

Práctica 1: MATLAB y las Señales Discretas

Curso 20045/2006

1. Introducción

En esta práctica presentaremos inicialmente una herramienta de MATLAB, llamada *Notebook*, que permite la creación de documentos *Word* en los que queden reflejadas sesiones de MATLAB. Seguiremos con una explicación de como una aplicación matemática destinada a manejar matrices puede ser utilizada para realizar tratamiento discreto de señales. Por último, nos centraremos en algunas cuestiones de eficiencia de MATLAB.

En esta práctica se supone que el alumno tiene conocimientos básicos sobre el lenguaje y la sintaxis de MATLAB. Como resultado de la práctica se generará una memoria en forma de *M-Book* (véase más abajo). Dicha memoria deberá contener todas las gráficas, listados, desarrollos, respuestas a preguntas, etc. que se pidan en la práctica.

Por último, antes de comenzar recuerde ir salvando periódicamente su trabajo.

2. La aplicación *Notebook* de MATLAB

El documento básico de la aplicación *Notebook* es lo que se conoce como *M-Book*. Un *M-Book* es simplemente un documento *Word* que ha sido creado utilizando la plantilla correspondiente. Al abrir uno de estos documentos, o crear uno nuevo, se arrancará automáticamente MATLAB.

En estos documentos se puede escribir texto normalmente, como en cualquier documento de *Word*. Además es posible escribir comandos de MATLAB. Si en una línea se escribe uno de dichos comandos y se pulsa Ctrl-Enter al final de la misma, dicha línea es ejecutada, y la salida generada es pegada automáticamente en el *M-Book*.

Abra el documento `cuaso1.doc`. Observará que dicho documento contiene una plantilla de la memoria que debe generar al final de la práctica. Al abrir el mismo comprobará que se abre también MATLAB. Sitúe el cursor en la zona adecuada del documento *Word* y pruebe a escribir:

`A=magic(7)` Ctrl-Enter

sin punto y coma al final. Verá que aparece un resultado. Dicho resultado ha sido generado por MATLAB. De hecho para comprobarlo, ejecute la instrucción `whos` en MATLAB y compruebe que existe una única variable llamada **A**. A los comandos de MATLAB ejecutados desde el *M-Book* se les denomina celdas de entrada mientras que a la salida generada por MATLAB se le denomina celdas de salida. Ambas se distinguen fácilmente del texto ordinario porque aparecen de otro color.

Un aspecto interesante que conviene tener presente, es que a diferencia de lo que sucede con las hojas de cálculo, en un *M-Book* no se modifican dinámicamente las celdas de salida. Así por ejemplo si escribe:

```
A=magic(3) Ctrl-Enter
```

observará que se genera una nueva celda de salida pero la original permanece inalterada. De hecho una vez generada la celda de salida, la celda de entrada correspondiente puede ser borrada. Retroceda el cursor y borre la línea que contenía `A=magic(3)`. Observe como la celda de salida permanece invariable. No obstante, si se cambia a MATLAB observará que la variable de salida tiene el valor correspondiente al último comando (aunque la celda de entrada que generó la variable haya sido borrada).

Si al escribir un comando comete un error y desea cambiar el contenido de una celda de entrada, simplemente tiene que modificar el contenido de la misma, situar el cursor al final de la línea y pulsar `Ctrl-Enter`. La correspondiente celda de salida también cambiará. Modifique el comando `A=magic(7)` para que diga `A=magic(8)` y actualice el valor de la variable **A**.

La aplicación *Notebook* funciona también con comandos gráficos. Pruebe a escribir:

```
plot(1:10) Ctrl-Enter
```

Cuando se abre un *M-Book*, desde el punto de vista de la sesión MATLAB, se está en el directorio donde esté el ejecutable de MATLAB. Compruébelo escribiendo en el *M-Book*:

```
cd Ctrl-Enter
```

Normalmente al principio de una práctica lo primero que deberá hacer es cambiarse al directorio donde estén los ficheros correspondientes a la misma. Para ello escribiría algo similar a:

```
cd c:\labtds\trabajo Ctrl-Enter
```

En ocasiones puede ser interesante agrupar varios comandos MATLAB en una única celda de entrada y que la salida producida por el conjunto de dichos comandos vaya a una única celda de salida. Para ello seleccione las celdas de entrada que desea agrupar y del menú **Notebook** ejecute **Group Cells**. Si al seleccionar las celdas de entrada, se seleccionan celdas de salida, éstas son eliminadas. Seleccione un par de celdas de entrada de las que tiene escritas, agrúpelas y evalúelas de nuevo. Para volver a evaluar una celda de entrada simplemente sitúe el cursor tras la misma y pulse `Ctrl-Enter`.

