Programación Visual Basic (VBA) para Excel y Análisis Numérico

M.Sc. Walter Mora F., M.Sc. José Luis Espinoza B.
Escuela de Matemática
Instituto Tecnológico de Costa Rica

Octubre 2005

Versión 0.1

Contents

Chapter 1

Programación Visual Basic (VBA) para Excel

1.1 Introducción

Microsof Excel© es un software para el manejo de hojas electrónicas agrupadas en *libros* para cálculos de casi cualquier índole. Entre muchas otras aplicaciones, es utilizado en el tratamiento estadístico de datos, así como para la presentación gráfica de los mismos. La hoja electrónica Excel es ampliamente conocida, en forma generalizada, por profesionales y estudiantes en proceso de formación, pero hay una gran cantidad de usuarios que no conocen a profundidad su gran potencial y adaptabilidad a los diferentes campos del conocimiento.

Para científicos e ingenieros, el Excel constituye una herramienta computacional muy poderosa. También tiene gran utilidad para ser utilizado en la enseñanza de las ciencias y la Ingeniería, particularmente, en la enseñanza de los métodos numéricos. Pese a que existen en el mercado programas computacionales muy sofisticados, tales como MATLAB, MATHEMATICA, etc., no están tan disponibles como Excel, que usualmente forma parte del paquete básico de software instalado en las computadoras que funcionan bajo el sistema Windows© de Microsoft.

A continuación se brinda al lector una breve introducción a algunas actividades de programación con macros escritos en VBA (una adaptación de Visual Basic para Office de Microsoft), definidos desde una hoja electrónica de Excel. Salvo pequeñas diferencias para versiones en inglés, el material puede ser desarrollado en cualquier versión.

1.2 Evaluación de funciones

1.2.1 Funciones definidas por el usuario

A manera de ejemplo, vamos a evaluar la función

$$f(x) = 2x^3 + \ln(x) - \frac{\cos(x)}{e^x} + \text{sen}(x)$$

1. Como al evaluar $f(x)$ se debe recurrir a varias funciones básicas que se invocan desde Excel, se puede tener acceso a su sintaxis, pulsando el ícono $\boxed{f_x}$ y seleccionar 'Matemáticas y Trigonométricas'.

2. Para escribir una fórmula, seleccionamos una celda para escribir el valor a ser evaluado; por ejemplo, podemos digitar el valor 1.1 en la celda B3.

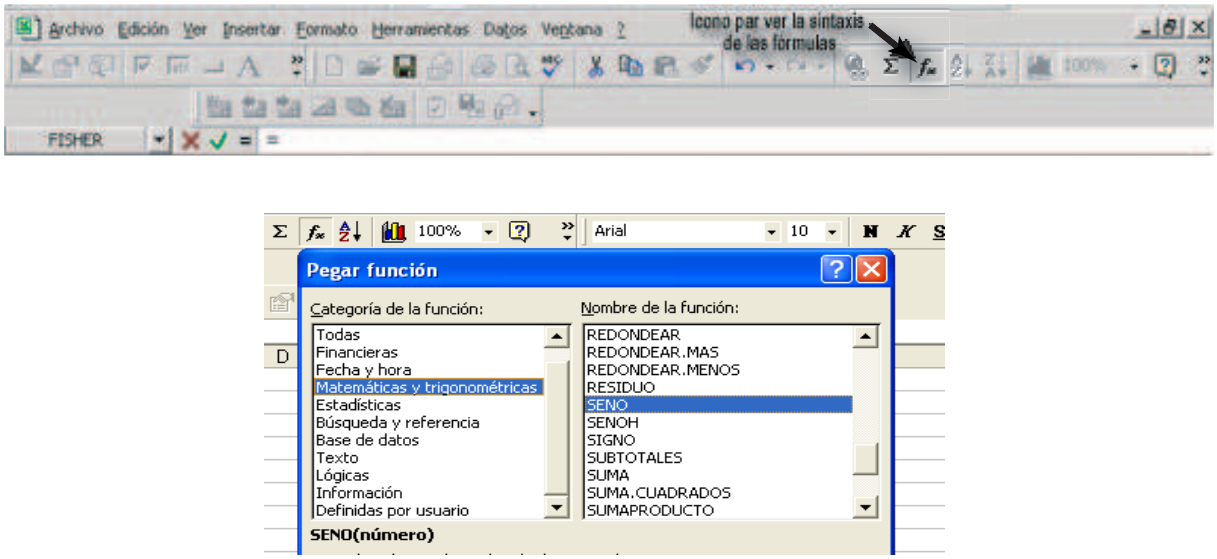


Figure 1.1: Funciones predefinidas en Excel.

