

Cálculo Numérico (521230) - Laboratorio 2  
**INTRODUCCION AL MATLAB II**

## 1. Programas en MATLAB

Un programa informático es un conjunto de instrucciones (comandos) escritas en un determinado lenguaje de programación. MATLAB ya incorpora muchos comandos para la solución de varios problemas matemáticos. La semana pasada vimos, por ejemplo, el comando `norm` para el cálculo de normas de vectores y matrices. Otro ejemplo es `roots` que permite calcular las raíces de un polinomio de grado  $n \geq 1$ . Con

```
1 >> edit roots
```

abrimos el archivo correspondiente, note que su nombre es `roots.m`. Él está formado por un conjunto de instrucciones en MATLAB. Las líneas que comienzan con `%` son comentarios. Si regresamos a la ventana de comandos y escribimos

```
1 >> help roots
```

MATLAB escribe un texto de ayuda para esta función. Note que este texto coincide con los primeros comentarios en `roots.m`. De acuerdo con esta ayuda, si queremos calcular las raíces del polinomio  $x^2 + 5x + 6$  con ayuda de la función `roots`, debemos darle como entrada un vector que contenga los coeficientes del polinomio, así con el llamado

```
1 >> roots([1 5 6])      % calcular raices de polinomio de grado
    2
```

MATLAB ejecutará las instrucciones en el archivo `roots.m` y calculará aproximaciones a las raíces de  $x^2 + 5x + 6$ . Como no guardamos la salida de la función en ninguna variable, las raíces del polinomio quedan guardadas en `ans`.

Si escribimos

```
1 >> x = roots([1 0 -1])  % calcular raices de polinomio de grado
    2
```

MATLAB volverá a ejecutar las instrucciones en `roots.m`, esta vez para calcular aproximaciones a las raíces del polinomio  $x^2 - 1$ , las cuales no quedarán guardadas en `ans`, sino en `x`.

Con este ejemplo hemos visto algunos detalles importantes de programas en MATLAB:

1. Todos los programas deben ser escritos en el editor de MATLAB, el cual se muestra en la figura 1. Para abrir el editor, en la ventana principal de MATLAB hacer click en *New Script* (El primer botón a la izquierda). El editor se abrirá en una ventana independiente.

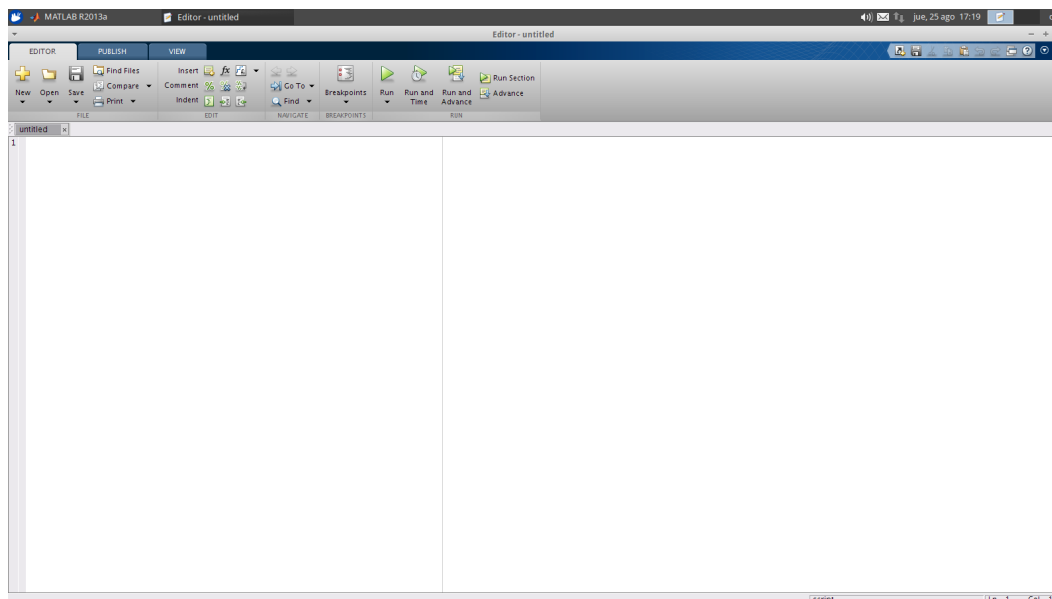


Figura 1: Editor de MATLAB.

2. Todos los programas MATLAB deben guardarse en archivos con extensión `.m`. Para ello, en la ventana del editor haga click en el icono *Save* (El tercer botón a la izquierda), y luego seleccione el directorio donde guardará su archivo.
3. Un programa debe ser ejecutado desde la ventana de comandos. Al llamar a un programa desde la ventana de comandos, MATLAB buscará el archivo con nombre igual al programa y extensión `.m` en su directorio de búsqueda (Con el comando `path` podremos ver cuál es este directorio, con `addpath`, `rmpath` podremos modificarlo. Evidentemente, en el path por defecto, MATLAB tiene guardados los programas y comandos que trae incorporado). Si el programa no se encuentra en el directorio de búsqueda, MATLAB lo buscará en el directorio de trabajo actual, el cual se puede ver en la barra superior de la ventana principal de MATLAB y también en la ventana *Current Directory*. En estas ventanas podemos cambiar y navegar por los directorios (también podemos verlos y cambiarlos con `cd`).
4. Una vez encontrado el archivo y si lo hemos llamado de manera correcta, MATLAB ejecutará las instrucciones en él. Note que si en la ventana de comandos de MATLAB escribimos

```
1 >> x = roots(1 0 -1)
```

MATLAB no podrá calcular las raíces del polinomio  $x^2 - 1$  pues no hemos llamado al programa de manera correcta.

