UNIVERSIDAD DEL BÍO-BÍO
FACULTAD DE CIENCIAS
DEPARTAMENTO DE MATEMÁTICA
Viernes 5, Abril 2019

Métodos Numéricos 220138

Laboratorio 2

Introducción al MATLAB – II

En este laboratorio discutiremos los tipos de programas que pueden hacerse en MATLAB y cómo almacenar datos.

Hay dos tipos de programas MATLAB: uno se donomina *rutero* y el otro *function*.

Supongamos que tenemos un directorio donde guardaremos nuestros programas. MATLAB debe estar direccionado a ese directorio. Un comando para cambiar de directorio dentro de MATLAB es:

```
>> cd directorio
```

Todos los archivos con programas MATLAB deben terminar con la extensión *.m*. Veamos un ejemplo:

Deseamos resolver la ecuación de segundo grado $3x^2 + 5x + 2 = 0$. Escribamos primeramente un programa tipo *rutero*. El programa puede ser el siguiente:

```
a=3;
b=5;
c=2;
D=b^2-4*a*c;
x(1)=(-b+sqrt(D))/(2*a);
x(2)=(-b-sqrt(D))/(2*a);
x
```

Guarde el programa con el nombre *eje1.m*. Para ejecutarlo escriba en MATLAB el nombre del archivo y obtendrá:

```
>> eje1

x =

    -0.6667    -1.0000
```

Este tipo de programas se conocen como *ruteros* y las variables son globales, es decir, quedan en la memoria despues de ejecutarse el programa. Para saber que hay en la memoria puede usarse el comando *whos*:

```
>> whos
 Name      Size          Bytes  Class

  D         1x1              8  double array
  a         1x1              8  double array
  b         1x1              8  double array
  c         1x1              8  double array
  x         1x2             16  double array
```

Una desventaja de este tipo de programas es que para resolver otra ecuación que utilice la misma fórmula debemos modificar el programa.

Los programas tipo *function* tienen una estructura más esquematizada y siempre comienzan de la siguiente forma:

```
function [salida1,salida2,...]=nombre(entrada1,entrada2,...)
```

El programa anterior escrito como function queda asi:

```
function x=eje2(a,b,c)
D=b^2-4*a*c;
x(1)=(-b+sqrt(D))/(2*a);
x(2)=(-b-sqrt(D))/(2*a);
```

Se almacena en un archivo *eje2.m* y se ejecuta del siguiente modo:

```
>> eje2(3,5,2)
ans =

      -0.6667   -1.0000
```

Este programa puede usarse, sin modificarlo, para resolver otras ecuaciones del mismo tipo. También puede usarse en otros programas (como veremos en otros laboratorios).

En este caso las variables son locales. Por ello si se ejecuta *whos* se obtiene:

```
>> whos
  Name      Size          Bytes  Class

  ans       1x2              16  double array
```

Es conveniente usar programas tipo *function*, cuando sea posible, pues permiten un ahorro de memoria. Otro ejemplo es `roots` que permite calcular las raíces de un polinomio de grado $n \geq 1$. Con

```
>> edit roots
```

abriremos 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

```
>> help roots
```

MATLAB escribirá 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 $x^2 + 5x + 6$, debemos escribir

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

MATLAB ejecutará entonces 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

```
>> 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`.

A continuación daremos los comandos más usados en programas:

- *for*. La sintaxis de este comando es

  ```
  for i=vi:in:vf
      instrucciones
  end
  ```

  donde `vi`, `in` y `vf` son el valor inicial, el incremento y el valor final de la variable escalar i. Cuando `in` está ausente, se presupone el valor 1: así, son equivalentes `for i=vi:vf` y `for i=vi:1:vf`.

- *while*. La sintaxis de este comando es

  ```
  while relación
      instrucciones
  end
  ```

  Las instrucciones se ejecutan reiteradamente mientras la relación sea verdadera.

- *if*. La sintaxis de este comando es

  ```
  if relación
      instrucciones
  end
  ```

Las instrucciones se ejecutan si la relación es verdadera. Otras formas de este comando son posibles. Por ejemplo,