3. Ahora en la celda C3 digitamos, de acuerdo a la sintaxis de la versión de Excel en español[1], la fórmula:

```
=2*B3^3+LN(B3)-COS(B3)/EXP(B3)+SENO(B3)
```

Una vez que ha sido digitada, simplemente se pulsa la tecla 'Entrar' o 'Enter'.

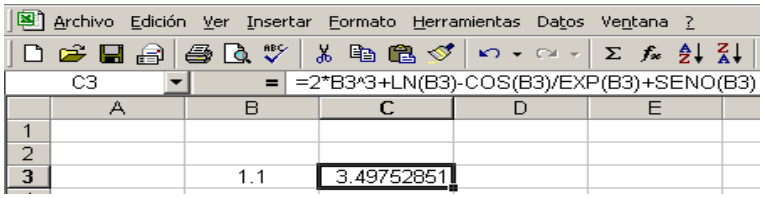


Figure 1.2: Al evaluar fórmulas, a menudo se requiere evaluar varias funciones predefinidas.

1.2.2 Errores comunes

Conforme vamos digitando nuestras primeras fórmulas, nos van apareciendo algunos errores que usualmente son debidos a un manejo inadecuado de la sintaxis o a la incompatibilidad con la configuración de la computadora. A continuación se describen algunas situaciones que pueden aparecer.

1. El valor de error `#¿NOMBRE?` aparece cuando Excel no reconoce texto en una fórmula. Deber revisar la sintaxis de dicha fórmula o, si es una macro, verificar que esté en un módulo de esta hoja.

2. El valor de error `#¡VALOR!` da cuando se utiliza un tipo de argumento (u operando) incorrecto. Este error se da por ejemplo, cuando evaluamos una función numérica en una celda que contiene algo que no sea un número (Por defecto, el contenido de una celda vacía es cero).

3. El valor de error `#¡NUM!` se aparece cuando hay un problema con algún número en una fórmula o función. Por ejemplo, si evaluamos una función logarítmica en cero o en un número negativo.

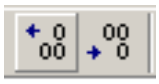
[1]La versión que estamos usando está en español. Por ejemplo, en la versión en inglés de Excel, se usa SIN(x) en lugar de SENO(x).

4. El valor de error `#¡DIV/0!` se produce cuando se divide una fórmula por 0 (cero).

5. El valor de error `#¡REF!` se da cuando una referencia a una celda no es válida.

6. Dependiendo de la forma en que esté configurado el sistema Windows, debe usarse punto o coma para separar la parte decimal de los números a evaluar. Para personalizarlo, se debe entrar al panel de control y en la 'Configuración regional' se selecciona 'Números'. En la primera cejilla, 'Símbolo Decimal' se selecciona el punto o la coma, según sea el caso. Finalmente, se presiona el botón 'Aplicar' y luego 'Aceptar'.

7. Una situación que a veces es confundida con un error se da cuando el sistema trabaja con poca precisión y se presentan valores numéricos no esperados. Por ejemplo, si el formato de una celda se ha definido para dos posiciones, entonces la operación `+1.999+1` efectuado en dicha celda dará como resultado el valor `2`, que no es otra cosa que el resultado de tal suma redondeado a dos decimales. El valor correcto se obtiene aumentado la precisión con el ícono correspondiente:



   También se puede cambiar la precisión en el menú `'Formato-Celdas-Número-Posiciones decimales'`.

   Estos cambios son sólo de apariencia, pues, independientemente del número de dígitos que sean desplegados, Excel manipula los números con una precisión de hasta 15 dígitos. Si un número contiene más de 15 dígitos significativos, Excel convertirá los dígitos adicionales en ceros (0).

1.2.3 Evaluando una función en varios tipos de parámetros

Muchas fórmulas a evaluar tienen argumentos de distinto tipo, pues algunos argumentos varían (a veces con un incremento determinado), mientras que otros permanecen constantes. Por lo general estos argumentos son tomados de celdas específicas, por lo que es importante saber manejar distintos escenarios para la evaluación de una función o fórmula.