5. En un programa todo lo que siga al símbolo `%` es interpretado como un comentario y los primeros comentarios que se escriban constituyen una ayuda al programa y se mostrarán cuando en la ventana de comandos se escriba `help <nombre del programa>`.
6. Existen dos tipos de programas en MATLAB: ruteros y funciones. Una función permite parámetros de entrada y salida, un rutero no (más aún, el rutero es una “recopilación” ordenada de comandos que pueden ser ejecutados con una sola en la ventana de comandos). La primera línea del programa en una función tiene que ser de la forma

```
function [salida] = <nombre funci\ 'on>(entrada)
```

**salida** es una lista con los nombres de los parámetros de salida de la función separados por comas, mientras que **entrada** es una lista con los nombres de todos los parámetros de entrada a la función también separados por comas. Observe que la primera línea en **roots.m** es

```
function r = roots(c)
```

lo que significa que **r** es la variable de salida de la función **roots**, en ella quedan guardadas las raíces del polinomio cuyos coeficientes se reciben en el vector de entrada **c**. En este caso, como la variable de salida es una sola, no se encerró su nombre entre corchetes.

**Obs. 1:** Note que el nombre de la función y el nombre del archivo **.m** son iguales.

Otra diferencia importante entre funciones y ruteros es que las variables que se usen en una función son locales a la función, mientras que las que se usen en un rutero son globales, es decir, seguirán existiendo una vez que termine la ejecución del programa.

**Obs. 2:** Una forma directa de crear una función desde la ventana principal de MATLAB es hacer click en *New* (El segundo botón a la izquierda) y luego en *Function*.

**Obs. 3:** Un programa tipo rutero puede ser ejecutado desde el editor haciendo click en el botón *Run*, ubicado en la cuarta posición desde la derecha (el triángulo verde). Observe que alrededor de ese botón aparecen otras opciones de ejecución más avanzadas (que no se verán en este curso).

## 2. MATLAB y sus funciones

La versatilidad de MATLAB se basa en sus funciones. En particular la función **help** es muy importante, esta función nos permite leer un resumen del uso de otras funciones, por ejemplo si ejecutamos

```
1 help abs
```

obtendremos una salida como

```
1 abs      Absolute value.
2 abs(X) is the absolute value of the elements of X. When
3 X is complex, abs(X) is the complex modulus (magnitude) of
4 the elements of X.
```

una breve lectura nos permite observar que la función **abs()** nos permite obtener el valor absoluto de los elementos de una matriz. Entre las principales funciones de matlab cuentan:

Sentencia	Significado
abs()	Valor absoluto
cos(),sin(),tan()	Funciones trigonométricas en radianes
if()	Verificador lógico
plot()	Permite dibujar figuras planas
ones()	Crea una matriz con todas las entradas en 1
zeros()	Crea una matriz con entradas todas nulas
find()	Encuentra los índices de una matriz que son no nulos
disp()	Muestra una cadena <sup>1</sup> en la ventana de comando
input()	Recibe una variable a través del teclado del ordenador
norm()	Norma Euclideana de un vector
cross()	Producto cruz de vectores
dot()	Producto interior o punto de vectores

Utilice la función **help()** para conocer los distintos funcionamientos de cada uno de estas funciones.

### 3. Ciclos y condicionales en MATLAB

1. Ciclo **for**: tiene la forma

```
1 for var = vi:incr:vf
2 <instrucciones>
3 end
```

Las instrucciones en el cuerpo de este ciclo se ejecutan tantas veces como valores tome la variable **var**, la que se inicializa con **vi** y toma los valores **vi**, **vi+incr**, **vi+incr+incr**, ..., **vf**. Si **incr** es 1, puede escribirse

```
1 for var = vi:vf
2 <instrucciones>
3 end
```

Si, queremos, por ejemplo, sumar los 10 primeros números naturales con ayuda de un ciclo **for** deberíamos escribir

```
1 suma = 0
2 for i = 1:10
3 suma = suma + i
4 end
```

Si, queremos, por ejemplo, guardar en un vector **p** los 10 primeros términos de la progresión geométrica (con primer término igual a 1 y razón igual a 3)

$$1, 3, 9, 27, 81, \dots$$

podríamos hacerlo con ayuda de un ciclo **for** de la siguiente manera

```
1 p = zeros(10,1);      % inicializar p
2 p(1) = 1;             % primer elemento de p es 1
3 for i = 2:10
4 p(i) = 3*p(i-1);
5 end
```

2. Ciclo **while**: tiene la forma

```
1 while <condici\ 'on>
2 <instrucciones>
3 end
```

Las instrucciones en el cuerpo del ciclo se ejecutan mientras **condici\ 'on** sea verdadera.

El vector **p** del ejemplo anterior también puede crearse con un ciclo **while** de la siguiente forma

```

1 p = zeros(10,1);      % inicializar p
2 p(1) = 1;             % primer elemento de p es 1
3 i = 2;
4 while i <= 10
5 p(i) = 3*p(i-1);
6 i = i + 1;
7 end

```

3. Una condicional en MATLAB tiene la forma general

```

1 if <condici\ 'on>
2 <instrucciones 1>
3 [elseif
4 <instrucciones 2>
5 else
6 <instrucciones 3>]
7 end

```

donde `condici\ 'on` es una expresión lógica e `instrucciones` es el conjunto de comandos que se ejecutará si `condici\ 'on` es verdadera. Los comandos encerrados entre corchetes son opcionales.

