Programación con OpenOffice.org BASIC y OOo Calc

Para métodos numéricos

Walter Mora F.,
Escuela de Matemática
Instituto Tecnológico de Costa Rica.

Revista Digital Matemática Educación e Internet (www.cidse.itcr.ac.cr/revistamate)
Textos Universitarios

Correo Electrónico: wmora2@itcr.ac.cr
Escuela de Matemática
Instituto Tecnológico de Costa Rica
Apdo. 159-7050, Cartago
Teléfono (506)25502225
Fax (506)25502493

Mora Flores, Walter.
Programación con OpenOffice.org BASIC y OOo Calc/Walter Mora F.
– 1 ed.
– Escuela de Matemática,Instituto Tecnológico de Costa Rica.
ISBN
1. Programación de macros, 2. OpenOffice.org Basic. 3. OOo Calc 4. Métodos numéricos

Contenido

Bibliografía 64

1.1 Introducción

En este primer capítulo nos ocupamos de los constructores básicos del lenguaje OpenOffice.org Basic (OOo Basic). Luego consideramos las funciones y las subrutinas con el propósito de empezar a construir *una biblioteca* con funciones y subrutinas de uso frecuente en la implementación de algoritmos en métodos numéricos.

OpenOffice.org 3.x (`http://es.openoffice.org/`) es una suite ofimática (procesador de textos, hoja de cálculo, presentaciones, etc.) libre, disponible para varias plataformas, tales como Microsoft Windows, GNU/Linux, BSD, Solaris y Mac OS X.

Figura 1.1 Inicio de OpenOffice.org

En lo que nos concierne, vamos a usar el lenguaje de programación OOo Basic y la hoja electrónica OOo Calc. OOo Basic es una abreviación de "OpenOffice.org Basic". Calc es una hoja de cálculo si-milar a Excel y la programación de macros es muy similar a VBA para Excel. No se puede decir, en realidad, si es más sencillo usar VBA u OOo Basic; ambos hacen las cosas sencillas, pero a su manera.

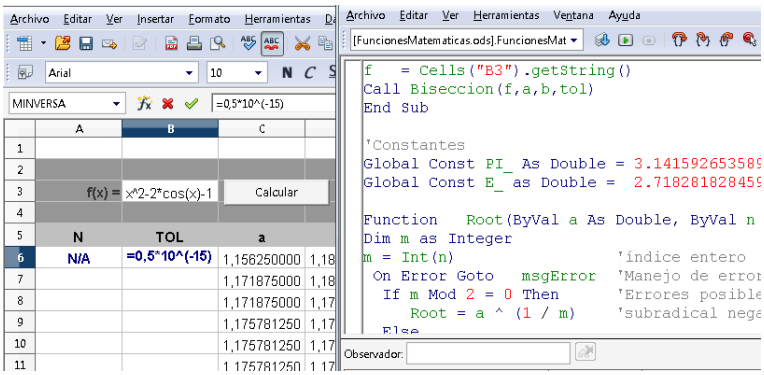


Figura 1.2 OOo Calc

VBA y OOo Basic son lenguajes de programación de la familia "Basic". Como tal, comparten los mismos constructores linguísticos básicos, por ejemplo, declaración de variables, ciclos (For, Do, While, Do While, Do Until,...), condicionales, operadores lógicos, funciones, etc. La diferencia está en el API, es decir, la manera de comunicarse con los documentos de OpenOffice.org. y el modelo de objetos de cada uno. Por tanto, si alguien ya está familiarizado con algún lenguaje de de la fa-

milia Basic (por ejemplo, VBA), se sentirá cómodo con OOo Basic.

Este capítulo está orientado a estudiar las construcciones necesarias para programar algoritmos en matemática e ingeniería, que se puedan usar en conjunto con OOo Calc. La programación de este capítulo está orientada a construir una biblioteca con subrutinas y funciones de propósito general y de uso frecuente en la programación de algoritmos en métodos numéricos y otras ramas.

1.2 Preliminares: Macros, funciones y subrutinas.

Una "macro" es una una colección ordenada de instrucciones o comandos. Las macros son programas, están constituidas por constantes, variables, instrucciones, subrutinas y funciones. Las subrutinas y funciones son programas y son un caso particular de macros.

1.2.1 Editar y ejecutar una macro.

Lo primero que hay que aprender (y recordar) es que las macros se guardan en *módulos*, estos a su vez se guardan y organizan en *bibliotecas*, las cuales, están contenidas dentro de documentos (cuadernos OOoCalc). Hay una biblioteca default llamada "`Standard`". Por ahora vamos a poner nuestras macros en un módulo en esta biblioteca `Standard`.

Sesión de programación en OOoCalc, con OOo Basic. Los pasos generales para hacer nuestro primer programa son: Preparamos la hoja OOoCalc, abrimos el entorno de programación, implementamos el programa (macro) y luego ejecutamos (y depurar si es el caso).

Nuestro primer programa es muy sencillo: Una macro que despliega una ventana con el mensaje `"Hola!"`.

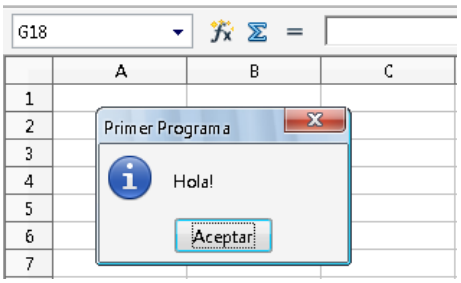


Figura 1.3 Primer programa: Una macro que despliega una ventana con el mensaje `"Hola!"`.

Pasos para implementar y ejecutar el programa.

a) Abrimos un libro OOoCalc y los guardamos, digamos como "LeerImprimir". Así, tendremos un archivo `LeerImprimir.ods` (el cual puede abrir en windows, Linux o Mac!).

b) Ahora vamos a crear un módulo llamado "`Module1`" (su nombre default) para editar subrutinas y funciones. Para esto vamos a

```
Herramientas>Macros>Organizar macros>OpenOffice.org Basic
```

y en la ventana `Macros básicas`, (1) seleccionamos nuestra hoja `LeerImprimir`, (2) hacemos clic en `'Nuevo'` y (3) hacemos clic en `Aceptar`. Esto nos llevará al "Entorno de Desarrollo Integrado" (IDE por sus siglas en inglés) de OOo Basic. La manera directa de hacer esto es presionar `Alt-F11-[Nuevo o Editar]`).

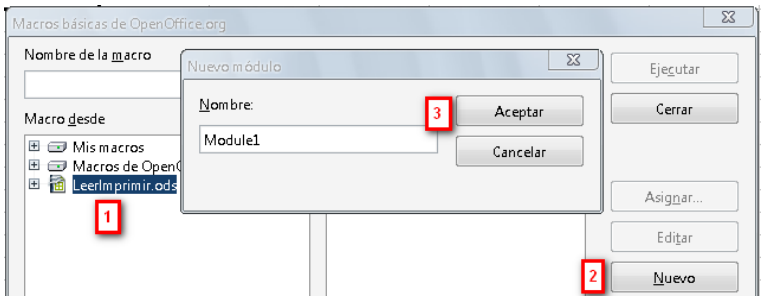


Figura 1.4 Crear Módulo

c) En el IDE de OOo Basic aparece la subrutina default "`Main`". Nuestro primer programa es muy simple: Enviamos un mensaje `Hola!` usando el comando `MsgBox`. Cuando entramos al IDE encontramos una `sub`rutina vacía

```
Sub  Main

 End Sub
```

Agregamos el código con el mensaje (el apóstrofo "`'`" se usa para poner comentarios),

Programa 1 *Utilizando la subrutina* `Main` *para levantar una ventana con un mensaje (figura 1.5).*

```
Sub  Main
'MsgBox "Mensaje", #=tipo de ventana, "título de la ventana"
  MsgBox "Hola!" , 64,"Primer programa"
End Sub
```

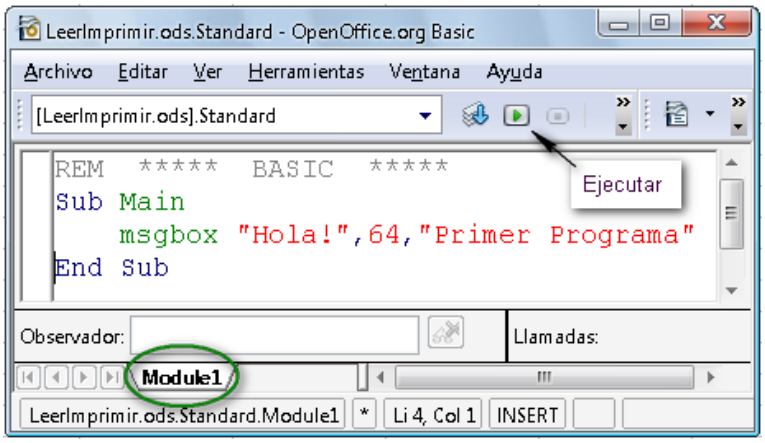


Figura 1.5 IDE de OOo Basic. Subrutina Main con un mensaje

d) Para *ejecutar* la subrutina `Main` podemos usar el botón de ejecución (ver figura 1.5) o, desde el cuaderno, usar la combinación de teclas `Alt-F11` seleccionar y presionar el botón de `Ejecutar`.

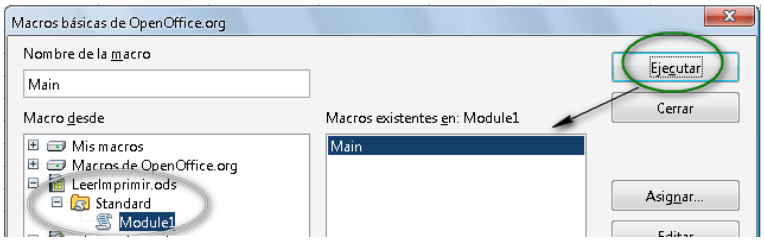


Figura 1.6 Ejecutar la subrutina con `Ctrl-F11`

Bibliotecas y módulos. Los módulos se deben poner en bi-bliotecas. La biblioteca default es la biblioteca `Standard`. En la figura 1.6 se observe que el `módulo 1` quedó en esta biblioteca. Más adelante volveremos sobre este tema.

Edición Los comentarios inician con `REM` o con un apóstrofo recto (`'`). El compilador no es sensitivo a las mayúsculas y minúsculas, es decir, es lo mismo escribir `MaxIteraciones` que `maxIteraciones`.

Ventanas de mensaje. Hay varios tipos de ventanas de mensaje. En el siguiente código se muestra cuatro opciones.

Programa 2 *Tipos de ventanas de mensaje*

```
Sub  Main
 MsgBox " ", 16, "Icono  Stop"        'Mensaje vacío
 MsgBox " ", 32, "Icono  pregunta"
 MsgBox " ", 48, "Icono  exclamación"
 MsgBox " ", 64, "Icono  información"
End Sub
```

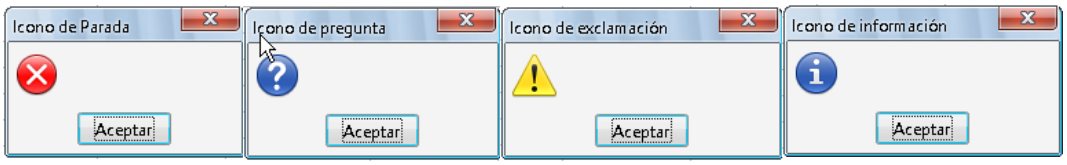


Figura 1.7 Iconos para MsgBox (mensaje vacío)

1.2.2 Subrutinas y funciones.

Una subrutina es una colección de instrucciones (es un caso especial de macro). Podemos usar una subrutina para incluir las instrucciones para leer una valor de una celda e imprimir en otra celda. Una función es una macro que devuelve (generalmente) un valor. Podemos usar una función para implementar las funciones usuales en matemática.

Cuando ejecutamos una subrutina, se ejecutan las instrucciones que contiene. En OOo Basic hay una subrutina default, la subrutina `Main`.

```
Sub  Main

End Sub
```

Esta es la primera subrutina que se ejecuta. En principio, como no tiene instrucciones, no pasa nada.

La estructura de una función es (lo que está entre "[ ]" es opcional),

```
Function NombreFuncion([argumentos])
    Variables
    instrucciones
  [ Exit  Function]
    instrucciones
  NombreFuncion= lo que retorna la función
End Function
```

La línea opcional `Exit Function` se usa en el caso de que, por alguna razón, queramos detener el cálculo y salir de la función

La estructura de una subrutina es (lo que está entre "[ ]" es opcional),

```
Sub  NombreSubrutina([argumentos])
    Variables
    instrucciones
    [ Exit Sub]
    instrucciones
End Sub
```

La línea opcional `Exit Sub` se usa en el caso de que, por alguna razón, queramos terminar ahí la subrutina.

Funciones matemáticas. Las funciones matemáticas se implementan de manera natural. Considere $f(x) = x^3 + x + 1$ y $g(x,y) = \sqrt{x^2+y^2}$. La función f tiene un argumento y la función g tiene dos argumentos.

Programa 3 *Implemetación de las funciones* $f(x) = x^3 + x + 1$ *y* $g(x,y) = \sqrt{x^2+y^2}$.

```
1 Function   F(x)
2       F = x^2+x+1
3 End Function
4
5 Function   G(x,y)
6      G = Sqr(x^2+y^2)
7 End Function
```

En la línea `6.` se usa la función Basic `Sqr` para la raíz cuadrada.

Usar una función en una hoja. Una vez definida la función en un módulo de la biblioteca `Standard`, ya es accesible en el cuaderno actual y se pueden usar en las fórmulas. Solo *un detalle*: En el cuaderno los argumentos se separan con punto y coma (";") mientras que en el ambiente OOo Basic se usa la coma corriente.

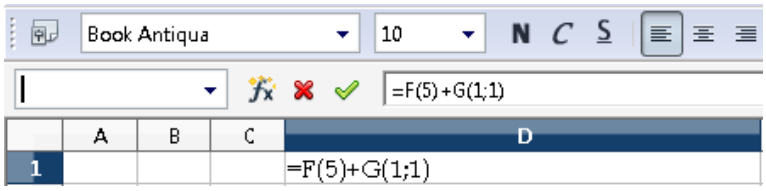


Figura 1.8 Usando una funciones en el cuaderno OOoCalc

1.2.3 Variables.

Las variables contienen valores que pueden cambiar en la ejecución de una macro. Aunque OOo Basic no nos obliga a declarar variables, en la práctica es mejor hacerlo.

Los nombre de las variables inician con una letra (A−Z o a−z). Se pueden usar números (0−9) y el guión bajo (_) pero no al principio.

"Option Explicit". Es bueno declarar las variables y también es muy práctico agregar la instrucción `"Option Explicit"` para detectar nombres que no corresponden a alguna variable. Esto va a ahorrar mucho tiempo a la hora de buscar errores en nuestro código.

Dim. Para declarar una variable se usa la palabra reservada "`Dim`" y se puede agregar el tipo `Integer` para enteros y `Long` para enteros grandes, `Double` para números en doble precisión, `String` para cadenas de caracteres, `Boolean` para variables que toman los valores verdadero o false, etc.

Variables tipo `Variant`. Sino se declara el tipo, se asume que es tipo "`Variant`" y todo se acomoda al tipo de dato. No siempre es bueno usar el default "`Variant`" porque a veces no es claro, después de algunas asignaciones, en que tipo de dato se va a convertir este "`Variant`".

Variable tipo `Integer`. Las variables tipo `Integer` se inicializan en 0 y soportan valores entre -32768 y 32767.

Variable tipo `Long`. Las variables tipo `Long` se inicializan en 0 y soportan valores entre -2147483648 y 2147483647.

Variable tipo `Double`. Las variables tipo `Double` se inicializan en $0,0$ y soportan números positivos y negativos entre $\pm 1,79769313486232 \times 10^{308}$ y $\pm 4,94065645841247 \times 10^{-324}$.

Los computadores comunes representan los números reales en este rango con un número aproximado. El error *relativo* entre el número real x y su aproximación $\widetilde{x}$ en el computador, es menor o igual que 0.5×10^{-15}. Los números positivos inferiores a $4.94065645841247 \times 10^{-324}$ se tratan

como 0.

El computador representa los números `Double` con a lo sumo 15 o 16 dígitos significativos.

		Número real	*Representación en el computador*
c_1	=	299792458112345699	2,99792458112346E+017
$c_2 = c_1 + 16$	=	299792458112345715	2,99792458112346E+017
20!	=	243290200817664	2,43290200817664E+018
21!	=	5109094217170944	5,10909421717094E+019

Los números c_1, c_2 tienen más de 15 dígitos significativos, en la suma real difieren pero la suma en el computador resulta igual. Hay que tomar en cuenta que en una aproximación, 16 es porcentualmente despreciable respecto al número 299792458112345699. Por otra parte, 20! se puede representar de manera exacta (como un `Double`) pero 21! (16 dígitos) se representa aproximadamente.

