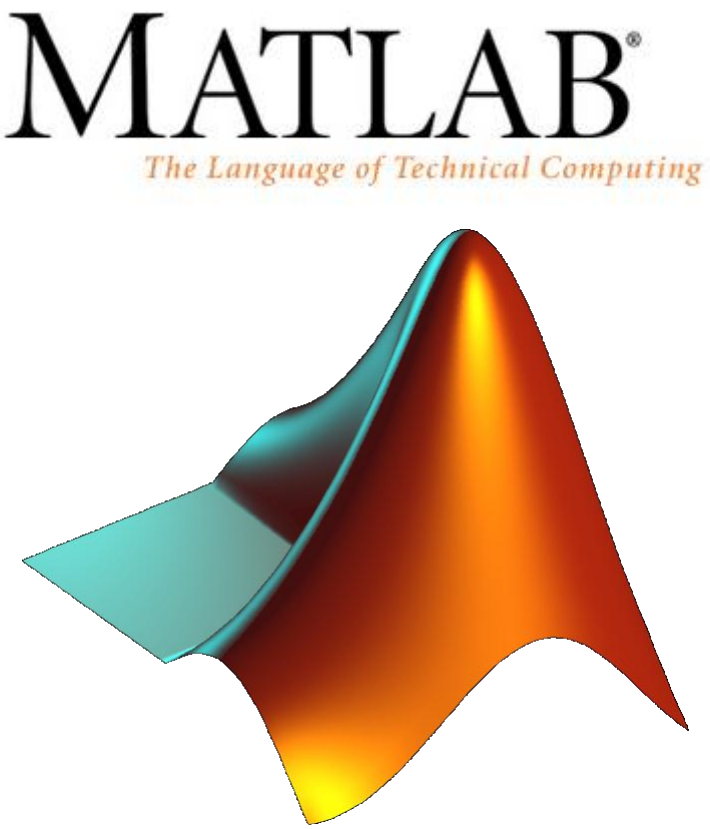


III CONVOCATORIA PARA LA REALIZACIÓN DE ACTIVIDADES DE FORMACIÓN DOCENTE EN CENTROS, TITULACIONES Y DEPARTAMENTOS

Taller Práctico

Uso de MATLAB para la docencia en Ingeniería Química y áreas afines

Miguel García Román

Universidad de Granada, 13 de septiembre de 2012

Índice

Parte I. Breve repaso del uso de MATLAB

Parte II .Ejemplos de aplicación de MATLAB para la docencia en Ingeniería Química

Parte I

Breve repaso del uso de MATLAB

1. Introducción y objetivos del taller

Según lo definen sus desarrolladores MATLAB es "un entorno de programación para el desarrollo de algoritmos, el análisis de datos, la visualización y el cálculo numérico". Las aplicaciones de MATLAB son por tanto muy extensas, lo que unido su gran potencia de cálculo y al hecho de poseer un lenguaje de programación propio, han hecho de este programa uno de los más utilizados en el cálculo técnico.

El principal objetivo de este taller es presentar las posibilidades de MATLAB como herramienta didáctica, así como analizar las ventajas e inconvenientes que comporta su uso en este sentido. Sin embargo, comenzaremos realizando un breve repaso sobre el manejo de MATLAB para aquellos que no estén familiarizados con el programa. Este repaso no será para nada exhaustivo, por lo que remito a la bibliografía que se aporta al final de estas notas, o a la propia ayuda del programa para ampliar la información aquí recogida.

2. El entorno de trabajo de MATLAB.

La pantalla principal de MatLab cuenta con tres subventanas que por defecto estarán presentes la primera vez que se abre. Éstas son:

- Command Window, que es la que nos permite introducir las órdenes o comandos en el programa.
- Command History Window, que muestra la secuencia de comandos ejecutados
- Current Directory Window/Workspace, muestran los programas presentes en el directorio actual, así como las variables asignadas con sus valores, respectivamente.

Una imagen de la pantalla principal puede observarse en la Figura 1. Aunque en mi opinión la mejor opción es trabajar con el escritorio por defecto, es posible personalizar la pantalla principal añadiendo o quitando subventanas, o modificando su tamaño. Todas estas opciones están disponibles a través de los iconos situados en la esquina superior derecha de cada una de las subventanas o en el menú Desktop (en la barra de menús).

El menú Desktop también contiene una opción que permite restaurar la organización por defecto (Desktop Layout -> Default), que resulta muy útil cuando un alumno ha perdido "sin saber cómo" alguna de las ventanas principales.

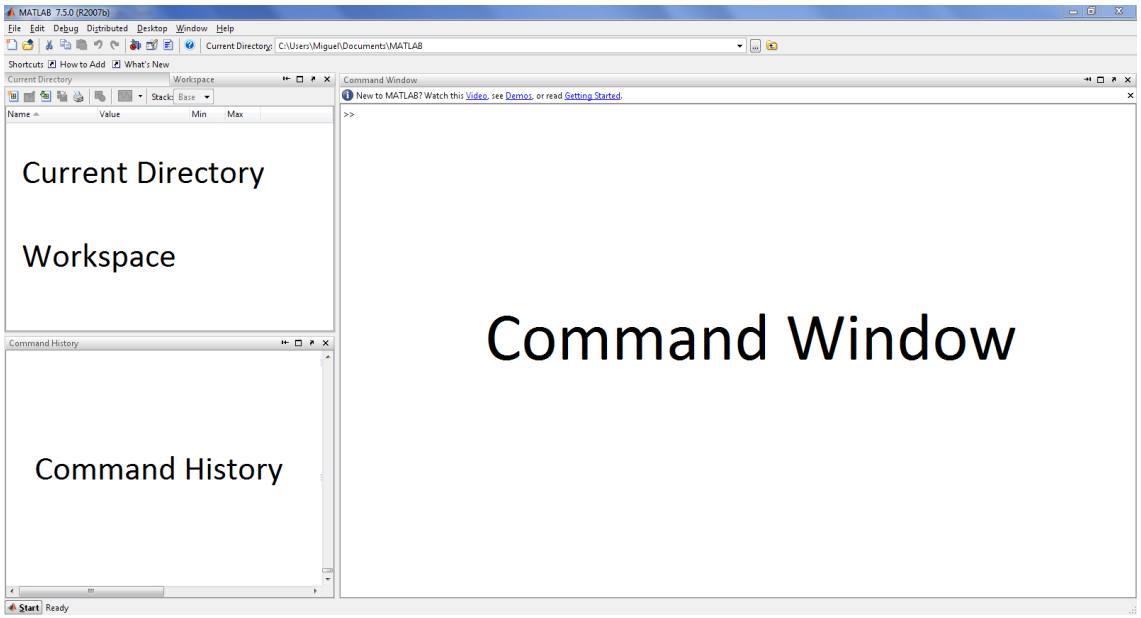


Figura 1. Pantalla principal de MATLAB

3. Tipos de datos y su introducción en MATLAB.

Las matrices constituyen la forma básica de dato en MATLAB, nombre que precisamente es un acrónimo de **Mat**rix **Lab**oratory. No obstante, muy frecuentemente trabajaremos con vectores fila o columna, más que con matrices bidimensionales, así como con escalares. Los elementos de una matriz pueden ser números, variables predefinidas, caracteres o funciones matemáticas

La introducción de los datos puede llevarse a cabo de forma directa, a través de la Ventana de Comandos, aunque existe la posibilidad importar datos desde otros programas, por ejemplo Excel. Para ello existen varias alternativas.

a. El comando **`dlmread`**, que permite importar datos desde ficheros ASCII delimitados. Funciona muy bien con ficheros 'csv' muy fáciles de generar desde EXCEL.
b. La opción 'Import data', del menú 'File'
c. Copiar y pegar directamente los datos. Para ello debemos crear previamente la variable en MATLAB y abrirla en el editor de matrices (simplemente haciendo doble clic sobre ella desde el 'Workspace'). El editor de matrices tiene una organización similar al EXCEL, por lo que resulta muy fácil pegar datos sobre él.