### Ejemplo 1.

Abramos un archivo nuevo en el editor de texto de MATLAB. En el primero escribamos las instrucciones siguientes:

```

1 function F = mifuncion1(x)
2 % funci\ 'on para evaluar f(x) = (1-cos(x))/x^2 en cada
3 % una de las componentes del vector de entrada v
4
5 % creando vector de ceros con mismo n\ 'umero de elementos que x
6 F = zeros(length(x),1);
7
8 % ciclo para evaluar f en componentes de x
9 for i=1:length(x)
10 % averiguar si componente i-esima de v es distinta de cero
11 if x(i) ~= 0
12 F(i) = (1-cos(x(i)))/x(i)^2;
13 else
14 F(i) = 1/2
15 end
16 end

```

Ésta es la función que permite evaluar una función  $f$ . Guardemos este programa en `/home/cfm/mifuncion1.m`.

**Observación:** El nombre del archivo donde guardamos la función y el nombre de la función (en 1ra línea del programa) coinciden. Durante todo el semestre mantendremos esta convención.

Regresemos a la ventana de comandos de MATLAB para llamar a `mifuncion1`. Si queremos, por ejemplo, averiguar los valores de  $f$  en 0, 0,1, 0,2, 0,3, 0,4, 0,5, podemos hacerlo mediante

```

1 >> mifuncion1(0:.1:.5) % evaluar f en [0,0.1,0.2,0.3,0.4,0.5]
2 >> whos % ni x ni F (locales a mifuncion1) aparecen
3 % como variables de memoria

```

Evaluemos  $f(x)$  en los puntos  $0,188 \times 10^{-7}$ ,  $0,189 \times 10^{-7}$ ,  $0,19 \times 10^{-7}$  llamando a `mifuncion1`

```

1 >> mifuncion1([0.188e-7, 0.189e-7, 0.19e-7])

```

Observe que los valores que retorna `mifuncion1.m` son todos mayores que 0,6.

**Ejemplo 2.** Escriba un rutero para graficar a la función  $f$  en (1) en 100 puntos equidistantes en  $[-\pi, \pi]$ .

Abramos un archivo nuevo, escribamos:

```

1 % rutero para graficar f(x) = (1-cos(x))/x^2 entre -pi y pi
2 x = linspace(-pi,pi);
3 Fx = mifuncion1(x);
4 % grafiquemos la funci'on
5 plot(x, Fx)

```

y guardémoslo en el archivo `/home/cfm/plotmifuncion1.m`. Éste es el rutero para graficar  $f(x)$ . Si escribimos en la ventana de comandos

```

1 >> help plotmifuncion1

```

MATLAB muestra los primeros comentarios en `plotmifuncion1.m`.

Llamemos al rutero para ver el gráfico de  $f(x)$  con  $x \in [-\pi, \pi]$

```

1 >> plotmifuncion1
2 >> whos % las variables x y Fx permanecen en memoria

```

Con el gráfico obtenido nos damos cuenta de que la función parece ser siempre menor o igual que  $\frac{1}{2}$ . Sin embargo, al evaluar  $f$  en los puntos  $0,188 \times 10^{-7}$ ,  $0,189 \times 10^{-7}$  y  $0,19 \times 10^{-7}$  obtuvimos valores mayores que 0,5. Éste es un ejemplo típico de *cancelación excesiva*, la cual puede ocurrir cuando se restan dos cantidades casi iguales. En la expresión  $(1 - \cos(x))/x^2$  si  $x \approx 0$ , entonces  $\cos(x) \approx 1$  y al hacer  $1 - \cos(x)$  se estarán restando dos números casi iguales.

**Ejemplo 3.** Escriba una función que evalúe  $e^x$  mediante su serie de Taylor  $e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!}$ .

Abramos un archivo nuevo, escribamos:

```

1 function y=miexp(x)
2 y=1;
3 sum=x;
4 n=1;
5 while (y+sum~=y)
6 y=y+sum;
7 n=n+1;
8 sum=x*sum/n;
9 end

```

y guardémoslo en el archivo `/home/cfm/miexp.m`. Explique por qué este programa siempre se detiene y compare los valores de esta función con los de la función de MATLAB `exp(x)` para distintos valores de  $x$ .

**Ejemplo 4.** Supongamos que deseamos tener un fichero que nos permita conocer el valor de las imágenes de la función  $y = \sin(x + x^2)$ , nos interesaría que dado un valor para  $x$  nos retorne el valor para  $y$ . En este caso podemos hacer:

```
function y=f(x)
y=sin(x+x.^2);
```

observe que en tal caso la operación potenciación debe anteponerse con el operador `.` para que la potenciación sea componente a componente. Obviamente la entrada  $x$  de esta función puede ser cualquier matriz, y la salida será una matriz de las mismas dimensiones.