Evaluación con argumentos variables

Continuando con el ejemplo que iniciamos en la sección 2.1, a partir de la celda B4 podemos continuar digitando valores, siempre en la columna B y con el cuidado de que estos números no se salgan del dominio de la función $f(x) = 2x^3 + \ln(x) - \dfrac{\cos(x)}{e^x} + \text{sen}(x)$, que en este caso es el conjunto de los números reales positivos. Una vez hecho ésto, se evalúa la función $f(x)$ en la celda C3, como se hizo previamente. Luego, seleccionamos esta misma celda C3 y se ubica el *mouse* en la esquina inferior derecha, arrastrándolo hasta la celda deseada. Otra posibilidad es hacer un doble clic en la esquina inferior derecha de la celda a copiar y esto realiza la copia automáticamente.

Evaluación con argumentos variables y/o constantes

Es común tener que evaluar funciones o fórmulas que dependen de varios parámetros, algunos de los cuales se mantienen fijos mientras que otros son variables.

Ejemplo 1

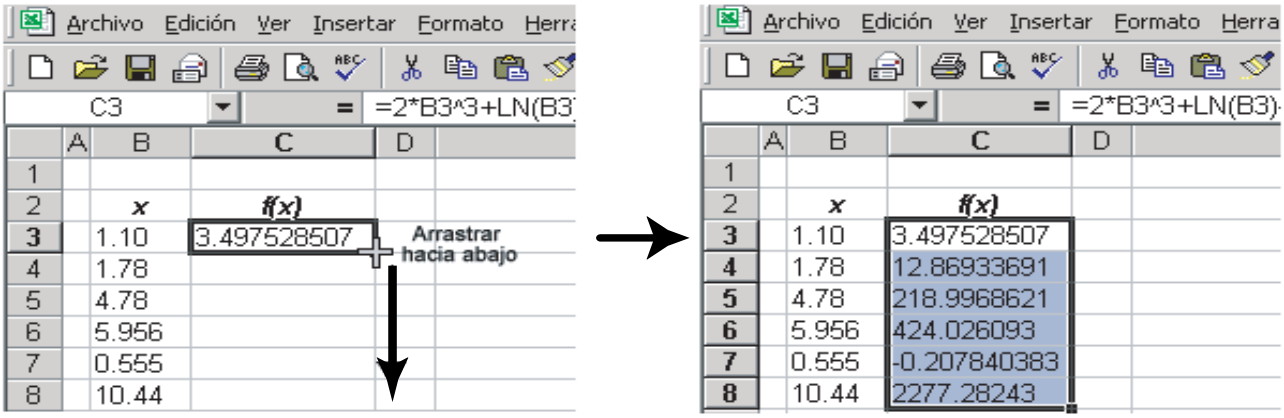


Figure 1.3: Copia de una fórmula en un grupo de celdas.

El siguiente ejemplo describe una función con dos parámetros y una variable.

La función $P(t) = \dfrac{K}{1 + Ae^{-kt}}$, con $A = \dfrac{K - P_0}{P_0}$, describe el tamaño de una población en el momento t.
Aquí:

- k es una constante de proporcionalidad que se determina experimentalmente, dependiendo de la población particular que está siendo modelada,
- P_0 es la población inicial y
- K es una constante llamada *capacidad de contención* o *capacidad máxima que el medio es capaz de sostener.*

Si queremos evaluar $P(t)$ para distintos valores del tiempo t en días, seguimos la siguiente secuencia de pasos:

1. Para empezar, es importante escribir encabezados en cada una de las columnas (o filas) donde vamos a escribir los datos que serán los argumentos de la función. En este caso, comenzando en la celda B3, escribimos las etiquetas

   ```
   P0      K      k     t      P(t).
   ```

2. A continuación escribimos los valores de los parámetros, comenzando en la celda B4

   ```
   100     1000   0.08  0.
   ```

3. Ahora escribimos la fórmula de la función $P(t)$ en la celda G4:

   ```
   =C$4/(1+((C$4-B$4)/B$4)*EXP(-D$4*E4))
   ```

   Como puede observarse, el único argumento variable es t y nos interesa mantener a los otros argumentos constantes. Para mantener un valor (que se lea en una celda) constante, se le agrega el símbolo $ antes del número de fila, como por ejemplo `C$4`.

   En nuestro ejemplo, los argumentos constantes son los que están en las celdas B4, C4 y D4, mientras que el valor de t en la celda E4, es variable.

	A	B	C	D	E	F
1						
2						
3		P0	K	k	t	P(t)
4		100	1000	0,08	0	100
5					5	142,18925
6					6	152,229
7					10	198,2569
8					15	269,48745
9					17	302,11964
10					20	354,97892
11					21	373,50145
12					30	550,52086
13					45	802,62394

Figure 1.4: Evaluación con parámetros constantes y un parámetro con un incremento variable.