A continuación se muestran ejemplos de la introducción manual de un vector fila, un vector columna y una matriz bidimensional.

```
>> A=[1 2 3]
A =
     1     2     3
>> B=[1;2;3]
B =
     1
     2
     3
>> C=[1 2 3;4 5 6]
C =
     1     2     3
     4     5     6
```

Otra forma de introducir vectores es especificar el primer término (*a*), la diferencia entre cada dos términos (*d*), y el último término (*z*), p. ej., **`x=[a:d:z]`**. Es también posible crear un vector especificando el primer y el último elemento (*a* y *b*, p. ej.), así como el número total de elementos del vector (*n*). La distancia entre dos elementos sucesivos dependerá, evidentemente, del número total de términos que deseemos. Para hacer esto se usa el comando **`linspace`**, de la forma siguiente: **`x=linspace(a,b,n)`**. Si no se especifica el número de elementos se considera 100 por defecto.

Una vez introducida un vector o matriz es conveniente conocer algunos comandos para trabajar con ellos:

- Para obtener el elemento i-ésimo de un vector A, ya sea fila o columna, escribiremos `A(i)`
- Si es una matriz, tendremos que especificar tanto la fila como la columna en la que se encuentra el elemento en cuestión. Así si se encuentra en la fila 'i' y en la columna 'j', escribiremos: `A(i,j)`. De esta forma es posible también asignar nuevos valores al elemento que queramos, sin modificar el resto de la matriz.
- Si queremos especificar un rango de elementos de un vector o matriz, usaremos "dos puntos", de la siguiente manera:
    - `A(m:n)`: Se refiere a los elementos comprendidos entre las posiciones 'm' a 'n' del vector A.
    - `M(:,n)`: Se refiere a todos los elementos de la columna 'n' de la matriz M.
    - `M(m,:)`: Se refiere a todos los elementos de la fila 'm' de la matriz M.

- Para conocer el número de elementos (filas o columnas) de un vector se usa el comando `length`. Para matrices podemos usar el comando `size`, que nos da las dimensiones de las mismas.

Mención especial merecen también en este punto los formatos numéricos en MATLAB. El formato por defecto (formato ***short***) muestra sólo cuatro decimales. Los otros formatos disponibles son:

- Formato ***long***: muestra hasta 14 decimales
- Formato ***short e***/***long e***: análogos ***short*** y ***long***, pero en notación científica.
- Formato ***short g*** y ***long g***: elige la mejor opción entre ***short*** o ***short e*** y ***long*** o ***long e***.

Para cambiar entre los distintos formatos se puede usar el comando **`format`** (el cambio sólo es válido para la presente sesión) o bien hacerlo desde el Menú File→Preferences, en el apartado Command Window (el cambio se mantiene al entrar y salir del programa).

Será necesario modificar los formatos cuando en una misma matriz coexistan datos de muy diferente orden de magnitud, que pueden dar lugar a problemas en la visualización de los resultados obtenidos, tal y como se observa en el siguiente ejemplo.

```
>> a=[1.5777889 0.0000005 548129453975]

a =

  1.0e+011 *

    0.0000    0.0000    5.4813

>> format short e
>> a=[1.5777889 0.0000005 548129453975]

a =

  1.5778e+000  5.0000e-007  5.4813e+011

>> format short g
>> a=[1.5777889 0.0000005 548129453975]

a =

       1.5778       5e-007  5.4813e+011
```

4. Operaciones en MATLAB

4.1. Operaciones con escalares y matrices

Los operadores básicos en MATLAB para escalares y matrices son los que se muestran en la Tabla 1. Las operaciones pueden realizarse directamente con números, o bien con variables, a las que previamente se les asigna un valor numérico (un escalar o una matriz).

Tabla 1. Operadores de MATLAB

Operador	Operación
+	Suma
–	Resta
*	Multiplicación
/ ó \	División
^	Potencia

Algunos operadores específicos para matrices son: **`inv`** (para obtener la matriz inversa), ' ' (para obtener la traspuesta) y **`det`** para obtener el determinante.

De todos los anteriores, el único operador que puede resultar novedoso es la división a la derecha (\), que implica que el dividendo se sitúe a la derecha del divisor, al contrario que en la división convencional, como puede verse en el ejemplo siguiente:

```
>> a=15/5

a =
     3
>> b=5\15

b =
     3
```

La principal utilidad de este operador es la resolución de sistemas de ecuaciones lineales, como se muestra en el ejemplo, cuya resolución en MATLAB admite dos formas posibles:

$$\begin{aligned} 3x+2y-z&=10\\ 8x-3y+4z&=21\\ 5x+2y+z&=13 \end{aligned} \qquad \begin{bmatrix} 3 & 2 & -1\\ 8 & -3 & 4\\ 5 & 2 & 1 \end{bmatrix} * \begin{bmatrix} x\\ y\\ z \end{bmatrix} = \begin{bmatrix} 10\\ 21\\ 13 \end{bmatrix}$$

Con la matriz inversa de la de coeficientes:

```
>> A=[3 2 -1;8 -3 4;5 2 1];

>> B=[10;21;13];

>> x=inv(A)*B

x =
        3.225
         -0.7
       -1.725
```

O bien con la división a la derecha (\):

```
>> x=A\B

x =
        3.225
         -0.7
       -1.725
```

4.2. Operaciones por elementos

Es muy frecuente en el manejo de MATLAB que necesitemos realizar una determinada operación sobre un conjunto de valores, que normalmente vendrán contenidos en un vector fila o columna. Así por ejemplo podemos multiplicar (o dividir) o sumar (o restar) un escalar por/a todos los elementos de un vector, usando los operadores normales (* ó +). Sin embargo, cuando se trata de, por ejemplo, elevar todos los elementos de un vector a una potencia determinada, el operador ^ no resulta válido, ya que el programa entenderá que se trata del producto del vector fila o columna por sí mismo, que no es posible en el álgebra matricial. Esto puede verse en el ejemplo siguiente.

```
>> a=[1 2 3];

>> b=5*a

b =

     5    10    15

>> c=a^2

??? Error using ==> mpower
Matrix must be square.
```

Al similar ocurre cuando se pretende dividir un escalar por los elementos de un vector o multiplica o dividir dos vectores con el mismo número de elementos. Para poder llevar a cabo las anteriores operaciones hay que indicar que lo que se pretende es realizarlas ***por elementos***, es decir, generar un nuevo vector de las mismas dimensiones que el/los

originales constituido por el resultado de realizar la operación pretendida sobre cada uno de los elementos de el/los vectores originales. Para ello basta anteponer un punto al operador correspondiente, como se observa a continuación:

```
>> c=a.^2
c =
     1     4     9
```

Las operaciones por elementos son bastante frecuentes en cualquier aplicación de MATLAB y suelen presentar dificultades para los alumnos que en muchos casos no llegan a entender qué representan exactamente, aunque no se trate nada complejo. Quizá estas dificultades se deban a que los alumnos no recuerdan el álgebra matricial o no acaban de entender la forma de operar de un determinado programa de MATLAB. La no utilización o el uso incorrecto de las operaciones por elementos genera mensajes de error del tipo "Matrix dimensions must agree", cuya causa inicial suele ser difícil de detectar, ya que son numerosos los fallos que pueden dar lugar a este tipo de mensaje. Por ello un recurso frecuente es poner el punto antes de todos los operadores, aunque no resulte necesario.

5. Representaciones gráficas

La generación de gráficos en dos dimensiones en MATLAB se lleva a cabo usando el comando `plot`, cuya sintaxis es la siguiente:

```
plot(x,y,'especificadores de línea','propiedades','valores')
```

- '**x**' e '**y**' son dos vectores con el mismo número de elementos, siendo el primero de ellos el que contiene los valores de la variable x (eje de abscisas) y el segundo de la variable y (eje de ordenadas).
- **Los especificadores de línea** son los argumentos que definen el tipo y aspecto de la línea y/o marcador que se usará para la representación de los datos. En la Tabla 2 siguiente se resumen los más importantes:

Tabla 2. Especificadores de línea para el comando `plot`

Estilo línea	Especif.	Color	Especif.	Marcador	Especif.
Continua	-	rojo	r	signo más	+
Discontinua	--	azul	b	círculo	o
De puntos	:	magenta	m	asterisco	*
Rayas y puntos	-.	verde	g	punto	.
		cian	e	cuadrado	s
		amarillo	y	diamante	d
		blanco	w	estrella de cinco puntas	p
		negro	k	estrella de seis puntas	h

Para utilizar estos especificadores hay que hacerlo en la secuencia: 'color → Estilo de línea → Estilo de marcador'. Por ejemplo: 'b-*' dibujaría una línea continua azul, con asteriscos azules como marcadores.