Si nos interesara dibujar tal función en el intervalo  $[-10, 10]$ , podemos hacer directamente

```
x=[10:0.1:10];
y=f(x);
plot(x,y);
```

También podemos definir funciones definidas por tramos usando este tipo de ficheros. Por ejemplo, si interesa ingresar a MATLAB la función

$$g(x) := \begin{cases} x^2 + 1 & x > 0 \\ \frac{1}{x^2+1} & x \leq 0 \end{cases}$$

podemos crear el fichero función

```
function y=g(x)
in=find(x>0);
y(in)=x.^2+1;
in=find(x<=0);
y(in)=1./(x.^2+1);
```

## 4. Funciones como objetos inline

Los objetos inline son otra forma de representar funciones en MATLAB. Por ejemplo, para crear un objeto inline que nos permita manejar la función  $g(x) = 33x^2 + \cos(e^x)$ , debemos simplemente ejecutar la instrucción

```
1 g=inline('33*x.^2+cos(exp(x))','x');
```

una vez creado este tipo de dato, podemos dibujarlo rápidamente haciendo uso de la función `ezplot()`, según

```
1 ezplot(g)
```

La primera cadena de las entradas de `inline()` es la forma vectorizada de la operación que define a la función, mientras que la segunda entrada y siguientes, especifican las variables que independientes de la función a definir.

## 5. Arreglo de funciones

La instrucción

```
1 arreglo=@nombre
```

retorna un arreglo de funciones llamado **arreglo** que tiene una función llamada **nombre**.

Un arreglo de función es un objeto de MATLAB que permite llamar a funciones indirectamente. Se pueden utilizar para definir otras funciones e inclusive se pueden agrupar dentro de arreglos tipo celda. Cuando queramos definir una nueva función a través de un arreglo de funciones utilizaremos funciones anónimas, estas se generan como se ejemplifica a continuación. Supongamos que nos interesa ingresar a MATLAB la función de la parábolas  $y = 2x^2 + 3x + 1$ , esto lo hacemos según

```
1 a = 2;  
2 b = 3;  
3 c = 1;  
4 parabola = @(x) a*x.^2 + b*x + c;
```

para poder dibujar este tipo de funciones hacemos uso de `plot()`.

```
1 x=linspace(0,10,100);  
2 plot(x,parabola(x));
```

## 6. La ventana gráfica

Como vimos en los ejemplos anteriores, sin importar el método que usemos para dibujar MATLAB siempre creará otra ventana, llamada por defecto **Figure 1** similar a la figura (2). Para crear una nueva

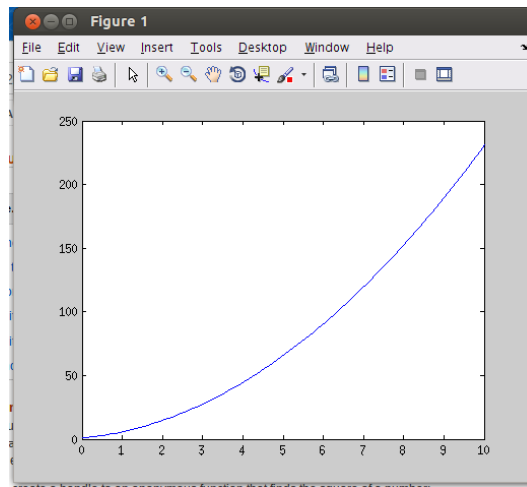


Figura 2: Ventana gráfica por defecto de MATLAB.

ventana gráfica donde dibujar hacemos uso de la función `figure()`. Por ejemplo las instrucciones

```
1 figure(1);  
2 ezplot(inline('x^2'));  
3 figure(2);  
4 ezplot(inline('x^2-1'));  
5 figure(3);  
6 ezplot(inline('x^2-2'));
```



crearán tres ventanas gráficas llamadas **Figure 1**, **2** y **3**, las cuales se abrirán en ese orden con tres gráficas muy similares. Si nos interesa adjuntar estos tres gráficos en una misma ventana gráfica hacemos uso del comando `hold on`, como se ilustra a continuación,

```
1 figure(4); hold on;
2 ezplot(inline('x^2'));
3 ezplot(inline('x^2-1'));
4 ezplot(inline('x^2-2'));
```

si finalmente no deseamos agregar más gráficos a esta figura ejecutamos

```
hold off;
```

Si se desea que las gráficas adjuntas cambien de colores automáticamente se puede usar la instrucción `hold all`; en vez de `hold on`;

Además como anotaciones podemos agregar texto en ciertos lugares de los gráficos de una figura. Mientras la figura en la que deseamos añadir un comentario esté seleccionada con el comando `figure()` ejecutamos el comando `gtext()`. Si conocemos la coordenada exacta donde queremos hacer un comentario, podemos utilizar la función `text()`.

### 6.1. Uso del comando `line()`, `label()` y opciones de estilo en gráficos

A modo de ejemplo, a continuación presentamos un código que sobrepone aproximaciones en serie de Taylor de la función  $y = \sin(x)$  y formatea un gráfico con varios detalles. Para acceder al rutero ingrese a

<http://www.udec.cl/~fmilanese/codigo4.m>  
léalo y ejecútelo.