4. Finalmente, escribimos varios valores para t en la columna E, seleccionamos la celda F4 y arrastramos para evaluar $P(t)$ en el resto de valores de t.

   Nota: $P(t)$ es la solución de la llamada *ecuación logística* $\dfrac{dP}{dt} = kP\left(1 - \dfrac{P}{K}\right)$.

Construyendo rangos con un incremento fijo

A menudo necesitamos evaluar una función en una secuencia de valores igualmente espaciados, por lo que a continuación se explica cómo hacerlo, modificando el ejemplo previo de crecimiento de una población.

1. Podemos seleccionar la columna C para poner los valores fijos P_0, K, k y el incremento h. En este caso, por ejemplo, $h = 5$ serviá como incremento entre un tiempo y el siguiente, iniciando con $t = 0$.

2. En la celda E4 escribimos el tiempo inicial $t = 0$ y en la celda E5 se escribe el nuevo tiempo con el incremento h:

   ```
   =+E4+C$6
   ```

   Debemos usar `C$6` para que el incremento se mantenga inalterado al copiar esta operación en otra fila celda situada en una fila diferente.

3. Ahora seleccionamos esta celda `E5` y la arrastramos hacia abajo para obtener los nuevos tiempos con el respectivo incremento.

	A	B	C	D	E	F
1						
2						
3		P0	K	k	t	P(t)
4		100	1000	0,1	0	100
5					5	142,1893
6	Incremento:	h=	5		10	198,2569
7					15	269,4875
8					20	354,9789
9					25	450,8531
10					30	550,5209
11					35	646,291
12					40	731,6039
13					45	802,6239

Figure 1.5: Evaluación con un parámetro de incremento fijo

Nota: Esto también se puede hacer escribiendo, en celdas consecutivas, un valor y luego el valor más el incremento y luego seleccionando ambas celdas y arrastrando. Sin embargo, en algunos algoritmos es más cómodo

tener una celda dónde leer el incremento.

1.3 Gráficas

Continuando con el ejemplo anterior en el que se ha evaluado la población $P(t)$ en un conjunto de valores del tiempo t, recordemos que en las columnas E y F se han escrito, respectivamente, los valores de t y $P(t)$. Para graficar $P(t)$ con respecto a t, pordemos seguir los siguientes pasos:

1. Seleccionamos el rango en el cual se encuentran los valores de t y $P(t)$. Este rango puede incluir las celdas que contienen los rótulos de las columnas.

2. Presionamos el icono  , que activa el asistente para gráficos.
   Hay varias opciones que podemos elegir para el gráfico y en nuestro caso podemos elegir el tipo `Dispersión.`

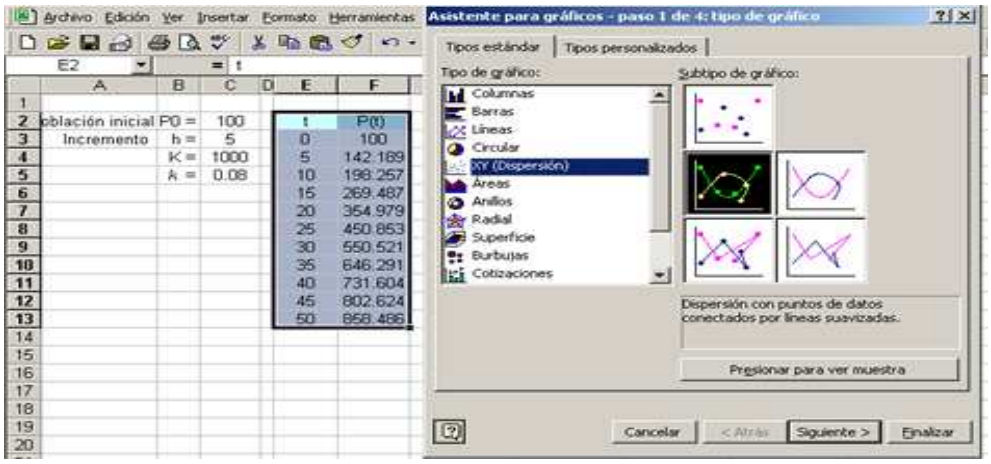


Figure 1.6: Selección del tipo de gráfico.

3. Presionamos el botón Siguiente y luego Finalizar. Antes de finalizar se pueden escoger distintas opciones para personalizar el gráfico. A continuación se muestra la salida del gráfico.