- **Las propiedades y valores** permiten modificar la apariencia de la curva o los marcadores, actuando concretamente sobre el grosor de la línea, el tamaño de los marcadores, el color de borde y de relleno de los marcadores.

Por ejemplo, si deseamos graficar la función $y=2x^2+1$ en el intervalo [0,20], lo haríamos de la forma siguiente:

```
>> a=linspace(0,20);
>> b=2*a.^2+1;
>> plot(a,b,'r')
```

Lo que daría lugar al gráfico que se muestra en la Figura 2.

Existen otras opciones para la representación gráfica, como es el comando `fplot`, que permite representar directamente una función predefinida de MATLAB, de las que hablaremos más adelante.

Cuando se desea agregar series a un gráfico se ha de usar el comando `hold on`, que se ha de ejecutar una vez creado el gráfico con la primera serie. Los gráficos que se generen posteriormente se agregarán al anterior como series sucesivas. El comando `hold off` tiene el efecto contrario.

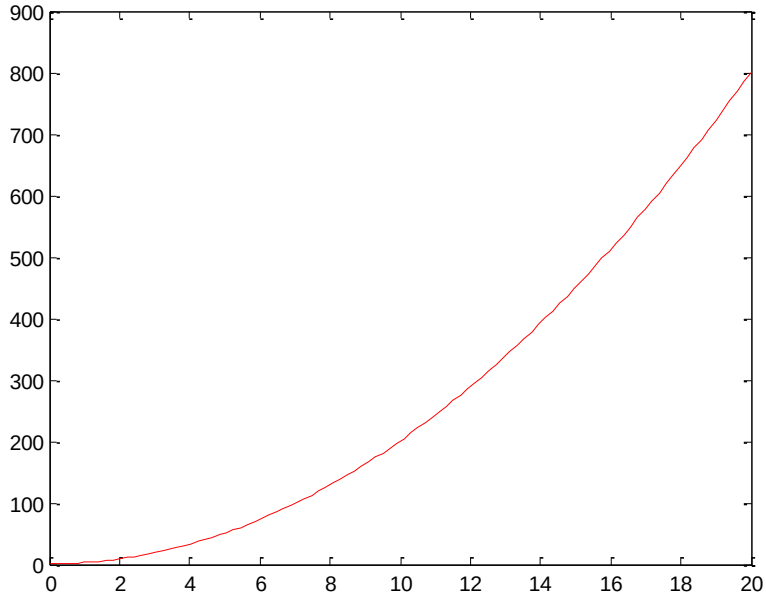


Figura 2. Representación grafica de la función $y=2x^2+1$ en el intervalo [0,20]

6. Funciones en MATLAB

Una gran cantidad de funciones matemáticas se encuentran predefinidas en MATLAB. Para usarlas hay que teclear el nombre de la función y entre paréntesis los argumentos correspondientes, de los cuales se puede obtener información detallada en la ayuda del programa. En la Tabla 3 se recogen algunas de las funciones matemáticas elementales en MATLAB.

Tabla 3. Funciones elementales en MATLAB

Función	Descripción
`sqrt(x)`	raíz cuadrada
`exp(x)`	exponencial(e^x)
`abs(x)`	valor absoluto
`log(x)`	logaritmo natural
`log10(x)`	logaritmo decimal
`sin(x)`	seno
`cos(x)`	coseno
`factorial(x)`	(factorial de x, x!)

Otras funciones importantes y de uso frecuente en MATLAB se relacionan a continuación:

- `fzero`: permite hallar las raíces de una función cualquiera

- `quadl`: para la integración numérica
- `trapz`: válida para la integración numérica pero usa la regla de los trapecios, partiendo de un conjunto de valores de la función a integrar.
- `fminbnd`: halla el mínimo de una función en un intervalo determinado. Sirve también para hallar el máximo, que será obviamente el mínimo de la función inversa.
- `ode45`: comando para la integración numérica de sistemas de ecuaciones diferenciales.

En los ejemplos que se trabajaremos en el presente taller se profundizará más en el manejo de estas funciones.

6.1. Creación de funciones (ficheros función)

A parte de las funciones predefinidas el usuario de MATLAB puede crear sus propias funciones que son realmente secuencias de código con una sintaxis definida. Una vez escrito el código de la función, para lo cual es aconsejable usar el editor de MATLAB (basta pulsar en el icono ***New M-file*** en la barra de herramientas superior), ésta se graba como un programa con el mismo nombre de la función creada y ya puede invocarse desde la ***Command window***. Es importante tener en cuenta que para poder usar una función debemos encontrarnos en el mismo directorio (***Current directory***) en el que fue guardada. Si no fuera así el programa no reconocerá el nombre de la función, lo que es un error también habitual entre los estudiantes que comienzan a usar el programa.

Cualquier fichero función ha de constar de las siguientes partes:

1) **Definición de la función:** Ha de incluirse en la primera línea de texto ejecutable. Tiene la siguiente estructura:

```
function [argumentos de salida] =
nombre_función (argumentos de entrada)
```

- La palabra '`function`', en minúsculas, debe ser la primera que aparezca.
- A continuación deben incluirse los argumentos de salida, es decir la/s variable/s donde MATLAB almacenará el resultado de los cálculos. Si son varios deben ir separados por comas y entre corchetes.

- Después del signo igual se escribe el nombre de la función, con el que luego se la invocará.
- A continuación del nombre se deben indicar el o los argumentos de entrada, es decir, la/s variable/s cuyos valores hay que dar a la función para que realice los cálculos u operaciones que posteriormente se especificarán. Si son varios los argumentos, deben ir separados por comas.

2) **Líneas de ayuda**, se trata de líneas de comentario. Van detrás de el signo % y en ellas se describe la función y como usarla. Son opcionales, pero de gran utilidad.
3) **Cuerpo de la función,** que define las operaciones y cálculos a realizar con los argumentos de entrada para obtener los argumentos de salida.

En el siguiente ejemplo se presenta el código de una función que se ha denominado `esg` y que permite calcular las raíces de una ecuación de segundo grado:

```
function y=esg(a,b,c)
%Función que calcula las raíces de una ecuación de 2º grado
y1=(-b+(b^2-4*a*c)^(1/2))/(2*a);
y2=(-b-(b^2-4*a*c)^(1/2))/(2*a);
y=[y1,y2];
```

La función `esg` tiene como argumento de salida el vector `y`, que contiene las raíces de la ecuación (`y1` e `y2`), y como argumento de entrada los tres coeficientes de cada uno de los términos de la ecuación de 2º grado (`a`, `b` y `c`).

La sintaxis para la definición de funciones en MATLAB es ciertamente poco intuitiva, por lo que suele resultar difícil de asimilar para los estudiantes que se inician en el manejo del programa. Sin embargo no deja de ser más que una sintaxis, por lo que una vez familiarizados con ella no debe presentar dificultades adicionales.

6.2. Las variables globales

Las variables usadas por cualquier función de MATLAB son por defecto propias de esa función, o dicho de otra forma, se gestionan de forma independiente de las usadas por otras funciones o las presentes en el ***workspace*** de MATLAB (estas variables del ***workspace*** son las asignadas a través de la ***command window*** o mediante programas). Así por ejemplo, la función `esg` que acabamos de definir asigna la salida a la variable 'y', pero dicha variable

no aparece en el ***workspace*** cuando la función se ejecuta, y si queremos conocer su valor y tecleamos 'y' en la ***command window*** obtendremos la salida:

```
??? Undefined function or variable 'y'.
```

De igual forma, las funciones no tienen acceso a las variables definidas en ***command window*** y almacenadas en el ***workspace***. Esto puede apreciarse en el siguiente ejemplo en el que se define una función, isotopo, que representa el proceso de desintegración de un isótopo radioactivo a partir de la vida media del mismo, dato que constituye el argumento de entrada. Otro dato necesario es el tiempo total para la representación, tt. Este último se introduce por la ***command window***, lo que produce el resultado siguiente al ejecutar la función:

```
function y=isotopo(x)
t=linspace(0,tt);
y=2.^(-t/x);
plot(t,y)
xlabel('t, dias')
ylabel('N/No')

>> tt=1000;
>> isotopo(138);
??? Undefined function or variable 'tt'.
Error in ==> isotopo at 2
t=linspace(0,tt);
```

Como puede apreciarse la variable tt no es reconocida por la función, y por tanto no se puede ejecutar ésta. En estos casos, que son muy frecuentes, MATLAB requiere la definición de la variable tt como ***global***, con lo que podrá ser asignada en la ***command window*** y leída por la función. Esta definición como global hay que hacerla tanto en el cuerpo de la función como en el propia ***command window***, como puede verse a continuación:

```
function y=isotopo(x)
global tt
t=linspace(0,tt);
y=2.^(-t/x);
plot(t,y)
xlabel('t, dias')
ylabel('N/No')

>> global tt
>> tt=1000;
>> isotopo(138);
```

Lo que genera la salida mostrada en la Figura 3.

