Introducción práctica a la programación con Octave

95.10 Modelación Numérica / 75.12 Análisis Numérico I

23 de marzo de 2020

Índice

1. Introducción

En este documento se realiza una primera introducción a la programación con Octave orientada a la resolución de problemas numéricos.

2. Conceptos básicos de programación

2.1. Programa

El núcleo de la comptadora digital es su procesador o CPU. Este es capaz de realizar un conjunto de instrucciones muy sencillas. Un programa informático o programa de computadora es una secuencia de estas instrucciones, escritas de modo de realizar una tarea específica.

2.2. Lenguaje de Programación

Para que la computadora ejecute el programa directamente, el programa debe estar escrito en *lenguaje de máquina*. Este consiste en una serie de números binarios, que son leídos por los circuitos en secuencia por los circuitos electrónicos del procesador e interpetrados como instrucciones (de muy bajo nivel), direcciones de memoria y constantes numéricas. Este lenguaje de máquina es específico para cada familia de procesadores.

Típicamente los programas no se escriben directamente en lenguaje de máquina, ya que el proceso resultaría engorroso para los programadores y sería propenso a errores. En su lugar se escriben los programas en algún *lenguaje de programación*. En estos lenguajes, las instrucciones y direcciones de memoria se representan con palabras fácilmente comprensibles por seres humanos. Además, dado que las instrucciones no deben ejecutarse de manera directa por un circuito, en muchos lenguajes se utilizan instrucciones de *alto nivel*, que representan en una sola instrucción procesos de cierta complejidad.

Los programas escritos en lenguaje de programación deben ser traducidos a lenguaje de máquina para poder ser ejecutados. Hay dos paradigmas básicos para realizar esa traducción:

- Traducir el programa una única vez, guardando el programa final en código de máquina en un *archivo ejecutable*, específico para el tipo de máquina y sistema operativo. Este proceso de traducción que se denomina compilación.

- Conservar el programa en su lenguaje original, y traducirlo en tiempo real a medida que se ejecuta cada vez. Este proceso se denomina *interpretación*. Como desventaja, el proceso de interpretación ralentiza la ejecución en cierta medida. No obstante existen varias ventajas: por un lado el proceso de modificar el programa y correrlo se hace más ágil, y por otro el código del programa puede correrse en cualquier dispositivo y sistema operativo que cuente con un intérprete sin necesidad de modificarlo. Octave es un lenguaje interpretado.

2.3. Variables

Seguramente esté familiarizado con el concepto de *variable* que se utiliza en matemática, en el cual esta representa o bien una incógnita o un término de una ecuación no definido explicitamente. En computación, en cambio, el valor de las variables nunca es "desconocido", sino que siempre está definido de manera explícita, aunque no siempre el programador pueda anticipar qué valor tendrá la variable en cada instante. Por lo tanto, hay poca relación entro ambos conceptos de variable.

En computación, una variable no es más ni menos que un nombre asignado a una *posición de la memoria* de la computadora en la cual puede almacenarse un valor. El nombre de variable hace referencia a que, si bien las variables pueden contener un único dato en un determinado instante, este puede cambiarse o "sobreescribirse" tantas veces como sea necesario durante la ejecución de un programa.

2.3.1. Declaración de una variable

Típicamente en una computadora se ejecutan continuamente programas distintos que almacenan sus datos en la memoria. En el momento en que nuestro programa se inicia la memoria que utilizaremos puede contener cualquier conjunto de información antecedente de manera aleatoria. Por lo tanto, típicamente antes de utilizar una variable en el programa es necesario inicializarla.

El proceso de tomar una posición de memoria, reservarla para su uso en el programa y asignarle un nombre se denomina *crear*, *alocar* o *declarar* una variable.

En algunos lenguajes de programación, entre ellos Octave, las variables se inicializan automáticamente a un valor por defecto cuando se declaran.

Otros, por razones de eficiencia, confian que el programa las inicializará de manera explícita cuando sea necesario. En estos últimos es siempre importante recordar inicializar las variables, ya que sino el programa utilizará los valores antecedentes aleatorios contenidos en la memoria, lo que puede producir comportamientos inesperados.

2.3.2. Asignación de una variable

Al proceso de tomar un valor y almacenarlo en una variable se lo llama *asignación*. En muchos lenguajes de programación asignar un valor por primera vez a una variable es la forma más sencilla de declarla.

En la mayor parte de los lenguajes de programación, incluyendo Octave, cada sentencia de asignación inicia con un nombre de variable, por ejemplo `ancho`, luego sigue un signo igual y por último finaliza una expresión. La computadora evalua primero la expresión, calcula su resultado y por último lo almacena en la variable ubicada a la izquierda del igual.

La expresión puede contener cualquier combinación de valores constantes, variables y resultados de *funciones* (discutiremos estas últimas más adelante), unidos por operaciones.