```
if relación
     instrucciones 1
else
     instrucciones 2
end
```

Si la relación es verdadera se ejecutan las instrucciones 1, caso contrario se ejecutan las instrucciones 2.

Las relaciones para los comandos *if* y *while* se construyen mediante los siguientes relacionadores:

`<`	menor que
`>`	mayor que
`<=`	menor o igual a
`>=`	mayor o igual a
`==`	igual a
`~=`	distinto a

y los siguientes conectivos lógicos:

`&`	y
`|`	o
`~`	no
`xor`	o excluyente

A continuación mostraremos un par de ejemplos de programas (prográmelos) y dejaremos algunos ejercicios (trate de hacerlos todos en el tiempo del laboratorio).

1. Construya un programa que evalúe la función $f(x) = \begin{cases} 2\,\mathrm{sen}^2(2x), & x \leq 0, \\ 1 - \mathrm{e}^{-x}, & x > 0. \end{cases}$

   Solución:

```
function y=fun1(x) % Si la entrada es un vector, la salida tambien lo es.
n=length(x);       % Determina la longitud del vector x.
                   % A continuacion se calcula el valor de la funcion
                   % componente a componente.
for i=1:n          % Al omitir el incremento este se asume igual a 1.
    if x(i)<=0
        y(i)=2*(sin(2*x(i)))^2;
    else
        y(i)=1-exp(-x(i));
    end
end
```

   Para hacer la gráfica de la función f en el intervalo $[-10, 10]$ puede utilizarse este programa del siguiente modo:

```
>> x=-10:.01:10;
>> plot(x,fun1(x))
```

   Como solución alternativa, tambien podemos considerar

```
function y=fun1b(x)
% funcion para evaluar f(x) por tramos
% entrada: numero real o vector con componentes reales
% salida: valor o valores de la funcion evaluada en "entrada"

% creando vector de ceros con mismo nmero de elementos que x
y = zeros(size(x));
% escogiendo de x los elementos distintos de cero
xmay0 = x(x<=0);
xmen0 = x(x>0);
% evaluando funcion en x menor o igual a cero
 y(x<=0) = 2*(sin(2*xmay0)).^2;
% evaluando funcion en x mayor que cero
y(x>0) = 1-exp(-xmen0);
```

2. Construya un programa que evalúe la función: $f(x) = \begin{cases} x-1, & x \leq -2, \\ 1-x^2, & -2 < x < 0, \\ -\dfrac{1}{x+1}, & x \geq 0. \end{cases}$

Solución:

```
function y=fun2(x)
n=length(x);
for i=1:n
    if x(i)<=-2
        y(i)=x(i)-1;
    elseif (x(i)>-2 & x(i)<0)
        y(i)=1-x(i).^2;
    else
        y(i)=-1/(x(i)+1);
    end
end
```

Utilice este programa para hacer gráficos de la función en diferentes intervalos.

3. Construya una función que genere una matriz de la forma $\boldsymbol{A} = \begin{pmatrix} 2 & -1 & & 0 \\ -1 & \ddots & \ddots & \\ & \ddots & \ddots & -1 \\ 0 & & -1 & 2 \end{pmatrix} \in \mathbb{R}^{n\times n}$.

Solución:

```
function A=matriz(n)
B=[zeros(n-1,1) eye(n-1);zeros(1,n)];
A=2*eye(n)-B-B';
```

genere y visualice la matriz $\boldsymbol{A}$ para $n = 8$. Genere el vector $\boldsymbol{b} = \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} \in \mathbb{R}^8$ y resuelva el sistema $\boldsymbol{Ax} = \boldsymbol{b}$ mediante el comando MATLAB:

```
>> x=A\b
```

Verifique que el vector $\boldsymbol{x}$ obtenido resuelve el sistema de ecuaciones anterior evaluando (obviamente en MATLAB) $\boldsymbol{Ax} - \boldsymbol{b}$. ¿Qué observa?

4. Haga un programa *function* que genere una matriz de la forma

$$A = \begin{pmatrix} \alpha\,\mathbf{I} & \varepsilon\,\mathbf{1} \\ \varepsilon\,\mathbf{1} & \alpha\,\mathbf{I} \end{pmatrix} \in \mathbb{R}^{2n\times 2n},$$

donde $\mathbf{1}$ denota la matriz de elementos todos 1. $\mathbf{I}$ denota a la matriz identidad de $\mathbb{R}^{n\times n}$. α, ε y n deben ser parámetros de la función.

5. Haga un programa *function* que genere una matriz de la forma

$$A = \begin{pmatrix} 2n & & \\ & \ddots & 1 \\ 1 & & 2n \end{pmatrix} \in \mathbb{R}^{2n\times 2n},$$

donde $\begin{matrix} 1 \end{matrix}$ indica que los elementos fuera de la diagonal principal de la matriz A son unos.

6. Haga un programa *function* eficiente que genere una matriz de la forma

$$A = \begin{pmatrix} 2\mathbf{I} & -\mathbf{I} & \mathbf{0} \\ -\mathbf{I} & 2\mathbf{I} & -\mathbf{I} \\ \mathbf{0} & -\mathbf{I} & 2\mathbf{I} \end{pmatrix} \in \mathbb{R}^{3n\times 3n},$$

7. Construya una función que evalúe e^x mediante su serie de Taylor: $\mathrm{e}^x = \sum_{n=0}^{\infty} \frac{x^n}{n!}$.

Solución:

```
function y=miexp(x)
y=1;
sum=x;
n=1;
while (y+sum~=y)
   y=y+sum;
   n=n+1;
   sum=x*sum/n;
end
```

Explique por qué este programa siempre se detiene.

Compare los valores de esta función con los de la función MATLAB `exp(x)` para distintos valores de x. Para visualizar más dígitos decimales utilice el comando:

```
>> format long
```

8. Usando el comando *help* de MATLAB estudie la sintaxis de los comandos *save* y *load*. Pruébelos con algunos de los programas realizados.

9. Dada una matriz $\boldsymbol{A}$, indique qué calculan las siguiente líneas MATLAB:

```
>> max(sum(abs(A')))
>> norm(A,inf)
```

Comentario sobre operaciones con números reales

En los computadores personales se utiliza la aritmética de punto flotante para almacenar los números reales. MATLAB representa cada número como *double*, es decir, usando 64 bits consecutivos de memoria. Las constantes `realmax` y `realmin` contienen el mayor y el menor números reales positivos que pueden almacenarse de esta forma. Note que ellos son aproximadamente iguales a 2^{1023} y 2^{-1023} respectivamente.

```
>> realmax
>> 2^1023
>> realmin
>> 2^(-1023)
```

Si el valor absoluto del resultado de una operación aritmética es mayor que `realmax`, ocurrirá *overflow*. Nos daremos cuenta de que esto ha ocurrido porque MATLAB dará `Inf` o `-Inf` como resultados de la operación, en lugar de un número real. El resultado en MATLAB será `Inf` si el resultado real de la operación aritmética es un número positivo mayor que `realmax`, se obtendrá un `-Inf` cuando el resultado real de la operación aritmética sea un número negativo menor que $-$`realmax`.

```
>> x = 2^512
>> x^2          % resultado real es 2^(1024), mayor que realmax
                % matlab devuelve Inf
>> x = -2^342
>> x^3          % resultado real es -2^(1026), menor que -realmax
                % matlab devuelve -Inf
```

Si el valor absoluto del resultado de una operación aritmética es menor que `realmin`, ocurre *underflow*. Pero en MATLAB se resuelve este problema de manera desapercibida para el usuario. Note que, por ejemplo, a pesar de que el resultado de la siguiente operación es menor que `realmin`, MATLAB muestra un número real como resultado y no un valor especial como en el caso anterior.

```
>> x = 2^(-512)
>> x^2
```

Si realizamos alguna operación aritmética cuyo resultado no pueda ser determinado, el resultado será `NaN` (*not a number*).

```
>> 1/0          % resultado es Inf
>> 0/0          % resultado es NaN
```

En MATLAB la precisión del computador se denota por `eps`, éste es el menor número real positivo x que satisface que el resultado de la operación $1 + x$ es distinto de 1.

```
>> eps
>> (1+eps)-1      % es distinto de cero
>> (1+eps/2)-1    % es cero
```