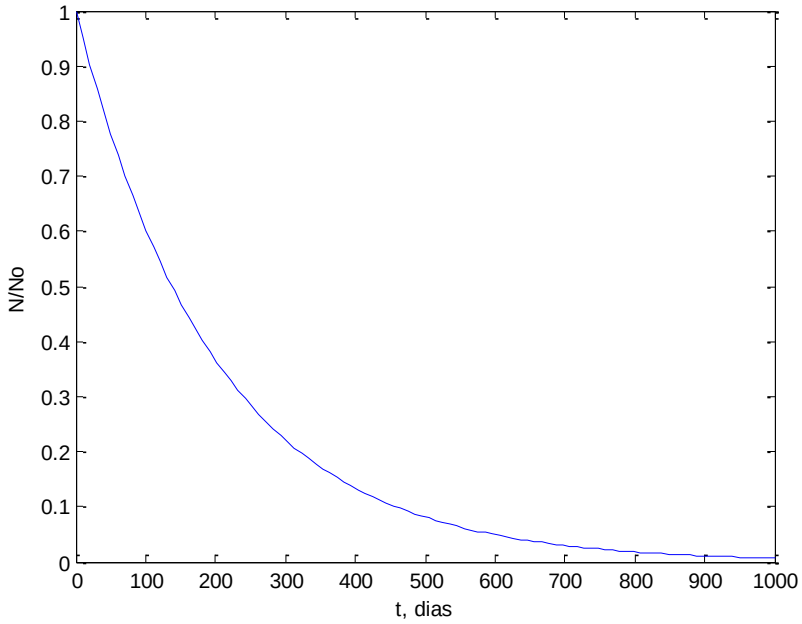


Figura 3. Salida de la función 'isotopo'

Es necesario familiarizar al alumno con el concepto de variables globales, ya que su no definición o definición incorrecta conlleva errores frecuentes y que no siempre son fáciles de detectar, sobre todo cuando es necesario asignar muchas variables.

7. Ficheros de programa y programación en MATLAB.

Hasta ahora hemos usado la ***command window*** para asiganr variables, ejecutar funciones, realizar operaciones, etc., es decir, para todo menos para la definición de funciones. Sin embargo, el uso de la ***command window*** obliga a ir introduciendo las instrucciones una a una, lo cual no es cómodo cuando se trata de un número elevado de ellas. Para estos casos se recurre a escribir programas ("scripts") que permiten, al ejecutarse, realizar una serie de acciones de una sola vez. Estos programas se escriben normalmente, aunque no obligatoriamente, en el editor de MATLAB y se deben guardar en el ***current directory*** para poder ejecutarlos, tal y como sucede con las funciones. Una vez guardado, para ejecutar un fichero script, basta con escribir su nombre en la ***command window***.

7.1. Operadores relacionales y operaciones lógicas.

En la Tabla 4 se incluyen los operadores relacionales que se utilizan en MATLAB. Estos operadores se usan en operaciones lógicas, cuya salida es siempre un 1 (verdadero) o un 0 (falso)

Tabla 4. Operadores relacionales en MATLAB

Operador relacional	Descripción
<	Menor que.
>	Mayor que.
<=	Menor o igual que.
>=	Mayor o igual que.
==	Igual a.
~=	Distinto de.

7.2. Sentencias de programación.

Abordaremos a continuación la sintaxis de los principales tipos de sentencias de programación en MatLab:

- **Sentencias condicionales:**

Permiten ejecutar una serie de comandos si se verifican unas condiciones previamente especificadas. La instrucción que se usa es '`if ... end`'. También es frecuente especificar una serie de instrucciones u otra según se cumplan unas u otras condiciones. Para ello se emplean las estructuras '`if ... else ... end`' o '`if ... elseif ... else ... end`', para 2 ó mas de 2 grupos de instrucciones, respectivamente.

En el siguiente ejemplo se presenta una función que utiliza sentencias condicionales para dar la calificación de un alumno en función de su nota numérica.

```
function y=nota(x)
if x>=9
    y=('sobresaliente');
elseif x>=7
    y=('notable');
elseif x>=5
    y=('aprobado');
else
    y=('suspenso');
end
```

- **Bucles:**

Permiten ejecutar una instrucción o conjunto de ellas un número determinado de veces. Los más importantes son los siguientes:

Bucle for-end

Su peculiaridad es que tenemos que conocer, desde el primer momento, el número de veces que la instrucción se repetirá. La estructura del mismo es:

```
for i=f:s:t
……
…… (serie de comandos en MATLAB)
……
end
```

`i` es la variable que se irá actualizando.

`f` es el primer valor que toma la variable `i`.

`s` es el incremento de la variable `i`. Si no se especifica, es 1 por defecto.

`t` es el valor máximo (o mínimo, si `s` es negativo) que alcanzará la variable `i`. Una vez alcanzado, los comandos comprendidos entre `for` y `end`, no se ejecutarán más.

En el ejemplo siguiente se presenta una función que usa el bucle for-end para construir un vector fila con el número deseado de unos.

```
function y=unos(x)
for i=1:x
    y(i)=1;
end
```

Bucle while-end

En este caso no se conoce el número exacto de ejecuciones de los comandos del bucle que serán necesarias, por lo que éstos se ejecutarán mientras se cumpla una determinada condición, que obviamente debe especificarse. Su estructura por tanto, es:

```
while expresión_condicional
……
…… (serie de comandos en MATLAB)
……
end
```

Para que esta estructura funcione es imprescindible que al menos una de las variables de la sentencia condicional cambie de valor cada vez que se ejecutan los comandos. En el siguiente ejemplo se presenta una función que utiliza un bucle while-end para comprobar si en una serie de datos (vector x) se supera un valor previamente especificado (m). En caso de que así sea informa del valor y su posición (índice) dentro del vector, la primera vez que esto ocurre. Si no es así muestra un mensaje indicando que no se supera dicho valor.

```
function y=comprueba(x,m)
j=1;
while j<=length(x)&&x(j)<m
    j=j+1;
```

```
end
if j>length(x)
    y=('No se supera el valor especificado');
else
y1=x(j);
y2=j;
y=[y1,y2];
end
```

8. Interfaces gráficas de usuario (GUI) en MATLAB.

Las interfaces gráficas de usuario (GUI) constituyen una utilidad muy interesante que permite ejecutar programas complejos sin necesidad de poseer conocimientos de MATLAB. La creación de una GUI se ve facilitada enormemente gracias al asistente GUIDE del que dispone MATLAB.

Aunque la creación de GUI puede llegar a ser un proceso complejo cuyo abordaje en detalle excede los objetivos de este taller, dedicaremos este último capítulo a describir los pasos elementales para su creación, ya que pueden tener gran utilidad en docencia, al permitir al alumno obviar los detalles de programas de MATLAB relativamente complicados y usar MATLAB solamente como una herramienta para comprender y profundizar los contenidos de la asignatura. Esto puede particularmente adecuado para alumnos de grado, especialmente en los primeros cursos, pero no tanto para los de máster, donde debería pretenderse que los alumnos desarrollaran la capacidad de escribir sus propios programas en MATLAB.

