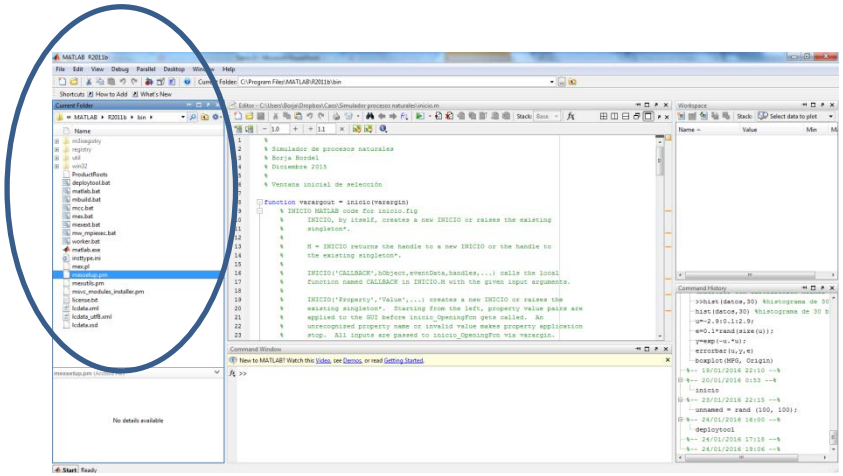


CONTROL DE VERSIONES EN MATLAB

- MATLAB se distribuye junto con la última versión de la herramienta *Apache Subversion* disponible en cada momento
- Es la forma “natural” de gestionar versiones en MATLAB, aunque no hay diferencias de rendimiento con el uso de *Git*
- Al tratarse de una funcionalidad totalmente integrada, puede activarse directamente en el panel *Source Control*

CONTROL DE VERSIONES EN MATLAB

- Para ello se hace click derecho en el navegador de ficheros

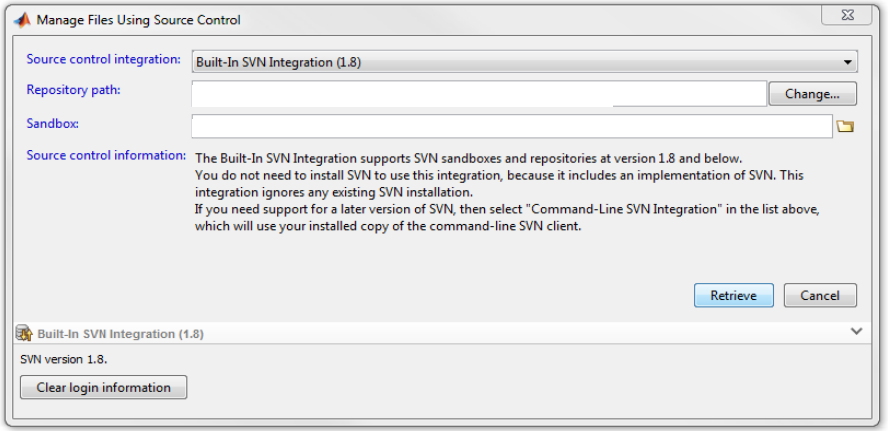


- Se accede al panel de gestión de ficheros, localizado en *Source Control > Manage Files*

CONTROL DE VERSIONES EN MATLAB

- Aparecerá una nueva ventana, donde se debe indicar
  - *Source control integration: Build-in SVN version*
  - *Repositor path:* La URL del repositorio donde se almacena el código. Si es preciso, al finalizar la edición del campo aparecerá una ventana de *log-in*
- Se pulsa *Retrieve* al finalizar