Como decíamos más arriba, si la representación del número real x es $\tilde{x}$, entonces

$$\left|\frac{x-\tilde{x}}{x}\right| \leq 0.5 \times 10^{-15}$$

Programa 4 *Declaración de variables y algunos cálculos.*

```
1  Option Explicit
2  Sub  Prueba()
3  Dim a,b                'a y b son de tipo Variant
4  Dim q1   As Integer    'q es de tipo entero
5  Dim q2   As Double     'q es de tipo Double
6
7  'Las variables se pueden inicializar en una
8  'sola línea si las  separamos con ":"
9   a  = 5665555737832145 : b = a+20
10 MsgBox b               '-> 5,66555573783216E+015
11  q1 = a/b              'División real
12 MsgBox q1              '-> 1
13  q2 = a\b               'División Entera: -> Error
14 End Sub
```

El cálculo en la línea `11.`, `q1=a/b` devuelve 1 pues por la precisión que maneja el computador (15 o 16 dígitos), `a=b+20`.

El cálculo en la línea `13.`, `q2=a\b` provoca un error en tiempo de corrida: La división entera espera dividir dos enteros y retornar un entero, si `a` y `b` no son enteros, deben ser truncados para convertirlos en enteros, pero en el ejemplo estos números están fuera del rango de los enteros. Se despliega una venta de error con el mensaje `"Tipo de datos o valor inadmisible. Desbordamiento."`

Cadenas de caracteres (String). En el manejo de texto es necesario conocer algunas operaciones sobre cadenas de caracteres ((String).) Los espacios *en blanco* cuentan como caracteres.

- Declaración de variables: `Dimtxt As String`
- Pasar un número x a String: `Str(x)`
- Concatenar: `txt = Str(x)& " es real"` o también `Str(x) + " es real"`.
- Accesar partes de un String:
  (a) `Left(txt, n)` muestra los primeros `n` caracteres de `txt`.
  (b) `Right(txt, n)` devuelve los últimos `n` caracteres de `txt`.
  (c) `Mid (txt, m, n)` devuelve los primeros `n` caracteres de `txt` desde la posición `m`.
  (d) `Len(txt)` devuelve el número de caracteres de `txt`.
  (e) `Trim(txt)` elimina espacios iniciales y finales. `Trim(" Hola! ") -> "Hola!"`.

Programa 5 *Ejemplo para mostrar el uso de algunos métodos en cadenas de caracteres.*

```
Sub  Texto()
Dim txt     As String
Dim rt      As String
Dim lg      As Integer

 txt = "Texto de prueba"
 rt  =  Left(txt,6)     'rt -> "Texto ", con un blanco al final.
 rt  =  Right(txt, 5)   'rt -> "rueba", sin blancos.
 rt  =  Mid(txt, 8, 5)  'rt -> "e pru".
 lg  =  Len(txt)        'lg -> 15.

'Mid con cuatro argumentos pasa a ser instrucción, es decir,
'modifica la tira txt pero no devuelve algo.
'La siguiente instrucción modifica txt, cambia " de " por "=".
 Mid(txt, 6, 4, "=")        'Ahora txt = "Texto=prueba"

End Sub
```

- Buscar y reemplazar: La instrucción `InStr(PosInicial, txt, txtBuscado)` devuelve una número con la posición en la que se encontró la primera aparición, desde la posición `PosInicial`, de la subcadena `txtBuscado` en la cadena `txt` . Este comando ignora mayúsculas y minúsculas. También se puede usar `Mid` con cuatro argumentos como se muestra en el ejemplo,

Programa 6 *Uso de* `InStr`.

```
Sub  Posicion()
Dim txt     As String
Dim i     As Integer

 txt = "Texto de prueba"
 i =  inStr(1,txt," de ")    '-> i=6, pues " de " inicia en un blanco.

 Mid(txt, 6, 4, "=")        'Ahora txt = "Texto=prueba"
```

```
 9    MsgBox  inStr(1,txt," de ") '-> i=0
10   End Sub
```

Ámbito de las variables. Las variables *locales* son las que se declararan dentro del cuerpo de una subrutina o función y se crean al invocar ésta y se destruyen al finalizar. Si estas variables se declaran `Static` (en vez de `Dim`) entonces conservan el último valor que tuvieron, entre llamada y llamada (siempre y cuando no cambie la macro que llama a la macro que la contiene). En el siguiente código se muestra la misma función pero con los dos tipos de variable.

```
Sub  Main
 MsgBox Dsucesor(1)   'Imprime 1
 MsgBox Dsucesor(1)   'Imprime 1
 MsgBox Dsucesor(1)   'Imprime 1
                      '---------
 MsgBox Stsucesor(1) 'Imprime 1
 MsgBox Stsucesor(1) 'Imprime 2
 MsgBox Stsucesor(1) 'Imprime 3
End Sub

Function  Dsucesor(n  As Integer)  As Integer
Dim el_sucesor  As Integer
 el_sucesor = el_sucesor+n
  Dsucesor = el_sucesor
End Function

Function  Stsucesor(n  As Integer)  As Integer
 Static el_sucesor  As Integer
 el_sucesor = el_sucesor+n
  Stsucesor = el_sucesor
End Function
```

Las variables *globales de dominio público* de un módulo se declaran al inicio del módulo y son visibles para todos los módulos de la biblioteca. Se pueden usar y modificar en las subrutinas y funciones. No conserva su último valor.

```
 Option Explicit
Dim ValorGeneral  As Integer

Sub Main
End Sub
```

Las variables *globales* son las mismas que las variables de dominio público excepto que mantienen su último valor aunque se termine la macro que la utilizó. Se declaran al inicio del módulo y son visibles para todos los módulos del Cuaderno Calc. Se pueden usar y modificar en las subrutinas

y funciones.

```
 Option Explicit
 Global ValorG  As Integer 'ValG es global y conserva su último valor

Sub Main
End Sub
```

1.2.4 Constantes

Para declarar una constante, usamos la palabra clave `Const` de la siguiente manera,

```
 Const PI  As Double = 3.14159265358979
 Const E   As Double =  2.71828182845905
```

Para que las constantes sean visibles para todos los módulos usamos `Global` (por ser constantes no se pueden modificar).

```
'PI_ y E_ son visibles para todos los módulos de este cuaderno
 Global Const PI  As Double = 3.14159265358979
 Global Const E  As Double =  2.71828182845905
```

Constantes para OOoCalc. Las constantes definidas en OOo Basic se pueden usar en la hoja OOo Calc si se implementan como una función, digamos en la biblioteca `Standard`. π ya tiene una implementación, la base e de los logaritmos naturales se implementa así,

Programa 7 *Constante e y función* `E_()`

```
1  Global Const E As Double =  2.71828182845905
2                        'π para la hoja: Ya está definida como PI()
3  Function E_()         'Constante e para la hoja. La función debe tener
4     E_=2.71828182845905 'un nombre diferente al de la constante.
5  End Function
```

Ahora podemos evaluar en la hoja fórmulas con esta función, por ejemplo `"=2*E_()^2"`.

Nota. En OOo Basic se usa `E` para escribir números en notación científica, pero esto no presenta problemas con la constante `E`. Por ejemplo OOo Basic entiende `0.1E+2` como `10` y entiende `0.1E+2+E` como `12.71828182...`

1.2.5 Operadores

Como es usual, las operaciones aritméticas son `+`, `-`, `*` y `^` para los exponentes. El operador `"+"` también se usa para concatenar cadenas de caracteres. En las tablas (**??**) y (**??**) se muestra una lista de operadores y funciones de uso frecuente.

Operador	*Significado*	*Ejemplo*
`Mod`	Resto de la división entera.	`-5 mod 3 -> -2`
`\`	División entera	`2\3 =0` y `7\4 =1.`
`&`	Concatenación de cadenas	`"xi =" & "g(xi)" -> "xi=g(xi)"`
`<=`	$\leq$	
`>=`	$\geq$	
`<>`	$\neq$	`4 mod 3 <> 0 -> true`
`Not`	Operador lógico $\neg$ (negación).	
`AND`	Operador lógico `"y"`	
`OR`	Operador lógico `"o"`	
`XOR`	Operador lógico `"ó"`	`True XOR True = False,True XOR False = True`
`ABS(x)`	El valor absoluto de un número	`Abs(-2) -> 2`
`CLng(x)`	Redondea al `Long` más cercano.	
`Fix(x)`	Trucar la parte decimal.	
`Int(x)`	Parte entera (entero de la izquierda).	`Int(3.99)= 3, Int(-0.47)= -1.`
`CInt(x)`	Redondea al entero más cercano.	`CInt(3.99)= 4, CInt(-0.47)= 0`
`Str(x)`	Convierte en `String`	`Str(2.45) ->"2.45"`
`CDbl(x)`	Convierte a `Double`	`CDbl("3,2")+3 -> 6,2; CDbl("0,45E3") -> 450`
`Val(x)`	Convierte en `Double`	`Val("3.2") +3 -> 6,2; Val("0.45E2") -> 450`
`Rnd()`	número aleatorio entre 0 y 1	`2*rnd() - 1 -> número aleatorio en [-1,1]`
`SGN(x)`	signo	`sgn(-5.5) -> -1,sgn(0)-> 0`
`SQR(x)`	$\sqrt{x}$	
`Log(x)`	El logaritmo natural de un número.	
`Exp(x)`	e^x	
`SIN(x)`	$\text{sen}(x)$	
`COS(x)`	$\cos(x)$	
`TAN(x)`	$\tan(x)$	
`ATN(x)`	Arcotangente	`atan(x)` $\in]-\pi/2,\pi/2[$

Notas. Según las reglas de conversión implícitas, `"3,2"+3` pasa a ser la cadena `"3,23"` mientras que `CDbl("3,2")+3` pasa a ser el número `6,2`. Hay que tener en cuenta que `CDbl` respeta la configuración regional de idioma y reconoce la coma como separador decimal, esto significa que en español ignora el punto: `CDbl("3.2")+3 ->35`. En cambio `Val` reconoce el punto como separador decimal pero devuelve el resultado de acuerdo a la configuración regional, por ejemplo `Val("3.2")+3 ->6,2; Val("0.1E2")-> 10` pero `Val("0,1E2")-> 100`.

1.2.6 Ciclos.

En métodos numéricos es muy frecuente acumular sumas o productos y manejar esquemas iterativos que requieren repetir algunos cálculos hasta que se cumpla alguna condición, esto se hace con estructuras de código que manejan ciclos.

Ciclo For. Este ciclo repite un bloque de instrucciones un número determinado de veces. La sintaxis general es (lo que está entre "[ ]" es opcional),

```
For  contador = inicio To fin [ Step ValordeSalto ]
      instrucciones...
      [ Exit For ]
Next [contador]
```

Acumular sumandos. Vamos a implementar una función `ElSumatorio(ini,n)` para calcular el sumatorio

$$\sum_{i=\text{ini}}^{n} 1/2^i.$$

Este ejemplo muestra una idea muy usada para acumular sumandos. Para calcular el sumatorio $\sum_{i=\text{ini}}^{n} 1/2^i$ procedemos de manera natural. Usamos una variable `suma` con valor inicial cero y acumulamos cada nuevo sumando, uno por uno,

$$\begin{array}{lrcrcrcl}
 & \texttt{suma} & = & 0, & & & & \\
i=0, & \texttt{suma} & = & \texttt{suma} & +1/2^0 & = & 0+1 & = & 1, \\
i=1, & \texttt{suma} & = & \texttt{suma} & +1/2^1 & = & 1+1/2 & = & 3/2, \\
i=2, & \texttt{suma} & = & \texttt{suma} & +1/2^2 & = & 3/2+1/4 & = & 7/4, \\
\ldots & & & & & & & &
\end{array}$$

Programa 8 *Un ejemplo de cómo acumular sumandos: Un sumatorio*

```
1 Function ElSumatorio(ini, n)
2 Dim i, suma
3     suma=0
4    For  i= ini To n
5         suma = suma + 1/2^i
6     Next i
7     ElSumatorio = suma
8 End Function
```

Acumular factores. Para mostrar la manera de acumular factores en un producto vamos a implementar la función `factorial(n)` . Recordemos que $n! = 2 \cdot 3 \cdots (n-1) \cdot n$.

Procedemos de manera natural. Usamos una variable `producto` con valor inicial 1, acumulamos cada nuevo factor, uno por uno,

$$\begin{array}{lrcrcrcl}
 & \texttt{producto} & = & 1, & & & & \\
i=2, & \texttt{producto} & = & \texttt{producto}*2 & = 1*2 & = & 2 & \\
i=3, & \texttt{producto} & = & \texttt{producto}*3 & = 2*3 & = & 6 & \\
i=4, & \texttt{producto} & = & \texttt{producto}*4 & = 6*4 & = & 24 & \\
\ldots & & & & & & & &
\end{array}$$

Programa 9 *Un ejemplo de cómo acumular factores: La función factorial.*

```
1 Function Factorial(n) '1 ≤ n ≤ 170. Valores exactos hasta n = 20.
2 Dim i, producto
3     producto=1
4    For  i= 2 To n
5         producto = producto*i
```

```
    Next i
    Factorial = producto
End Function
```

Podemos usar la misma idea del programa anterior para implementar una función `CoefBinomial(s,n)`. Recordemos que si $s \in \mathbb{R}$ y $n \in \mathbb{N}$, $\binom{s}{0} = 1$ y $\binom{s}{1} = s$ y en general,

$$\binom{s}{n} = \frac{s(s-1)(s-2)\dots(s-n+1)}{n!}.$$

Procedemos de manera natural. Usamos una variable `producto` con valor inicial 1, acumulamos cada nuevo factor, uno por uno. El código es,

Programa 10 *Coeficiente binomial* $\binom{s}{n}$.

```
Function CoefBinomial(s,n)
Dim i, producto

    producto=1
    If n>0 Then
     For  i= 0 To n-1
            producto = producto*(s-i)/(i+1)
      Next i
    End If
    CoefBinomial = producto
End Function
```

Ciclo Do...Loop. Este ciclo viene en diferentes sabores. Se utiliza para hacer la ejecución de un bloque de código mientras o hasta que una condición se cumpla. El uso más común es verificar la veracidad de la condición antes de ejecutar el código. El código se ejecuta repetidamente mientras la condición sea `true`. Si la condición es falsa, el código nunca se ejecuta. La sintaxis general es (lo que está entre "[ ]" es opcional),

```
Do While condición
  instrucciones...
  [ Exit Do]
  instrucciones...
Loop
```

En otra forma del ciclo `Do...Loop`, el código se ejecuta repetidamente mientras la condición es falsa. En otros palabras, el código se ejecuta hasta que la condición se convierte en verdad. *Si la condición se evalúa como verdadera al inicio, el ciclo nunca se ejecuta.* La sintaxis general es (lo que está

entre "[ ]" es opcional),

```
Do Until condición
  instrucciones...
 [ Exit Do]
  instrucciones...
Loop
```

Se puede colocar el control al final del ciclo, en cuyo caso *el bloque de código se ejecuta al menos una vez*. En este caso, el bucle se ejecuta al menos una vez y luego se ejecuta varias veces mientras la condición sea verdadera. La sintaxis general es (lo que está entre "[ ]" es opcional),

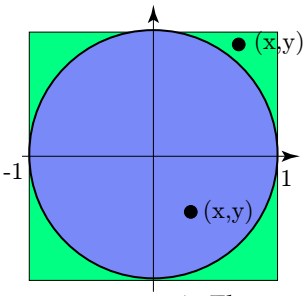
```
Do
   instrucciones...
   [ Exit Do]
   instrucciones...
Loop While condición
```

Para ejecutar el ciclo al menos una vez y luego continuar mientras la condición sea falsa, utilice el siguiente constructor (lo que está entre "[ ]" es opcional),

```
Do
   instrucciones...
   [ Exit Do]
   instrucciones...
Loop Until condición
```

Ciclo While ... Wend. Este ciclo se usa cuando se quiere repetir un bloque de código mientras una condición es `true`. Este ciclo no ofrece beneficios adicionales al ciclo `Do...Loop`, por ejemplo no hay un "`Exit While`". La sintaxis general es

Números aleatorios en un círculo. Para obtener pares (x,y) aleatoriamente distribuidos en un círculo de radio 1 se generan dos números aleatorios x, y entre -1 y 1. Este par estará en un cuadrado de lado 2 centrado en el origen. Este círculo tiene área π mientras que el cuadrado tiene área 4. Si generamos un par (x,y) aleatorio con $-1 < x,y < 1$, la probabilidad de que este par quede dentro del círculo es $\pi/4$. Los pares (x,y) en el círculo de radio 1 tienen la propiedad $x^2+y^2 \leq 1$.



Como `rnd()` genera un número aleatorio entre 0 y 1, entonces $-1 \leq$ `2*rnd()-1` ≤ 1. El par aleatorio lo generamos con `x = 2*rnd()-1` y `y = 2*rnd()-1`. Esto lo hacemos *mientras que*

$\sqrt{x^2+y^2} > 1$ (el programa que sigue usa arreglos, una exposición más amplia de los arreglo se puede ver en 1.6.5).

Programa 11 *Generar pares (x,y) aleatoriamente distribuidos en un círculo de radio 1*

```
Sub  Main
  Dim p()
    p =  xyCAleatorio()'generamos el par aleatorio en el círculo
  MsgBox  Str(p(0))+"  "+Str(p(1))
End Sub

Function  xyCAleatorio()
Dim x,y
Dim par()
  par = Array(0,0) 'inicializamos el par
Do 'ciclo se ejecuta de nuevo solo si (x,y) queda fuera del círculo
  x = 2*rnd()-1
  y = 2*rnd()-1
Loop While Sqr(x^2+y^2)>1
 par(0)= x 'p(0) es la componente x
 par(1)= y 'p(1) es la componente y
  xyCAleatorio=par()
End Function
```

El número esperado de veces que se ejecuta el ciclo `Do` es $4/\pi \approx 1.27$ veces!