8.1. GUI para sumar dos números.

Es un ejemplo muy sencillo, por lo que lo usaremos de introducción. Siempre que se crea una GUI lo ideal es hacer previamente un esquema en papel de su aspecto final, que en este caso se muestra en la Figura 4.

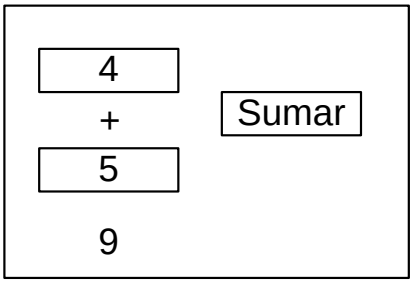


Figura 4. Esquema de la GUI para sumar dos números

Para construir esta GUI usaremos el asistente GUIDE, que se inicia escribiendo 'guide' en la Command Window o bien mediante el icono correspondiente en la barra de menús.

Una vez en GUIDE seleccionamos la opción 'Blank GUI (default)', con lo que se abrirá el panel de diseño completamente vacío. A la izquierda tenemos los controles para insertar los distintos componentes de la interfaz, que para identificar más fácilmente conviene que incluyan el nombre de cada uno. Para ello entramos en File -> Preferences y activamos la opción 'Show names in component palette'.

La GUI que pretendemos diseñar contiene:

- Dos cuadros de texto editable ('Edit text'), que servirán para introducir los números a sumar.
- Un cuadro de texto estático para poner el símbolo '+' (esto es simplemente un "adorno")
- Un cuadro de texto estático ('Static text') para mostrar el resultado (que el texto sea estático no quiere decir que no cambie, sino que no se puede editar por el usuario.
- Un botón para ejecutar la suma (`Push button').

Estos componentes se toman del menú (paleta) de la izquierda y se arrastran a la posición deseada, debiendo quedar un diseño (´layout') similar al de la Figura 5.

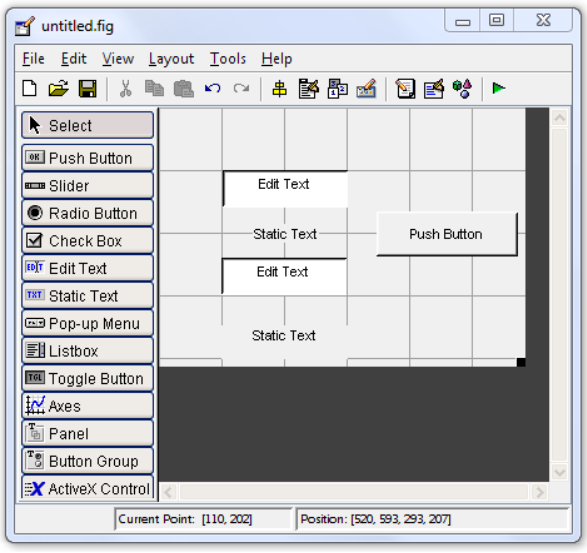


Figura 5. Aspecto del layout de la GUI suma

Para alinear los elementos que forman el Layout de la GUI (no muy necesario en este caso) se puede usar la opción 'Align Objects' a la que se accede mediante el menú Tools o el icono correspondiente.

Las propiedades de cada uno de los componentes se pueden modificar con el 'Property Inspector' que se despliega al hacer doble clic en el componente deseado. Entre las propiedades hay dos muy importantes:

- 'String': Es el texto que se muestra en cada componte.
- 'Tag': Es el nombre con el que se identifica el componente. Luego será necesario para hacer referencia al mismo en el código de MATLAB asociado a la GUI.

Vamos a cambiar los dos textos editables para poner 0 en cada uno de ellos, que será lo que aparezca cuando se inicie la GUI, así como en el texto estático donde se mostrará el resultado. En el otro texto estático pondremos '+' y en el 'push button' pondremos sumar, con lo que la GUI toma el aspecto de la Figura 6.

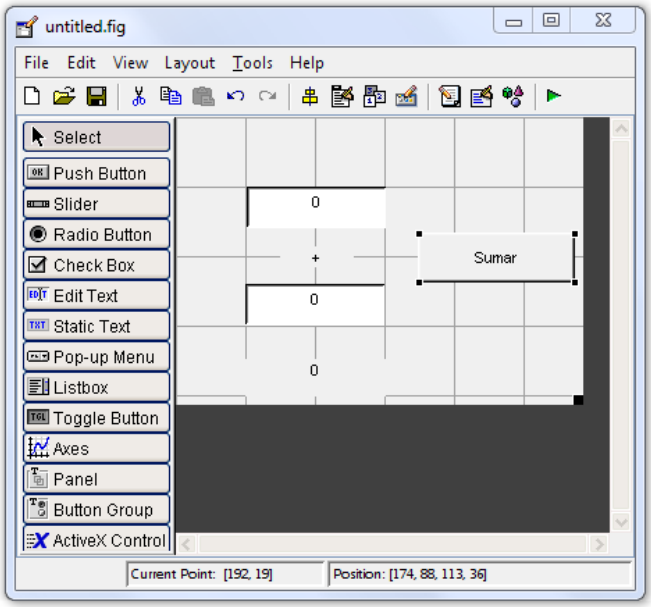


Figura 6. Aspecto de la GUI suma tras editar las propiedades de los componentes.

Ya podemos grabar la GUI para realizar las modificaciones necesarias sobre el código de MATLAB. Le daremos el nombre 'suma'. Al hacerlo se crean dos archivos, un archivo suma.fig, que contiene el diseño realizado, y otro suma.m con el código asociado. Para ejecutar la GUI siempre se debe ejecutar el archivo .m, no el .fig.

Al guardar se abre automáticamente el archivo suma.m que tendremos que modificar para que realice las operaciones que deseamos. En síntesis hay que indicarle que lea los datos que introducimos en las celdas editables, y al pulsar el 'pushbutton' los sume y muestre el resultado en la celda estática designada para ello.

Para obtener el valor de la celda editable y convertirlo a formato 'double' el código necesario es:

```
get(handles.nombre_elemento,'nombre_propiedad')
```

Por ejemplo:

```
get(handles.edit1,'String')
```

El término 'handles' hace referencia a un objeto de MATLAB, que en concreto es el edit1. El problema es que en la celda de texto lo que tenemos es una cadena de texto, por lo que habrá que transformarlo en formato 'double', para lo que se usa la orden 'str2double'

Para fijar el valor de un campo de texto estático de forma que muestre el resultado de un cálculo, se puede usar la orden:

```
set(handles.nombre_elemento,'nombre_propiedad',nombre_variable)
```

Por ejemplo:

```
set(handles.text2,'String',sum)
```

Como todas estas operaciones las tiene que hacer el programa al pulsar el 'push button' este código se debe escribir en la parte del m-file, concretamente una función, que describe la acción de dicho componente. En la jerga de las GUI esta función asociada a los componentes (no todos las tienen) se llama 'Callback'. Para localizar la del 'push button' usamos el icono 'Show functions' de la barra de menús, y hacemos clic sobre 'pushbutton1_Callback', lo que nos lleva a la zona del código que vamos a editar. En ella insertamos lo siguiente:

```
a=str2double(get(handles.edit1,'String'));
b=str2double(get(handles.edit2,'String'));
sum=a+b;
set(handles.text2,'String',sum);
```

Ya solo queda guardar nuevamente y ejecutar la GUI que hemos construido.

8.2. GUI para representar gráficamente una función.

Este ejemplo muestra el uso del componente 'axes. Lo que hará la GUI es representar gráficamente la desintegración radioactiva de un isótopo de vida media conocida, la cual se suministrará por pantalla. Su aspecto será el de la Figura 7.

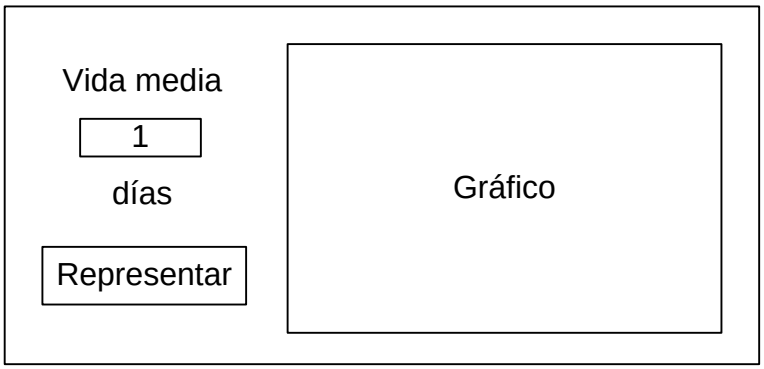


Figura 7. Esquema de la GUI para representar la desintegración radioactiva.