Para finalizar esta sección comentaremos brevemente algunos aspectos adicionales de *Notebook*, que no veremos en esta práctica, pero que deberá tal vez utilizar en prácticas futuras:

- Se pueden definir comandos que se ejecutan automáticamente al abrir el documento.
- Se puede variar el formato de las celdas de salida (número de decimales, tamaño de los gráficos, etc.)
- Se pueden definir zonas de cálculo independientes.

Todas estas funciones se encuentran disponibles en el menú **Notebook**, y para más información puede consultar la ayuda.

Recuerde: un *M-Book* no es más que un reflejo de una sesión con MATLAB.

El resto de la práctica deberá ejecutarlo desde el *M-Book* para que así quede reflejado su trabajo en la memoria.

Importante: Precauciones con *Notebook*

Es importante tener en cuenta una serie de precauciones al trabajar con esta aplicación:

- Recuerde poner “;” al final de las instrucciones que puedan generar mucha información de salida, como por ejemplo vectores muy largos (de más de 200 elementos). El programa tiende a colgarse.
- El programa también tiende a colgarse al avanzar o retroceder fragmentos largos del documento, especialmente si posee muchas gráficas. Para evitarlo es mejor utilizar la facilidad de zoom que posee *Word*, reduciéndolo por ejemplo a un 25 %, para dirigirse más directamente a la gráfica o parte del documento buscada, tras lo cual se puede restaurar el zoom al tamaño adecuado para la lectura.
- Recuerde salvar frecuentemente el documento, por si pese a todo no siguiera estas indicaciones. Para hacerlo de forma rápida puede pulsar CTRL-G o utilizar el botón correspondiente de *Word*.

3. Señales Discretas en MATLAB

En esta sección veremos como se puede usar MATLAB, un lenguaje orientado a matrices, para la manipulación de señales discretas. En primer lugar revisaremos algunos conceptos básicos:

- **Matriz MATLAB** : Es un conjunto de números reales o complejos, ordenados en forma de matriz de $M \times N$ elementos, donde M es el número de filas y N es el número de columnas. Cuando el número de columnas N es uno tendremos un vector (columna) unidimensional. También podemos tener un vector (fila) unidimensional si el número de filas es uno. En las prácticas:

- Procuraremos que los vectores que contengan muestras de señales sean vectores columna, es decir matrices de una única columna.
 - Las funciones que programemos y que reciban un vector como argumento de entrada, *deberán funcionar indistintamente con vectores fila o columna*.
- **Señal discreta:** Una señal discreta es un conjunto infinito de números ordenados. Dicho de otra forma el índice o variable independiente varía de $-\infty$ a ∞ . Como en MATLAB la cantidad de números que puede contener un vector es finito, lo que haremos es que un vector representará *un trozo de la secuencia*. Lo que ocurra fuera del trozo contenido en el vector MATLAB, es decir el valor que toman las muestras de la señal no contenidas en el vector, en principio nos es indiferente pudiendo suponerse una de estas dos cosas:
- La señal es nula fuera del intervalo.
 - No importa lo que la señal valga fuera del intervalo.

En resumen, MATLAB nos permite trabajar con *fragmentos de señales discretas* en la práctica.

En MATLAB los índices de las matrices empiezan en uno. Por lo tanto NO se debe escribir en MATLAB las siguientes órdenes:

```
x(-3)
x(0)
```

ya que producirán un mensaje de error.

En general, esta limitación no representa demasiados problemas prácticos pues en la mayoría de los casos el origen de tiempos de las señales es arbitrario.

3.1. Generación de señales en MATLAB

En esta sección vamos a aprender cómo se puede utilizar MATLAB para generar señales discretas. Cuando se desea generar un vector **x** en primer lugar es necesario generar un vector de índices. Para ello veremos primero cómo generar series de números con MATLAB.

3.1.1. Generación de series de números equiespaciados

Para generar números equiespaciados con MATLAB la forma más sencilla es:

```
n=0:5
```

Pruebe la anterior orden y observe el vector generado. Una variante de la anterior instrucción, consiste en generar números con paso diferente de 1. Pruebe las siguientes órdenes:

```
n1=0:2:6
n2=0:2:7
n3=0:0.2:3
```

¿Coincide en todos los casos el paso con el valor solicitado? ¿Coincide el valor inicial en todos los casos con el valor indicado?, ¿y el valor final?

Una posible aplicación de las anteriores ideas es invertir el orden de los elementos de un vector. Genere un vector A que contenga los números [1 2 4 8 16 32] SIN utilizar bucles. Genere a partir de A, un vector B que contenga los mismos números que A pero en orden inverso.