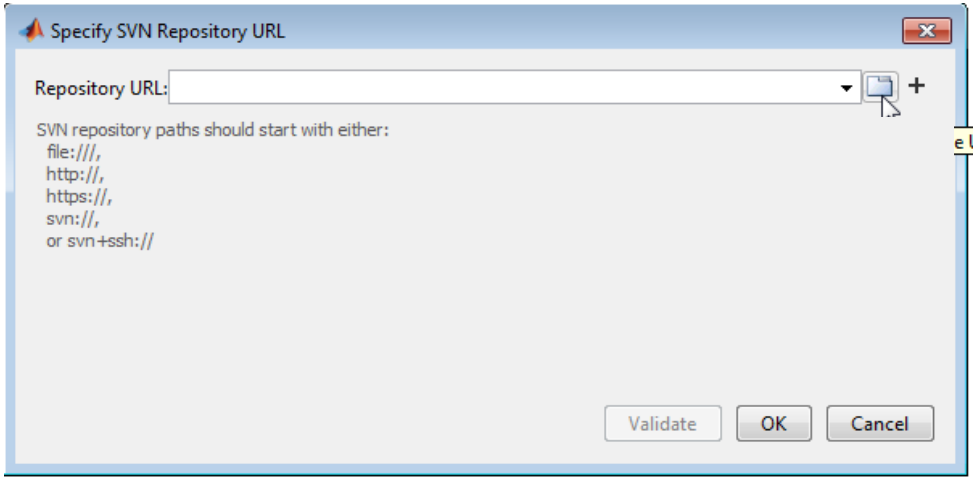
CONTROL DE VERSIONES EN MATLAB



- También es posible modificar el repositorio en caso de error.
- Para ello se accede a *Source Control > Manage Files*

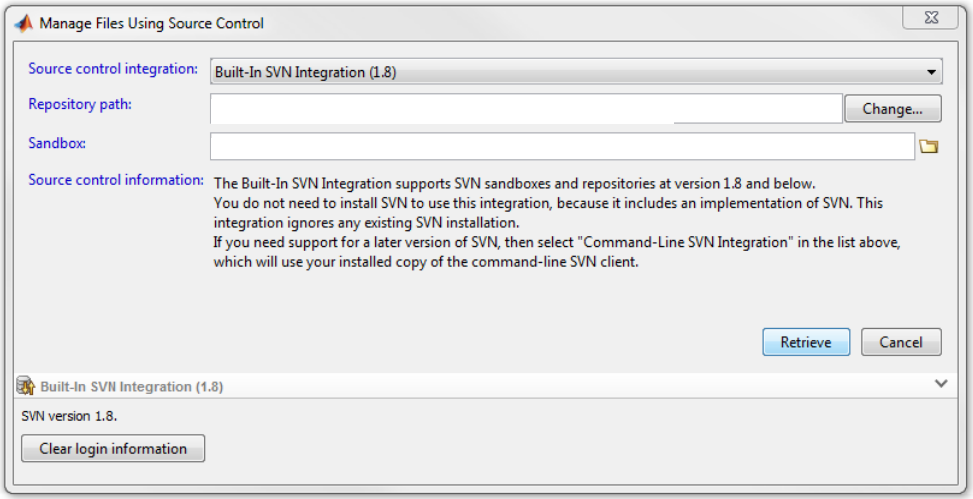
CONTROL DE VERSIONES EN MATLAB

- Si la ruta que aparece no coincide con la del repositorio deseado, pulsamos *Change*
- Aparecerá una nueva ventana