### 6.2. Opciones de estilo para dibujar curvas en 2D

Las opciones de estilo del comando `plot()` son una cadena que consiste de hasta tres caracteres que especifican color, el estilo de línea y forma de marcar los puntos de una gráfica. Estos caracteres son resumidos en la siguiente tabla

Estilo de color		Estilo de línea		Estilo de marcador	
y	amarillo			+	signo de suma
m	magenta	-	solida	o	circulo
r	rojo	-	rayada	*	asterisco
g	verde	:	punteada	x	con una equis
b	azul	-.	raya punto	.	con un punto
w	blanco	none	sin línea	^	triángulo
k	negro			s	cuadrado
				d	diamante

los cuales puede ser usados como

```
plot(x,y,'r*') %dibuja con línea roja continua y con marcadores asterisco
plot(x,y,'b--') %dibuja con línea azul rayada
plot(x,y,'+') %dibuja puntos no conectados con un marcador +
```

## 7. Subgráficas en una figura

Si se quiere hacer un cuantos gráficos dentro de una misma figura, pero no sobreponerlos, se utiliza el comando `subplot()`, este comando requiere tres entradas de números enteros `subplot(m,n,p)` y representa dividir la ventana gráfica en  $m \times n$  subventanas y elegir la subventana  $p$ -ésima, contadas por filas, como ventana de dibujo.

Por ejemplo

1 `subplot(2,2,3)`

dividirá la ventana gráfica actual en cuatro subventanas y dibujará lo siguiente en la ventana inferior izquierda.

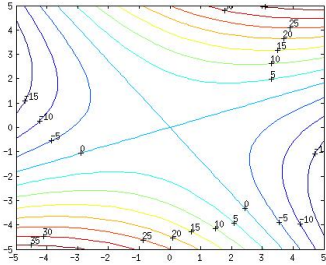
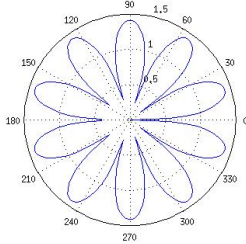
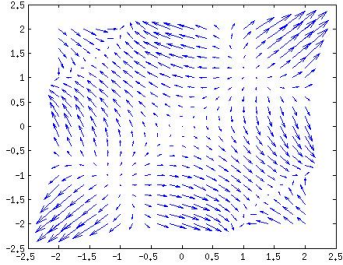
## 8. Resumen de las principales funciones para graficar 2D

En la siguiente tabla se encuentran las principales funciones para hacer gráficos 2D.

Nombre	Descripción
<code>fplot()</code>	Crea un gráfico de una función de una variable
<code>bar()</code>	Crea un gráfico de barras vertical
<code>contour()</code>	Crea un gráfico de curvas de nivel
<code>contourf()</code>	Crea un gráfico de curvas de nivel llenas
<code>polar()</code>	Dibuja curvas descritas en coordenadas polares
<code>quiver()</code>	Dibuja campos vectoriales

a continuación ejemplos de usos de estos códigos.

Función	Código	Salida
<code>fplot()</code>	<pre>f='sin(x.^2+3*x)'</pre> <pre>fplot(f,[-10,pi])</pre> <p>Notar que <math>f</math> debe estar definida en <math>x</math>.</p>	
<code>bar()</code>	<pre>t=linspace(0,2*pi,200);</pre> <pre>r=sqrt(abs(2*sin(5*t))) ;</pre> <pre>y=r.*sin(t);</pre> <pre>bar(t,y);</pre> <pre>axis([0 pi 0 inf]);</pre>	

<pre> contour() r = -5:.2:5; [X,Y]=meshgrid(r,r) ; Z=-.5*X.^2 +X.*Y+Y.^2; cs=contour(X,Y,Z); clabel(cs); </pre>	
<pre> polar() t=linspace(0,2*pi,200); r=sqrt(abs(2*sin(5*t))) ; polar(t,r); </pre>	
<pre> quiver() r=-2:.2:2; [X,Y]=meshgrid(r,r) ; Z=X.^2-5*sin(X.*Y)+Y.^2; [dx,dy]=gradient(Z,.2,.2); quiver(X,Y,dx,dy,2); </pre>	

## 9. Errores comunes en MATLAB

Los errores son una parte fundamental de nuestras vidas, interactuemos o no con Matlab. La única diferencia es que, cuando interactuamos con ordenadores, se nos indican inmediatamente nuestros errores. Hay que advertir que programar en MATLAB requiere de tiempo y concentración. Si no tenemos el tiempo o la concentración adecuada se puede volver muy irritante, si por un momento no hacemos la distinción entre (, [ ó { o simplemente no distinguimos entre : y ;, o nos olvidamos de considerar que **a** es distinto de **A**, entonces programar en MATLAB se volverá largo y tedioso.

A continuación se enlistan los errores mas comunes con los que nos encontraremos mientras trabajamos con MATLAB. Se adjuntan ejemplos de los errores y explicaciones de ellos.

### 9.1. Indices de asignación no coinciden

```

1 >> D=zeros(3); d=[1,2];
2 D(2:3,:)=d;
3 Subscripted assignment dimension mismatch.

```

Este error sucede cuando hay asignaciones de matrices y las matrices que están a ambos lados del signo = no tienen la misma dimensión. Utiliza el comando **size()** para chequear las dimensiones de ambos elementos y asegurarte que coincidan. En este ejemplo tendríamos que

```

1 >> size(D(2:3,:))
2
3 ans =

```

```

4
5 2      3
6
7 >> size(d)
8
9 ans =
10
11 1      2

```

## 9.2. Mal uso de (,[,{

```

1 >> (x,y)=sphere(5)
2 (x,y)=sphere(5)
3 |
4 Error: Expression or statement is incorrect--possibly unbalanced (, {,
   or [.

```

Aquí lo que sucede es que MATLAB se confunde con las variables de salida de `circlefn()`. Al verlas dentro de paréntesis ( ellas representan índices de una matriz y no variables. Las salidas de una función deben ser encerrada en paréntesis [.