La única novedad con respecto al ejemplo anterior la supone la introducción del gráfico. Por lo demás el aspecto final del layout tras seguir un procedimiento similar al del ejemplo anterior, se muestra en la Figura 8.

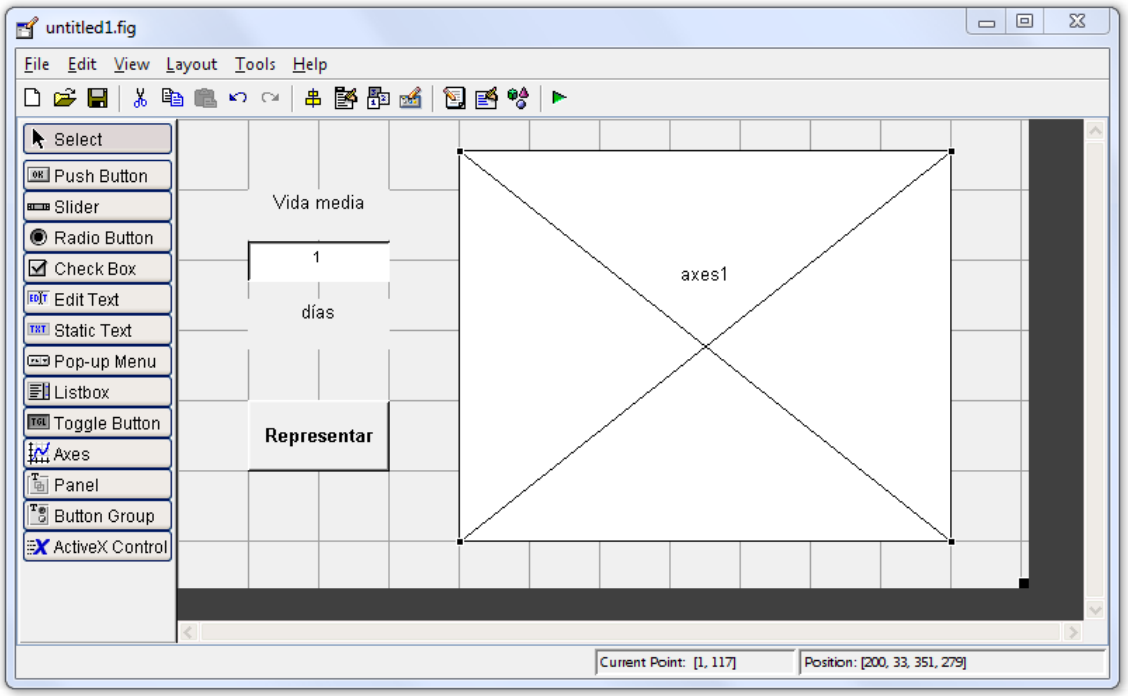


Figura 8. Layout de la GUI para representar la desintegración radioactiva.

Seguidamente grabamos la GUI llamándola 'desinteg' y editamos el código. En este caso es conveniente que inicialmente el gráfico desplegado muestre alguna representación. En el m-file de una GUI existe siempre una función denominada 'nombre_GUI'_OpeningFcn, en este caso desinteg_OpeningFcn, que sirve para este fin. La localizamos y le añadimos:

```
axes(handles.axes1); %activa el gráfico cuyo Tag es axes1
t=linspace(0,10);
y=2.^(-t/1); %función desintegración para vida media de 1 día
plot(t,y)
set(handles.axes1,'YTick',[0;0.1;0.2;0.3;0.4;0.5;0.6;0.7;0.8;0.9;1.0])
%fija la escala para el eje y
xlabel('t, días')
ylabel('N/No')
```

Finalmente en la Callback del 'pushbutton1' añadimos:

```
vm=str2double(get(handles.edit1,'String'));
axes(handles.axes1);
t=linspace(0,10*vm);
y=2.^(-t/vm);
plot(t,y)
set(handles.axes1,'YTick',[0;0.1;0.2;0.3;0.4;0.5;0.6;0.7;0.8;0.9;1.0])
xlabel('t, días')
ylabel('N/No')
```

Hay que tener en cuenta que cada vez que se dibuja un gráfico se borra todo contenido anterior por eso es necesario volver a indicar los títulos de los ejes respectivos y la escala del eje y.

8.3. Generación de otras GUI.

Obviamente existen muchos más ejemplos y componentes cuya programación no podemos abordar aquí. Un excelente tutorial on-line sobre este tema se encuentra en:

http://www.weizmann.ac.il/matlab/techdoc/creating_guis/creating_guis.html

Ejercicios de repaso.

1) Crear en la memoria USB personal o en el escritorio del ordenador una carpeta con el nombre Taller_MATLAB y convertirla en el directorio de trabajo (*current directory*).
2) Introducir mediante la ventana de comandos los dos vectores siguientes: A=(1,5,-3,6,7) y B=(2,8,4,3,-5), siendo A un vector fila y B un vector columna.
3) Introducir la siguiente matriz usando para ello el editor de matrices (*array editor*):

$$\begin{bmatrix} 1 & 5 & 8 \\ -3 & 2 & 0 \\ 4 & 1 & 1 \end{bmatrix}$$

4) Determinar la longitud del vector A.
5) Crear un vector de 50 elementos regularmente espaciados comprendidos entre 1 y 25.
6) Resolver el siguiente sistema de ecuaciones:

$$x + 2y + 3z = 12$$
$$-4x + y + 2z = 13$$
$$9y - 8z = -1$$

7) Usando el siguiente conjunto de datos que corresponden a la viscosidad de la glicerina a diferentes temperaturas:

T, °C	μ, Pa·s
15	2.33
20	1.49
25	0.954
30	0.629

   a. Representar gráficamente la viscosidad frente a la temperatura
   b. Representar gráficamente el logaritmo neperiano de la viscosidad frente a la inversa de la temperatura (en K)
   c. Obtener los parámetros A y B de la ecuación Andrade mediante el ajuste de los datos a la ecuación linealizada usando la función de MATLAB `polyfit`.

$$\mu = A \cdot \exp\left(\frac{B}{T}\right)$$

8) Calcular las raíces de la función siguiente en el intervalo [1,10]

$$y = 8 + x - 2x \cdot \ln\left(\frac{x}{2}\right)$$

9) Integrar la función anterior en el intervalo [1,5]. Usar para ello las funciones de MATLAB `quadl` y `trapz`. Comparar los resultados obtenidos con cada una de ellas.
10) Encuentre el mínimo y máximo de la función

$$y = x^3 - 6x^2 - 4x + 25$$

en el intervalo [-2,6].

11) Escribir una función en MATLAB que permita calcular el volumen de un cilindro a partir de altura y diámetro del mismo.
12) Crear una función en MATLAB para calcular el valor de la suma siguiente:

$$1 + x^2 + x^3 + \cdots + x^n$$

13) Construir una GUI que calcule el número de Reynolds para un fluido circulando por una conducción e indique el régimen de flujo. Los datos para el cálculo del Reynolds se suministrarán por pantalla.

Parte II

Ejemplos de aplicación de MATLAB para la docencia en Ingeniería Química

Ejemplo 1.

Simulación de un reactor discontinuo mezcla perfecta

Este primer ejemplo está tomado del trabajo que se propone a los alumnos de la asignatura Reactores Químicos, de 4º curso de Ingeniero Químico.

El objetivo del trabajo es que diseñen (determinen el diámetro) de un reactor discontinuo mezcla perfecta que cumpla dos objetivos: (a) que se alcance la productividad requerida, y (b) que no se supere una determinada temperatura.

Este es el enunciado de uno de los trabajos propuestos:

RDMP, fase líquida, densidad constante de la mezcla reaccionante,

Reacción exotérmica, Tanque encamisado.

