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. 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 programas para la solución de varios problemas matemáticos. La semana pasada vimos, por ejemplo, el programa `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
```

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 raíces 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 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 leer el nombre de un programa de 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.

Escribamos una función en MATLAB que permita evaluar

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

y un rutero que grafique esta función en el intervalo $[-\pi, \pi]$.
Abramos dos archivos nuevos en el editor de texto de MATLAB . En el primero escribamos las instrucciones siguientes:

```
function val = mifuncion1(x)
% función para evaluar f(x) = (1-cos(x))/x^2 en x
% entrada: número real o vector con componentes reales
% salida: valor o valores de la función en la entrada

% cambiando x a vector columna (si x es una matriz se tranforma
% en vector columna)
x = x(:);
% creando vector de ceros con mismo número de elementos que x
val = zeros(length(x),1);
% escogiendo de x los elementos distintos de cero
xdiff0 = x(x~=0);
% evaluando función en x distintos de cero
val(x ~= 0) = (1-cos(xdiff0))./xdiff0.^2;
% en cero la función es 1/2
val(x == 0) = .5;
```

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.
En el segundo archivo escribamos:

```
% rutero para graficar f(x) = (1-cos(x))/x^2 entre -pi y pi
x = -pi:.01:pi;
valx = mifuncion1(x);
% grafiquemos la función
plot(x,valx)
```

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 mifuncion1
>> help plotmifuncion1
```

MATLAB mostrará los primeros comentarios en los archivos `mifuncion1.m` y `plotmifuncion1.m`. Para evaluar $f(x)$ en los puntos $0{,}188 \cdot 10^{-7}$, $0{,}189 \cdot 10^{-7}$, $0{,}19 \cdot 10^{-7}$

```
>> mifuncion1(0.188e-7)   % evaluar f en 0.188e-7
>> whos                   % ni x ni val (locales a mifuncion1) aparecen
                          % como variables de memoria
>> mifuncion1(0.189e-7)   % evaluar f en 0.189e-7
>> mifuncion1(0.19e-7)    % evaluar f en 0.19e-7
```

Observe que los valores que retorna `mifuncion1.m` son todos mayores que 0,6. Llamemos al rutero para ver el gráfico de $f(x)$ en $[-\pi, \pi]$

```
>> plotmifuncion1
>> whos              % las variables x y valx permanecen en memoria
```

Con el gráfico obtenido nos damos cuenta de que la función parece ser siempre menor o igual que $\frac{1}{2}$. De hecho puede demostrarse que $f(x) \leq \frac{1}{2}$, $\forall x \in \mathbb{R}$. 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: Encuentre una expresión matemáticamente equivalente a $\frac{1-\cos(x)}{x^2}$ y en la cual no pueda ocurrir cancelación excesiva, es decir, en la cual no se resten números que puedan ser casi iguales. Escriba una nueva función, llamémosla `mifuncion2.m`, para evaluar $f(x)$ mediante esta nueva expresión. Escriba un rutero para graficar esta nueva función en $[-\pi, \pi]$ y evalúe la nueva función en $0{,}188 \cdot 10^{-7}$, $0{,}189 \cdot 10^{-7}$, $0{,}19 \cdot 10^{-7}$ para comprobar que sus valores son efectivamente menores o iguales que $\frac{1}{2}$.

CONDICIONALES EN MATLAB

En el ejemplo anterior, en los comentarios de `mifuncion1.m` escribimos que la función admite como entrada un vector de números reales. Sin embargo, entre las instrucciones de la función no incluimos ninguna para comprobar que las componentes de `x` sean sólo números reales.
Si, por ejemplo, escribimos

```
>> mifuncion1(['a',0.1])
```

nuestro programa `mifuncion1.m` intentará evaluar `cos('a')` y esta función detectará la invalidez de la entrada.
Las funciones `isnumeric`, `isreal` e `isfinite`, así como la construcción `if` (condicional) de MATLAB , nos permitirán detectar parámetros de entrada erróneos. La función `isnumeric` devuelve 1 si todos los elementos del vector que recibe como entrada son números, `isreal` devuelve 1 si todas las componentes del vector de entrada son números reales. La salida de `isfinite` es un vector con el mismo número de componentes que el vector de entrada, las cuales serán 1 si el correspondiente elemento en el vector de entrada es distinto de `Inf` y `-Inf` y 0 en caso contrario, es decir, si la componente correspondiente en el vector de entrada es `Inf` o `-Inf`.
Escribamos

```
>> isnumeric([2,2+i,3,-10])
>> isreal([2,2+i,3,-10])
>> isfinite([2,2+i,3,-10])
>> isnumeric(['a',1])
>> isfinite([1/0,1])
```