Números armónicos. Los números armónicos son los números $H_N = \sum_{n=1}^{N} 1/n$. En análisis matemático se establece que $H_N \approx \ln(N) + 0.57721$ para N grande (ver [8]). El siguiente programa calcula el $N-$ésimo número armónico usando un ciclo `Do...Loop Until`,

Programa 12 *Números armónicos* $H_N = \sum_{i=1}^{N} 1/i$

```
Function NHarmonico(N)
Dim i As Long
Dim suma As Double
    suma=0 : i=1
    Do
      suma = suma+ 1/i
         i = i+1
    Loop Until  i > N
    NHarmonico = suma
End Function
```

=NHARMONICO(I2)

I	J	K
N	H_N	Ln(N) +0,57721
10	2,9289682539682500	2,8797950929940500
100	5,1873775176396200	5,1823801859880900
10000	9,7876060360443500	9,7875503719761800
100000	12,0901461298633000	12,0901354649702000
1000000	14,3927267228650000	14,3927205579643000

Figura 1.9 Comparando H_N con $\ln(N) + 0.57721$

1.2.7 Condicionales.

La condición `If` es usado para ejecutar un bloque de código de acuerdo a si se cumple o no una condición. La forma más sencilla de esta instrucción es

```
If condición Then
        instrucciones...
End If
```

La condición puede ser cualquier expresión que se evalúa con `true` o `false`. La sintaxis general es (lo que está entre "[ ]" es opcional),

```
If condición1 Then
        instrucciones...
[Else If condición2 Then]
        instrucciones...
[Else]
        instrucciones...
End If
```

Si la primera condición que se evalúa es `true`, se ejecuta el primer bloque de código. Se pueden usar varias declaraciones del tipo `ElseIf` para probar otras condiciones. La declaración `Else` se ejecuta si ninguna de las otras condiciones se evalúa como `true`.

Una función a trozos. Vamos a implementar la función

$$T(a,x) = \begin{cases} a\ln(x) & \text{si} \quad x > 0, \\ \cos(a/x) + \operatorname{sen}(a/x) + e^x & \text{si} \quad x < 0, \\ 0 & \text{si} \quad x = 0. \end{cases}$$

No podemos usar `T(a,x)` pues `T` ya está reservado para una función que trabaja con texto. Para no tener problemas, cambiamos el nombre a `fT`. El código podría ser,

Programa 13 *Función* `T(a,x)`

```
Function  fT(a,x)
 Dim Tax
 If x>0 Then
         Tax=Log(x)
 Else If x<0 Then
         Tax=cos(a/x)+sin(a/x)+Exp(x)
 Else
         Tax=0
 End If
      fT= Tax
End Function
```

1.3 Leer e imprimir en una celda.

En lo que nos concierne, lo primero que tenemos que aprender es cómo leer *datos numéricos* de las celdas y como imprimir datos numéricos en una o varias celdas. Para leer datos de una celda se debe indicar la hoja que vamos a usar, para esto usamos el objeto `ThisComponent` que designa el documento desde donde vamos a llamar la macro que editamos. La hoja 1 corresponde a la hoja 0.

Leer o imprimir en la celda usando el nombre. Para leer o imprimir el valor numérico almacenado en una celda usando el nombre de la celda, se usa el método `getCellRangeByName` y la propiedad `Value`.

Programa 14 *Leer* x *en* `"A4"` *e imprimir* $f(x)$ *en* `"B4"`, desde la subrutina `Main`.

```
Sub  Main
Dim Hoja, x
 Hoja = ThisComponent.Sheets(0)              'Hoja 0
 x    = Hoja.getCellRangeByName("A4").Value'Lee el valor numérico en "A4"
                                            'y lo almacena en x.
 Hoja.getCellRangeByName("B4").Value = f(x)'Imprime f(x) en "B4"
End Sub

Function   F(x)
      F = x^2+x+1
End Function
```

	A	B	C
1			
2			
3	x	f(x)	
4	4	21	

Figura 1.10 Leer $x = 4$ en "A4" e imprimir $f(4) = 21$ en "B4"

Leer o imprimir en la celda usando la posición. La hoja electrónica se puede ver como una matriz.

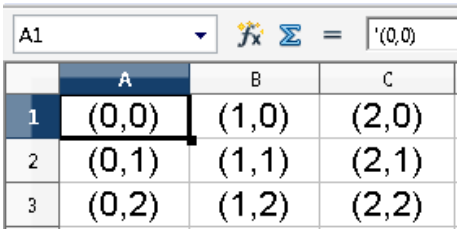


Figura 1.11 La celda `A1` corresponde celda $(0,0)$.

La celda (c,f) corresponde a la columna c y la fila f. La columna `A` es la columna 0, la columna `B` es la columna 1, etc. La celda `"A1"` es la entrada $(0,0)$, la celda `"A2"` es la celda $(0,1)$,la celda `"A3"` es la celda $(0,2)$ y la celda `"An"` es la celda $(0,n-1)$. La celda `"Bn"` es la celda $(1,n-1)$, etc.

Para leer o imprimir un *valor numérico* almacenado en una celda usando su posición en la matriz, se usa el método `getCellByPosition` y la propiedad `Value`.

Programa 15 *Leer x en `"A4"=(0,3)` e imprimir $f(x)$ en `"B4"=(1,3)`*, desde la subrutina `Main`.

```
Option Explicit
Sub Main
Dim Hoja, x
  Hoja = ThisComponent.Sheets(0)          'Corresponde a la Hoja 1
  x    = Hoja.getCellByPosition(0,3).Value'Contenido numérico de "A4"
  Hoja.getCellByPosition(1,3).Value = f(x)'imprime f(x) en "B4"
End Sub
```

Podemos ahora imprimir una comparación entre los números armónicos y la aproximación $H_N \approx \ln(N)+0.57721$. En el cuaderno de la figura (1.12) los números armónicos se imprimen en las celdas `J2,J3,J4,...` que corresponde a las celdas `(9,i)` con $i=1,2,...$ y el valor de comparación se imprime en las celdas `K1,K2,...` que corresponde a las celdas `(10,i)` con $i=1,2,...$

	J	K
1	H_N	Ln(N) +0,57721
2	2,9289682539682500	2,8797950929940500
3	5,1873775176396200	5,1823801859880900
4	7,4854708605503400	7,4849652789821400
5	9,7876060360443500	9,7875503719761800
6	12,0901461298633000	12,0901354649702000

Figura 1.12 Comparando H_N con $\ln(N)+0.57721$

Programa 16 *Imprimir Números armónicos* $H_N = \sum_{i=1}^{N} 1/i$ *vs* $\ln(N) + 0.57721$

```
Sub Main()
 Dim i
 For i=1 To 5
  ThisComponent.Sheets(0).getCellByPosition(9,i).Value = NHarmonico(10^i)
  ThisComponent.Sheets(0).getCellByPosition(10,i).Value = Log(10^i)+0.57721
 Next i
End Sub


Function NHarmonico(N)
Dim i As Long
Dim suma As Double
    suma=0 : i=1
    Do
      suma = suma+ 1/i
         i = i+1
    Loop Until  i > N
    NHarmonico = suma
End Function
```

Para leer o imprimir *texto* se usa el método `setString`, por ejemplo

Programa 17 *Leer e imprimir texto en una celda*

```
Sub Main
Dim txt
  txt= "Si x= "+Str(x)+",  f(x)= "+Str(f(x))
  ThisComponent.Sheets(0).getCellByPosition(1,3).setString(txt)
End Sub
```

	A	B
1		
2		
3	x	f(x)
4	4	Si x= 4, f(x)= 21

Figura 1.13 Imprimir texto

Leer cadenas de caracteres de una celda Para leer el contenido de una celda como una cadena de caracteres (String) se usa el método `getString`, por ejemplo

```
Dim txt  As String
 txt    = ThisComponent.Sheets(0).getCellRangeByName("A4").getString()
```

Color. De paso, para cambiar el color de fondo y el color de la fuente en la celda usamos `CharColor` y `CellBackColor`,

```
1 Dim oCell
2  oCell = ThisComponent.Sheets(0).getCellRangeByName("A4")
3  oCell.Value = 2.343403000102
4 'Color del texto: blanco
5  oCell.CharColor = RGB(255,255,255)
6 'Color de fondo: azul
7  oCell.CellBackColor = RGB(0,0,255)
```

1.4 Ejecutar una subrutina (o una función) desde un botón.

Para dar un ejemplo de cómo se inserta un botón en la hoja y de cómo se le asigna una subrutina (de tal manera que cuando el usuario hace clic en el botón, se ejecuta la subrutina), vamos a implementar una subrutina de nombre `"LeerImprimir()"`. Agregamos las instrucciones en la subrutina para leer un valor x_0 en la celda `"A4"` e imprimir $f(x_0)$ en la celda `"B4"`. La función f será $f(x) = x^3 + x + 1$.

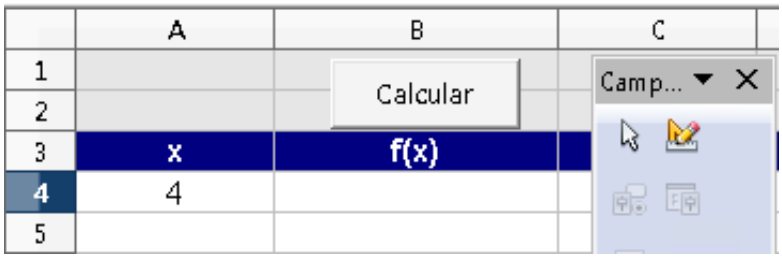


Figura 1.14 Leer e imprimir en una celda

El código de la subrutina podría ser

Programa 18 *Subrutina para leer e imprimir en una celda*

```
1 Sub LeerImprimir()
2    Dim Hoja,x
3     Hoja = thisComponent.Sheets(0)
4     x    = Hoja.getCellRangeByName("A4").Value
5     Hoja.getCellRangeByName("B4").Value = f(x)
6 End Sub
```

Después de digitar el código procedemos a la *compilación* para revisar la sintaxis. La sintaxis, en el ambiente computacional, se refiere al "conjunto de reglas que definen las secuencias correctas de

los elementos de un lenguaje de programación". Estas reglas son las que estamos aprendiendo en el camino. Observe que la sintaxis no tiene que ver con la lógica del cálculo. Lo que tratamos de hacer es que la sintaxis esté bien y el cálculo sea correcto.

Agregar el botón. Ahora sigue insertar el botón que "dispara" el cálculo. Debemos tener visible la barra "`Campos de control de formulario`". Si no esta visible, se habilita con

`Ver>Barra de herramientas>Campos de control de formulario.`

Para crear un botón, seleccionamos el botón en la barra `Campos de control` y arrastramos el ratón (presionando el botón izquierdo) en cualquier parte del cuaderno.

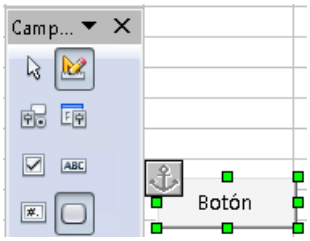


Figura 1.15 Selecciona el botón y arrastra el ratón...

En este momento, el botón está en "modo diseño". Para poner la etiqueta "Calcular" abrimos la ventana de propiedades del botón (con clic derecho sobre el botón, abrimos el menú "`Campo de control`")

Figura 1.16 Campos de control.

y editamos el campo "`Título`".

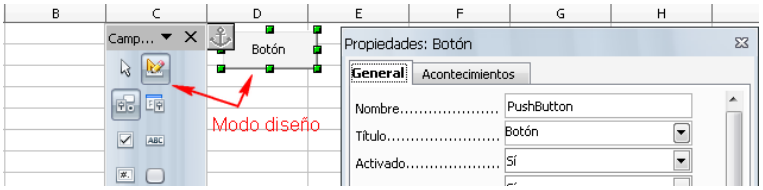


Figura 1.17 El botón está en modo diseño, podemos agregar propiedades.

Ahora, seleccionamos la cejilla "`Acontecimientos`" y asignamos una acción: "`Al iniciar`" (cuando el usuario hace clic en el botón). Elegimos la macro, es decir, la función o subrutina que se

va a ejecutar cuando el usuario hace clic en el botón. En nuestro caso, seleccionamos la subrutina `LeerImprimir()`.

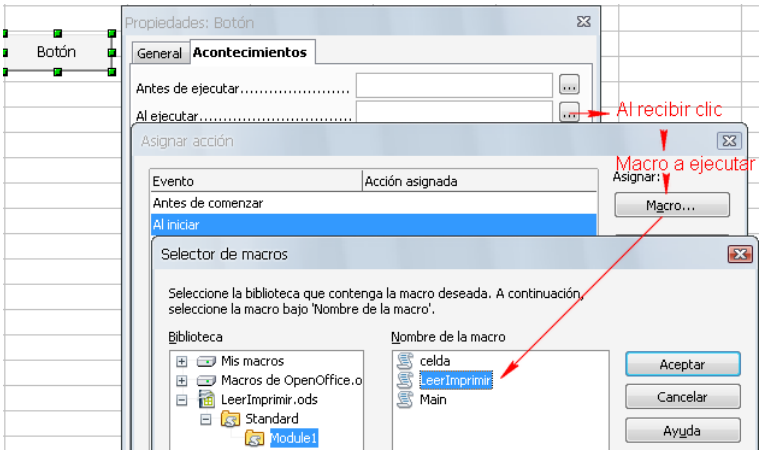


Figura 1.18 Asignar la subrutina que se ejecuta cuando se hace clic en el botón.

Salir de modo diseño. Luego de hacer clic en "`Aceptar`" y cerrar la ventana "`Propiedades: Botón`", debemos habilitar el botón (que hasta ahora ha estado en "modo diseño"): hacemos clic en  (en la barra "`Campos de control de formulario`"). Ahora ya podemos hacer clic en el botón para realizar el cálculo e imprimir. Una subrutina se puede ejecutar también con `ALT-F11`.

1.5 Crear, exportar, importar y cargar bibliotecas.

Recordemos de nuevo que las macros se guardan en módulos, estos a su vez se guardan y organizan en bibliotecas, las cuales están contenidas dentro de documentos. Hasta ahora, todos nuestras subrutinas y funciones las hemos editado en un módulo en la biblioteca `Standard`. La biblioteca `Standard` *no* se puede exportar (ni eliminar), así que si queremos tener disponibles nuestras funciones especiales para otros cuadernos, una opción es exportarlas en otra biblioteca. Así otro cuaderno las puede importar.

Para ver el proceso de exportar e importar, vamos a reunir varias funciones matemáticas y otras utilitarias e introducirlas en un módulo de una nueva biblioteca. Esta nueva biblioteca la llamaremos aquí `BblMatematica`. Luego *exportamos* esta biblioteca para que cualquier otro cuaderno la pueda *importar*. Los pasos son,

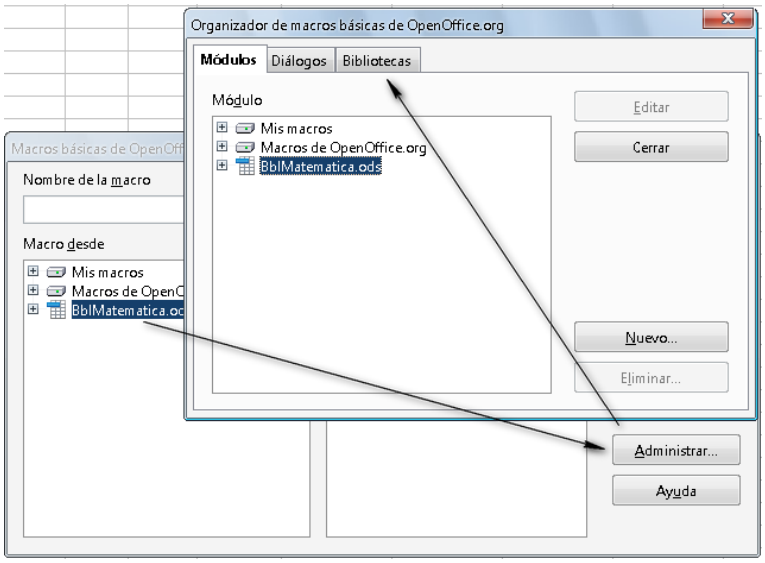
1. Crear una nueva biblioteca `BblMatematica`
2. Exportar la biblioteca
3. Importar y *cargar* la biblioteca (en otro cuaderno)

1.5.1 Crear una biblioteca.

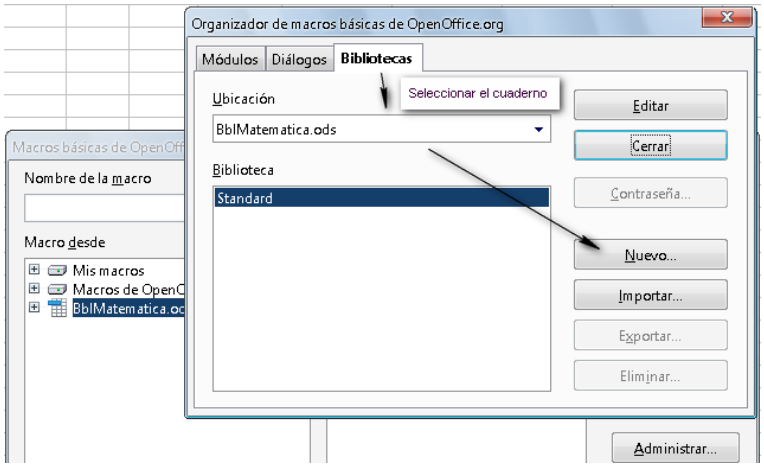
En lo que sigue, vamos a suponer que tenemos un cuaderno llamado `BblMatematica`. En este cuaderno tenemos las funciones y las macros especiales que queremos exportar como una bib-

lioteca. Vamos a suponer que todos estos programas están en la biblioteca `Standard`.