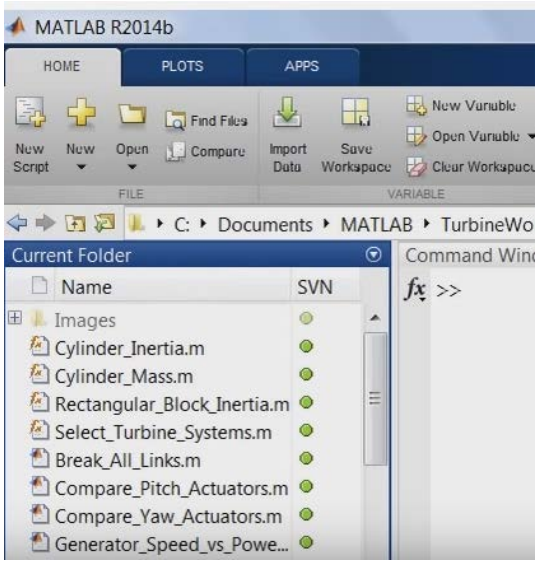
CONTROL DE VERSIONES EN MATLAB

- Introducida la URL se presiona *Validate*
- Si el repositorio lo requiere, aparecerá una ventana de *log-in*
- Finalmente se mostrará un resumen de la configuración. Pulsamos *Retrieve*



CONTROL DE VERSIONES EN MATLAB

- Una vez MATLAB ha accedido al repositorio, en el navegador de ficheros aparecerá una nueva columna, indicando el estado de cada fichero
- Un círculo verde indica que el fichero está actualizado a la última versión

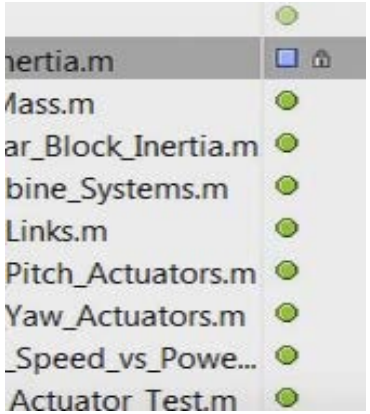


CONTROL DE VERSIONES EN MATLAB

- Puesto que *Subversion* trabaja sobre un único repositorio central, para evitar problemas de edición simultánea, los archivos deben ser bloqueados antes de editarlos
- Basta hacer click derecho sobre el archivo en el navegador y seleccionar *Source Control > Get File Locked*
- En la columna dedicada al control de versiones aparecerá un candado

CONTROL DE VERSIONES EN MATLAB

- Una vez se ha editado el fichero, el círculo verde cambiará por un cuadrado azul, indicando que hay cambios pendientes de notificar



CONTROL DE VERSIONES EN MATLAB

- Haciendo click derecho sobre el archivo en el navegador y seleccionando *Source Control > Commit to SVN repository* se envían los cambios al repositorio central y el archivo se desbloquea automáticamente
- Dentro del panel *Source Control* de cada fichero aparecen otras opciones como “reversión de cambios” o “comparación de versiones” que pueden ser de utilidad

CONTROL DE VERSIONES EN MATLAB

- Si se desea utilizar *Git* como gestor de cambios es necesario primero instalar esta herramienta en el sistema
- En Linux basta con teclear

```
sudo apt-get install git
```