También, cuando los paréntesis no están escritos correctamente el mismo error es mostrado, por ejemplo

```

1 >> (x,y]=sphere(5)
2 (x,y]=sphere(5)
3 |
4 Error: Expression or statement is incorrect--possibly unbalanced (, {,
   or [.

```

## 9.3. Errores de índice y dimensión

```

1 >> x=1:100;
2 v=0:3:45;
3 s=x(v);
4 Subscript indices must either be real positive integers or logicals.

```

El primer elemento del vector de índices `v` es cero. En consecuencia, estamos intentando obtener el 0<sup>avo</sup> elemento de `x`, pero cero no es un índice válido para un vector ni para una matriz. El mismo error sucede cuando un número negativo aparece como índice. Además, cuando un índice excede el la dimensión de una variable, también sucede un error, siguiendo el ejemplo anterior:

```

1 >> x=1:100;
2 v=1:3:103;
3 s=x(v);
4 Index exceeds matrix dimensions.

```

Los ejemplos aquí ilustrados son casi triviales. La mayoría de las veces, estos errores aparecen cuando los índices son creados, incrementados o manipulados dentro de un ciclo.

## 9.4. Operaciones matriciales

```
1 >> x=1:10; y=20:-2:1;
2 x*y
3 ? ? Error using ==> mtimes
4 Inner matrix dimensions must agree.
```

Cuando realizamos la multiplicación matricial  $x*y$ , el número de columnas de  $x$  debe coincidir con el número de filas de  $y$ . En el ejemplo anterior  $x$  e  $y$  son ambos vectores de tamaño  $1 \times 10$ , por lo tanto no se pueden multiplicar. Sin embargo,  $x*y'$  y  $x'*y$  si se puede ejecutar, los que representan los productos exterior (cruz) e interior (punto) del vector  $x$  con el vector  $y$ .

Muchas otras operaciones que involucren matrices de dimensiones no apropiadas producirán un error similar a este. Una regla general para solucionar este problema es escribir la operación en papel y pensar si la operación tiene sentido. Si no tiene sentido, muy probablemente MATLAB muestre un error. Por ejemplo  $A^2$  sólo tiene sentido si la matriz  $A$  es cuadrada.

Una fuente muy común de este tipo de errores es usar operaciones matriciales cuando lo que se busca es una operación componente a componente, por ejemplo  $y.^x$  nos entrega la potenciación componente a componente, mientras que

```
1 >> y^x
2 ? ? ? Error using ==> mpower
3 At least one operand must be scalar
```

## 9.5. Mal llamado de una función

```
1 creador();
2 ? ? ? Input argument 'n' is undefined.Error in ==> CREADOR at 3MATRIX(1:n,1:n)=0;
```

Este error sucede cuando un fichero tipo función no ha sido ejecutado con las entradas adecuadas. Este es uno de los pocos errores en los cuales el mensaje del error nos provee exactamente la línea del fichero en el que se produce el error.

## 9.6. Mal llamado de un rutero

```
1 >> x=codigo1(4)
2 ??? Attempt to execute SCRIPT codigo1 as a function:
```

Aquí `codigo1` es un rutero. Las entradas y salidas no pueden ser especificadas en un rutero. A continuación presentamos un caso más interesante que produce el mismo error. Esto sucede cuando se intenta ejecutar la siguiente función

```
1 funcion [x,y]=circle(r)
2 %CIRCLE -dibuja un circulo de radio r
3 theta = linspace(0,2*pi,100); %Declaro vector angular
4 x=r*cos(theta); %Coordenadas x
5 y=r*sin(theta); %Coordenadas y
6 plot(x,y); %Dibujo
```

al hacer

```

1 >> circle(4)
2 ??? Attempt to execute SCRIPT circle as a function

```

El lector dirá: Pero si no es un script!. Y tiene razón, no es un script pero tampoco es un función. La primera palabra de una función debe ser **function** y no **funcion**!.

## 9.7. Variables no definidas

```

1 >> x=c+22;
2 Undefined function or variable 'c'.

```

En este ejemplo la variable `c` no está definida al momento de la operación. Este mensaje es preciso. Cuando este mensaje aparece en funciones o ficheros que nosotros hemos escrito puede deberse a que tal función o fichero se encuentra en otro directorio distinto al directorio actual. Puedes usar los comandos `what()`, `dir` o `ls` para enlistar los archivos de tu directorio actual. Si el archivo no está en la lista entonces MATLAB no puede acceder a él, deberás localizar el directorio y cambiar el directorio actual al directorio indicado, para esto puedes usar el comando `cd`.

## 9.8. Prácticas recomendables

Siempre que trabajemos con ficheros estaremos trabajo con la memoria local. Esto tiene ventajas y desventajas, por un lado tenemos la ventaja de que podemos modificar directamente las variables que están en el fichero. Pero en algunas ocasiones esto puede ser una desventaja, puesto puede producir resultados no esperados en el código. Baje el fichero ubicado en la siguiente URL y ejecútelo.