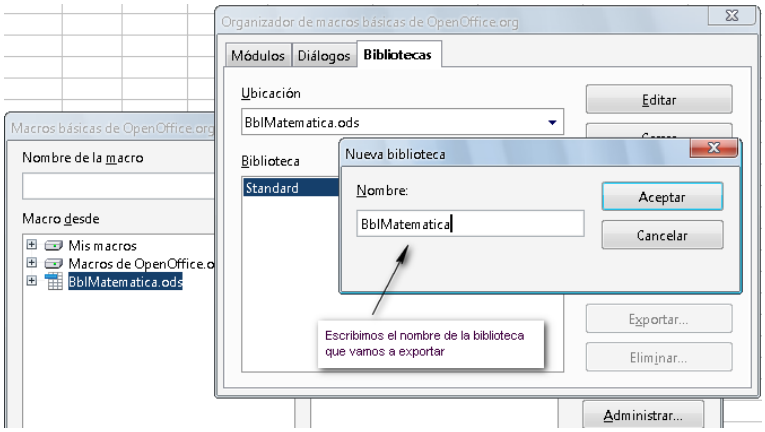
Para crear esta biblioteca (que contendrá el módulo o los módulos con las funciones especiales u otras macros) abrimos la ventana `Macros básicas` con `ALT-F11`. Hacemos clic en el botón `Administrar`.
En la nueva ventana `Organizador de macros...` elegimos la pestaña `Bibliotecas` (`Library`),



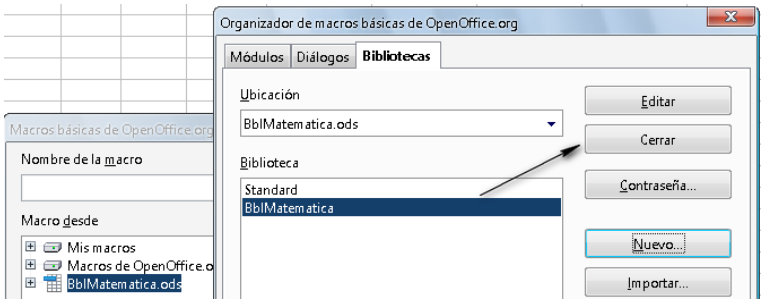
en la cejilla `Ubicación`(`Location`) elegimos el cuaderno actual,



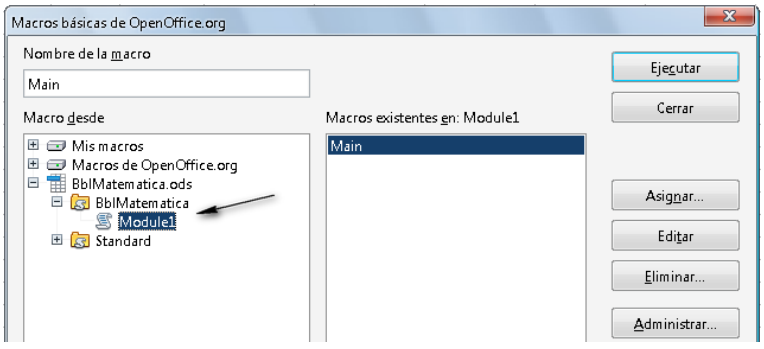
y presionamos el botón `Nuevo`. Luego, en la nueva ventana ponemos el nombre a nuestra biblioteca, en el campo correspondiente. Puede ser cualquier nombre, en nuestro caso le pondremos el mismo nombre del cuaderno de trabajo: `BblMatematica`.



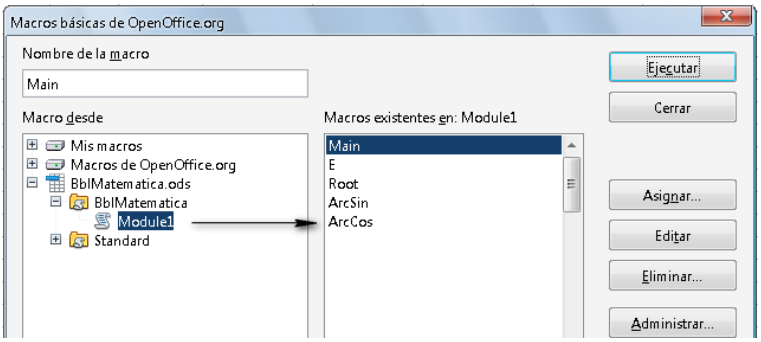
Aparece la biblioteca agregada. Podemos hacer clic en `Cerrar (Close)`.



La nueva biblioteca tiene ya un `module 1`.

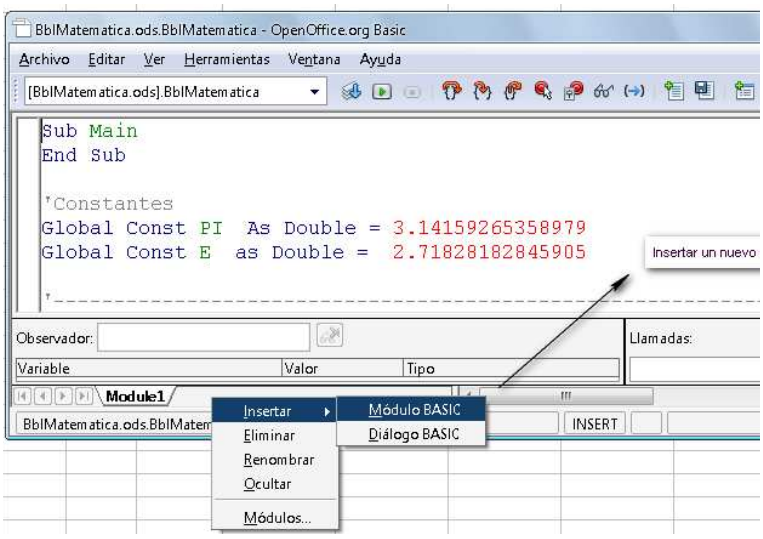


Presionamos el botón `Editar` para introducir el código de algunas funciones y subrutinas en este módulo y para editar cualquier otra función nueva.



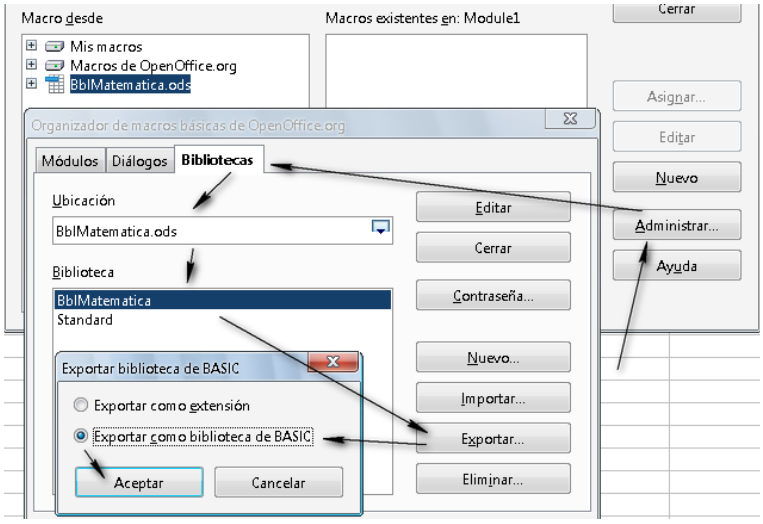
1.5.2 Agregar un nuevo módulo.

Para crear un nuevo módulo hacemos clic derecho en la barra en la que aparece el nombre del módulo y elegimos `Insertar -> Módulo Basic.`



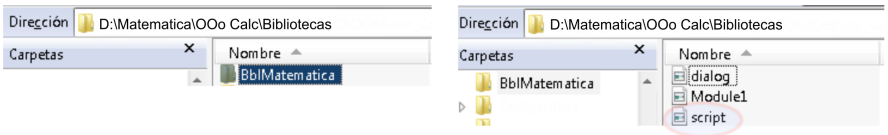
1.5.3 Exportar una biblioteca.

Para exportar una biblioteca (con nuestras macros personales u otro tipo de macro) primero abrimos el cuadro `Macros básicas...` con `ALT-F11`. Hacemos clic en el botón `Administrar`. En la nueva ventana `Organizador de macros...` elegimos la pestaña `Bibliotecas`, en la cejilla `Ubicación` elegimos el cuaderno que tiene la biblioteca y luego en la cejilla `Biblioteca` elegimos la biblioteca que vamos a exportar y presionamos el botón `Exportar`. En la nueva ventana elegimos `Exportar como biblioteca Basic` y hacemos clic en `Aceptar`. Inmediatamente se abre el explorador para ubicar la carpeta en la vamos a guardar la biblioteca.



Una vez elegida la ubicación, hacemos clic en `Aceptar` y luego hacemos clic en el botón `Cerrar`.

En nuestro caso, la biblioteca queda guardada como una carpeta con tres archivos `dialog` `Module 1` y `script`. Esta biblioteca está lista para *importar* en otro cuaderno.



1.5.4 Cargar una biblioteca

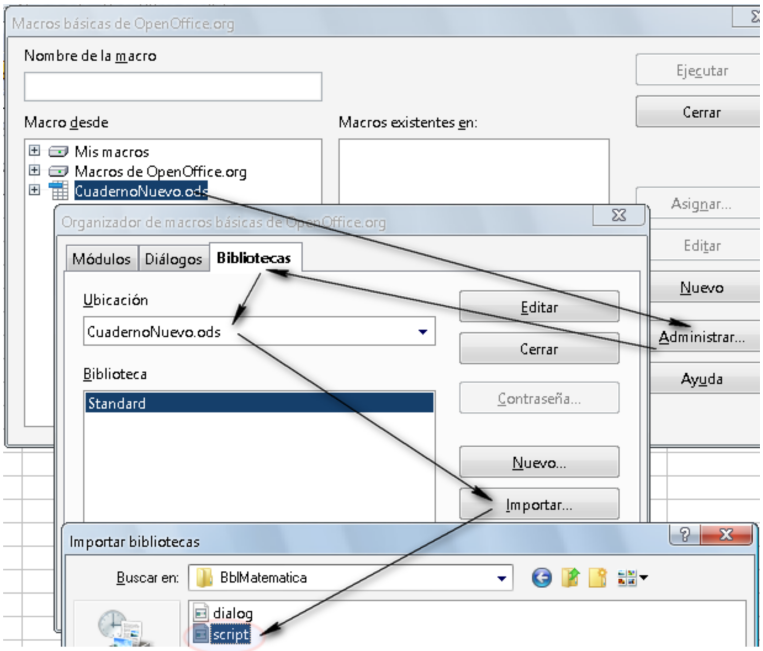
Importar una biblioteca no significa que este disponible. Cuando inicia OpenOffice.org solo se carga la biblioteca `Standard`. Este es un mecanismo para acelerar la inicialización. Para acceder a las bibliotecas importadas, se deben cargar (y el 'motor' Basic las compila). Por ejemplo, para cargar la biblioteca (ya importada) `BblMatematica` en la subrutina `Main`, se usa el código

```
'Cargar la biblioteca "BblMatematica" (si ya fue importada).
Sub  Main
  BasicLibraries.loadLibrary("BblMatematica" )
End Sub
```

Si no hacemos esto, usar alguna función de esta biblioteca provoca un error en tiempo de corrida: `Property or method not found (propiedad o método no encontrado).`

1.5.5 Importar una biblioteca.

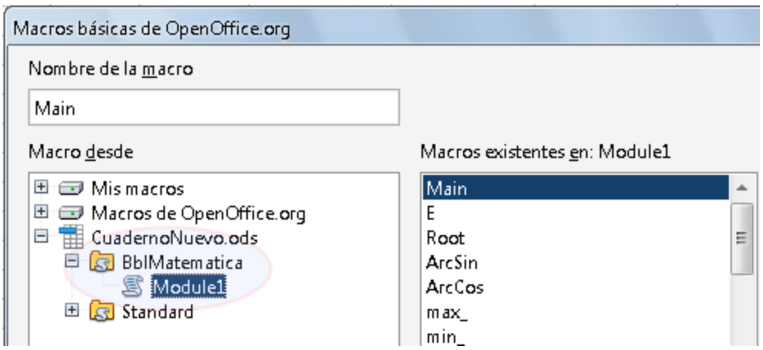
Para importar una biblioteca en un cuaderno `CuadernoNuevo`, primero abrimos el cuadro `Macros básicas...` con `ALT-F11`, presionamos el botón `Administrar`. En la nueva ventana `Organizador de macros...` elegimos la pestaña `Bibliotecas`, en `Ubicación` elegimos nuestro (nuevo) cuaderno y hacemos clic en el botón `Importar`.



Se abre el explorador y vamos a la carpeta en la que está la biblioteca, la abrimos y seleccionamos, en nuestro caso, el archivo `script.xlb`. Hacemos clic en `Abrir`.

En la nueva ventana, hacemos clic en `Aceptar` y luego cerramos la última ventana que queda.

Ahora, además de la biblioteca `Standard`, la nueva biblioteca está disponible,



Recordemos que para poder usar la biblioteca, ésta se debe cargar. En el código de la figura que sigue se usan la funciones `Cells` y `Root(x,n)` de la biblioteca `BblMatematica` (ver la sección **??**) en un módulo de la biblioteca `Standard` del nuevo cuaderno.

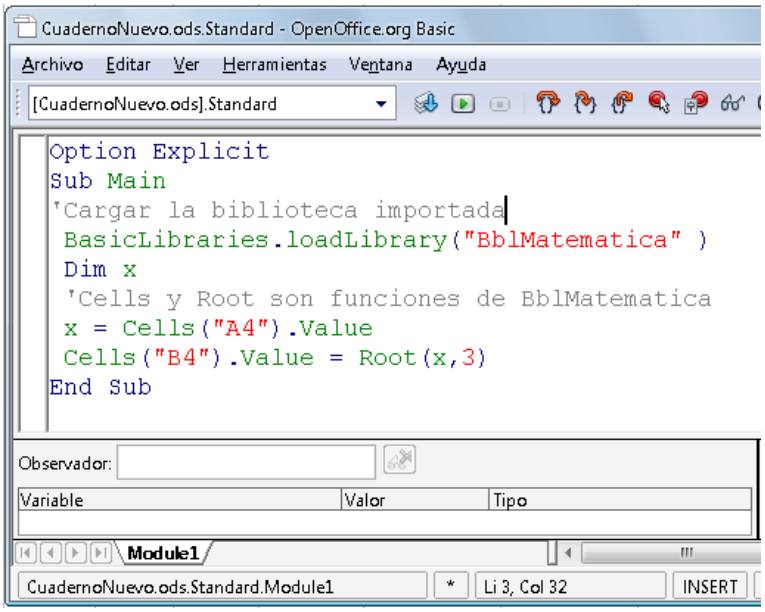


Figura 1.19 Cargar y usar nuestra la biblioteca `BblMatematica`.

1.6 Subrutinas y funciones

En esta sección estudiamos aspectos más específicos de las funciones y subrutinas con el propósito de iniciar la implementación de algunas funciones especiales para una biblioteca de funciones para métodos numéricos.

1.6.1 Pasar parámetros a una subrutina o una función.

Pasar parámetros por valor y por referencia. A una subrutina o a una función se le pueden pasar los argumentos o parámetros de dos maneras, una es por valor y otra por referencia. Cuando pasamos los argumentos *por valor*, en realidad lo que se hace es pasarle una "copia" del valor de la variable, de tal manera que las modificaciones que sufra la copia no afecta a la variable original. En cambio, cuando los argumentos se pasan *por referencia*, lo que estamos haciendo es pasarle la "ubicación" de la variable en la memoria y entonces la variable original si se puede modificar.

Por default, salvo que se indique lo contrario, *los argumentos se pasan por referencia.* Para pasarlos por valor hay que agregar `ByVal`,

```
 Function  Root( ByVal a As Double,  ByVal n As Double) As Double
...
End Function
```

Parámetros Opcionales. En las funciones y las subrutinas se pueden poner variables opcionales de cualquier tipo. Se debe comprobar si se 'paso' o no el argumento para que en su defecto,se asigne un valor por default a dicho argumento, para verificar si se paso o no un argumento se usa la fun-

ción de OOo Basic `IsMissing( Argumento)`.

División con resto. Si $b \neq 0$, la división de a por b se representa de dos maneras $a = qb + r$ o como $a/b = q + r/b$ donde q es el cociente y r el resto. En el programa que sigue se hace una división entera. El parámetro opcional es `n`, si el parámetro no esta presente o si $n = 0$, el resultado se imprime como $a = qb + r$. Opcionalmente se puede escoger imprimir $a/b = q + r/b$ si agregamos el parámetro opcional con valor `n =1`.