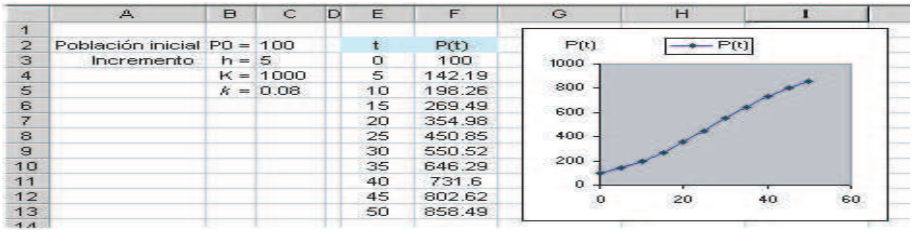


Figure 1.7: Curva obtenida para el modelo logístico de crecimiento de una población.

La curva obtenida se llama *curva logística* o *sigmoide*, por la forma de *'S'* que tiene.

1.4 Programación de macros

1.4.1 Introducción

El lenguaje Visual Basic para Aplicaciones (VBA), en el contexto de Excel, constituye una herramienta de programación que nos permite usar código Visual Basic adaptado para interactuar con las múltiples facetas de Excel y personalizar las aplicaciones que hagamos en esta hoja electrónica.

Las unidades de código VBA se llaman `macros`. Las macros pueden ser *procedimientos* de dos tipos:

- Funciones (**Function**)

- Subrutinas (**Sub**)).

Las funciones pueden aceptar argumentos, como constantes, variables o expresiones. Están restringidas a entregar un valor en una celda de la hoja. Las funciones pueden llamar a otras funciones y hasta subrutinas (en el caso de que no afecten la entrega de un valor en una sola celda)

Una subrutina realiza acciones específicas pero no devuelven ningún valor. Puede aceptar argumentos, como constantes, variables o expresiones y puede llamar funciones.

Con las subrutinas podemos entregar valores en distintas celdas de la hoja. Es ideal para leer parámetros en algunas celdas y escribir en otras para completar un cuadro de información a partir de los datos leídos.

Editar y ejecutar macros.

Las funciones y las subrutinas se pueden implementar en el editor de Visual Basic (Alt-F11).

Para usar una función en una hoja de Excel se debe, en el editor de VB, insertar un módulo y editar la función en este módulo. Esta acción de describe más adelante. De la misma manera se pueden editar subrutinas en un módulo.

Una función se invoca en una hoja, como se invoca una función de Excel o una fórmula. Una subrutina se puede invocar por ejemplo desde la ventana de ejecución de macros (Alt-F8) o desde un botón que hace una llamada a la subrutina (como respuesta al evento de hacer clic sobre él, por ejemplo).

El código que ejecuta un botón puede llamar a subrutinas y a las funciones de la hoja. El código del botón no está en un módulo. En la hoja de edición donde se encuentra el código del botón, se pueden implementar funciones para uso de este código pero que serán desconocidas para la hoja (mensaje de error `#¿NOMBRE?`).

Nota: un error frecuente es editar una función en un módulo que corresponde a una hoja y llamarlo desde otra hoja. En este caso se despliega el error (mensaje de error `#¿NOMBRE?`).

1.4.2 Funciones

Una función tiene la siguiente sintaxis:

Function NombreFun(*arg1, arg2,...,argn*)

> Declaración de Variables y constantes
>
> *Instruccción 1*
> *Instruccción 2*
> ...

Instruccción k
NombreFun = *Valor de retorno* 'comentario

End Function

Una función puede tener o no tener argumentos, pero es conveniente que retorne un valor. Observe que se debe usar el nombre de la función para especificar la salida:

$$\textbf{NombreFun} = \textit{Valor de retorno}$$

Nota 1: Al interior de las funciones, se pueden hacer comentarios utilizando (antes de éstos) la comilla (').

Nota 2: Para el uso de nombres de variables o de cualquier otra palabra reservada de VBA, no se discrimina entre el uso de letras mayúsculas y minúsculas.

Ejemplo 1: implementar una función.