Existen otras alternativas para generar números equiespaciados. Si lo que conocemos es el valor inicial y final deseado y el número de puntos total podemos hacer:

```
t=linspace(0,10,20)
```

¿Coincide el valor inicial con el valor indicado? ¿y el valor final? ¿Cuanto vale el paso (diferencia entre dos elementos consecutivos del vector)?

Otra función que resulta de interés conocer es la que permite generar números en progresión geométrica:

```
f=logspace(log10(1),log10(2),13)
f2=440*f
```

Compruebe que el cociente entre dos elementos consecutivos cualesquiera del vector f es constante. El vector f2 contiene la frecuencia en hercios de las doce notas de una octava musical empezando por La a intervalos de semitonos. La tabla 1 muestra la relación entre las notas y las frecuencias.

Ind. MATLAB	1	2	3	4	5	6	7	8	9	10	11	12	13
Nota	La	La#	Si	Do	Do#	Re	Re#	Mi	Fa	Fa#	Sol	Sol#	La

Cuadro 1: Relación entre los elementos del vector f2 y las notas de la escala musical.

3.1.2. Generación de señales discretas con MATLAB

Es posible generar con MATLAB dos tipos de señales discretas con relativa facilidad:

- Señales de las que se dispone expresión analítica.
- Señales aleatorias.

Veamos un ejemplo de cada una de ellas. Ejecute las siguientes órdenes:

```

n1=-100:100;
x1=sinc(0.1*n1);
plot(n1,x1);
n2=0:100;
x2=(0.9).^ n2;
plot(n2,x2);

```

En cuanto a las señales aleatorias utilice

```

r1=rand(1000,1);

```

para generar un vector de 1000 muestras de ruido blanco con distribución uniforme entre 0 y 1. Represente el valor de las muestras y el histograma de 10 intervalos con las instrucciones

```

plot(r1)
hist(r1)

```

Compruebe que los gráficos corresponden con lo esperado. Si desea que el ruido sea blanco entre $\pm A$ haga

```

A=10;
r2=(r1-0.5)*2*A;

```

Comente el sentido de las operaciones anteriores. Represente los nuevos valores de las muestras obtenidas y su histograma de 10 intervalos y compruebe que corresponden con lo esperado. También es posible generar números aleatorios con distribución normal (gaussiana) utilizando la función `randn()`. Consulte el manual o la ayuda “on line” o “help” de las diferentes funciones empleadas, incluyéndola en la memoria de la práctica:

```

help rand
help randn
help hist

```

Genere un vector de 1000 muestras de ruido blanco gaussiano de media 1 V y potencia 4 W. Represente los valores de las muestras obtenidas y su histograma de 10 intervalos y compruebe que corresponden con lo esperado. Indique qué forma tendría la densidad espectral de potencia de dicho ruido.

3.2. Aplicación: síntesis de notas musicales

Vamos a tratar de realizar en este punto un sintetizador musical muy sencillo. Para ello vamos a generar notas de acuerdo con el modelo siguiente:

$$x(t) = e(t) \sin(2\pi f_k t) \quad 0 \leq t \leq 2 \text{ segundos}, \quad (1)$$

donde f_k representa la frecuencia de la nota en hercios, y $e(t)$ la envolvente. Supondremos que la frecuencia de muestreo a emplear es de 8000 hercios. Generaremos notas con envolvente exponencial, similar a la que tienen en la práctica instrumentos como la guitarra o el piano:

$$e(t) = u(t) e^{-t/\tau},$$

Con $\tau = 0,4$ segundos.

- Genere el vector **t** de instantes de muestreo correspondiente a 2 segundos de señal. NO olvide poner el “;” en el *Notebook*.
- Genere el vector **e** correspondiente a las muestras de la envolvente.
- Genere las sinusoides correspondientes a las notas La (grave), Do#, Mi, La (agudo), correspondientes a lo que se conoce como acorde de *La Mayor*. NO olvide poner el “;” en el *Notebook*. Cree 4 vectores **s1**, **s2**, **s3**, **s4**, cada uno con una senoide.
- Cree 4 vectores **n1**, **n2**, **n3**, **n4**, cada uno con una nota (ver ecuación (1)).
- Genere un vector llamado **arpeggio** que sea la concatenación de las 4 notas.
- Genere un vector llamado **acorde** que sea la suma (dividida por 4) de las 4 notas.
- Utilizando la función **wavwrite()** guarde el resultado en 2 ficheros .WAV.
- Oiga ambos ficheros.