Un ejemplo de sentencias de asignación en Octave es el siguiente:

```
ancho = 5.0
alto = 3.0
area = ancho * alto
```

En este sencillo programa estamos definiendo y asignando valores iniciales a tres variables. A las primera dos variables se les asignan valores constantes, mientras que a la tercera se le asigna el resultado de la multiplicación de las dos variables anteriores, que este caso será un valor de 15.

2.3.3. Sobreescritura de una variable

Es muy común encontrar sentencias de asignación en las cuales aparece las misma variable a ambos lados del signo igual, por ejemplo:

```
i = i + 1
```

esto puede resultar en principio confuso si lo pensamos como una expresión matemática. No obstante, hay que recordar que se trata de una sentencia de aignación y no una ecuación. Al igual que con los ejemplos anteriores la

computadora lee primero la expresión a la derecha del igual, calcula su resultado y por último lo almacena. Si por ejemplo el valor de `i` es inicialmente igual a 5, la computadora le sumará 1, obteniendo un 6, y lo almacenará finalmente en `i`, sobreescribiendo el valor anterior.

2.3.4. Tipos de datos

En las variables se pueden almacenar datos de distintos tipos. En algunos lenguajes de programación, llamados de *tipo fijo* (o *fixed type*) cada variable puede contener solo datos de un tipo específico definido por el programador. En cambio en otros, como Octave, a las variables puede asignárseles datos de cualquier tipo en cualquier momento.

Algunos tipos de datos que existen en Octave son:

- Escalar (`scalar`): Es un número individual, que puede ser entero, de punto flotante o complejo.
- Matríz (`matrix`): es un arreglo de números, con tantas filas y columnas como el usuario indique `N x M`. Si solo posee una columna podemos considerarlo un vector, pero técnicamente sigue siendo una matriz de `N x 1`.
- Cadena (`string`): es un conjunto de letras.

A continuación veremos ejemplos de asignación de variables con distintos tipos de datos:

```
miEntero = 2
miFlotante = 5.0
miMatriz = [0.1, 0.5; 0.2, 2; 0.3, 0]
miOtraMatriz = [1; 2; 3; 4]
miCadena = 'Esto es una cadena de texto'
```

2.3.5. Acceder a los elementos de una matriz

El tipo de datos matríz es sumamente importante en Octave. Un uso más o menos obvio es representar una matriz matemática para realizar cálculo algebraico. Pero además, si hay que realizar una misma operación N veces, resulta muchas veces conveniente almacenar cada resultado en una fila o columna distinta de una matriz. Si luego queremos recuperar el resultado de

una fila y columna en particular, es posible hacerlo indicando los índices de la matriz en las que se encuentra el valor en cuestión.

Si queremos por ejemplo recuperar el valor ubicado en la 2 fila y primera columa de la matriz definida en el ejemplo anterior, lo haremos de la siguiente manera:

```
valor = miMatriz(2, 1)
```

Esta misma metodología puede también usarse para escribir o sobreescribir un elemento específico de la matriz. Por ejemplo, para almacenar un valor en la fila 3, columna 2 podemos hacer:

```
miMatriz(3, 2) = valor
```

2.4. Estructuras de control

Ejecutar una lista de instrucciones individuales de manera secuencial puede no ser suficiente para resolver todos los problemas. No obstante, todos los lenguajes de programación inluyen la posibilidad de agregar *estructuras de control*. Estas son instrucciones especiales que permiten controlar el flujo del programa, permitiendo a este tomar decisiones, siempre basadas en las condiciones establecidas por el programador.

Las estructuras de control más comunes son los condicionales y los bucles.

2.4.1. Condicional If

Una estructura condicional permite seleccionar si una parte del programa debe ejecutarse o no de acuerdo a la evaluación de una condición.

Una condición es una expresión que al evaluarse puede ser *verdadera* o *falsa*. El tipo de condición más común involucra una comparación, por ejemplo si determinada variable es mayor o menor que tal valor, o si es igual a tal otro. Las condiciones pueden involucrar cualquier cantidad de variables y/o operaciones matemáticas y lógicas, funciones, etc

En Octave la estructura de control condicional más sencilla inicia con la instrucción `if` (en inglés *si*), seguida por la condición entre paréntesis. Luego se escriben las instrucciones a ejecutar en caso de que la condición sea verdadera, finalmente se cierra el bloque con la instrucción `endif` (*finalizar si*). Un ejemplo sencillo podría ser el siguiente:

```
if (i > 5)
```

```
    i = 5
  endif
```

En este caso, el programa se pregunta si el valor de la variable `i` es mayor que 5. Si lo es, la variable tomará un valor fijo igual a 5.

En ocasiones el algoritmo exige ejecutar un conjunto de instrucciones si la condición se cumple, y un conjunto distinto si no se cumple. Para estos casos, el lenguaje Octave provee la palabra clave `else` (en inglés *sino*). Por ejemplo:

```
if (i > 5)
  i = 5
else
  i = i + 1
endif
```

En este caso el valor de `i` se incrementa si no es menor que 5.