Vamos a implementar como una macro la función con la que se trabajó previamente:

$$f(x) = 2x^3 + \ln(x) - \frac{\cos(x)}{e^x} + \text{sen}(x)$$

Para su definición y utilización, se siguen los pasos:

1. Ingresamos al menú y en la opción `Herramientas` seleccionamos `Macros`. Luego se elige `Editor de Visual Basic`:

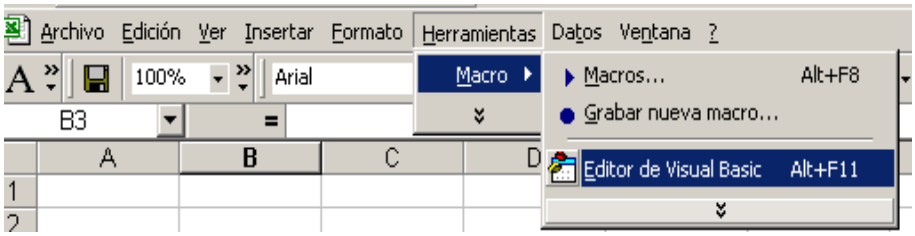


Figure 1.8: Primeros pasos para la definición de una macro.

También puede usar `Alt - F11`

2. Nuevamente, en el menú de la ventana que se abre, se elige `Insertar`, para luego seleccionar `Módulo`:
3. Ahora en la pantalla de edición del módulo, escribimos el siguiente código:

```
Function f(x)
  f = 2 * x ^ 3 + Log(x) - Cos(x) / Exp(x) + Sin(x)
End Function
```

4. Una vez que ha sido editado el código del macro, se salva y salimos del ambiente de programación en Visual Basic para volver a la hoja electrónica de donde partimos. Esto se hace en el menú `Archivo`, seleccionando `Cerrar y Volver a Excel`.

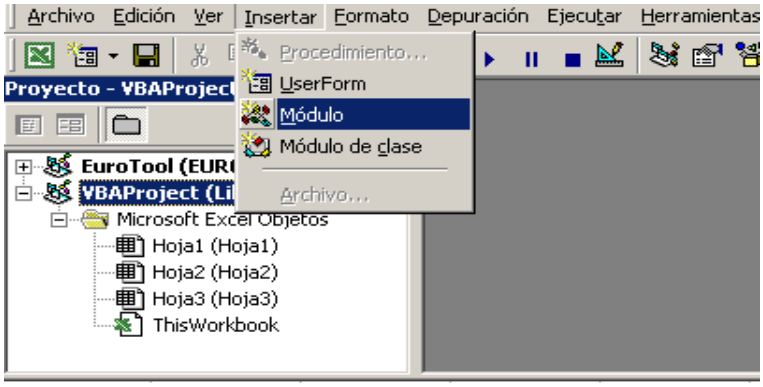


Figure 1.9: Se inserta un módulo en el que se escribirá el código de las macros.

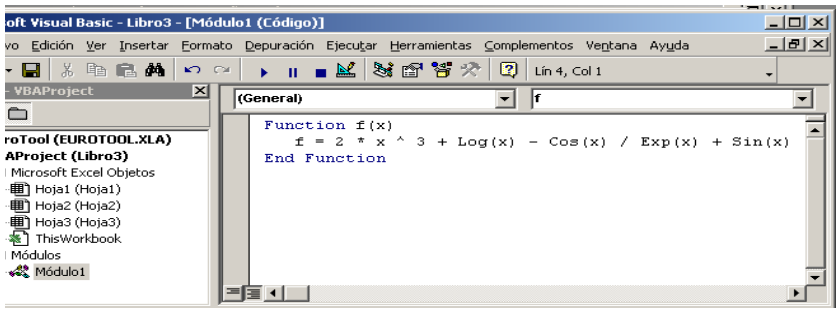


Figure 1.10: Escritura del código de una macro.

5. Para evaluar la función $f(x)$ en algunos valores que se localicen, por ejemplo, desde la fila 3 hasta la fila 6 de la columna B, nos situamos en una celda en la que nos interese dejar el resultado de dicha evaluación y se digita `+f(B3)`. Luego se arrastra hasta C6 para copiar la fórmula, quedando:

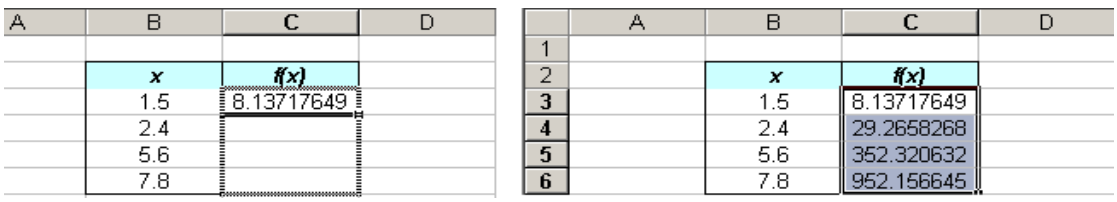


Figure 1.11: Evaluación de una función definida por el usuario.