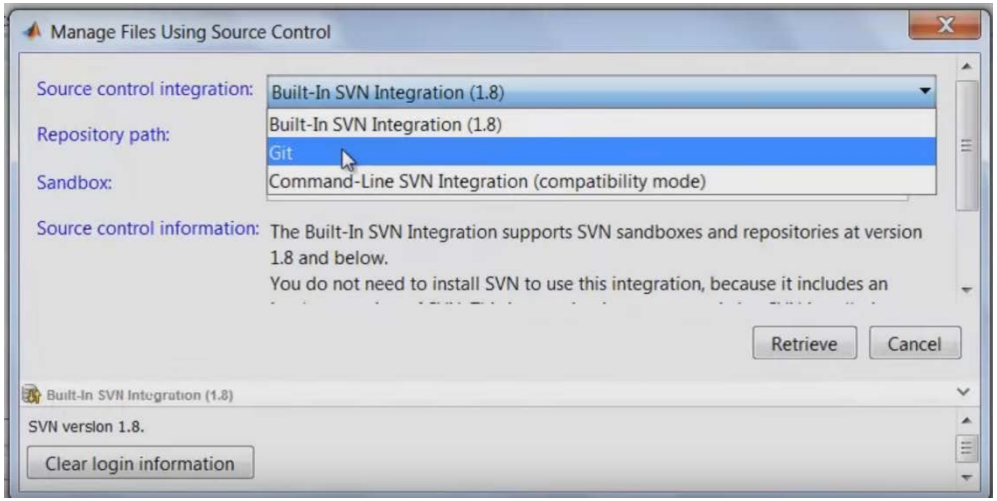
CONTROL DE VERSIONES EN MATLAB

- En Windows, en la ventana de instalación hay que marcar las opciones
  - Use Git from the Windows Command Prompt
  - Line-ending conversions > Checkout as-is, commit as-is
- Para activar el control de versiones con *Git* se hace click derecho en el navegador de ficheros y se selecciona *Source Control > Manage Files*

CONTROL DE VERSIONES EN MATLAB

- Aparecerá una nueva ventana, donde se debe indicar
  - *Source control integration: Git*
  - *Repositor path:* La URL del repositorio común donde se almacena el código. Si es preciso, al finalizar la edición del campo aparecerá una ventana de *log-in*
- Se pulsa *Retrieve* al finalizar

CONTROL DE VERSIONES EN MATLAB



- A partir de aquí el modo de proceder es igual al visto para *Apache Subversion*

CONTROL DE VERSIONES EN MATLAB

- Como única diferencia, en *Git* no existe la opción de “bloquear archivo” ya que cada usuario cuenta con su propia copia local del repositorio
- Dentro del panel *Source Control* de cada fichero aparecen otras opciones que pueden ser de utilidad como “refresco de estado” (para ver los últimos cambios producidos por otros usuarios) o “reversión de cambios”

MARCOS DE PRUEBAS UNITARIAS PARA MATLAB

- Tradicionalmente, las pruebas de algoritmos escritos en MATLAB se realizan mediante *frameworks* externos
  - Sobre todo se emplean marcos del tipo xUnit
    - mIUnit http://sourceforge.net/projects/mlunit/
    - Lombardi's Munit
    - Legland's MUnit
  - Doctest

MARCOS DE PRUEBAS UNITARIAS PARA MATLAB

- Con la versión R2008a se desarrolló un primer intento de marco de pruebas propio
  - MATLAB xUnit Test Framework
- Resultó muy limitado y su uso fue escaso
- No formaba parte del paquete MATLAB

MARCOS DE PRUEBAS UNITARIAS PARA MATLAB

- Finalmente, con la versión R2013a, se desarrolló y distribuyó como parte integrante de MATLAB un nuevo *framework*
  - *MATLAB Unit Testing*
  - Su código es abierto (aunque no libre) y puede encontrarse en el paquete *matlab.unittest*

MARCOS DE PRUEBAS UNITARIAS PARA MATLAB

- *MATLAB Unit Testing* permite escribir pruebas unitarias en tres formas distintas
  - Scripts
  - Funciones
  - Clases
- Cada tipo de pruebas permite niveles de complejidad distintos
  - Las más simples serán scripts, las más complejas clases

MARCOS DE PRUEBAS UNITARIAS PARA MATLAB

- Las pruebas en forma de scripts se emplean para comprobar que los resultados producidos por scripts, funciones y clases son los esperados
- Sólo dos funciones están permitidas en estas pruebas
  - *assert*, para comprobar el valor, tipo , tamaño, etc. de los resultados
  - *runtests*, para ejecutar las pruebas

MARCOS DE PRUEBAS UNITARIAS PARA MATLAB

- Típico uso de la función *assert*

```
assert (salida == esperado, 'Mensaje en caso de error')
```

- Invocación con *runtests*

```
resultado = runtests ('Nombre del script de pruebas');
```

MARCOS DE PRUEBAS UNITARIAS PARA MATLAB

- Ejecutando el script de pruebas mediante la función *runtests* nos aseguramos de que se ejecutan todas las pruebas
- Si el script de pruebas se ejecutase como un script ordinario, MATLAB finalizaría la ejecución al fallar el primer *assert*

MARCOS DE PRUEBAS UNITARIAS PARA MATLAB

- Las pruebas en forma de funciones también se emplean para comprobar que los resultados producidos por scripts, funciones y clases son los esperados
- El tipo de comparaciones que permiten estas pruebas se sitúa alrededor de la centena
  - Relación de igual, mayor, menor, contenido, fallos incondicionales, etc.