Programa 19 *División con resto. El tipo de salida está controlada por un parámetro opcional*

```
Function  Division_yResto(a, b,  Optional n  As Integer)  As String
Dim tipoSalida As Integer

If Not IsMissing(n) Then 'Si el parámetro n está presente
  tipoSalida  = n
Else 'Si n  no está, usamos el valor default del parámetro
  tipoSalida  = 0
End If

If tipoSalida =0 Then    'Salida  b*q + r
  Division_yResto = Str(a)+"="+Str(b)+"*"+Str(a)+"+"+Str(a Mod b)
End If
If tipoSalida =1 Then     'Salida  a/b = q + r/b
  Division_yResto = Str(a)+"/"+Str(b)+"="+Str(a/b)+"+"+Str(a Mod b)+"/"+Str(b)
End If
End Function
```

Al ejecutar `Division_yResto(5,4)` devuelve `"5=4*1+1"` mientras que `Division_yResto(5,4,1)` devuelve `"5/4=1+1/4"`.

1.6.2 Manejo de errores.

Los errores en tiempo de ejecución (como divisiones por cero, desbordamiento, etc.) se pueden controlar con el manejador de errores ('ErrorHandler') de OOo Basic. Cuando el manejador de errores detecta un error, ejecuta la salida de la función (o la subrutina) en la línea de código del error e inmediatamente ejecuta el código que hemos asignado al manejador de errores. Aquí vamos a usar, en principio, un manejador de errores genérico: Cuando detectamos un error, salimos de la función (o la subrutina) y enviamos una mensaje con la descripción del error. El código sería el siguiente,

```
Function  nombreFuncion()  'Manejador genérico de errores

On Error Goto  msgError
  '...
  '...el código de la función va aquí
```

```
    nombreFuncion =....
 Exit  Function
'Código para 'manejar' el error
 msgError:
     'Si Err=0 no hubo error,
     'si Err<>0 hubo algún error en tiempo de corrida.
     If Err <> 0 Then
     'Mensaje con # de error, descripción del error y # de línea
      MsgBox "Error #: "& Err & Chr(13) & Error & Chr(13) &"Línea " & Erl
     End If
      On Error Goto  0 'reinicializar las variables de error,
                                  'es decir Err, Error y Erl.
 End  Function
```

El manejador de errores puede tener cualquier nombre válido, aquí lo llamamos `msgError`, y tiene tres variables: `Err`=número de error, `Error`= descripción del error y `Erl`= número de línea del error en el código.

Raíces $n-$ésimas. A menudo la función a^x se implementa usando la fórmula $a^x = e^{x \ln a}$, $0^x = 1$ (con $0^0 = 1$!); pero no se acepta el caso $a < 0$. Hay que implementar una nueva función `Root(a,n)` para que maneje raíces como $(-8)^{1/3} = \sqrt[3]{-8}$.

Si n es impar y $a < 0$ usamos la fórmula $a^{1/n} = \text{sgn}(a) \cdot |a|^{1/n}$. En otro caso usamos $a^{1/n}$. La función es sencilla de implementar, pero la vamos a usar para mostrar un ejemplo de cómo usar el manejador de errores:

Si n es par, dejamos que se encargue la función default `a^(1/n)`. Si $a < 0$ o $n = 0$, se dispara un error.

Si n es impar, `a^(1/n)=Sgn(a) * Abs(a)^(1/n)` sin importar el signo de a.

Programa 20 `Root(x,n)` *con manejador de errores.*

```
1  Function  Root(ByVal a As Double, ByVal n As Double) As Double
2    Dim m  As Integer
3     m = Int(n)    'índice entero
4     On Error Goto  msgError  'Manejo de errores
5    If m Mod 2 = 0 Then       'Errores posibles: división por cero o
6               Root = a ^ (1 / m)    'subradical negativo.
7       Else
8                  Root = Sgn(a) * Abs(a) ^ (1 / m)
9    End If
10    Exit  Function
11    msgError:
12   If Err <> 0 Then
13   MsgBox "Error #: "& Err & Chr(13) & Error & Chr(13) & "En la línea " & Erl
14   End If
15    On Error Goto  0 'reinicializar las variables de error,
16                                  ' es decir Err, Error y Erl.
17 End Function
```

1.6.3 Usando la funciones de OOo Calc en OOo Basic.

La lista de funciones disponibles en OOo Calc se puede acceder con `Ctrl F2`. Muchas de estas funciones *no* se encuentran en OOo Basic pero aún así se pueden invocar desde una macro. Usar estas funciones pueden ser muy conveniente porque usualmente están muy bien implementadas.

El nombre a nivel de programación. OOo Calc usa un nombre en la hoja para cada función pero usa otro nombre a nivel de programación. Por ejemplo, en mi versión en español OOo Calc entiende la fórmula `=REDONDEAR(4,4544;2)` pero no entiende la fórmula `=ROUND(4,4544;2)`. Sin embargo, *a nivel de programación* debemos usar el nombre de programación ("programmatic name") e invocar la función `ROUND`.

Para acceder a *todas* las funciones de OOo Calc necesitamos crear un *servicio*. En este caso, el servicio `FunctionAccess`. Este servicio nos provee de propiedades y métodos. Aquí nos interesa el método `callFunction`. El siguiente código general hace una llamada a una función `"NombreFuncion"`. El nombre de la función *siempre es una tira de texto* y el *segundo argumento siempre es un array*

```
Dim oFunction,x
oFunction    = createUnoService("com.sun.star.sheet.FunctionAccess")
          x = oFunction.callFunction("NombreFuncion", Args())
```

En el siguiente ejemplo se muestra la llamada a varias funciones.

Programa 21 *Usando funciones de OOo Calc con el método* `callFunction` *del servicio* `FunctionAccess`.

```
1  Sub  Main
2  Dim oFunction, oCell 'Variant
3  Dim args(1 To 3)    'Array
4  'Inicializamos el arreglo
5   args(1)=4 : args(2)=2.1 : args(3)=0.9
6  'oCell es la celda A3
7   oCell = ThisComponent.Sheets(0).getCellRangeByName("A3")
8  'Crear el servicio
9   oFunction    = createUnoService("com.sun.star.sheet.FunctionAccess")
10 'Usar el método callFunction del servicio para llamar a la función "SUM"
11  oCell.Value = oFunction.callFunction("SUM",Args())
12
13 'También es válido usar
14  oCell.Value  = oFunction.callFunction("SUM",Array(4,2.1,0.9)) 'Imprime 27
15
16 'Aunque la función sea de un solo argumento, se debe usar un array
17  oCell.Value  = oFunction.callFunction("Cos",Array(3.1416))
18
```

```
'Aplicación a Rangos
Dim oRange
'oRange es el rango B4:B6
 oRange      = ThisComponent.sheets(0).getcellrangebyname("B4:B6")
'Aplicamos la prueba Z al rango con  parámetros μ = 2.5 y σ = 1.
 oCell.Value = oFunction.callFunction("ZTEST", Array(oRange, 2.5, 1.0))
End Sub
```

Nuevas funciones. Podemos incorporar las funciones de OOo Calc como nuevas funciones en nuestra biblioteca `BblMatematica`. Por ejemplo, la función que calcula el valor mínimo de un rango podría ser,

Programa 22 *Función* `MIN` *de OOo Calc.*

```
Function CMIN(oRange)
 Dim oFunction
  oFunction = createUnoService("com.sun.star.sheet.FunctionAccess")
  CMIN = oFunction.callFunction("MIN",oRange)
End Function
```

La podríamos utilizar así,

```
  v = Array(1,2,3,4)
  x = CMIN(v) 'Retorna 1
```

Algunas de las funciones disponibles son,

Tabla 1.1 Funciones matemáticas disponibles en OOo Calc

COS	SIN	TAN	COT	ACOS	ACOT	ASIN	ATAN
RADIANS	PICOSH	SINH	TANH	COTH	ACOSH	ACOTH	ASINH
ROUND	ROUNDDOWN	ROUNDUP	CEILING	FLOOR	EVEN	ODD	MROUND
MOD	EXP	POWER	LOG	LN	LOG10	ABS	COMBIN
CONVERT_ADD	COUNTBLANK	COUNTIF	DELTA	ERF	ERFC	FACT	FACTDOUBLE
GESTEP	ISEVEN	ISODD	LCM	LCM_ADD	MULTINOMIAL	PRODUCT	RAND
SIGN	SQRT	SQRTPI	SUBTOTAL	SUM	SUMIF	SUMSQ	SERIESSUM
ATAN2	DEGREES	ATANH	TRUNC	INT	QUOTIENT	COMBINA	CONVERT
BESSELI	BESSELJ	BESSELK	GCD	GCD_ADD	RANDBETWEEN		

En la tabla que sigue se muestran algunas funciones de la hoja para trabajar con matrices. Este tema se desarrolla con detalle en la sección 1.6.8.

Tabla 1.2 Funciones para matrices (arrays) en OOo Calc

FREQUENCY	GROWTH	LINEST	LOGEST	MDETERM	MINVERSE	MMULT	MUNIT
SUMPRODUCT	SUMX2MY2	SUMX2PY2	SUMXMY2	TRANSPOSE	TREND		

La lista completa de funciones se puede encontrar en,

http://wiki.services.openoffice.org/wiki/Documentation/How_Tos/Calc:_Functions_listed_by_category

1.6.4 Un evaluador de funciones matemáticas ("Math Parser").

La idea aquí es digitar una expresión tal como `x^3+2*x-3*log(x)+1` en una celda y leer y evaluar esta función en algún valor de x. Para hacer esto necesitamos una *evaluador* de funciones matemáticas ("Math parser"). Sin embargo hacer un evaluador requiere el uso de estructuras de datos más complejas de las que queremos usar en este libro. Hay un par de soluciones sencillas: Usar el evaluador de fórmulas de una celda o usar un evaluador de funciones de otro lenguaje, por ejemplo el evaluador de `javascript`.

Un evaluador `Eval(f,x)`. Una manera de evaluar una función es con el método `formula` de las celdas. La estrategia es sencilla: La fórmula de la función es una tira de texto (string), leemos esta expresión y sustituimos la variable por el valor a evaluar y evaluamos la fórmula que nos queda con el método `formula`. Esta manera de proceder requiere una celda que almacene el resultado de la evaluación. Nuestra función `Eval` nos va a permitir hacer cálculos como,

```
f = "2,455*Root(x;3)+x^2"   '"Root" requiere ";" pues evaluamos en la hoja
x = Eval(f,x0)              'Devuelve el valor f(x0). "x" es variable default
f = "t^2+1"
x = Eval(f,t0,"t")          'Devuelve el valor f(t0). "t" es la variable
```

En el siguiente código se muestra como implementar una tal función `Eval(f,x)`. La implementación requiere dos funciones adicionales para manejar `Strings`.

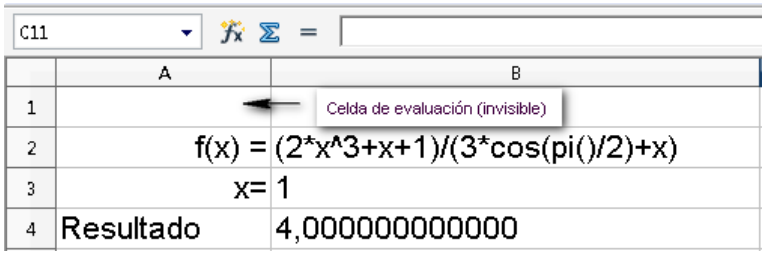


Figura 1.20 Usando la función `Eval(f,x)`.

Programa 23 *Evaluador de funciones*

```
'Evalúa funciones usando OOo Calc, por tanto las fórmulas deben ajustarse
'a la sintaxis de la hoja y a la configuración regional del idioma.
'Evalúa fórmulas funcionales como:"2*x^3-2*cos(x)+log(x+1)", "t^2+3*Exp(t)", etc.
'Puede usar funciones de las bibliotecas: 2,4332*x -3*Root(x;3).
'Observar el uso de "," debido a la configuración regional y el ";".
'Requiere una hoja y una celda auxiliar (i,j) para evaluar (por default es (0,0)).
'La hoja default es la hoja 0
'Uso: f= oCell.String o f="2*x^2+x+1"
' Eval(f,valor) o Eval(f, valor, "variable")
```

```
1 Function  Eval(f As String, valor As Double, Optional variable As String,_
2                           Optional j, Optional i, Optional numhoja)
3     Dim valorVar As String
```

```
    Dim nombreVar As String
    Dim formula As String
    Dim CellAuxiliar As Variant
    Dim co, fi,nh, cl
    'Posición de la La celda auxiliar (para evaluar)
    co=0 : fi =0 : nh =0 : nombreVar="x" 'default
    If Not IsMissing(i) And Not IsMissing(j) Then
       co=j : fi =i
    End If
    If Not IsMissing(numhoja) Then
       hn=numhoja
    End If

    If Not IsMissing(variable) Then
           nombreVar = variable
    End If
    CellAuxiliar=ThisComponent.Sheets(nh).getCellByPosition(co,fi)
    'Color de texto = color de fondo
     cl = CellAuxiliar.CellBackColor
     If cl = -1 Then
        CellAuxiliar.CharColor = RGB(255,255,255)
     Else
        CellAuxiliar.CharColor=cl
     End If
    'cambiamos a String para la sustitución
    valorVar= Str(valor)
    formula = f
    'Sustituye  valor  por variable
    Call strSustituir(formula, nombreVar, valorVar)
    ' Las fórmulas inician con "="
    formula="="+formula
    'Evaluar la fórmula
    CellAuxiliar.SetFormula(formula)
 Eval= CellAuxiliar.Value
End Function

'-------------------------------------------------------------------------
'Subrutina para sustituir
Sub strSustituir( ByRef str1 As String, str2 As String, str3 As String)
Dim i As Long, s1 As String, s2 As String, L1 As Long, L2 As Long
 str1 = " " & str1 & " "
 L2 = Len(str2)
 i = 0
 Do
    i = InStr(i + 1, str1, str2)
    InStr(i + 1, str1, str2, 1)
   If i = 0 Then  Exit Do
    s1 = Mid(str1, i - 1, 1)
   If Not  IsLetter(s1) Then
```

```
        s2 = Mid(str1, i + 1, 1)
        If Not  IsLetter(s2) Then 'Sustituir,
            s1 = Left(str1, i - 1)
            s2 = Right(str1, Len(str1) - i - L2 + 1)
            str1 = s1 & str3 & s2
        End If
    End If
Loop
str1= Trim(str1)
End Sub

Function  IsLetter(ByVal char As String) As Boolean
Dim code As Long
  code = Asc(char)
 IsLetter=(65<=codeAnd code<=90) Or(97<=code And code<=122)Or char="_"
End Function
```

Programa 24 *Usando el evaluador en Main*

```
Sub  Main
 'Usando la función Eval(f,x) para evaluar f en 'valor'
  'La fórmula debe respetar la sintaxis permitida en la HOJA
Dim f, valor
'Leemos la fórmula en la celda "B2"
 f = ThisComponent.Sheets(0).getCellByPosition(1,1).getString()
'Leemos el valor de "x"
 valor =ThisComponent.Sheets(0).getCellByPosition(1,2).Value
 'Evaluamos con Eval(f,valor). "x" es la variable default
 ThisComponent.Sheets(0).getCellByPosition(1,3).Value= Eval(f,valor)

 'También puede introducir la fórmulas directamente,
 f="2,455*Root(t;3)+t^2" '"Root" es de nuestra biblioteca BblMatematica
 MsgBox  Eval(f,valor,"t")
End Sub
```

Evaluar funciones de dos variables. La misma idea del evaluador para funciones de una variable se puede usar en dos variables: Sustituimos cada variable una a la vez y luego evaluamos. La función la llamamos `Eval2`, las variables default son x e y.

Nuestra función `Eval2` nos va a permitir hacer cálculos como,

```
f = "x^2+y^2+1"
MsgBox  Eval(f,x0,y0) '"x" e "y" son variables default, en ese orden.
f = "t^2+y^2+1"
MsgBox  Eval(f,t0,y0,"t","y")
```

Programa 25 *Eval2 para evaluar funciones de dos variables*

```
' Eval2 = evalúa una función de dos variables.
' Eval(fxy,x0,y0) -> f(x0,y0)
' Eval2(ftw,t0,w0,"t","w") -> ftw(t0,w0)
```

```
Function Eval2(f As String, valor1 As Double, valor2 As Double,_
            Optional var1 As String,  Optional var2 As String,_
             Optional j,  Optional i, Optional numhoja)
    Dim valorVar1 As String
    Dim valorVar2 As String
    Dim formula As String
    Dim nombreVar1 As String
    Dim nombreVar2 As String
    Dim CellAuxiliar As Variant
    Dim co, fi,nh, cl
    'Posición de la La celda auxiliar (para evaluar)
    co=0 : fi =0 : nh =0 : nombreVar1="x" : nombreVar2="y" 'default
    If Not IsMissing(i) And Not IsMissing(j) Then
       co=j : fi =i
   End If
    If Not IsMissing(numhoja) Then
       hn=numhoja
    End If

    If Not IsMissing(var1) Then
       nombreVar1=var1
    End If
    If Not IsMissing(var2) Then
       nombreVar2=var2
    End If
    CellAuxiliar=ThisComponent.Sheets(nh).getCellByPosition(co,fi)
    'Color de texto = color de fondo
     cl = CellAuxiliar.CellBackColor
     If cl = -1 Then
        CellAuxiliar.CharColor = RGB(255,255,255)
     Else
        CellAuxiliar.CharColor=cl
     End If
    'cambiamos a String para la sustitución
    valorVar1 = Str(valor1) :  valorVar2 = Str(valor2)
    formula = f
    'Sustituye  valorVari por nombreVari
    Call strSustituir(formula, nombreVar1, valorVar1)
    Call strSustituir(formula, nombreVar2, valorVar2)
    ' Las fórmulas inician con "="
    formula="="+formula
    'Evaluar la fórmula
    CellAuxiliar.SetFormula(formula)
 Eval2= CellAuxiliar.Value
End Function
```

1.6.5 Vectores, matrices y rangos.

Los "arreglos" (`array`) son de uso frecuente no solo en cálculos matriciales, muchos cálculos requieren este formato, por ejemplo en interpolación donde se debe seleccionar un rango de datos.