Nota: Para conocer con detalle la sintaxis de las funciones matemáticas estándar que se pueden evaluar en Visual Basic, puede usarse la `Ayuda` del `Editor de Visual Basic`. Esta sintaxis es un poco diferente a la que maneja Excel para las mismas funciones. Como ya vimos, para implementar la función

$$f(x) = 2x^3 + \ln(x) - \frac{\cos(x)}{e^x} + \text{sen}(x)$$

- en Excel la sintaxis es: `2*B3^3+LN(B3)-COS(B3)/EXP(B3)+SENO(B3)`

- en VBA la sintaxis es `2 * x ^ 3 + Log(x) - Cos(x) / Exp(x) + Sin(x)`

Observe, por ejemplo, que la función logaritmo natural $\ln(x)$, en Excel se escribe `LN` mientras que en VBA se escribe `Log`.

Ejemplo 2: lectura de parámetros en celdas

Una vez más vamos a trabajar con el modelo de crecimiento poblacional descrito anteriormente. La función

$$P(t) = \frac{K}{1 + Ae^{-kt}},$$

con

$$A = \frac{K - P_0}{P_0}.$$

Ahora evaluaremos $P(t)$ para distintos valores del tiempo t en días, pero esta vez haremos dicha evaluación mediante una macro para definir $P(t)$.

Los parámetros los vamos a leer desde unas celdas ubicadas en la columna C. Para hacer referencia a una celda, se usa el código

```
Cells(fila,columna)
```

pero escribiendo 'columna' en formato numérico. Por ejemplo, la celda C5 se invoca como

```
Cells(5,3)
```

Lo primero que hacemos es escribir, en el editor de VBA, la fórmula de $P(t)$, luego la invocamos en la celda F3 (de nuestra hoja de ejemplo) y arrastramos. Para ésto, se siguen los siguientes pasos:

1. En primer lugar, abrimos una hoja Excel, que se llame por ejemplo `Poblacion.xls`. Luego se escriben los valores de los parámetros, tal y como puede observarse en la siguiente figura:

	A	B	C	D	E	F	G	H	I	J
1										
2	Población inicial	P0 =	100		t	0	5	10	15	20
3	Incremento	h =	5		P(t)					
4		K =	1000							
5		*k* =	0.08							
6										

Figure 1.12: Ubicación inicial de los parámetros.

2. Ahora ingresamos al menú y en la opción `Herramientas` seleccionamos `Macros`. Luego se elige `Editor de Visual Basic`. Nuevamente, en el menú de la ventana que se abre, se elige `Insertar`, para luego seleccionar `Módulo` y escribir el siguiente código:

```
Function P(t)
  P0 = Cells(2, 3)      'P0 est\'a en la celda C2
  LimPobl = Cells(4, 3) 'K  est\'a en la celda C4
  k = Cells(5, 3)       'k  est\'a en la celda C5
  A = (LimPobl - P0) / P0
  P = LimPobl / (1 + A * Exp(-k * t))
End Function
```

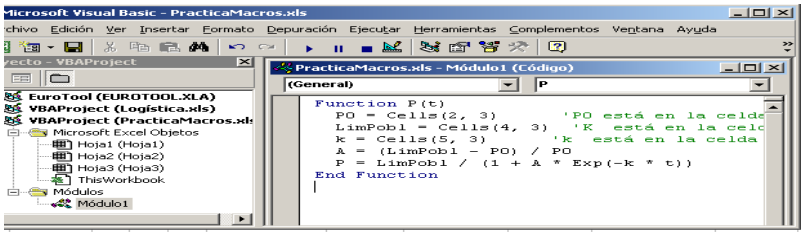


Figure 1.13: Captura de algunos parámetros constantes de las celdas de la hoja.

De esta forma, la ventana de edición de Visual Basic, quedaría así:

3. Una vez que ha sido editado el código de la macro, se guarda y salimos del ambiente de programación en Visual Basic para volver a la hoja electrónica de donde partimos. Este retorno se hace siguiendo el menú `Archivo` y seleccionando `Cerrar y Volver a Excel`.

4. Para evaluar la función $P(t)$ en los valores de t que están en la fila que inicia en F2, nos situamos en la celda F3 y se digita `+P(F2)`. Luego se arrastra hasta J2 para copiar la fórmula, quedando:

F3 = =+P(F2)

	A	B	C	D	E	F	G	H	I	J
1										
2	Población inicial	P0 =	100		t	0	5	10	15	20
3	Incremento	h =	5		P(t)	100	142.18925	198.2568985	269.487452	354.9789231
4		K =	1000							
5		*k* =	0.08							

Figure 1.14: Resultado final al evaluar la macro del modelo poblacional.