MARCOS DE PRUEBAS UNITARIAS PARA MATLAB

- Las pruebas en forma de funciones se basan en una función principal acompañada de tantas funciones locales como sea necesario
- La función principal tiene un prototipo y un código fijo

```
function tests = nombre_a_alegir
       tests = funtiontests (localfunctions);
end
```

MARCOS DE PRUEBAS UNITARIAS PARA MATLAB

- El nombre de la función principal debe ser el nombre del fichero de pruebas
- Esta función crea un array de celdas conteniendo manejadores de funciones (punteros) para cada una de las funciones locales incluidas
- En dichas funciones se realizarán las pruebas

MARCOS DE PRUEBAS UNITARIAS PARA MATLAB

- Puede haber tantas funciones locales como se desee
- Su contenido es libre, pero su prototipo no

```
function nombre_a_alegir (testCase)
        …
end
```

- Por claridad, cada función suele asociarse con un caso de prueba

MARCOS DE PRUEBAS UNITARIAS PARA MATLAB

- Para comparar los resultados obtenidos del script, función o clase bajo pruebas con los resultados esperados se emplean las funciones de calificación
- Las funciones de calificación pueden clasificarse en cuatro tipos
  - Verificaciones: Ejecuta el test y, si falla, no producen ninguna excepción. El test continua

MARCOS DE PRUEBAS UNITARIAS PARA MATLAB

- Supuestos: El test sólo se ejecuta si se da una condición. Si se produce un fallo, el test continua
- Aseveraciones: Comprueban una determinada condición e informan
- Aseveraciones fatales: Comprueban una determinada condición. Si no se cumple, el test se aborta

- http://es.mathworks.com/help/matlab/matlab_prog/types-of-qualifications.html

MARCOS DE PRUEBAS UNITARIAS PARA MATLAB

- Existen, finalmente, cuatro funciones especiales que pueden ser incluidas en las pruebas si se considera necesario
  - *File Fixture Functions*
    - *setupOnce*, se ejecuta antes de iniciar todas las pruebas
    - *teardownOnce*, se ejecuta al finalizar todas las pruebas
  - *Fresh Fixture Functions*
    - *setup*, se ejecuta antes de iniciar cada prueba
    - *teardown*, se ejecuta al finalizar cada prueba

MARCOS DE PRUEBAS UNITARIAS PARA MATLAB

- El contenido de estas funciones es libre, pero su prototipo no

```
function setupOnce (testCase)
        …
end
```

```
function teardownOnce (testCase)
        …
end
```

MARCOS DE PRUEBAS UNITARIAS PARA MATLAB

```
function setup (testCase)
        …
end
```

```
function teardown (testCase)
        …
end
```

MARCOS DE PRUEBAS UNITARIAS PARA MATLAB

- La invocación de la ejecución de las pruebas es ligeramente diferente en este caso

```
resultado = runtests ('nombre_funcion_principal.m');
```

- Es muy importante incluir la extensión .m al invocar la batería de pruebas

MARCOS DE PRUEBAS UNITARIAS PARA MATLAB

- Se puede suprimir la extensión .m, pero en tal caso el nombre de la función principal se pasa como manejador, no como cadena de caracteres

```
resultado = runtests (nombre_funcion_principal);
```

MARCOS DE PRUEBAS UNITARIAS PARA MATLAB

- Las pruebas en forma de clases permiten comprobaciones de mucha más profundidad que las alternativas anteriores
- En este caso se trabaja directamente sobre las funciones del paquete *matlab.unittest*
- El enfoque y estructura de estas pruebas es igual al visto para xUnit
  - JUnit en Laboratorio de Programación