<http://www2.udec.cl/~fmilanese/codigo2.m>

Podrá percartarse que este código realiza el llenado de una matriz diagonal llamada *Ac* con los número 1, 2, 3 y 4. Observe que si usted ejecuta más de una vez este código la matriz *Ac* pierde esta característica. Esto sucede por qué al ejecutarse por segunda vez las instrucciones del fichero se sobrescribe la variable *Ac*. Para evitar este error MATLAB posee un comando que borra todas las variables que están en la memoria local, basta ejecutar la sentencia

```

1 clear all

```

o mas precisamente puede eliminar únicamente la variable *Ac*, ejecutando

```

1 clear Ac

```

Muchas veces el código que en un momento nos puede parecer sencillo al momento en que se escriba, al ser reléido se olvida parte del razonamiento que lo produjo. Por esto, los lenguajes de programación permiten comentar en el mismo código. En matlab los códigos se comentar con el símbolo de porcentaje%. Por ejemplo, descargue el código ubicado en la siguiente URL y lealo en el editor.

<http://www.udec.cl/~fmilanese/codigo3.m>

## 10. Ejercicios

1. Escriba un código que construya una matriz de orden  $98 \times 10$  cuyas filas sean los número del 1 al 10 y viceversa, alternadamente, es decir

$$\begin{bmatrix} 1 & 2 & 3 \dots \\ 10 & 9 & 8 \dots \\ 1 & 2 & 3 \dots \\ 10 & 9 & 8 \dots \\ \vdots & \vdots & \vdots \end{bmatrix}$$

2. Escriba un código que gire una matriz 90 grados en sentido antihorario.
3. Describa las características de la variable  $A$  en cada uno de los siguientes códigos.

a)

```

1 for i=1:10
2     for j=1:10
3         A(i,j)=1;
4     end
5 end

```

```

1 for i=1:10
2     for j=1:10
3         if(i>j)
4             A(i,j)=1;
5         else
6             A(i,j)=-1;
7         end
8     end
9 end

```

b)

```

1 for i=1:10
2     for j=1:10
3         if(i>2*j)
4             A(i,j)=1;
5         else
6             A(i,j)=-1;
7         end
8     end
9 end

```

d)

```

1 for i=1:10
2     j=1;
3     while j<7

```

```

4         if(i>2*j)
5             A(i,j)=1;
6         else
7             A(i,j)=-1;
8         end
9         j=j+1;
10    end
11 end

```

e)

```

1
2 A(1:10,1)=3;
3 for i=1:10
4     j=1;
5     while j<2
6         if(i>1&& i<3)
7             A(i,j)=A(i,j)+1;
8         else
9             A(i,j)=0;
10        end
11        j=j+1;
12    end
13 end

```

d)

```

1 A=1;
2 for i=1:10
3     A(:,end+2)=1
4     A(end+1,:)=0;
5 end

```

4. Identifique errores lógicos en los siguientes códigos e interprete algún posible significado.

a)

```

1 j=0;
2 while j>=0
3     A(j)=A(j)+1;
4 end

```

b)

```

1 A=[1,2;3,4];
2 B=[A;1,2];
3 C=[[1;2],A];
4 D=[[A]];

```

c)

```

1 for i=0:3

```

```

2     for j=0:3
3         A(i,j)=i*j;
4     end
5 end

```

d)

```

1 j=-9;
2 while j>10
3     A(j)=10;
4 end

```

e)

```

1 i=0;
2 count=i;

```

```

3 | while count <= i
4 |     A(i) = 8;
5 |     i = i + 1;

```

```

6 |     count = i - 1;
7 | end

```

5. Construya un programa que evalúe la función.  $f(x) = \begin{cases} \sin^2(x) & \text{si } x \leq -2 \\ 1 - e^{-x} & \text{si } -2 < x < 2 \\ \frac{1}{x+1} & \text{si } x \geq 2 \end{cases}$
6. Cree un programa tipo function que determine el ángulo existente entre dos vectores de  $\mathbb{R}^n$ . Testee con los vectores base de  $\mathbb{R}^3$ .
7. Construya un programa que calcule el producto cruz **sólo** entre dos vectores de  $\mathbb{R}^3$ . Utilice este programa para calcular el área contenida por el paralelepípedo generado por ambos vectores.
8. Construya un programa tipo function que calcule el volumen del paralelepípedo generado por tres vectores de  $\mathbb{R}^3$ .
9. Construya un programa tipo function que calcule la distancia de un punto a una recta.
10. Construya un programa tipo function que calcule la distancia de un punto a un plano.
11. Construya un programa tipo function que calcule la distancia entre dos rectas cualquiera.
12. Construya un programa que reciba como entrada una función  $f : \mathbb{R} \rightarrow \mathbb{R}$  y un conjunto de puntos de  $\mathbb{R}^2$ ,  $\{(x_i, y_i)\}_{i=1}^n$  y realice lo siguiente.
  - a) Entregue como salida un vector con los puntos que pasan por la curva.
  - b) Grafique la curva de  $f$ , y los puntos dados, diferenciando explícitamente aquellos que pasan por la curva de los que no mediante colores.

#### Sugerencias:

- La función  $f$  puede ingresarse como una variable tipo **string**, por ejemplo.

```

1 | f = 'x+1'

```

y dentro del programa se transforma a función mediante el comando `fcnchk()` o `inline()`.