4. Representación gráfica de señales discretas

En esta sección se pretende que explore algunas de las posibilidades que ofrece MATLAB para la representación de señales unidimensionales. Para una referencia completa de las funciones se recomienda consultar el manual, pues algunas de ellas tienen opciones que pueden resultar interesantes. Ejecute las siguientes órdenes MATLAB:

```
n=(0:50)';  
x=sin(2*pi/10*n);  
plot(n,x)  
plot(n,x,'*')
```

```

plot(n,x,n,x,'*')
plot(n,[x,2*x])
plot(n,x,n,2*x)
stem(n,x)

```

Para representar señales complejas, MATLAB ofrece diversas posibilidades. Para experimentarlas, considere la siguiente señal compleja:

$$y[n] = \begin{cases} 0,9^n e^{j2\pi n/12,7} & 0 \leq n \leq 15 \\ 0 & \text{resto} \end{cases}$$

Forme el vector de índices `ny` y de amplitudes `y` para representar la secuencia, tal como se explicó en el apartado anterior (sin usar bucles). Ejecute las siguientes funciones en el *M-Book*:

```

plot(ny,y)
plot(ny,real(y))
plot(ny,real(y),ny,imag(y))
subplot(2,1,1), plot(ny,real(y)), subplot(2,1,2), plot(ny,imag(y))
subplot(2,1,1), plot(ny,abs(y)), subplot(2,1,2), plot(ny,angle(y))
subplot;% Para quitar el subplot
compass(y);
plot(y)
feather(y)

```

Tras cada gráfica anote una pequeña explicación de lo que dibuja cada comando. Note que en ocasiones la utilización de `subplot` puede reducir considerablemente el número de gráficas en las memorias, y facilitar la comparación de varias de ellas a la vez en pantalla.

5. Consideraciones de velocidad y memoria en MATLAB

5.1. Velocidad

MATLAB es un lenguaje de programación interpretado. Esto quiere decir que cada instrucción es traducida del lenguaje fuente a código máquina antes de ser ejecutada. Ello hace, que por ejemplo, los bucles necesiten traducir las instrucciones de su cuerpo tantas veces como iteraciones tenga el bucle, lo que ralentiza su ejecución. Por ello:

SIEMPRE QUE SEA POSIBLE EVITAREMOS LOS BUCLES.

MATLAB es un lenguaje orientado a matrices que permite eliminar los bucles en muchas de las ocasiones en que estos resultarían necesarios en otro lenguaje de alto nivel. Para ello sólo tendremos que tratar de *vectorizar* nuestros programas. La clave está en que las instrucciones vectoriales están compiladas en el núcleo de MATLAB y por lo tanto su ejecución es mucho más rápida.

No todos los programas son vectorizables. Generalmente se podrá vectorizar si los elementos del vector de salida de una función no dependen los unos de los otros o precisan ser calculados en una secuencia concreta. Desde el punto de vista del programador MATLAB, éste se comporta como un *procesador paralelo*. Como resumen de las principales operaciones vectoriales considere que:

- A, B, C, ... son matrices (o vectores).
- a, b, c, ... son escalares (es decir matrices de una fila y una columna).

A continuación se indica el significado de una serie de expresiones que son útiles:

- A' : Matriz traspuesta conjugada de A. Análogamente con vectores. Evidentemente si A es una matriz real, A' coincide con su traspuesta.
- $A.'$: Matriz traspuesta. Análogamente con vectores.
- $A(:)$ (A vector): Vector columna con los elementos de A, sea éste un vector fila o columna.
- $A(:, :)$ (A matriz): Vector columna con los elementos de A, tomados de columna en columna.
- $A+B$: Suma elemento a elemento de los valores de las matrices. (Idem con la resta)
- $a+A$: Suma el escalar a cada elemento de la matriz. (Idem con la resta)
- $A*B$: Producto de matrices en el sentido algebraico.
- $A.*B$: Producto elemento a elemento de los valores de la matriz.
- $A./B$: Cociente elemento a elemento de los elementos de la matriz A: entre los de B.
- $A.\backslash B$: Cociente elemento a elemento de los elementos de la matriz B entre los de A.
- A/a : Cociente de cada elemento de la matriz A entre a.
- $a./A$: Cociente de a entre cada elemento de la matriz A.
- $A.\backslash a$: Cociente de cada elemento de la matriz A entre a.
- $a.^A$: Eleva el escalar a cada uno de los elementos de la matriz.

- `A.^a`: Eleva cada uno de los elementos de la matriz al escalar.
- `sin(A)`: Calcula el seno de cada uno de los elementos de la matriz. Idem con la mayoría de las funciones como coseno, logaritmos, raíz cuadrada, etc.

