Introducción

Los métodos númericos son herramientas que nos permiten resolver problemas matemáticos utilizando operaciones aritméticas y lógicas.

En esta materia, aprederemos cómo crear "instrucciones" para que la computadora resuelva estos problemas. Estás instrucciones estarán escritas en un lenguaje determinado, en nuetro caso MATLAB. Una completa introducción a MATLAB puede ser encontrada en el github de la materia.

Dicho material, presupone que al alumno le resultan familiares ciertos temas relacionados con algoritmos y programación. De no ser el caso (ya sea, porque no se cursó algoritmos y programación o porque simplemente esos conocimientos desaparecieron) este articulo puede servir de introducción previa a la introducción a Matlab (algo así como la introducción de la introducción).

Breve intro a los algoritmos

Un algoritmo no es más que un conjunto ordenado de instrucciones que se llevan a cabo para solucionar un problema. Un ejemplo puede ser... una receta de cocina.

- Paso 1: Coloque el agua al fuego,
- Paso 2: Agregue un poco de sal y un chorrito de aceite.
- Paso 3: Cuando rompe el hervor agregué los fideos.
- Paso 4: Deje cocinar por 10'.
- Paso 5: Cuele.
- Paso 6: Agregue un poco de manteca.

Luego de 6 pasos, ya tengo mis fideos listos y solucione el problema (en este caso, hambre... o ganas de comer). Pueden encontrar más recetas en mi libro: algoritmos de cocina en 6 pasos (link) (https://www.youtube.com/watch?v=wpSxH3RxdLU).

El exito de los métodos numéricos, se basa justamente en que las computadoras son unas expertas en seguir recetas (algoritmos), y lo hacen de manera muy eficiente. Por eso, nos van a ayudar a resolver las recetas que escribamos en la materia. Hay 3 puntos a tener en cuenta:

1. En nuestro caso, las instrucciones van a ser de tipo aritmetico (por ejemplo, tomá este número, restale 2 y multiplicalo por 3), o lógico (hace esta cuenta solo si el número es par).
2. Las instrucciones tienen que estar escritas en un cierto lenguaje (Matlab) y ser escritas de manera precisa. Si yo te pido que coloques el agua al fuego, que pongas el agua a hervir, que pongas el agua a calentar, o incluso, si te digo que que le mandes mecha, me vas a entender. Bueno, con la compu no es tan facil. Tenemos que ser claros y estructurados, pero no te preocupes, que eso es algo que vamos a ir aprendiendo.
3. Muchas instrucciones son muy repetitivas (del estilo: a este número, sumale 2, y ahora sumale 2, y 2 más, y otros 2, así unas 100000 veces). Por lo que tendremos que aprender algunos trucos para no escribir 100000 pasos cuando haya instrucciones muy repetitivas.

Veamos entonces, algunas intrucciones básicas que podemos dar:

In [1]:
```
%Hace esta súma

2+2
```

```
ans =  4
```

In [2]:
```
%Hace esta multiplicación

2*3
```

```
ans =  6
```

In [3]:
```
%Crea el vector x=[1 2 3 4 5]

x=[1 2 3 4 5]
x=1:5
x=linspace(1,5,5)
```

```
x =

   1   2   3   4   5

x =

   1   2   3   4   5

x =

   1   2   3   4   5
```

In [4]:
```
%Hace este producto de matrices

[1 2; 3 4].*[1 0;0 1]
```

```
ans =

   1   0
   0   4
```

Un algoritmo, entonces, no va a ser más que un conjunto de estas instrucciones:

In [5]:
```
%Paso 1: A la variable x dale el valor 2
x=2
%Paso 2: A la variable y dale el valor 3.
y=3
%Paso 3: a x sumale 2, a y restale 1
x=x+2
y=y-1
%Paso 4: Calcula x^y y guardalo en la variable resp
resp=x^y
```

```
x =  2
y =  3
x =  4
y =  2
resp =  16
```

Resolviendo problemas

Ok, ya estamos listos para empezar a resolver problemas. Empecemos por uno sencillo:

Problema 1: Encuentre la suma de todos los numeros del 1 al 15.

La solución podría ser algo así:

In [6]:
```
1+2+3+4+5+6+7+8+9+10+11+12+13+14+15
```

```
ans =  120
```

Bien, llegamos a la respuesta. Pero digamos que, no de un modo muy elegante. Veamos entonces una herramienta que nos va a ahorrar mucho trabajo, en especial con instrucciones muy repetitivas.

Bucle "for"

La estructura de un "for", es algo así:

> *Para cada elemento de este "grupo":*
> *Hacé esto.*

Por ejemplo, podría ser:

> A cada elemento de [4 5 6 7 8]:
> Multiplica el elemento por 2

En matlab, se escribiría asi:

In [7]:
```
for i=[4 5 6 7 8]
    i*2 % aquí va la instrucción
end
```

```
ans =  8
ans =  10
ans =  12
ans =  14
ans =  16
```

Lo que hizo el **for**, fue tomar cada elemento del vector, asignarlo a la variable i, y realizar las cuentas que yo le indiqué. Puedo poner muchas instrucciones para cada elemento:

In [8]:
```
for i=[4 5 6 7 8]
    x=i-2;
    y=i+2;
    z=x*y
end
```

```
z =  12
z =  21
z =  32
z =  45
z =  60
```

Buenisimo!!! Entonces con un **for** podemos encontrar la solucion al problema 1 :) sería algo así:

In [ ]:
```
x=0;
for i=[1 2 3 4 5 6 7 8 9 10 11 12 13 14 15]
    x=x+i;
end
disp(x) % disp muestra el resultado de x
```

Bien. Llegamos al mismo resultado que antes, pero escribir [1 2 3 4 5 6 7 8 9 10 11 12 13 14 15] tampoco es muy elegante, por suerte, hay varias formas de escribir el mismo vector de una forma más corta, por ejemplo, si escribo 1:15, obtengo el mismo vector, mirá:

In [10]:
```
1:15
```

```
ans =

    1    2    3    4    5    6    7    8    9   10   11   12   13   14   15
```

Entonces, puedo llegar a la solucion así:

In [ ]:
```
x=0;
for i=1:20
    x=x+i;
end
disp(x)
```

Esta forma, además de ser mucho más elegante, me permite dejar de preocuparme por el número de pasos, si me piden por ejemplo, que sume todos los números del 1 al 10000, puedo hacerlo facilmente:

In [12]:
```
x=0;
for i=1:10000
    x=x+i;
end
disp(x)
```

```
 50005000
```

Condicional: If

Pasemos al siguiente problema:

Problema 2: Encuentre la suma de todos los numeros del 1 al 10000 que sean multiplos de 3.

A ver, tengo que sumar 3+6+...+9996+9999. Hay varias formas de hacer esto, pero intentemos usar lo que aprendímos recien: el **ciclo for**. Debería ser algo asi:

```
x=0;
for i=1:10000
   si i es divisible por 3, hacé esta cuenta: x=x+1
end
disp(x)
```

Cómo vemos, agregué un paso lógico a mis instrucciones. Solo se sumarán las i que son divisibles por 3. Es justo lo que necesitaba. El único problema, es que si lo escribo así, Matlab no me va a entender. Veamos cómo indicarle condiciones lógicas a matlab:

```
if condición 1:
    instrucción 1
elseif condición 2:
    instrucción 2
else:
    instrucción 3
end
```

Es decir:
Si (**if**) la condición 1 es verdadera, se ejecuta la instrucción 1.
Sino (**elseif**), verifico la condición 2:
Si la condición 2 es verdadera, se ejecuta la instrucción 2.
Sino (**else**) ejecuto la instrucción 3.

La condición debe ser un valor booleano. Es decir, *True* o *False*, por ejemplo:

- 2==3 ..... False
- 2==2 ..... True
- 2>3 ..... False
- 2<3 ..... True
- mod(12,3)==0 ..... True --- (la funcion mod(a,b) devuelve el resto de dividir a/b, por ejemplo, el resto de 7/4 es 3)
- mod(13,3) ..... False

Veamos un ejemplo:

In [13]:
```
x=2;
y=15
if x==1 %Condición 1
    y*2 %instrucción 1
elseif x==2  %Condición 2
    y*3 %instrucción 2
else
    y*5 %instrucción 3
end
```

```
y =  15
ans =  45
```

Vemos que x==1 es "False", entonces la instrucción 1 no se ejecuta. Se verifica la condición 2, es True, por lo cual, se ejecuta la instrucción 2.

Habiendo entendido esto, puedo volver al problema 2, cuya solución sería:

In [14]:
```
x=0;
for i=1:10000
    if mod(i,3)==0 %Condición=será "True" con cada i multiplo de 3
        x=x+i;
    end
end
disp(x)
```

```
 16668333
```

Funciones

Pasemos al siguiente problema:

Problema 3: Encuentre la suma de todos los numeros primos del 1 al 1000.

Teniendo en cuenta lo visto anteriormente, sabemos que la solución sería algo así:

```
x=0;
for i=1:10000
   if (i es primo?),
      x=x+1
    end
end
disp(x)
```

Aquí, el problema es que necesito saber cuando i es primo, pero Matlab no va a entender la expresión (i es primo?). Necesito ir un poco más alla. Necesito escribir algo (y aqui vienen las funciones a salvarnos), que cuando le pregunte si 67 es primo, me diga True, y cuando le pregunte si 9 es primo, me diga False. Veamos, entonces, cómo crear **funciones**.

La estructura general de una función es esta:

```
function y=nombre(x)
   instrucciones
end
```

Veamos un ejemplo simple para entender cómo funciona:

In [1]:
```
function resultado=por6mas2(x)
    aux=x*6;
    resultado=aux+2;
end

por6mas2(2)
por6mas2(9)
```

```
ans =  14
ans =  56
```

En el ejemplo, la función **por6mas2**, toma a x, lo multiplica por 6 y luego, a ese resultado se le suma 2. El valor de **resultado** es lo que devuelve la función.

Puedo entonces, escribir una función que me indique si un número es primo:

In [2]:
```
function resultado = primo(x)

%Los numeros 1 y 2, deben ser tratados de manera particular:
if x == 1
    resultado = false;
elseif x == 2
    resultado = true;
else

%Busco en todos los numeros entre 2 y x-1, si hay algun divisor de x. S
%i existe al menos 1 numero, x no es primo, por lo que el resultado es falso
    resultado = true;
    for i=2:x-1,
        if mod(x,i)==0
           resultado = false;
        end
    end
end
end
```

Teniendo esta función, finalmente, puedo resolver el **problema 3**:

In [3]:
```
x=0;
for i=1:1000
    if primo(i)
        x=x+i;
    end
end
disp(x)
```

```
 76127
```

En desarrollo...