Arrays. Un `array` ("arreglo") es una estructura de datos que se usa para indexar datos. Por ejemplo columnas, filas o tablas de números. Es obligatorio declarar los arreglos antes de usarlos. Para declarar un arreglo se usa `Dim` y se usa paréntesis para definir y accesar los elementos del arreglo. En la tabla (1.6.5) se muestra la manera de definir un arreglo y una breve descripción.

Declaración	Elementos	Descripción
`Dim v(n) As Integer`	$n+1$	Enteros $v(0), v(1), ..., v(n)$
`Dim v(1 To n) As Integer`	n	Enteros $v(1), v(2), ..., v(n)$
`Dim w(-n To n) As Double`	$2n+1$	$w(-n), w(-n+1), ..., w(0), w(1), ..., w(n)$
`Dim M(n, m)`	$(n+1)\times(m+1)$	$M(i,j)=a_{ij}$, $\begin{pmatrix} a_{00} & a_{01} & \dots & a_{0,m} \\ a_{10} & a_{11} & \dots & a_{1,m} \\ & & \vdots & \\ a_{n0} & a_{n1} & \dots & a_{n,m} \end{pmatrix}$
`Dim M(1 To n,1 To m)`	$n\times m$	$M(i,j)=a_{ij}$, $\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1,m} \\ a_{21} & a_{22} & \dots & a_{2,m} \\ & & \vdots & \\ a_{n1} & a_{n1} & \dots & a_{n,m} \end{pmatrix}$

Tabla 1.3

Un arreglo se puede declarar de manera directa, por ejemplo

```
Dim v()
v=Array(cos(1),2.3,4.2)    'v(0)=cos(1), v(1)=2.3 y v(2)=4.2
```

Aunque podemos crear una tabla como un arreglo "de arreglos", es mejor hacerlo de manera directa, por ejemplo,

```
Dim v(0 To 1, 0 To 2)
 v(0, 0) = 1 : v(0, 1) = 2 : v(0, 2) = 3
 v(1, 0) = 3 : v(1, 1) = 5 : v(1, 2) = 2.3
```

ReDim. Es frecuente no conocer las dimensiones del arreglo antes de seleccionar un rango o leer una o más variables. Podemos declarar el arreglo sin dimensiones y redimensionarlo en cuando tengamos el dato, por ejemplo,

```
Dim v()
...
ReDim v(1 To n)
```

LBound, UBound. Para recorrer un vector `w` de tamaño desconocido podemos usar `LBound(w)` y `UBound(w)` para tener acceso al primer y último subíndice, respectivamente, del arreglo.

También podemos leer las dimensiones de una arreglo 2-dimensional (una matriz) A. Si se declara `ReDim A(f1 to fn, c1 to cn)`, entonces `LBound(A,1)` devuelve `f1`, `UBound(mA,1)` devuelve `fn`, `LBound(A,2)` devuelve `c1` y `UBound(mA,2)` devuelve `cn`. Por ejemplo,

Programa 26 *Usando arreglos*

```
Sub  Main
Dim i
Dim w()
ReDim w(1 To n)
...
'La primera entrada de w es w(LBound(w))  y la última es w(UBound(w))
For i=LBound(w) To  UBound(w)
    w(i)=0
Next i

Dim A()
ReDim A(10,15) '11 filas, 16 columnas

MsgBox LBound(mA,1)  '-> 0  (fila 0)
MsgBox UBound(mA,1)  '-> 10 (fila 10)
MsgBox LBound(mA,2)  '-> 0  (columna 0)
MsgBox UBound(mA,2)  '-> 15 (columna 15)
End Sub
```

Preserve. Podemos aumentar la dimensión del arreglo conservado los datos ya alamcenados en el arreglo. Esto se hace con `ReDim Preserve A`. Por ejemplo,

Programa 27 *Usando* `Preserve`

```
Sub  Main
Dim i
Dim A()
ReDim A(1 To 2, 1 To 2)

   A(1,1)=11 : A(1,2)=12 : A(2,1)=21 : A(2,2)=22 'A = 11  12
                                                 '    21  22
ReDim Preserve A(2,2)
                                                 '    _   _   _
                                                 'A = _   11  12
                                                 '    _   21  22
End Sub
```

Array de Array's. A veces tenemos una matriz como un un array con array's. Lo bueno de esto es que las filas se pueden accesar como vectores, por ejemplo

Programa 28 *Arreglo de arreglos*

```
Sub  Main
Dim T()
Dim fila
  T = Array(Array(3,5,7),Array(4,5,6))
  fila = T(0)     '3 5 7
MsgBox fila(2) '-> 7
  fila = T(1)     '4 5 6
MsgBox fila(0) '-> 4
End Sub
```

También podemos escribir,

Programa 29 *Arreglo de arreglos. Otra manera.*

```
Sub  Main
Dim T()
Dim fila
  T(0)=Array(3,5,7)
  T(1)=Array(4,5,6))
  fila = T(0)     '3 5 7
MsgBox fila(2) '-> 7
  fila = T(1)     '4 5 6
MsgBox fila(0) '-> 4
End Sub
```

1.6.6 Funciones que reciben o devuelven arreglos.

Una función puede recibir arreglos como argumentos y entregar números o arreglos. El arreglo que recibe se declara Variant.

En el código que sigue se implementa la norma de un vector v y el producto escalar. Si $v = (v_1, v_2, ..., v_n)$ su norma es $||v|| = \sqrt{v_1^2 + v_2^2 + ... + v_n^2}$ y $k \cdot v = (k \cdot v_1, k \cdot v_2, ..., k \cdot v_n)$.

Programa 30 *Norma y producto escalar*

```
Function Norma(V() As Variant)
  Dim suma,i
   suma = 0
  For  i=LBound(V) To UBound(V)
      suma =suma+V(i)^2
```

```
6     Next i
7      Norma = Sqr(suma)
8   End Function
9
10  Function Escalar(V(), k )
11    Dim W()
12    Dim i,n
13
14    n = UBound(V)
15   ReDim W(n)
16   For  i=LBound(V) To UBound(V)
17      'V no se modifica
18       W(i) = k*V(i)
19   Next i
20   Escalar = W()
21  End Function
```

Un ejemplo de su uso es,

Programa 31 *Cálculos con la norma y el producto escalar*

```
1   Sub  Main
2     Dim x
3     Dim B()
4     Dim S()
5
6     B = Array(0,3,4)
7     x = Norma(B)          'Retorna 5
8     S = Escalar(B, 2)     'Retorna (0,6,8)
9   End Sub
```

1.6.7 Rangos.

Una operación frecuente es seleccionar con el ratón un rango, contar las filas y aplicar alguna fórmula. La selección del usuario se puede almacenar en una variable `rango` y contar las filas de la selección, de la siguiente manera,

```
Dim rango
Dim n
 rango = ThisComponent.getCurrentSelection()
 n     = rango.Rows.getCount()
```

Como antes, se puede accesar el rango por columnas y filas,

```
rango.getCellByPosition(0, i).Value
```

nos devuelve el valor en la primera columna del rango (columna 0) y en la fila i (contando desde cero). El valor en columna `j` del rango y en la fila `i` (ambos, contando desde cero), se obtiene con

```
rango = ThisComponent.getCurrentSelection()
rango.getCellByPosition(j, i).Value
```

Estadística básica. Supongamos que tenemos una muestra de N datos $x_0, x_1, ..., x_{N-1}$ de una población. El promedio muestral $\overline{x} = (x_0 + x_1 + \cdots + x_{N-1})/N$ es una estimación del promedio poblacional μ (el valor esperado del promedio de todas las muestras). La varianza poblacional σ^2 se estima con la varianza muestral $s^2 = \sum_{i=0}^{N-1}(x_i - \overline{x})/N$. La desviación estándar de la población se estima con s. Otros valores de interés son el valor máximo y el valor mínimo y la mediana.

En el cuaderno que se muestra en la figura (1.21) hay una columna con los datos x_i. El botón `Estadísticas` ejecuta la subrutina `Estadisticas()`. En esta subrutina se lee el rango seleccionado por el usuario y se calcula el máximo, la media muestral, la varianza muestral y la desviación estándar muestral.

B	C	D	E	F
	Estadísticas			
x_i	Máximo	Media	Varianza	Desviacion Std
76,56	93,75	79,68	154,88	12,44515
78,12				
87,5				
85,93				
87,5				
73,43				
85,93				
93,75				
87,5				
62,5				
60,93				
51,56				
68,75				
84,37				
85,62				
95,31				
62,5				

Figura 1.21 Estadística básica

Programa 32 *Estadística básica*

```
1  Sub Estadisticas()
2    Dim n, i
3    Dim Rango
4    Dim X()
5    Rango = ThisComponent.getCurrentSelection()
6        n = Rango.Rows.getCount()
7    ReDim X(0 To n-1)
8    For i=0 To n-1
9      X(i)=Rango.getCellByPosition(0,i).Value
```

```
  Next i
  ThisComponent.Sheets(0).getCellRangeByName("C3").Value = MaxVect(X)
  ThisComponent.Sheets(0).getCellRangeByName("D3").Value = Media(X)
  ThisComponent.Sheets(0).getCellRangeByName("E3").Value = Var(X)
  ThisComponent.Sheets(0).getCellRangeByName("F3").Value = DevStd(X)
End Sub
'-------------------------------------------------------------------------
Function MaxVect(X())
 Dim mx, im
  mx = X(0)   : im = UBound(X)

   For i = 0 To im
    If X(i) > mx Then
        mx = X(i)
    End If
  Next i
 MaxVect = mx
End Function

Function Media(X())
 Dim suma, i0,im
   suma = 0 : im = UBound(X) '# datos = im+1

  For i = 0 To im
         suma = suma + X(i)
  Next i
 Media  = suma/(im+1)
End Function

Function Var(X())
 Dim promedio, suma, im
  promedio = Media(X): suma = 0   : im = UBound(X) '# datos = im+1

  For i = 0 To im
         suma = suma +(X(i)-promedio)^2
  Next i
 Var   = suma/im
End Function

Function DevStd(X())
         DevStd = Sqr(Var(X))
End Function
```

En el programa anterior se pasó los valores del rango a un vector. Aunque no es necesario, a veces es sumamente cómodo hacer esto porque las fórmulas y los algoritmos son más fáciles de seguir y depurar.

Por ejemplo, En el siguiente código pasamos los valores de la primera columna (columna 0) a un vector `X()` y pasamos los valores de la segunda columna (columna 1, suponiendo que el rango tiene dos o más columnas) a un vector `Y()`,

Programa 33 *Pasar las columnas de un rango a vectores*

```
Dim rango
Dim n, i
Dim X(), Y()
 rango = ThisComponent.getCurrentSelection()
 n     = rango.Rows.getCount()

ReDim X(1 To n)
ReDim Y(1 To n)

For i=1 To n
   X(i)=rango.getCellByPosition(0, i-1).Value
   Y(i)=rango.getCellByPosition(1, i-1).Value
Next i
```

En el código que sigue, se pasa la totalidad de una rango a una matriz `A`.

Programa 34 *Pasar una rango a una matriz*

```
Dim rango
Dim n,m, i,j
Dim Y()
Dim A()
 rango = ThisComponent.getCurrentSelection()
 n     = rango.Rows.getCount()
 m     = rango.Columns.getCount()
ReDim A(1 To n, 1 To m) 'Filas x columnas!, A=(a_ij), i=1..n,j=1..n
For i=1 To n
   For j=1 To m
            A(i,j)=rango.getCellByPosition(j-1, i-1).Value
   Next j
Next i
```

Con solo que haya una celda seleccionada, ya el rango tendrá al menos una fila y una columna. En la práctica, muchas de las operaciones de rango requieren que la selección tenga dos o más filas (o columnas). Si este es el caso, se puede agregar una instrucción que envíe un mensaje si no se ha seleccionado la cantidad mínima de datos y además salir de la función o la subrutina (pues no habría nada que hacer!). Para este propósito podemos usar el código

Programa 35 *Mensaje para advertir sobre la cantidad de datos seleccionados.*

```
Sub  SeleccionarDatos()
   Dim rango
   Dim n
    rango = ThisComponent.getCurrentSelection()
    n     = rango.Rows.getCount()

If n<=1 Then
     MsgBox  "Por favor, seleccione los datos."
     Exit Sub
End If
    ...
End Sub
```

Hacer una copia de una Matriz. Si tenemos dos matrices `M1()` y `M2()`, para hacer una copia en `B` de `A`, lo mejor es usar un ciclo `For` y hacer la copia componente a componente:

Programa 36 *Subrutina para hacer una copia B de una matriz A*

```
Sub MCopiar(B(),A())
  Dim f1, fn, c1, cm, i, j
  f1  = LBound(A,1)  ' primera fila
  fn =  UBound(A,1)  ' última fila
  c1 =  LBound(A,2)  ' primera columna
  cm =  UBound(A,2)  ' última columna
  ReDim B(f1 To fn, c1 To cm )

  For i=f1 To fn
    For  j=c1 To cm
        B(i,j)=A(i,j)
   Next j
 Next i
End Sub
```

La razón de hacer esto así es porque la asignación `B()= A()` hace que estas dos matrices queden vinculadas, es decir, los cambios en `A` se reflejan en `B` y viceversa. Para algunos cálculos esto no es nada conveniente.

Programa 37 *Probando la subrutina* `MCopiar`

```
Sub  Main
  Dim A()
  Dim B()
  Dim C()
  Dim i,j
  ReDim A(1 To 5, 1 To 5)
  For i=1  To 5
     For  j=1  To 5
```

```
        A(i,j) = 0
      Next j
   Next i
  MCopiar(B,A)   'B es una copia de A
  C = A          'C está vinculada con A
  A(1,1)= 1
  MsgBox A(1,1) '-> 1
  MsgBox C(1,1) '-> 1, C cambió igual que A
  MsgBox B(1,1) '-> 0, B no cambia con A
End Sub
```

1.6.8 Funciones para operaciones con matrices.

Como ya vimos en la sección (1.6.3), podemos implementar funciones que usen las funciones para el manejo de rangos de la hoja OOo Calc. Puede ser bueno usar estas funciones porque generalmente son muy bien implementadas. Otras funciones las tendremos que implementar según los requerimientos de los algoritmos que estudiemos en el futuro.

Función Det(A). En el código que sigue, implementamos una función `CDET(A)` para calcular el determinante de la matriz $A_{n\times n}$. Usamos la función $MDETERM$ de la hoja. Solo hay que recordar que la matriz A se debe recibir como un array, es decir, como `Array(A)`.

Programa 38 *Función determinante usando la función* `MDETERM` *de OOo Calc*

```
Function CMATRIXDET(oRange)
 Dim oFunction
   oFunction = createUnoService("com.sun.star.sheet.FunctionAccess")
   CMATRIXDET = oFunction.callFunction("MDETERM",Array(oRange))
End Function
```

Un ejemplo de su uso se muestra en el código,

Programa 39 *Usando la función* `CMATRIXDET`

```
Sub  Main
  Dim A() As Double
  Dim x
  ReDim A(1 To 2,1 To 2)
    A(1,1)= 2 : A(1,2) = 2 : A(2,1)= 3 : A(2,2) = -3
    x = CMATRIXDET(A)        'Retorna -12

  'oRange es un rango en la hoja 0
  Dim oRange
      oRange = ThisComponent.sheets(0).getcellrangebyname("C6:D7")
           x = CMATRIXDET(oRange)
End Sub
```

Funciones que devuelven arreglos anidados. Hay que hacer algunos arreglos si la función de OOo Calc devuelve una array anidado (Ver [10]), como es el caso de la multiplicación de matrices y el cálculo de la inversa. En este caso lo que devuelve la función de OOo Calc es un array del tipo

```
Array(filas(columnas(0 to n-1))),
```

por lo que debemos *recuperar los datos por filas*. Por ejemplo,

Programa 40 *Inversa de una matriz, primera versión.*

```
Function  CMATRIXINVERSE(oRange)
 Dim Fila, Filas, mM, nf, i, j
 Dim oFunction
  oFunction = createUnoService("com.sun.star.sheet.FunctionAccess")
  Filas = oFunction.callFunction("MINVERSE",Array(oRange))
  '"Filas" es un array anidado
  'nf = número de filas
 nf = UBound(Filas)
 ReDim mM(0 To nf, 0 To nf)
 For i = 0 To nf
     Fila = Filas(i)
     For j = 0 To nf
        mM(j,i)= Fila(j)
     Next j
  Next i
 CMATRIXINVERSE= mM
End Function
```

En vez de poner todo este código, podemos proceder como en (Ver [10]) para editar de una manera más limpia las funciones que invocan funciones de OOoCalc que devuelven un array anidado (y también para las otras). Se necesitan dos funciones, la primera es `calc_Func` para llamar la función de OOo Calc y la otra es `DataArray2PlainArray` que lo que hace es convertir el array anidado en un array corriente de dimensiones `(1 to n, 1 to m)`. En el ejemplo que sigue, primero presentamos una subrutina `Test()` que usa las funciones implementadas más abajo (las cuales agregamos a nuestra biblioteca `BblMatematica`!).