MARCOS DE PRUEBAS UNITARIAS PARA MATLAB

- Las pruebas en forma de clases permiten test paramétricos, etiquetar las pruebas o emplear bloques de código compartido por varias pruebas
- Su uso, estructura, comprensión y manejo es mucho más complejo
- Aprender a emplear este tipo de pruebas no forma parte de este curso

DISTRIBUCIÓN DE CÓDIGO Y APLICACIONES MATLAB

- Una vez se ha finalizado y probado una rutina, función, aplicación, clase… es necesario decidir qué hacer con ese nuevo código
- Si se opta por su distribución existen tres alternativas
  - Como código abierto
  - Como librería compilada
  - Como aplicación ejecutable

DISTRIBUCIÓN DE CÓDIGO Y APLICACIONES MATLAB

- Sea cual sea el caso, MATLAB pone a disposición de todos los programadores MATLAB una comunidad en la que distribuir las nuevas contribuciones
  - http://es.mathworks.com/matlabcentral/?s_tid=gn_mlc

DISTRIBUCIÓN DE CÓDIGO Y APLICACIONES MATLAB

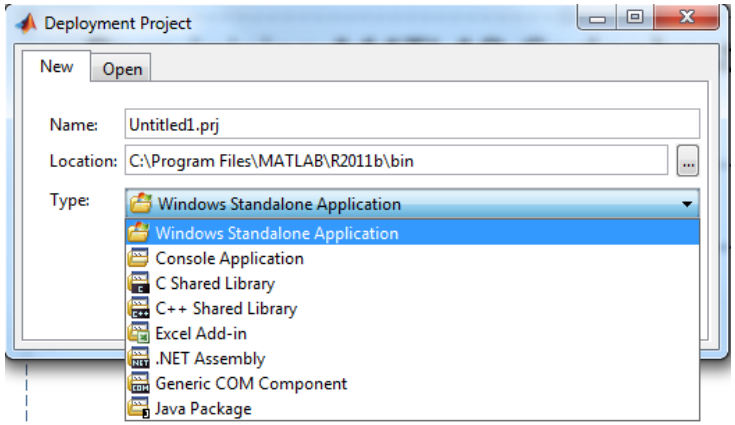
- Si se desea una distribución en código abierto, basta publicar los ficheros fuente
- Si se desea distribuir una librería compilada se debe hacer en forma de fichero JAR de Java
  - Recordemos que el lenguaje M es interpretado o compilado Just-in-time (según la versión)
  - El proceso se explicó en el Tema 2

DISTRIBUCIÓN DE CÓDIGO Y APLICACIONES MATLAB

- Existe una alternativa, en desuso: los fichero .p
- Son fichero pre-compilados que no permiten ver el código escrito en M
- Tienen un gran problema: no son independientes de la plataforma de compilación
- No se van a explicar al estar en vías de desaparición

DISTRIBUCIÓN DE CÓDIGO Y APLICACIONES MATLAB

- Finalmente, puede hacerse una distribución en forma de aplicación ejecutable
- Para ello se hará uso de la herramienta *MATLAB Coder* vista en el tema 2



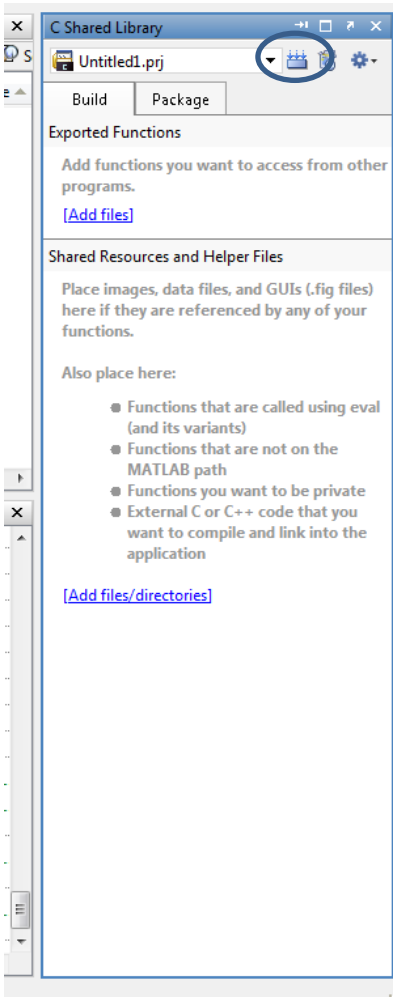
DISTRIBUCIÓN DE CÓDIGO Y APLICACIONES MATLAB