Dado que en la materia trabajaremos mayormente con variables numéricas es importante describir brevemente las condiciones matématicas básicas. Las más sencillas son:

- `a < b`: a menor que b
- `a > b`: a mayor que b
- `a == b`: a igual a b
- `a != b` o `a ~= b`: a distinto de b
- `a <= b`: a menor o igual a b
- `a >= b`: a mayor o igual a b

Distintas condiciones pueden ser combinadas en una sola utilizando los siguientes operadores lógicos:

- `&&`: and, en inglés *y*
- `||`: or, en inglés *o*
- `!`: not, en inglés *no*

A continuación se presentan distintas condiciones a modo de ejemplo:

```
% Si b es menor o igual a 2
if (b <= 2)

% Si a es igual a 5 y b es menor que 2
if ((a == 5) && (b < 2))

% Si a no es igual a 5 o b es menor que 2
if ((a != 5) || (b < 2))
```

2.4.2. Bucles For

Un bucle es un conjunto de instrucciones que se repiten un determinado número de veces. En particular, el bucle iniciado por la instrucción `for` se utiliza cuando la cantidad de veces a repetir el bloque es conocida en el momento de iniciar su ejecución.

La instrucción `for` (en inglés *para*) se sigue de una variable conocida como `indice`, y una expresión que define qué valores tomará la variable en cada repetición. Luego siguen las instrucciones a repetir y finalmente se cierra el bloque con la instrucción `endfor` (*finalizar para*)

Por ejemplo:

```
suma = 0
for i = 1:10
  suma = suma + i
endfor
```

En este caso el bucle se repite 10 veces: en la primera repetición i toma un valor de 1, en la segunda un valor de 2 y así hasta la última, en que toma un valor de 10. Luego de este pequeño fragmento de código la variable j contendrá la suma de los números naturales del 1 al 10.

Las expresiones para el bucle for pueden incluir dos o tres términos numéricos separados por dos puntos. Cuando solo se incluyen dos términos, como en el ejemplo anterior, el primero indica el valor inicial del índice y el segundo el valor final, y en cada iteración se incrementa el valor de la variable una unidad.

Cuando la espresión del bucle incluye tres términos, los términos iniciales y finales siguen en este caso indicando los valores iniciales y finales del iterador. El término central define en cambio el incremento de la variable, es decir, cuanto se modificará la variable en cada repetición. Por ejemplo:

```
suma = 0
for i = 1:2:10
  suma = suma + i
endfor
```

En este ejemplo la variable i iniciará en 1 y luego se incrementará de dos en dos mientras el valor resultante sea menor o igual que 10. Por lo tanto el bucle se repetirá 5 veces, con valores de `i` de 1, 3, 5, 7 y 9.

Es importante destacar que el incremento puede ser negativo, por ejemplo:

```
suma = 0
for i = 10:-1:1
  suma = suma + i
endfor
```

repetirá el bucle diez veces, con valores del índice de 10, 9, 8, ..., 2, 1.

2.4.3. Bucles While

A diferencia del bucle `for`, el bucle `while` se utiliza cuando el número de repeticiones no se conoce a priori. En cambio, el bloque de código dentro de estos bucles se repite tantas veces como haga falta mientras se cumpla una condición suministrada por el programador.

La sintaxis de estos bucles inicia con la palabra clave `while` (en inglés *mientras*), seguida de la condición expresada entre paréntesis. El cuerpo del bucle se cierra con la palabra clave `endwhile` (en inglés *fin mientras*).

A continuación se presenta un ejemplo. Este pequeño fragmento calcula cuantos números naturales consecutivos deben sumarse para totalizar un resultado de 100 o más.

```
i = 0
suma = 0
while (suma < 100)
  i = i + 1
  suma = suma + i
endwhile
```

Es importante destacar otra diferencia del bucle `while` respecto del bucle `for`. Mientras el bucle `for` incluye una variable de índice de manera implícita como parte de su sintaxis, el bucle `while` no lo hace. Por lo tanto, si se necesita incluir una variable para, por ejemplo, contar el número de repeticiones, esta debe declararse e incrementarse explícitamente.

2.4.4. Estructuras de control anidadas

Las estructuras de control pueden combinarse tantas veces como haga falta. Esto se hace colocando estructuras de control dentro de los cuerpos de otras estructuras de control. El proceso de colocar estructuras de control unas adentro de otras se conoce como *anidar*.

Si por ejemplo queremos sumar todos los números entre 1 y 100 que sean divisible por 3 y por 4:

```
suma = 0
for i = 1:100
  if (rem(i, 3) == 0)
    if (rem(i, 4) == 0)
      suma = suma + i
    endif
  endif
endfor
```

donde `rem(x, y)` es una función que devuelve el resto de la división entera x/y.