Programa 41 *Operaciones con matrices usando las funciones de OOo Calc*

```
Sub  Test()
  Dim i,j
  Dim mA(2)
  mA(0) = Array( 1, 2 )
  mA(1) = Array( 4, 5 )
  mA(2) = Array( 7, 8 )
 Dim mB(1)
  mB(0) = Array ( 3, 2, 1 )
  mB(1) = Array ( 6, 5, 4 )
 Dim mC()
```

```
  'Cálculo de la transpuesta de mA
   mC = CTRANSPOSE(mA)
  For i=1 To UBound(mC,2)
    For j = 1 To UBound(mC,1)
       MsgBox mC(j,i)
   Next j
  Next i
  'Cálculo de mA·mB
  mC=CMATRIXMULTIPLICATION(mA,mB)
  For i=1 To UBound(mC,2)
    For j = 1 To UBound(mC,1)
      MsgBox mC(j,i)
   Next j
 Next i
End Sub
'-------------------------------------------------------------------------
'Función para llamar la función 'MMULT' (multiplicación matricial) de la hoja
Function  CMATRIXMULTIPLICATION(A(), B())
 Dim  mM()
   mM = calc_Func("MMULT",Array(A(),B()))
   DataArray2PlainArray mM()
 CMATRIXMULTIPLICATION = mM
End Function

'Función para llamar la función 'TRANSPOSE' de la hoja
Function  CTRANSPOSE(A())
  Dim  mM()
   mM = calc_Func("TRANSPOSE",Array(A()))
   DataArray2PlainArray mM()
 CTRANSPOSE = mM
End Function

'Función para llamar las funciones de la hoja vía el servicio 'FunctionAccess'
Function  calc_Func(sFunc$,args())
 Dim oFA As Object
  oFA = createUNOService("com.sun.star.sheet.FunctionAccess")
 calc_Func = oFA.callFunction(sFunc,args())
End Function

'Cambiar array anidado a array corriente
Sub  DataArray2PlainArray(aRows())
 Dim i%,j%, aCols(),aTmp()
 ReDim aTmp(1 To UBound(aRows())+ 1,  1 To UBound(aRows(0)) +1)
 For i = 0 To UBound(aRows())
   aCols = aRows(i)
   For j = 0 To UBound(aCols())
      aTmp(i +1,j +1) = aCols(j)
   Next
Next
```

```
60    aRows() = aTmp()
61  End Sub
```

1.7 Bibliotecas especiales.

Hasta a hora hemos implementado algunas funciones y subrutinas de ejemplo. En ingeniería y ciencias se necesitan bibliotecas de funciones espaciales: De análisis de datos, métodos numéricos, etc. La primera biblioteca que vamos a implementar es una biblioteca con funciones y subrutinas generales de uso frecuente. Por supuesto, las primeras funciones que podemos agregar son las funciones que ya hemos implementado más arriba. Además de estas funciones, necesitamos algunas funciones adicionales. Recordemos que este capítulo corresponde al capítulo introductorio de un curso de métodos numéricos, así que que las nuevas funciones de la bibliotecas irán apareciendo en el camino.

1.7.1 Biblioteca `BblMatematica` de funciones de uso frecuente.

Esta biblioteca contiene funciones especiales y funciones misceláneas. Lo que hacemos es abrir un cuaderno nuevo, crear la biblioteca y luego exportarla. Luego podemos usar este mismo cuaderno o importar la biblioteca. En todo caso, antes de usar las funciones de la biblioteca, hay que cargarla en la subrutina Main,

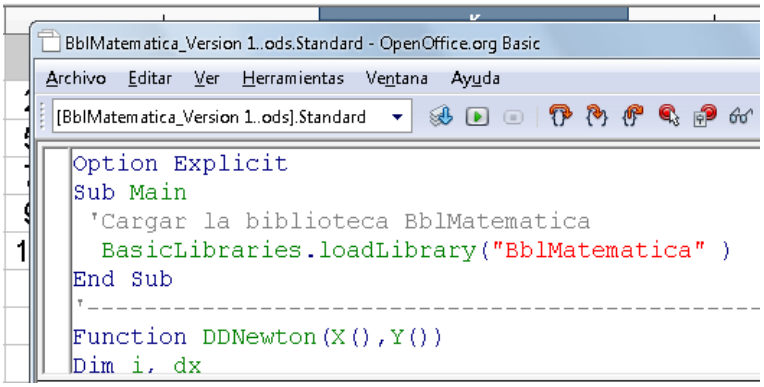


Figura 1.22 Importar y cargar la biblioteca `BblMatematica`.

Ahora, podemos agregar varias de las funciones y subrutinas que hemos implementado a esta biblioteca.

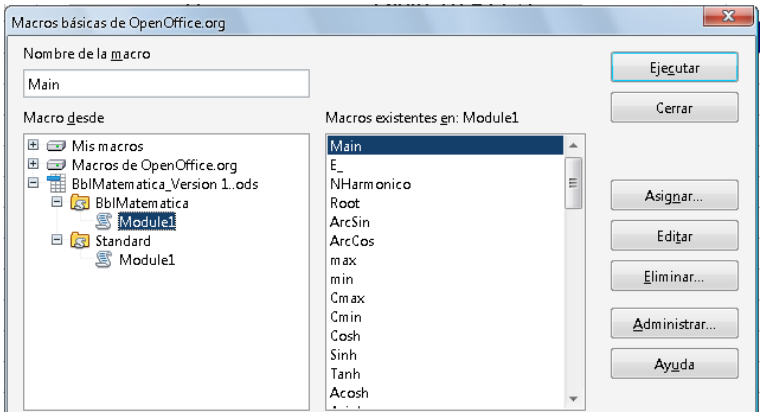


Figura 1.23 Biblioteca `BblMatematica`.

1.7.2 Algunas funciones especiales

Raíces $n-$ésimas. Si n es impar y $a < 0$ usamos la fórmula $a^{1/n} = \text{sgn}(a) \cdot |a|^{1/n}$. En otro caso usamos $a^{1/n}$.

Programa 42 `Root(x,n)` *con manejador de errores.*

```
Function  Root(ByVal a As Double, ByVal n As Double) As Double
  Dim m  As Integer
   m = Int(n)    'índice entero
   On Error Goto  msgError  'Manejo de errores
  If m Mod 2 = 0 Then       'Errores posibles: división por cero o
            Root = a ^ (1 / m)    'subradical negativo.
     Else
                Root = Sgn(a) * Abs(a) ^ (1 / m)
  End If
   Exit  Function
   msgError:
  If Err <> 0 Then
  MsgBox "Error #: "& Err & Chr(13) & Error & Chr(13) & "En la línea " & Erl
  End If
   On Error Goto  0 'reinicializar las variables de error,
                                ' es decir Err, Error y Erl.
End Function
```

Funciones trigonométricas inversas. En OOo Basic tenemos únicamente las funciones $\cos(x)$, $\text{sen}(x)$, $\tan(x)$ y $\arctan(x)$. El resto de funciones trigonométricas se deben implementar usando éstas o usando las funciones de la hoja. Muchas fórmulas se pueden encontrar en libros de tablas y fórmulas matemáticas como [1].

ArcSen(x) y ArcCos(x). Para implementar estas funciones necesitamos una identidad que las relacione con $arctan(x)$. Consideremos la figura (1.24),

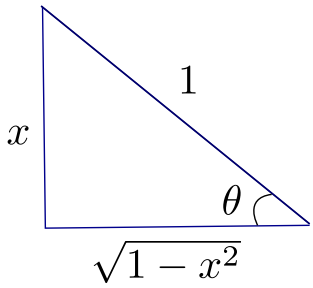


Figura 1.24

Como $\operatorname{sen}\theta = x$ entonces $\operatorname{arcsen}(x) = \arctan\left(\frac{x}{\sqrt{1-x^2}}\right)$, si $0 < x < 1$. Como $-\operatorname{arcsen} x = \operatorname{arcsen}(-x)$, $-\arctan x = \arctan(-x)$ y $\arccos x = \pi/2 - \operatorname{arcsen} x$, entonces

$$\begin{aligned} \operatorname{arcsen}(x) &= \arctan\left(\frac{x}{\sqrt{1-x^2}}\right) \quad \text{si } -1 < x < 1, \\ \arccos(x) &= \arctan\left(\frac{-x}{\sqrt{1-x^2}}\right) + \pi/2 \quad \text{si } -1 < x < 1. \end{aligned}$$

Programa 43 *Funciones* `arcsen(x)` *y* `arccos(x)`

```
Function Acos(ByVal a As Double) As Double
  If a = 1 Then
    Acos = 0
  Else If a = -1 Then
    Acos = PI
  Else
    Acos = Atn(-a / Sqr(-a * a + 1)) + PI / 2
  End If
End Function

Function Asin(ByVal a As Double) As Double
  If Abs(a) = 1 Then
    Asin= Sgn(a) * PI / 2
  Else
    Asin = Atn(a / Sqr(a * a + 1))
  End If
End Function
```

Min, Max. Mínimo y máximo de dos valores.

Programa 44 *Funciones* `Min` *y* `Max`

```
Function Min( p1, p2 )
   If p1 < p2 Then
      Min = p1
```

```
4     Else
5        Min = p2
6     End If
7  End Function
8
9  Function Max( p1, p2 )
10    If p1 > p2 Then
11         Max = p1
12    Else
13     Max = p2
14    End If
15 End Function
```

Random entero entre `nMin` y `nMax`. `Rnd()` devuelve un número 'aleatorio' entre 0 y 1. Para generar enteros 'aleatorios' entre `nMin` y `nMax` se usa la fórmula

```
Int(nMin + RND() * (nMax - nMin))).
```

Por ejemplo, para generar enteros aleatorios entre 10 y 30 se usa `Int(10+Rnd() * 20)`.

Programa 45 *Número entero aleatorio entre* `nMin` *y* `nMax`.

```
1  Function IntRandom( ByVal nMin As Double, ByVal nMax As Double ) As Double
2     IntRandom= Int(nMin + RND() * (nMax - nMin))
3  End Function
```

Función Gamma y factorial. La función Gamma se define como

$$\Gamma(z) = \int_0^\infty t^{z-1} e^{-t} dt$$

Una de las propiedades importantes de esta función es

$$\Gamma(z+1) = z\Gamma(z)$$

En particular, como $\Gamma(1) = 1$ entonces $\Gamma(n+1) = n!$ si $n \in Z$. Esta última fórmula no es adecuada para calcular $n!$ puesto que este caso particular de la función Γ se puede calcular de manera más sencilla.

La función Γ se calcula usualmente con la aproximación (Lanczos, 1964. [12, pág 256]),

$$\Gamma(z+1) = (z+\gamma+\frac{1}{2})^{z+\frac{1}{2}} e^{-(z+\gamma+\frac{1}{2})} \cdot \sqrt{2\pi}\left(c_0 + \frac{c_1}{z+1} + \frac{c_2}{z+2} + \cdots + \frac{c_N}{z+N} + \epsilon\right) \quad \text{si} \quad z > 0.$$

La fórmula también es válida si $z \in \mathbb{C}$ y $\mathrm{Re}\, z > 0$. P. Godfrey calculó un conjunto de 14 coeficientes a_i con los cuales el error $|\epsilon| < 10^{-15}$, suficiente para nuestros cálculos en el computador.

Es conveniente calcular $\ln\Gamma(z)$ para evitar los desbordamientos tempranos (overflow). Sea $A = z+\gamma+\frac{1}{2}$, entonces

$$\begin{aligned}\ln\Gamma(z) &= \ln\left(\frac{\Gamma(z+1)}{z}\right)\\ &\approx \frac{(z+0.5)\ln A - A + \ln\left[\sqrt{2\pi}\left(c_0 + \frac{c_1}{z+1} + \frac{c_2}{z+2} + \cdots + \frac{c_14}{z+14}\right)\right]}{z}\end{aligned}$$

donde, $a_0 = 0.999999999999997092$, $\gamma = 671/128$ y

$$\begin{aligned}a_1 &= 57.1562356658629235\\ a_2 &= -59.5979603554754912\\ a_3 &= 14.1360979747417471\\ a_4 &= -0.491913816097620199\\ a_5 &= 0.339946499848118887\times 10^{-4}\\ a_6 &= 0.465236289270485756\times 10^{-4}\\ a_7 &= -0.983744753048795646\times 10^{-4}\\ a_8 &= 0.158088703224912494\times 10^{-3}\\ a_9 &= -0.210264441724104883\times 10^{-3}\\ a_{10} &= 0.217439618115212643\times 10^{-3}\\ a_{11} &= -0.164318106536763890\times 10^{-3}\\ a_{12} &= 0.844182239838527433\times 10^{-4}\\ a_{13} &= -0.261908384015814087\times 10^{-4}\\ a_{14} &= 0.368991826595316234\times 10^{-5}\end{aligned}$$

Programa 46 *Función* $\ln(\Gamma(x)$ *(aproximación de Lanczos).*

```
Function  LnGamma(x As Double)
 Dim tmp As Double
 Dim suma As Double
 Dim Cf(14)
 Dim i
    Cf(0) = 0.999999999999997
    Cf(1)=57.1562356658629235
    Cf(2)=-59.5979603554754912
    Cf(3)=14.1360979747417471
    Cf(4)=-0.491913816097620199
    Cf(5)=0.339946499848118887E-4
    Cf(6)=0.465236289270485756E-4
    Cf(7)=-0.983744753048795646E-4
    Cf(8)=0.158088703224912494E-3
    Cf(9)=-0.210264441724104883E-3
```

```
    Cf(10)=0.21743961811521264 3E-3
    Cf(11)=-0.16431810653676389 0E-3
    Cf(12)=0.84418223983852743 3E-4
    Cf(13)=-0.26190838401581408 7E-4
    Cf(14)=0.36899182659531623 4E-5
    tmp = x + 5.2421875        'x+γ+1/2, γ = 671/128
    tmp =(x+0.5)*Log(tmp)-tmp
    suma = Cf(0)
    For i = 1 To 14
        suma = suma + Cf(i) / (x + i)
    Next i
     LnGamma = tmp+Log(2.5066282746310005*suma/x)
End Function
```

Para calcular usamos `Exp(LnGamma(x))` con $x < 172$.

```
  For i=1  To 5                  '2,67893853470775
      MsgBox Exp(LnGamma(i/3))   '1,3541179394264
  Next i                         '1
                                 '0,89297951156925
                                 '0,902745292950934
  MsgBox Exp(LnGamma(171.6))     '1,58589690966708E+308
  MsgBox Exp(LnGamma(172))       '-> "Desbordamiento"
```

La función factorial la habíamos implementado antes,

Programa 47 *La función factorial.*

```
Function Factorial(n)
Dim i
Dim producto As Double
    producto=1
   For  i= 2 To n
        producto = producto*i
    Next i
    Factorial = producto
End Function
```

Se puede calcular `factorial(n)` con $1 \leq n < 171$ pero solo es exacta hasta $n = 20$, pues $20! = 243290200817664$ (15 dígitos) y `factorial(20)=2,43290200817664E+018`. Para valores más grandes de n el computador recurre a una aproximación, por ejemplo $21! = 5109094217170944$ (16 dígitos) mientras que `factorial(21)=5,10909421717094E+019`.

1.7.3 Funciones y subrutinas misceláneas

Una función `Cells`. La acción de leer y escribir en una celda es muy frecuente. Es conveniente tener un par de funciones para simplificar la lectura y la escritura.

Programa 48 *Funciones para celdas*

```
Function Cells(txt  As String, Optional numhoja)
  Dim nh
  If Not IsMissing(numhoja) Then
      nh = numhoja
  Else nh=0
  End If
  Cells = ThisComponent.Sheets(nh).getCellRangeByName(txt)
End Function

Function CellsCF(columna, fila,  Optional numhoja)
  Dim nh
  If Not IsMissing(numhoja) Then
      nh = numhoja
  Else nh=0
  End If
   CellsCF = thisComponent.Sheets(nh).getCellByPosition(columna, fila)
End Function
```

La función `Cells` se puede usar para leer y escribir en las celdas de una hoja usando el nombre de la celda. La hoja default es la hoja 1. Por ejemplo, para leer la celda `A5` de la hoja 1 escribimos

```
Dim x
x = Cells("A4").Value.
```

La función `CellsCF` se puede usar para lo mismo haciendo referencia a la posición de la celda. Por ejemplo, para escribir el valor `f(x)` en la celda `B5` de la hoja 1, escribimos

```
CellsCF(1,4).Value=f(x)
```

Una función para limpiar un rango. Hay cálculos que llenan un rango. Un cálculo posterior puede llenar un rango menor y causar confusiones entre los datos nuevos y los viejos. Una manera de evitar este problema es implementando una subrutina que *limpie* las celdas (ver figura 1.25).

La subrutina `CleanRange(co,fi,nc)` limpia un rango iniciando en la celda (co, fi). Desde esta celda baja borrando `nc` columnas a la derecha, hasta que se encuentre una celda en blanco, es decir, una celda con cero caracteres.