- Dado un escalar o vector **x**, una función se evalúa de la forma

```

1 | y = feval(f, x);

```

- El conjunto de puntos pueden ingresarse como una matriz  $P \in \mathcal{M}_{2,n}$  de la forma.

$$P = \begin{bmatrix} x_1 & x_2 & \cdots & x_n \\ y_1 & y_2 & \cdots & y_n \end{bmatrix}.$$

13. Utilice la función `plot()` para dibujar la ecuación de la curva  $y = \cos(x^2)$  entre  $[0, \pi]$ . Utilice las funciones `legend()` y `title()` para agregar un título y leyenda al gráfico.
14. Construya una función que grafique la curva  $y = \cos(x^n)$  para un número  $n$  dado. ¿Qué sucede para  $n$  muy grandes?
15. Modifique el fichero disponible en <http://www.udec.cl/~fmilanese/saludador.m> de forma tal que sus entradas sean letras y no números, y que chequee que la entrada es una letra. Utilice la función `isnumeric()`.



16. Utilize la función `creador()`, descrita anteriormente, para construir otra función que construya una matriz que es la suma de dos matrices de  $10 \times 10$ , una que tiene slo el número 7 en su quinta columna y la otra sólo el número 6 en su tercera fila.
17. Baje el rutero disponible en <http://www.udec.cl/~fmilanese/codigo7M.m> y corrija todos los errores.

18. Escribamos una función en MATLAB que, dado un vector  $x \in \mathbb{R}^n$ ,  $n \in \mathbb{N}$ ,  $x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}$ , devuelva el

vector  $F \in \mathbb{R}^n$ ,  $F = \begin{pmatrix} F_1 \\ F_2 \\ \vdots \\ F_n \end{pmatrix}$ , tal que  $F_i = f(x_i)$ ,  $i = 1, 2, \dots, n$  siendo

$$f(x) = \begin{cases} \frac{1-\cos(x)}{x^2}, & x \neq 0, \\ \frac{1}{2}, & x = 0. \end{cases} \quad (1)$$

19. Considere la siguiente matriz **tridiagonal** y el siguiente vector:

$$\mathbf{A}_n = \begin{pmatrix} n & 1 & & & \\ 1 & n-1 & 2 & & \\ & 2 & n-2 & \ddots & \\ & & \ddots & \ddots & n-1 \\ & & & n-1 & 1 \end{pmatrix} \in \mathbb{R}^{n \times n}, \quad \mathbf{b}_n = \begin{pmatrix} 1/n \\ 2/(n-1) \\ 3/(n-2) \\ \vdots \\ n/1 \end{pmatrix} \in \mathbb{R}^n$$

Escriba un función en MATLAB que para una entrada  $n \in \mathbb{N}$  ejecute las siguientes tareas:

- Construya la matriz  $\mathbf{A}_n$  y el vector  $\mathbf{b}_n$ .
  - Resuelva el sistema  $\mathbf{A}_n \mathbf{x} = \mathbf{b}_n$  con el comando `x=A\b`.
  - Devuelva como salida el valor  $\|\mathbf{A}_n \mathbf{x} - \mathbf{b}_n\|_2$ .
20. Sea

$$g(x) = \begin{cases} \frac{2 \sin^2\left(\frac{x}{2}\right)}{x^2}, & x \neq 0, \\ \frac{1}{2}, & x = 0 \end{cases}$$

Note que  $\forall x \in \mathbb{R}$  se cumple  $f(x) = g(x)$ . Escriba una función `mifuncion2` para evaluar  $g$  en las componentes de un vector de entrada `x`. Evalúe a  $g$  en `[0.188e-7, 0.189e-7, 0.19e-7]`. ¿Son los valores retornados mayores que 0,5?

Note que, a pesar de que las funciones  $f$  y  $g$  son teóricamente equivalentes, numéricamente no lo son.

21. Considere la siguiente matriz:

$$\mathbf{M} = \begin{pmatrix} \mathbf{A}_m & \mathbf{\Theta} \\ \mathbf{\Theta}^t & \mathbf{A}_n \end{pmatrix} \in \mathbb{R}^{(m+n) \times (m+n)}$$

donde  $\mathbf{\Theta}$  es la matriz nula de orden  $m \times n$  y cada matriz  $\mathbf{A}_k$  tiene la forma:

$$\mathbf{A}_k = \begin{pmatrix} k & k-1 & k-2 & \cdots & \cdots & 2 & 1 \\ k-1 & k-1 & 0 & \cdots & \cdots & 0 & 0 \\ k-2 & 0 & k-2 & 0 & \cdots & \vdots & \vdots \\ \vdots & \vdots & 0 & \ddots & \cdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \cdots & \ddots & 0 & \vdots \\ 2 & \vdots & \vdots & \cdots & \cdots & 2 & 0 \\ 1 & 0 & 0 & \cdots & \cdots & 0 & 1 \end{pmatrix} \in \mathbb{R}^{k \times k}$$

Escriba un programa MATLAB, tipo *function*, que para  $m, n \in \mathbb{N}$  dados, genere la matriz  $\mathbf{M}$  y el vector  $\mathbf{b} = (b_i) \in \mathbb{R}^{m+n}$ , cuyas componentes están dadas por los  $m+n$  primeros elementos de la sucesión convergente al número de Euler  $e$ , esto es por:

$$b_i = \left(1 + \frac{1}{i}\right)^i, \quad \forall i = 1, \dots, m+n.$$

Luego, en un programa tipo rutero, llame a la función con  $n = 6$  y  $m = 4$ , resuelva el sistema  $\mathbf{M}\mathbf{x} = \mathbf{b}$  y calcule  $\|\mathbf{M}\mathbf{x} - \mathbf{b}\|_p$ , para  $p = 1$ ,  $p = 2$  y  $p = \infty$ .