Fluido de transmisión de calor isotérmico: primero vapor condensando, hasta que se alcanza la temperatura de operación, después agua hirviendo. Admitir que es posible cambiar de forma instantánea de alimentar vapor saturado a la camisa a alimentar condensado saturado a la misma temperatura de operación.

Estequiometría y Cinética:

$$A + B \rightleftarrows P \qquad (-r_A) = k_d\left(C_A C_B - \frac{1}{K_E} C_P\right)$$

$$k_d = 2.542\times10^{12}\, exp\left(-\frac{132000}{RT}\right),\ \frac{\text{L}}{\text{mol h}} \qquad K_E = 9.189\times10^{-8}\, exp\left(\frac{68000}{RT}\right),\ \frac{\text{L}}{\text{mol}}$$

Ecuación válida en el intervalo 300-480 K, aunque debido a que a partir de 475 K se produce una reacción paralela no deseada se ha decidido no alcanzar esa temperatura.

Alimentación:

C_{Ai} = 9.0 mol/L C_{Bi} = 10.0 mol/L C_{Pi} = 0.0 mol/L T_i = 330 K

El tiempo de servicio entre cargas se estima en 4 horas, se desea producir 120 moles/h de P.

Mezcla reaccionante: Disoluciones poco viscosas, Coeficiente global de transmisión de calor: U = 235 W/(m^2 K)

Propiedades de la mezcla reaccionante: ρ= 1.05 g/cm^3 C_P = 2.18 J/(g K)

$$\mu = 15.0 - 0.058\cdot T + 0.00006T^2\ mPa\cdot s$$

La conversión por ciclo debe ser superior al 50 % para facilitar el funcionamiento del equipo de separación del producto y los reactivos.

El comportamiento dinámico del reactor viene dado por las dos ecuaciones diferenciales:

$$\frac{dx_A}{dt} = -\frac{r_A}{C_{A0}}$$

$$\frac{dT}{dt} = \frac{(-\Delta H)C_{A0}}{\rho C_p}\frac{dx_A}{dt} + \frac{US(T_a - T)}{V\rho C_p}$$

con las condiciones iniciales: t=0, x_A=0 y T=T_i

y además se considera una relación de esbeltez igual a 1, lo que determina que:

$$\frac{S}{V} = \frac{4.53}{D}$$

La resolución en MATLAB del caso propuesto requiere de la integración del sistema de ecuaciones diferenciales. Para ello se crea un fichero programa (script) que tiene tres funciones principales:

a. Introducir los valores de las diferentes variables del sistema.
b. Realizar la integración propiamente dicha, mediante el comando ode 45.
c. Representar gráficamente los resultados obtenidos.

Este programa trabaja con dos funciones auxiliares:

- La función cinetica, que contiene la ecuación cinética de la reacción que tiene lugar.
- La función odefile, que contiene las ecuaciones diferenciales a integrar

A continuación se presentan los tres programas, el principal y las dos funciones auxiliares, necesarios para resolver el trabajo:

Programa principal: Trabajo1

```
%programa para la integración del sistema de ecuaciones
%en este mismo programa se introduce el valor de todos los parámetros,
%excepto D y Ta, que serán introducidos por la ventana de comandos
%para hacer las iteraciones necesarias.
global CAo CBo CPo delta_HA rho Cp U
CAo=9;
CBo=10;
CPo=0;
delta_HA=-68000;
rho=1050;
Cp=2.18e3;
U=8.46e5;
Ti=330;
Tmax=475;
tserv=4;
```

```
tmax=25;
te=0:0.1:tmax;
[t,x]=ode45('odefile',te,[0,Ti]);
subplot(3,1,1);
plot(t,x(:,1))
xlabel('t, h')
ylabel('x_A')
subplot(3,1,2);
plot(t,x(:,2))
hold on
plot([0,tmax],[Tmax,Tmax],'r')
xlabel('t, h')
ylabel('T, K')
V=0.8663*D^3;
P=(CAo*x(:,1)*V*1000)./(te'+tserv);
subplot(3,1,3);
plot(t,P)
xlabel('t, h')
ylabel('P, mol/h')
disp('     t, h        x      T, K      P, mol/h  ')
disp([t,x,P])
```

Función: cinetica

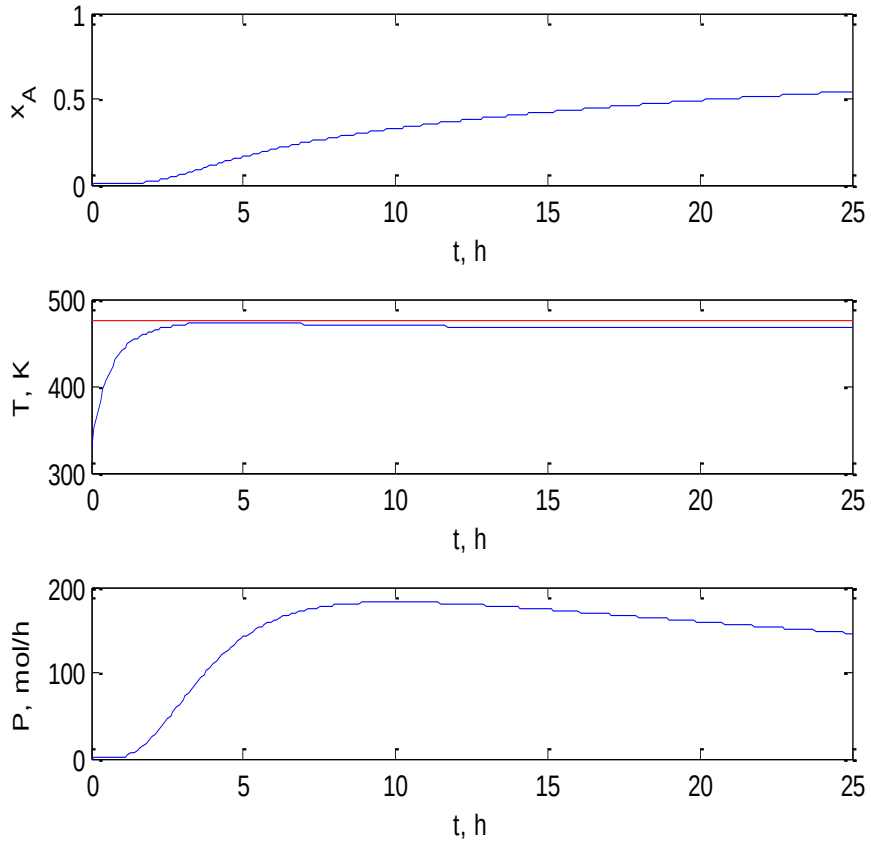
```
%funcion donde se introduce la cinetica de la
%reaccion. Servira como funcion auxiliar para la
%integracion del sistema.
function y=cinetica(x)
global CAo CBo CPo
kd=2.542e12*exp(-132000./(8.314*x(2)));
Ke=9.189e-8*exp(68000./(8.314*x(2)));
y=kd*(CAo*(1-x(1))*(CBo-CAo*x(1))-(1./Ke)*(CPo+CAo*x(1)));
```

Función: odefile

```
%funcion que contiene las ecuaciones diferenciales a integrar
function y=odefile(t,x)
global CAo delta_HA rho Cp U Ta D
y1=cinetica(x)/CAo;
y2=((-delta_HA*CAo*1000)/(rho*Cp)*y1)+(1/D)*(4.53*U*(Ta-
x(2)))/(rho*Cp);
y=[y1;y2];
```

Para obtener una solución los alumnos deben únicamente introducir por la ***command window*** un par de valores de las dos variables con las que van a tantear: el diámetro del reactor (D) y la temperatura de la camisa termostática (Ta), obteniendo la salida que se muestra a continuación:

```
>> global D Ta
>> D=1;Ta=465;
>> Trabajo1
     t, h         x       T, K       P, mol/h
        0         0   330.0000              0
   0.1000    0.0000   350.8117         0.0000
   0.2000    0.0000   368.4157         0.0004
.
.
.
   4.4000    0.1337   473.0342        124.0767
   4.5000    0.1388   473.0443        127.2698
   4.6000    0.1438   473.0379        130.3368
.
.
.
  20.7000    0.4993   466.8955        157.5963
  20.8000    0.5004   466.9611        157.3050
  20.9000    0.5015   467.0237        157.0148
.
.
.
  24.8000    0.5414   466.5928        146.5588
  24.9000    0.5423   466.5826        146.3029
  25.0000    0.5432   466.5698        146.0479
```



Como puede observarse con los valores especificados se supera la productividad de P requerida (concretamente a las 20.8 h y con una conversión de 0.5004 la productividad de P es de 157.3 mol/h), pero casi se alcanza la temperatura máxima permitida (473 K a las 4.5 h). Por ello sería necesario seguir tanteando, por ejemplo reduciendo Ta para llegar a una solución adecuada.