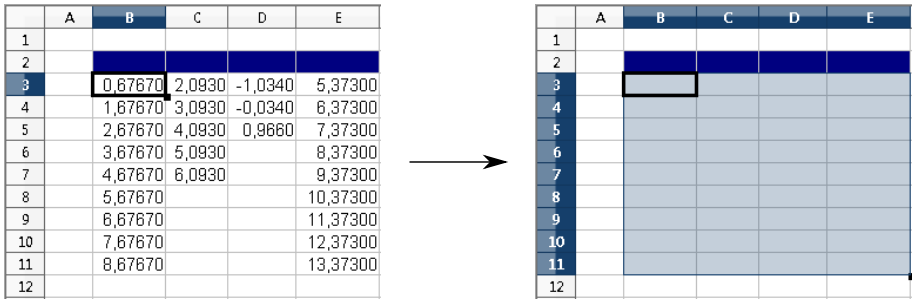


Figura 1.25 `Call CleanRange(1,2,4).`

Programa 49 *Subrutina para limpiar un rango*

```
'Limpia Rango
'cini         = columna en inicia la limpieza
'fini         = fila en que inicia la limpieza
'nc           = número de columnas (a la derecha) a borrar desde cini inclusive
'numhoja es el número de hoja, valor default = 0 (hoja 1).
Sub CleanRange(cini,fini, nc, Optional numhoja)
 Dim k,j, nh, lg
 Dim oCell, Hoja

 If Not IsMissing(numhoja) Then
       nh = numhoja
 Else nh=0
 End If
  Hoja = thisComponent.Sheets(nh)
  k=0
 Do
              'Recorremos la columna cini
       OCell = Hoja.getCellByPosition(cini, fini+k)
       lg = Len(oCell.String)
              'lg = 0 si la celda está en blanco
     If lg <> 0 Then
      For j = 0 To nc
             'Borra la fila "nc" columnas a la derecha
                Hoja.getCellByPosition(cini+j, fini+k).setString("")
      Next j
    End If
       k = k + 1
            'Hasta que encuentre la primera celda en blanco
Loop Until lg=0
End Sub
```

1.8 Gráficos.

El propósito de esta sección es desplegar la representación gráfica de una función desde una subrutina. Con OOo Basic se puede mostrar datos en forma de "diagramas" (gráficos de barras, pie, etc.), es decir, se crean vínculos gráficos con los datos, en forma de barras, sectores, líneas, etc.

Para hacer la representación gráfica de una función necesitamos un rango con los algunos pares ordenados (x_i, y_i) en el gráfico de la función. Luego estos pares se interpolan con un trazador cúbico (ver capítulo 2). Después de calcular este conjunto de pares ordenados, hay que hacer varias cosas. Primero necesitamos especificar la región dónde están los datos (en la hoja actual), en términos de filas y columnas: Fila de inicio, fila final, columna de inicio, columna final. Luego creamos un objeto gráfico, un rectángulo, especificando el largo, el ancho y la posición en la hoja en términos de su distancia al margen izquierdo y al margen superior de la hoja. En este rectángulo determina la posición de nuestro gráfico en la hoja. En este caso queremos que cada nuevo gráfico sustituya al anterior. Una vez que tenemos todos los elementos, incrustamos el "diagrama" e indicamos el tipo de gráfico (en nuestro caso `XYDiagram` con solo líneas). Finalmente hacemos algunos ajustes que tienen que ver tipo y tamaño de fuentes, color de fondo, etc.

Primero vamos a mostrar el código y luego se explica en detalle las partes de este código.Para la implementación vamos a usar como referencia la hoja de la figura (1.26),

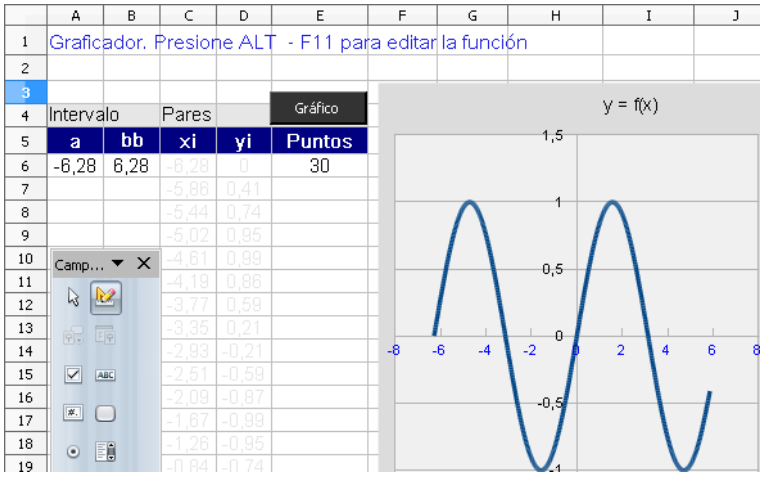


Figura 1.26

El código completo de la subrutina `Graficar()` es,

Programa 50 *Subrutina para graficar*

```
Function f(x)
     f= x^3+x+1
End Function

Sub  Graficar(rango)
Dim Rango(0) As New com.sun.star.table.CellRangeAddress
Dim Rectangulo As New com.sun.star.awt.Rectangle
Dim Chart    As Object
Dim oChart   As Object
Dim Hoja
Dim a,b,xi, n
```

```
12
13  Hoja  = thisComponent.Sheets(0)
14      a = Hoja.getCellRangeByName("A6").Value
15      b = Hoja.getCellRangeByName("B6").Value
16      n = Hoja.getCellRangeByName("E6").Value
17
18      'Pares ordenados en las columnas C, D
19        For  i=1 To n
20          xi=a+(i-1)*(b-a)/n
21          'Color de la fuente en celda: gris claro
22          Hoja.getCellByPosition(2,4+i).CharColor = RGB( 230,230,230 )
23          Hoja.getCellByPosition(3,4+i).CharColor = RGB( 230,230,230 )
24
25          Hoja.getCellByPosition(2,4+i).Value=xi
26          Hoja.getCellByPosition(3,4+i).Value=f(xi)
27      next i
28      'Insertar Gráfica
29      'Medidas en centésimas de milímetro
30   With Rectangulo
31      .X = 6500        'Distancia desde la izquierda de la hoja
32      .Y = 1000        'Distancia desde la parte superior
33      .Width = 10000   'El ancho del gráfico
34      .Height = 10000 'El alto del gráfico
35
36                       'Anclaje de los datos
37   With Rango(0)
38      .Sheet =        0     'Hoja 1
39      .StartColumn = 2      'C
40      .EndColumn =    3     'D
41      .StartRow =     5     'C6
42      .EndRow =       5+n   'C[6+n]
43   End With
44
45  Chart = thisComponent.Sheets(0).Charts
46  Chart.removeByName("Grafico")
47  Chart.addNewByName("Grafico",Rectangulo,Rango(),False, False)
48  oChart = Chart.getByName("Grafico").embeddedObject
49  oChart.diagram =oChart.createInstance("com.sun.star.chart.XYDiagram")
50  oChart.diagram.DataRowSource = com.sun.star.chart.ChartDataRowSource.COLUMNS
51
52                                 'Propiedades adicionales
53   With oChart
54        'Color de fondo, región externa.
55        .getArea().FillBackground = True
56        .getArea().FillStyle = RGB(220,220,220)
57        .getArea().FillColor = RGB(220,220,220)
58         'Color de fondo, región interna
59        .Diagram.getWall.FillBackground = True
60        .Diagram.getWall.FillStyle = RGB(240,240,240)
```

```
61          .Diagram.getWall.FillColor = RGB(240,240,240)
62
63          .Diagram.SplineType = 1   'Interpola con trazadores cúbicos
64          .Diagram.SymbolType =-1   'Símbolo para pares ordenados (none)
65           'Ejes y textos
66          .Diagram.HasXAxisTitle = True
67          .Diagram.XAxisTiTle.string = "X"
68          .Diagram.XAxisTiTle.CharHeight = 10
69          .Diagram.XAxis.CharHeight = 8
70          .Diagram.XAxis.Marks = 1
71           '.Diagram.XAxis.StepMain = 1
72          .Diagram.XAxis.StepHelp = 0
73          .Diagram.XAxis.CharColor = RGB(0,0,255)
74          .Diagram.XAxis.AutoStepMain = False
75          .Diagram.XAxis.AutoStepHelp = False
76          .Diagram.YAxis.CharHeight = 8
77          .Diagram.YAxis.Origin=0
78          '.Diagram.HasYAxisTitle = True
79          '.Diagram.YAxisTiTle.string = ""
80          '.Diagram.YAxisTiTle.CharHeight = 0
81          .HasMainTitle = True
82          .Title.string = "y = f(x)"
83          .Title.CharHeight = 10
84   End With
85  End Sub
```

Para calcular los pares ordenados necesitamos la función f, un intervalo $[a,b]$ y la cantidad n de puntos (por default será de 30). El código para el cálculo de los pares ordenados sería,

```
 For i=1 To n
         xi = a +(i-1)*(b-a)/n
         Hoja.getCellByPosition(2,4+i).Value=xi
         Hoja.getCellByPosition(3,4+i).Value=f(xi)
 Next i
```

Como son n nodos, dividimos el intervalo $[a,b]$ en n partes iguales, todas de tamaño $(b-a)/n$. Luego $x_0 = a$, $x_1 = a + (b-a)/n$, $x_2 = a + 2(b-a)/n, ..., x_n = b$.

Para el manejo del gráfico usamos cuatro objetos.

"`Rango()`" es un objeto que tiene soporte para usar los servicios del "servicio"[1] `com.sun.star.table.CellRangeAddress`, por tanto podremos tener acceso a la región de la hoja donde están los pares ordenados (estos serían los servicios del "servicio" `CellRangeAddress`),

[1]Los "servicios" son los componentes de OpenOffice. Un "servicio" es algo parecido a una clase o un "tipo" en Java. Las propiedades o métodos serían los servicios del "servicio". Ver [11].

```
Dim Rango(0) As New com.sun.star.table.CellRangeAddress
  ...
With Rango(0)
   .Sheet =       0     'Hoja 1
   .StartColumn = 2     'C
   .EndColumn =   3     'D
   .StartRow =    5     'C6
   .EndRow =      5+n   'C[6+n], después de n datos.
End With
```

El objeto `Rango` lo inicializamos en 0 si solo necesitamos ubicar una matriz de datos,en este caso los datos (x_i, y_i). Si necesitamos hacer referencia a $n+1$ bloques de datos en distintas partes (de la hoja o en otras hojas), ponemos `DimRango(n)` y el bloque de datos j se describe en términos de hoja, filas y columnas con `Rang(j).Sheet,...,Rango(j).EndRow`. Así, `Rango()` "tendrá" todos los datos.

Por ejemplo, si queremos la representación gráfica de dos funciones f y g en el mismo sistema y si, por alguna razón los datos $g(x_i)$ están algo lejos de las columnas `C,D` (en la misma hoja o otra hoja), entonces podríamos inicializar la matriz de la variable `Rango` en 1; de esta manera, con `Rango(0)` indicamos la ubicación de los datos (x_i, y_i) y con `Rango(1)` la ubicación de los datos $g(x_i)$. Así, el objeto `Rango()` "tendrá" tres columnas y el tipo de gráfico que elegimos (de 'dispersión') representa los pares $(x_i, f(x_i))$ y los pares $(x_i, g(x_i))$. En nuestro chart aparecerán los dos gráficos.

"`Rectangulo`" es un objeto que tiene soporte para usar los servicios del "servicio" `com.sun.star.awt.Rectangle`, por tanto tenemos acceso a las propiedades (servicios) que nos permite dotar de dimensiones al rectángulo y posicionarlo en al hoja. Este rectángulo determina las dimensiones y la posición de nuestro gráfico.

```
Dim Rectangulo As New com.sun.star.awt.Rectangle
  ...
                    'Medidas en centésimas de milímetro
With Rectangulo
   .X = 6500        'Distancia desde la izquierda de la hoja
   .Y = 1000        'Distancia desde la parte superior
   .Width = 10000   'Ancho del rectángulo
   .Height = 10000 'Altura del rectángulo
End With
```

El objeto `Chart` tiene los "Charts" (gráficos) de la hoja. Usando `Chart` creamos la representación gráfica de los datos de `Rango(0)`. La posición de esta representación está determinada por `Rectangulo`. Para hacer todo esto se usa el método `addNewByName`,

```
Chart = thisComponent.Sheets(0).Charts
Chart.removeByName("Grafico")  'Remover gráfico anterior
Chart.addNewByName("Grafico",Rectangulo, Rango(),False, False)
```

Los parámetros `addNewByName` son: `"Nombre"`, `Rectangulo`, `Rango()`, `EncabezadoColumna` y `EncabezadoFila`.

`oChart` tiene acceso al gráfico, usando el nombre. Desde `oChart` especificamos algunas propiedades

```
oChart = Chart.getByName("Grafico").embeddedObject
oChart.Diagram =oChart.createInstance("com.sun.star.chart.XYDiagram")
oChart.Diagram.DataRowSource = com.sun.star.chart.ChartDataRowSource.COLUMNS
...
```

Ejercicios

1.1 Implementar una subrutina `Graficar(f,a,b)` que recibe una función `f` y la evalúa con la función *Eval* (de nuestra biblioteca `BblMatematica`) y hace la representación gráfica en $[a,b]$.

1.9 Modelo de Objetos de OOo.

Esta es una sección muy general que trata de dar una idea del manejo interno de OOo. En Java o VBA por ejemplo, uno puede crear clases, luego crear objetos para accesar las propiedades y métodos de la clase. En OOo Basic las cosas son algo diferentes. OOo Basic obtiene toda su funcionalidad de componentes "UNO" (Universal Network Objects, objetos de red universales). Un UNO esta compuesto de Servicios, Interfaces y Propiedades.
Un *servicio* es un componente de un UNO. Cada servicio consiste de una o más interfaces, las interfaces son conjuntos de métodos, para interactuar con los clientes (nuestros programas). Las propiedades son Constantes, Excepciones, Estructuras (strucs) y Enumeraciones.

Los servicios UNO están agrupados jerárquicamente en módulos, este capítulo todos los módulos que usamos están en un módulo central `com.sun.star.`

Por ejemplo, para accesar celdas de una hoja necesitamos el UNO `com.sun.star.table`. Este UNO tiene los servicios `Cell, CellCursor, CellRange, etc.`

Por ejemplo, el servicio `CellRange` tiene la interface `XCellRange` y esta interface tiene los métodos `getCellByPosition, getCellRangeByName` y `getCellRangeByPosition.`

Cuando en una macro necesitamos crear una instancia de un servicio, usamos la función `createUnoService()`.
Una vez que referenciamos un servicio, podemos usar sus métodos y propiedades.

Por ejemplo, en nuestra `BblMatematica` instanciamos el servicio `FunctionAccess` del módulo `sheet` para accesar el método `callFunction.`

```
oFunction = createUnoService("com.sun.star.sheet.FunctionAccess")
cmax = oFunction.callFunction("MAX",oRange)
```

Todos los componentes de OpenOffice.org se pueden implementar en cualquier lenguaje que soporte UNO's. En C++ y Java se puede implementar componentes UNO. Cuando se instancia un UNO implementado en Java se levanta la máquina virtual de Java dentro de OpenOffice.org para atender la demanda.

Los componentes UNO *no* se pueden implementar con OOo Basic, solo los manejadores de eventos (Listeners). OpenOffice.org Basic es un lenguaje de scripts desarrollado para integrar directamente en OpenOffice.org pero no es la mejor opción para grandes proyectos aunque combina bien con otros lenguajes que requieren atajos rápidos y eficientes en ciertas tareas dentro de OOo.

Bibliografía

[1] A. Jeffrey, Hui-Hui Dai. *Handbook of Mathematical Formulas and Integrals.* 4th edition. Academic Press. 2008.

[2] A. Channelle. *Beginning OpenOffice 3. From Novice To Professional.* Apress. 2009.

[3] A. Pitonyak. *OpenOffice.org Macros Explained*. Hentzenwerke Publishing. 2004.

[4] L. Godart, Bernard Marcelly. *Programmation OpenOffice.org 2. Macros OOo Basic et API.* Editions Eyrolles. 2006.

[5] M. Abramowitz, I. Stegun. *Handbook of mathematical functions.* Dover Publications, inc., New York,1972.

[6] M.Baeza S. *Aprendiendo OOo Basic*. En http://www.universolibre.org/node/1

[7] M. A. Bain. *Learn OpenOffice.org. Spreadsheet Macro Programming OOo Basic and Calc Automation.* Packt Publishing. 2006.

[8] R. Graham, D. Knuth y O.Patashnik. *Concrete Mathematics: A Foundation for Computer Science*, 2nd ed. Addison-Wesley, pp. 272-282, 1994.

[9] OOo Forum.org. "Solved:Sheet Cell Array Functions in Basic." http://www.oooforum.org/forum/viewtopic.phtml?t=23520. Consultada el 18 de agosto 2010.

[10] OOo Forum.org. "How to use callFunction()?." http://www.oooforum.org/forum/viewtopic.phtml?p=389084#389084. Consultada el 24 de agosto 2010.

[11] OpenOffice.org Wiki. "OpenOffice.org BASIC Programming Guide." En http://wiki.services.openoffice.org/wiki/Documentation/BASIC_Guide

[12] W. Press W. et al., *"Numerical Recipes in C, The Art of Scientific Computing.* Cambridge University Press, New York, 1988.