Para comprobar la eficiencia de la vectorización, realice los siguientes experimentos:

1. Programe dos funciones: `sin1(A)` y `sin2(A)` que calculen ambas el seno de los elementos de la matriz A . Ambas incluirán las líneas necesarias para determinar el tiempo de ejecución. La primera, `sin1(A)`, utilizará operaciones de matrices de MATLAB mientras la segunda deberá implementarla con bucles.

Dichas funciones incluirán instrucciones para medir el tiempo de cálculo. Para ello, use las funciones `etime()` y `clock()`¹. Construya una matriz de números aleatorios de $N \times N$ elementos². Ejecute las funciones programadas y anote los tiempos de ejecución. ¡NO olvide poner “;” al final de las instrucciones!

2. Programe dos funciones MATLAB: `busca1(A,b)` y `busca2(A,b)` que permitan determinar el número de elementos de una matriz A mayores que una cierta constante b . Ambas funciones devolverán dicho número de elementos y el tiempo empleado en encontrarlo³. Una de ellas utilizará un bucle para recorrer los elementos de la matriz, y la otra utilizará funciones internas de MATLAB. Coja la matriz del apartado anterior, y encuentre cuántos elementos tiene la matriz de $N \times N$ mayores que 0,35. (Véase función `find()`)

Confeccione una tabla de los tiempos de ejecución para cada uno de los casos anteriores, tanto utilizando la forma matricial recomendada como realizando los dos bucles anidados que recorran las matrices. ¿Es capaz de apreciar la ventaja de no utilizar bucles?

Obtenga una cifra media de la ventaja de evitar los bucles.

5.2. Consideraciones de memoria

MATLAB es un lenguaje que permite no reservar memoria para las variables por anticipado, como ocurre en la mayoría de los lenguajes. Esto resulta cómodo en la mayoría de los casos, sobre todo si utilizamos las formas matriciales de las operaciones. Sin embargo, si alguna vez debemos emplear bucles, la ventaja se puede tornar en problema. Esto es porque cada vez que MATLAB necesita aumentar el tamaño de una matriz, debe reservar memoria para una nueva matriz, copiar los valores de la antigua matriz en los primeros valores de la nueva, y finalmente rellenar con el valor correcto los nuevos elementos. Para ello, busca *espacio contiguo* en memoria. Si encuentra la cantidad suficiente no hay problema, en caso contrario dará un error de memoria.

¹Ver manual.

²El valor de N se le indicará en la práctica.

³Consulte el manual sobre cómo programar funciones que devuelvan más de un resultado.

Puede suceder con frecuencia que MATLAB disponga de memoria suficiente pero ésta no sea contigua. En ese caso normalmente, empezará a usar memoria virtual (si tiene), lo que ralentizará mucho el proceso. Para compactar la memoria se utiliza la orden `pack`. Esta orden lo que hace es:

1. Guardar en un archivo temporal en disco todas las variables.
2. Borrar todas las variables de la memoria.
3. Cargar del disco de nuevo todas las variables sin dejar huecos en memoria.
4. Borrar el archivo temporal del disco.

Dicho proceso es lento, y además no es fácilmente previsible dónde se debe realizar dentro de un programa.

Como regla general,

DEBEMOS EVITAR BUCLES QUE HAGAN QUE LAS MATRICES CREZCAN.

Con ello:

- El programa será más rápido, pues no tendrá que emplearse tiempo en reservar memoria y copiar los valores antiguos.
- Se reduce el riesgo de quedarse sin memoria.

Para entender lo anterior considere los programas contenidos en FICH1.M y FICH2.M. Copie el código de los anteriores ficheros en la memoria de la práctica.

Observe que la recursión que contienen sólo es posible realizarla con bucles. Ejecute ambos programas con un argumento de 10000; comente la diferencia en la velocidad de ejecución, y justifíquela. Con la cantidad de memoria de las máquinas actuales es muy difícil quedarse sin memoria, debido a que si el sistema operativo no encuentra memoria física, se utiliza memoria virtual, por lo que el segundo efecto negativo de ir creciendo el tamaño de los vectores es difícil que aparezca.

Como resumen final de esta sección podemos decir que:

Aunque no se requiere reservar memoria en MATLAB, puede ser conveniente hacerlo para evitar que haya vectores que crezcan de tamaño.

6. Resumen

Tras realizar la práctica, deberá haber aprendido:

- Qué es el *Notebook* y su manejo básico.
- Cómo generar señales discretas con MATLAB.
- Los principales tipos de representaciones gráficas posibles con MATLAB.
- Cómo realizar programas eficientes en velocidad y memoria con MATLAB.