UNIVERSIDAD DE CONCEPCION
FACULTAD DE CIENCIAS
FISICAS Y MATEMATICAS
DEPARTAMENTO DE INGENIERIA MATEMATICA

Cálculo Numérico (521230)

Laboratorio 2

Introducción al MATLAB – II

En este laboratorio discutiremos los tipos de programas que pueden hacerse 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; por ejemplo, en el disco C tenemos el directorio *numerico* (es aconsajable tener un directorio diferente del directorio MATLAB, ya que si guardamos programas en el directorio donde se encuentra MATLAB, lo estamos modificando y esto nos puede crear problemas).

MATLAB debe estar direccionado al directorio donde guardamos los programas. Eso se logra con el siguiente comando:

```
>> cd c:\numerico
```

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* (para escribir un programa se puede usar cualquier editor de texto que permita guardar los archivos como ascii; por ejemplo, `notepad`). El programa es:

```
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 en el directorio *numerico*, 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 se usa 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 así:

```
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.

A continuación daremos los comandos más usados en programas:

1. *for*. La sintáxis 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.

2. *while*. La sintáxis de este comando es

```
while relación
    instrucciones
end
```

   Las instrucciones se ejecutan reiteradamente mientras la relación sea verdadera.

3. *if*. La sintáxis 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 y dejaremos algunos ejercicios (trate de hacerlos todos en el tiempo del laboratorio).

Ejemplos (Prográmelos)

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))
```

2. Construya una función que genere una matriz de la forma $\mathbf{A} = \begin{pmatrix} 2 & -3 & & 0 \\ 4 & \ddots & \ddots & \\ & \ddots & \ddots & -3 \\ 0 & & 4 & 2 \end{pmatrix} \in \mathbb{R}^{n\times n}$.

   (2.1) De manera full.

   (2.2) De manera esparsida (usando los comandos *sparse, spdiag, speyes*)

   (2.3) Utilice el comando `whos` para determinar la cantidad de memoria que requiere el almacenamiento de de A para $n = 30$ en los dos formatos anteriores. Indique cuál resulta más conveniente.

   (2.4) El comando `spy` permite "espiar" la estructura de una matriz mediante un gráfico con la distribución de sus entradas no nulas. "Espíe" la estructura de A para $n = 30$ en los dos formatos anteriores.

   (2.5) Para $n = 30$ genere el vector $\mathbf{b} = \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} \in \mathbb{R}^{10}$ y resuelva el sistema $\mathbf{Ax} = \mathbf{b}$ mediante el comando MATLAB:

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

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

Solución:

```
function A=matriz(n)
A=2*eye(n)+4*diag(ones(n-1,1),-1)-3*diag(ones(n-1,1),1);
```

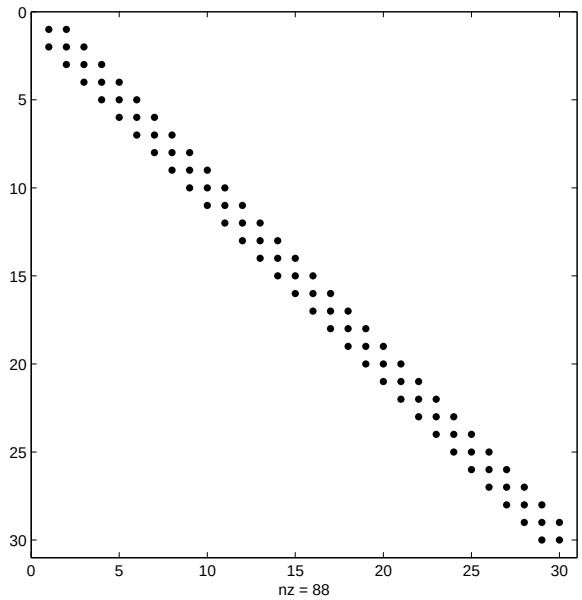
```
function A=smatriz(n)
A=4*spdiags(ones(n-1,1),-1,n,n)+2*speye(n)-3*spdiags(ones(n-1,1),-1,n,n)';
```

```
>> A=matriz(30);
>> SA=smatriz(30);
>> whos
  Name      Size                    Bytes  Class

  A        30x30                    7200  double array
  SA       30x30                    1180  double array (sparse)

Grand total is 988 elements using 8380 bytes

>> spy(A)
```



3. El siguiente programa

```
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
```

evalúa e^x mediante su serie de Taylor: $e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!}$.

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 uno de los comandos:

```
>> format long
>> format long e
>> format short g
```

4. (4.1) Haga un programa *function* que genere una matriz tridiagonal de orden n de la forma

$$\mathbf{A} = \begin{pmatrix} a & b & & 0 \\ c & \ddots & \ddots & \\ & \ddots & \ddots & b \\ 0 & & c & a \end{pmatrix}.$$

Los datos de entrada deben ser los valores de n, a, b y c, y la salida la matriz $\mathbf{A}$.

(4.2) Mediante el comando MATLAB *rank*, determine si la matriz tridiagonal y simétrica $\mathbf{A}$ correspondiente a $n = 10$, $a = 4$ y $b = c = 1$ es no singular.

(4.3) Resuelva el sistema $\mathbf{Ax} = \mathbf{b}$ con la matriz tridiagonal anterior y un segundo miembro $\mathbf{b}$ cualquiera, de los siguientes tres modos:

4.3.1.
```
>> x=A\b
```

4.3.2.
```
>> [L,U]=lu(A);
>> x=U\(L\b)
```

4.3.3.
```
>> R=chol(A);
>> x=R\(R'\b)
```

Compare las soluciones obtenidas y calcule los residuos $\mathbf{r} = \mathbf{b} - \mathbf{Ax}$ respectivos.

(4.4) Justifique el que la matriz **A** es definida positiva a partir de lo anterior.

Solución:
File `trid.m`:

```
function A=trid(n,a,b,c);
A=a*eye(n)+b*diag(ones(n-1,1),1)+c*diag(ones(n-1,1),-1);
```

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

6. Dada una matriz **A**, indique qué calculan las siguiente líneas MATLAB:

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

GBG/MCP/RRS/MSC
http://www.ing-mat.udec.cl/pregrado/asignaturas/521230/
27/08/03