Ejemplo 2.

Determinación del coeficiente global de transferencia de materia en un reactor de ozonización

Se trata de un trabajo propuesto a los alumnos de la asignatura optativa "Tratamiento de Aguas Residuales Industriales" de 5º curso de Ingeniero Químico.

La idea del trabajo es ajustar los datos procedentes de un artículo científico en el que se estudia la absorción del ozono en diferentes condiciones usando un reactor discontinuo en el que se burbujea una corriente gaseosa que contiene el ozono. En concreto se usó el trabajo de Sotelo et al. "Henry's Law Constant For The Ozone-Water System", (Water Research, 23, 1239-1246, 1989).

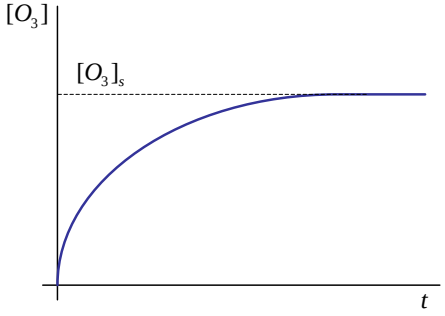
Debido a la descomposición del ozono, este proceso es, aunque sea en agua pura, una absorción con reacción química. La reacción de descomposición es normalmente considerada irreversible y de orden 1.5 ó 2 respecto del ozono en la mayoría de los estudios cinéticos publicados. Además el hecho de que en todos los casos se detecte la presencia de ozono en el agua, indica que la reacción transcurre en un régimen cinético lento, es decir, que la reacción de descomposición tiene lugar en el seno de la fase acuosa.
En estas condiciones, un balance al ozono en la fase acuosa tendrá la forma:

$$\frac{d[O_3]}{dt} = N_{O_3} - r_{O_3}$$

Siendo r_{O_3} la velocidad de descomposición del ozono (mol/s·m^3) que en principio consideraremos de orden n respecto del ozono (concentración acuosa) y N_{O_3} la velocidad de absorción del ozono (mol/s·m^3). Teniendo en cuenta la forma de la ecuación de velocidad de absorción del ozono en estas condiciones, tenderemos:

$$\frac{d[O_3]}{dt} = k_L a\left([O_3]^* - [O_3]\right) - k[O_3]^n$$

En estos casos la concentración de ozono en el reactor aumenta hasta alcanzar la saturación, lo que correspondería al estado estacionario, como se muestra en la siguiente figura:



De este modo, al alcanzarse la concentración de saturación, $[O_3]_s$, el proceso habrá alcanzado el estado estacionario, y se verificará:

$$k_L a\left([O_3]^* - [O_3]_s\right) = k[O_3]_s^n$$

La ecuación anterior nos permite poner $[O_3]^*$ en función $[O_3]_s$, mucho más fácil de determinar:

$$[O_3]^* = [O_3]_s\left(1 + \frac{k}{k_L a}[O_3]_s^{n-1}\right)$$

Lo que conduce a:

$$\frac{d[O_3]}{dt} = k\left([O_3]_s^n - [O_3]^n\right) + k_L a\left([O_3]_s - [O_3]\right)$$

La anterior ecuación puede integrarse, conocidos los valores de k y n (lo que es posible mediante experimentos cinéticos previos), para determinar los valores de la concentración de saturación de ozono, $[O_3]_s$, y del coeficiente de transferencia de materia, $k_L a$. Para ello debemos emplear algún programa que nos permita realizar integración numérica y optimización, tal como el MATLAB.

Para ello se pueden usar el programa principal y las dos funciones auxiliares siguientes:

Programa principal: Det_kLa

```
%Programa para buscar los valores de
%kLa y Os para la absorcion de ozono
global k n te Oexp Ocalc data
x1=input('Introduzca un valor inicial para kLa: ');
x2=input('Introduzca un valor inicial para Osat: ');
k=0.8258;
n=2;
te=(data(:,1))';
Oexp=(data(:,2));
%x0 contiene los valores iniciales de las constantes
x0=[x1,x2];
[s,feval]=fminsearch('desviacion',x0);
kLa=s(1,1)
Osat=s(1,2)
feval
plot(te,Oexp,'o',te,Ocalc,'-')
xlabel('t, s');
ylabel('[O3], mol/L');
```

Función: desviacion

```
function y=desviacion(x)
global te Oexp kLa Os Ocalc
kLa=x(1,1);
```

```
Os=x(1,2);
[t,z]=ode45('ozone',te,0);
y=sum((z-Oexp).^2);
Ocalc=z;
```

Función: ozone

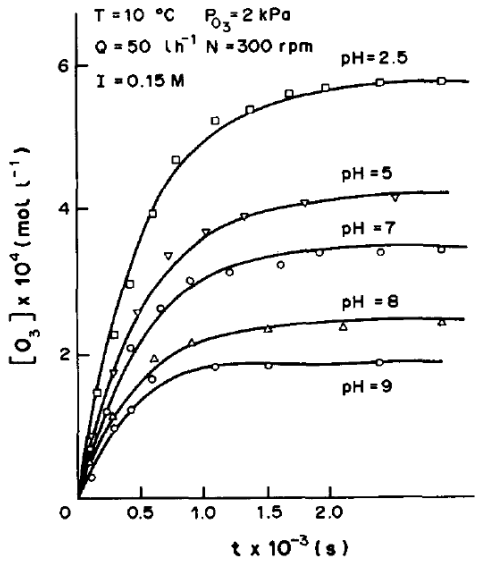
```
function y=ozone(t,z)
global k n kLa Os
y=k*(Os^n-z(1)^n)+kLa*(Os-z(1));
```

A los alumnos se les suministran los programas de MATLAB y se les indica la forma de emplearlos, y a parte se les indican los datos del artículo que deberán usar para su trabajo, que para cada alumno son diferentes de los del resto de compañeros. La tarea a realizar por ellos consiste en:

1) Digitalizar los datos del artículo mediante un programa adecuado (por ejemplo el DigitizeIt y crear una variable de MATLAB, denominada `data`, con dichos valores.
2) Buscar en el artículo los valores de k y n (constante cinética y orden de reacción) correspondientes a la serie de datos que les ha correspondido. También deben localizar la temperatura, fuerza iónica y concentración de secuestrantes de radicales usados en el experimento.
3) Ejecutar los programas de MATLAB, tanteando con diferentes valores iniciales de la concentración de saturación de ozono, $[O_3]_s$, y del coeficiente de transferencia de materia, k_La, hasta llegar a la mejor solución posible. El programa es muy sensible a los valores iniciales escogidos, por lo que suele ser necesario llevar a cabo varios tanteos para ir mejorando la solución obtenida.

A continuación se muestra un ejemplo del trabajo a realizar:

Usaremos los datos de la serie a pH 7, de la siguiente figura:



Para esta serie podemos obtener a partir del artículo los valores de k y n, que resultan ser de 0.8258 L/mol·s y 2, respectivamente.

Para introducir los datos experimentales en MATLAB se crea una nueva variable, `data`, y se edita en el ***array editor***, pegando los datos desde EXCEL. Dicha variable ha de ser una matriz de dos columnas con el tiempo en la columna izquierda y la concentración de ozono en la derecha.

Una vez creada la variable `data`, se puede ejecutar el programa, obteniéndose los resultados siguientes después de varios tanteos:

```
>> Det_kLa
Introduzca un valor inicial para kLa: 5e-3
Introduzca un valor inicial para Osat: 3.5e-4

kLa =

    0.0019


Osat =

  3.3019e-004


feval =

  1.1710e-009
```