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

Cálculo Numérico (521230) - Laboratorio 2

INTRODUCCION AL MATLAB II

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

```
>> 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

```
>> 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

```
>> roots([1 5 6])       % calcular raíces 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

```
>> x = roots([1 0 -1])   % calcular raíces 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 primcipal de Matlab hacer click en *New Script* (El primer botón a la izquierda). El editor se abrirá en una ventana independiente.

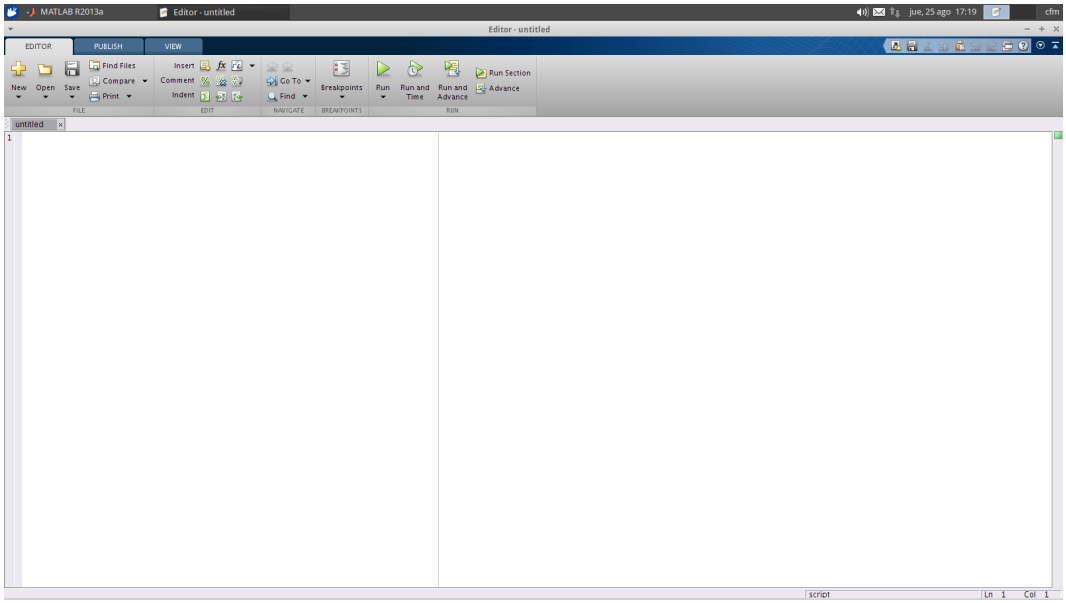

Figura 1: Editor de MATLAB.

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 insctrucciones en él. Note que si en la ventana de comandos de MATLAB escribimos

```
>> 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 función>(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).

Ejemplo 1. 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 $F_i = f(x_i),\ i = 1, 2, \ldots, n$ siendo

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

Antes de escribir esta función necesitamos introducir los ciclos y las condicionales en MATLAB.

Ciclos y condicionales en MATLAB

1. Ciclo `for`: tiene la forma

```
for var = vi:incr:vf
   <instrucciones>
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

```
for var = vi:vf
   <instrucciones>
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)

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

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

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

2. Ciclo `while`: tiene la forma

```
while <condición>
   <instrucciones>
end
```

Las instrucciones en el cuerpo del ciclo se ejecutan mientras `condición` sea verdadera.

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

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

3. Una condicional en MATLAB tiene la forma general

```
if <condición>
   <instrucciones 1>
[elseif
   <instrucciones 2>
else
   <instrucciones 3>]
end
```

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

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

```
function F = mifuncion1(x)
% función para evaluar f(x) = (1-cos(x))/x^2 en cada
% una de las componentes del vector de entrada v

% creando vector de ceros con mismo número de elementos que x
F = zeros(length(x),1);

% ciclo para evaluar f en componentes de x
for i=1:length(x)
   % averiguar si componente i-esima de v es distinta de cero
   if x(i) ~= 0
      F(i) = (1-cos(x(i)))/x(i)^2;
   else
      F(i) = 1/2
   end
end
```

Ésta es la función que permite evaluar $f(x)$. 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

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

```
>> 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:

```
% rutero para graficar f(x) = (1-cos(x))/x^2 entre -pi y pi
x = linspace(-pi,pi);
Fx = mifuncion1(x);
% grafiquemos la función
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

```
>> help plotmifuncion1
```

MATLAB muestra los primeros comentarios en `plotmifuncion1.m`.
Llamemos al rutero para ver el gráfico de $f(x)$ con $x \in [-\pi, \pi]$

```
>> plotmifuncion1
>> 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:

```
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
```

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.

