Universidad de Concepción
Facultad de Ciencias Físicas y Matemáticas
Departamento de Ingeniería Matemática

Cálculo Numérico (521230)

Laboratorio 2: Introducción al MATLAB (MATLAB como lenguaje de programación)

Generalidades

1. Comenzar MATLAB: En la esquina superior izquierda: `Inicio > Desarrollo > Matlab R2013a`.
2. Salir de MATLAB: `exit` o `quit`.
3. Ayuda en MATLAB: `helpdesk` o `helpwin` o `help <nombre de comando>`.
4. En cualquier momento del trabajo con MATLAB, con el comando `whos` podremos ver las variables en memoria y con el comando `clear all` podremos eliminarlas de memoria.
5. Una vez abierto MATLAB, con el comando `cd` podremos ver el directorio de trabajo y con `cd <nuevo directorio>` podremos cambiarlo.

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 MATLAB deben guardarse en archivos con extensión `.m`.
2. 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). Si el programa no se encuentra en el directorio de búsqueda, MATLAB lo buscará en el directorio de trabajo actual (podemos verlo y cambiarlo con `cd`).
3. 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.

4. 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>`.
5. Existen dos tipos de programas en MATLAB: ruteros y funciones. Una función permite parámetros de entrada y salida, un rutero no. 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.

   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.

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/<login>/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 0.188×10^{-7}, 0.189×10^{-7}, 0.19×10^{-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/<login>/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 0.188×10^{-7}, 0.189×10^{-7} y 0.19×10^{-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.
Ejercicio 1: 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.