en la ventana de comandos para entender mejor estas funciones.
Una condicional en MATLAB tiene la forma general

```
if <condición>
  <instrucciones>
end
```

donde `condición` es una expresión lógica e `instrucciones` es el conjunto de comandos que se ejecutará si `condición` es verdadera.
Añadiendo

```
if ~isnumeric(x) || ~isreal(x) || any(~isfinite(x))
        error('entrada debe ser vector de números reales')
end
```

después de `x = x(:)` en `mifuncion1.m` y `mifuncion2.m`,
si algún elemento de `x` no es un número $\vee$ algún elemento de `x` no es un número real $\vee$ algún elemento de `x` no es finito, **entonces** con el llamado a la función `error` estaremos generando un mensaje de error y parando la ejecución del programa.
Si hacemos

```
>> mifuncion1(['a',0.1])
```

ahora nuestra función detecta la invalidez del parámetro de entrada y para la ejecución del programa sin intentar evaluar `cos('a')`.
Otro tipo de condicional en MATLAB es

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

En esta nueva construcción, si `condición 1` es verdadera, se ejecutarán las instrucciones en `instrucciones 1`. De lo contrario, se evalúa `condición 2`. Si ella es verdadera, se ejecutan las instrucciones en `instrucciones 2`. De lo contrario, se ejecutan las instrucciones en `instrucciones 3`.

CICLOS EN MATLAB

Consideremos la sucesión de Fibonacci

$$f_1 = 1, \quad f_2 = 2, \quad f_n = f_{n-1} + f_{n-2}, \quad n \geq 3.$$

Esta sucesión fue descrita por Fibonacci como solución a un problema de cría de conejos, f_n es el número de parejas de conejos en un recinto cerrado después de n meses si se supone que de cada par de conejos nace, después de 1 mes de convivencia, una nueva pareja de conejos.
Ejercicio 2: Escriba, con ayuda de las construcciones `for` y/o `while` descritas a continuación, una función en MATLAB (llámela `fibonacci`) que, dado $n \in \mathbb{N}$ devuelva los primeros n elementos de sucesión de Fibonacci. Llame a la función con distintos valores de n y grafique las sucesiones resultantes.

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`. Ella se inicializa con `vi` y toma los valores `vi`, `vi+incr`, `vi+vi+incr`, ..., `vf`.
Si, queremos, por ejemplo, sumar los n primeros números naturales con ayuda de un ciclo `for` deberíamos escribir

```
suma = 0
for i = 1:n
  suma = suma + i
end
```

2. Ciclo `while`: tiene la forma

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

Las instrucciones en el cuerpo del ciclo se ejecutan mientras `condición` sea verdadera.
Si, queremos, por ejemplo, sumar los n primeros números naturales con ayuda de un ciclo `while` deberíamos escribir

```
suma = 0
i = 1
while i <= n
  suma = suma + i
  i = i + 1
end
```

Trabajo con matrices

En clases vimos que la regla de Cramer no es una variante óptima para la solución numérica de sistemas de ecuaciones lineales por su alto costo operacional. Veamos un ejemplo para demostrar esto.

Ejercicio 3: Escriba una función en matlab, llamémosla `comparar_cramer_gauss` que reciba como parámetro de entrada un número natural n y haga lo siguiente:

1. Genere la matriz $A \in \mathbb{R}^{n\times n}$ y el vector $b \in \mathbb{R}^n$

$$A = \begin{pmatrix} 2 & -1 & 0 & 0 & \cdots & 0 & 0 \\ -1 & 2 & -1 & 0 & \cdots & 0 & 0 \\ 0 & -1 & 2 & -1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \cdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 2 & -1 \\ 0 & 0 & 0 & 0 & \cdots & -1 & 2 \end{pmatrix}, \quad b = \begin{pmatrix} 1 \\ 1 \\ 1 \\ \vdots \\ 1 \\ 1 \end{pmatrix}.$$

   Sugerencia: Sugerencia: Vea `diag` y `ones`.
2. Con ayuda de las funciones `det, tic, toc` de MATLAB , calcule el tiempo que transcurre al calcular los determinantes necesarios para obtener la solución al sistema de ecuaciones con la regla de Cramer (no es necesario que calcule la solución al sistema de ecuaciones, sólo queremos averiguar el costo de calcular los $n + 1$ determinantes involucrados en la solución con el método de Cramer).
3. Con `A\b` resuelva el sistema de ecuaciones mediante eliminación gaussiana con pivoteo, calculando también el tiempo necesario para ello.
4. Llame a su función con distintos valores de n y observe como, a medida que n aumenta la diferencia en tiempo transcurrido crece.

Índice alfabético