Ejercicio 1. Considere la siguiente matriz **tridiagonal** y el siguiente vector:

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

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

1. Construya la matriz $\boldsymbol{A}_n$ y el vector $\boldsymbol{b}_n$.
2. Resuelva el sistema $\boldsymbol{A}_n\boldsymbol{x} = \boldsymbol{b}_n$ con el comando `x=A\b`.

3. Devuelva como salida el valor $||\boldsymbol{A_n x} - \boldsymbol{b_n}||_2$.

Ejercicio 2. Sea

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

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.

Ejercicio 3. Considere la siguiente matriz:

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

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

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

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

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

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

ERRORES EN MATLAB

Siguiendo con lo visto en el laboratorio 1, se mostrarán los errores más comunes que se pueden encontrar en la programación en MATLAB. Al momento de llamar a una función, nos podemos encontrar con el siguiente error:

```
>> x = mifuncion(1);
Undefined function or variable 'mifuncion'.
```

"*La función o variable* `mifuncion` *no está definida*". Esto quiere decir que `mifuncion` no es un comando de MATLAB, o bien, no se ha escrito el programa `mifuncion.m` o este se encuentra en un directorio distinto al cual se está trabajando.
Considere ahora el siguiente programa:

fibonacci.m

```
function F = fibonacci(N)
if (N == 0)
    F = 0;
elseif (N == 1)
    F = 1;
else
    f0 = 0 ; f1 = 1;
    for i = 2:n
        f2 = f1 + f0;
        f0 = f1 ; f1 = f2;
    end
    F = f2;
end
```

el cual calcula el $n-$ésimo término de la sucesión de Fibonacci, dada por:

$$f_0 = 0, \qquad f_1 = 1 \qquad \text{y} \qquad f_n = f_{n-1} + f_{n-2}, \qquad \forall\, n \geq 2.$$

Sin embargo, al ejecutar este programa para calcular el sexto término de la sucesión, tenemos:

```
>> fibonacci(6)
Undefined function or variable 'n'.

Error in fibonacci (line 8)
    for i = 2:n
```

MATLAB nos indica que el error es que no está definida la variable `n` y se encuentra en la línea 8 de la función `fibonacci` (incluso nos muestra qué contiene esa línea). Un aspecto importante de MATLAB es la diferencia entre variables con mayúsculas y variables con minúsculas. En este ejemplo, si bien `N` es una entrada de `fibonacci`, para MATLAB esta es una variable distinta a `n`. Por lo tanto, se debe cambiar en la línea 8 `n` por `N`.
A veces los errores no necesariamente son errores en la sintaxis de un comando, si no que son errores conceptuales, propios del problema que se quiere resolver. Por ejemplo, si se ejecuta el programa `fibonacci` con una entrada no entera, tenemos

```
>> fibonacci(6.5)

ans =

     8
```

Esto es, el programa acepta entradas no enteras, a pesar de que los términos de la sucesión de Fibonacci tienen índices enteros. MATLAB ofrece el comando `error`, el cual detiene un programa y muestra un mensaje de error escrito por el programador. Por ejemplo, podemos modificar nuestra función `fibonacci.m` de la siguiente forma:

fibonacci.m

```
1  function F = fibonacci(N)
2  if (N ~= round(N) || N < 0)
3      error('La entrada debe ser un entero positivo o cero')
4  end
5  if (N == 0)
6      F = 0;
7  elseif (N == 1)
8      F = 1;
9  else
10     f0 = 0 ; f1 = 1;
11     for i = 2:N
12         f2 = f1 + f0;
13         f0 = f1 ; f1 = f2;
14     end
15     F = f2;
16 end
```

El mensaje de error se mostrará cuando la entrada sea no entera o sea negativa (condiciones que se pueden ver en la línea 2). Así, al ejecutar este programa en la ventana de comandos:

```
>> fibonacci(6.5)
Error using fibonacci (line 3)
La entrada debe ser un entero positivo o cero
```

Finalmente, otro tipo de errores se presentan al ingresar mas entradas de las que acepta una función:

```
>> fibonacci(1,2)
Error using fibonacci
Too many input arguments.
```

o al requerir mas salidas de las que puede otorgar la función:

```
>> [x , y] = fibonacci(6)
Error using fibonacci
Too many output arguments.
```

Esto es válido no solo para las funciones que usted pueda programar en MATLAB, si no que también es válido para todas las funciones y comandos que este programa trae por defecto.