- En el desplegable se selecciona *Windows Standalone Application*
  - O el equivalente que aparezca en cada plataforma
- A continuación se puede optar por
  - Una aplicación que sólo se ejecute en sistemas que tenga MATLAB instalado
  - Una aplicación totalmente independiente de MATLAB

DISTRIBUCIÓN DE CÓDIGO Y APLICACIONES MATLAB

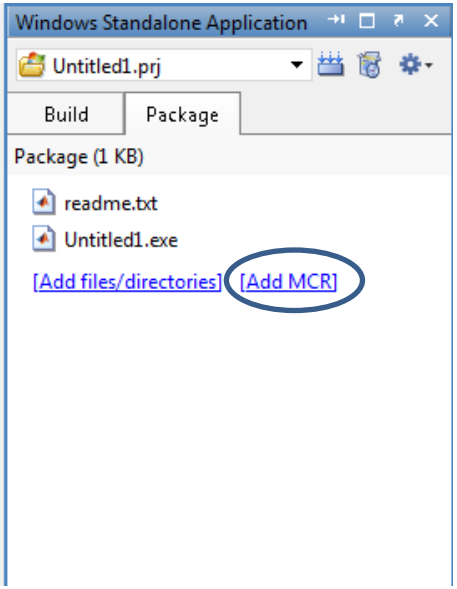
- Al aceptar en la ventana *popup*, se abrirá un nuevo marco en la interfaz de MATLAB donde se deberá seleccionar el archivo principal y todas las funciones de las que haga uso
- Si la aplicación va a necesitar de un MATLAB instalado en el sistema destino, seleccionados los ficheros se presiona el icono de *Build*

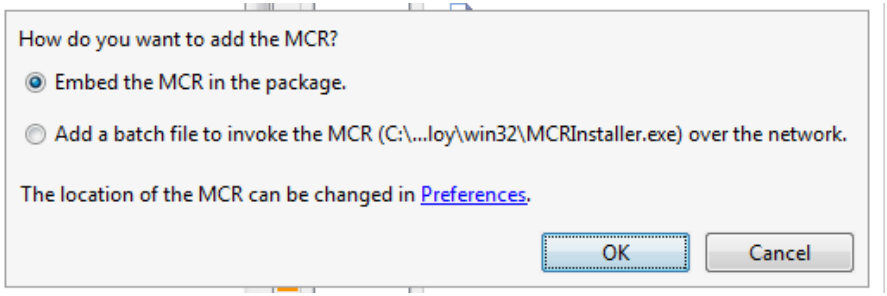
DISTRIBUCIÓN DE CÓDIGO Y APLICACIONES MATLAB



DISTRIBUCIÓN DE CÓDIGO Y APLICACIONES MATLAB

- Si se desea crear una aplicación independiente de MATLAB, en la pestaña *Packaje* antes se debe seleccionar “Add MCR”





DISTRIBUCIÓN DE CÓDIGO Y APLICACIONES MATLAB

- Al añadir el MCR (*MATLAB Compiler and Runtime*), junto con el código propio de nuestra rutina o aplicación se incluye el motor de ejecución de MATLAB
- En el sistema destino, al lanzar la aplicación, en primer lugar se desplegará el motor, donde luego se ejecutará nuestro código