1.5 Elementos de programación en VBA

Un programa computacional escrito mediante cualquier lenguaje de programación puede verse a grandes rasgos como un flujo de datos, algunos jugando el papel de datos de entrada, otros son datos que cumplen alguna función temporal dentro del programa y otros son datos de salida. A lo largo del programa es muy frecuente que sea necesaria la entrada en acción de otros programas o procesos. A mayor complejidad del problema que resuelve el programa, mayor es la necesidad de programar por aparte algunos segmentos de instrucciones que se especializan en una tarea o conjunto de tareas.

Hay tres tipos de estructuras básicas que son muy utilizadas en la programación de un algoritmo, a saber, la estructura secuencial, la estructura condicional y la repetitiva.

A continuación se explica, con ejemplos programados como macros de Excel, estas estructuras. También se incluyen los programas en seudocódigo y diagramas de flujo para explicar de un modo más gráfico la lógica del programa. El uso de estos últimos es cada vez menor, pues el seudocódigo por lo general es suficientemente claro y se escribe en lenguaje muy cercano al lenguaje natural.

1.5.1 Flujo secuencial

El flujo secuencial consiste en seguir una secuencia de pasos que siguen un orden predeterminado.

Por ejemplo, un programa que a partir de un número N de días, calcula la cantidad de segundos que hay en esta cantidad de días. Este programa se puede ver como una secuencia de varios pasos:

- Inicio: Ingresa el número N de días
- Paso 1: H = 24*N, para determinar la cantidad de horas
- Paso 2: M = 60*H, para determinar la cantidad de minutos.
- Paso 3: S = 60*M, para determinar la cantidad de segundos.
- Paso 4: Retorne S.
- Fin.

La macro correspondiente a esta secuencia de cálculos puede escribirse como sigue:

```
Function CalculeSegundos(Dias)
    CantHoras = 24 * Dias
    CantMinutos = 60 * CantHoras
    CalculeSegundos = 60 * CantMinutos
End Function
```

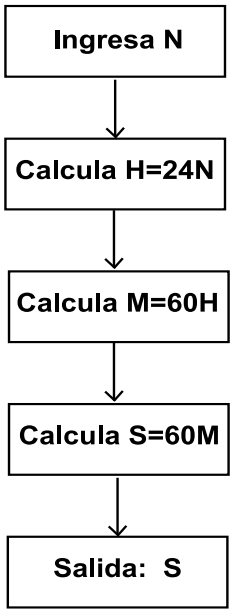


Figure 1.15: Flujo secuencial

1.5.2 Flujo condicional (`If - Else`)

Un flujo condicional se presenta en un programa o procedimiento que debe escoger una accón o proceso a ejecutar, dependiendo de condiciones que puedan cumplirse.

El caso más sencillo ocurre cuando el programa verifica si una condición se cumple y en caso de ser verdadera ejecuta un proceso, en tanto que si es falsa ejecuta otro proceso.

En VBA tenemos la instrucción

`If...Then...Else`

Ejecuta condicionalmente un grupo de instrucciones, dependiendo del valor de una expresión.

Sintaxis

`If` *condición* `Then`
instrucciones
`Else` *instrucciones-else*

Puede utilizar la siguiente sintaxis en formato de bloque:

`If` *condición* `Then`
instrucciones
`ElseIf` *condición* `Then`
instrucciones-elseif
...
`Else` *instrucciones-else*
`End If`

Nota: En la ayuda del editor de Visual Basic, tenemos acceso a la referencia del lenguaje.

Ejemplo 2

En este ejemplo veremos cómo usar la instrucción `If...Then...Else`

Obtener un programa que calcule aproximaciones de $\sqrt{2}$, sabiendo que la sucesión $\{x_n\}_{n\in\mathbb{N}}$ converge a $\sqrt{2}$, definida en forma recurrente mediante la relación:

$$\begin{cases} x_{n+1} &= \frac{1}{2}(x_n + \frac{2}{x_n}) \\ x_0 &= 1 \end{cases}$$

El programa deberá estimar el error absoluto de las aproximaciones y será capaz de escribir un mensaje de éxito o de fracaso, dependiendo de si el error absoluto es o no menor que una tolerancia dada.

Para los resultados que aparecen en la gráfica anterior pueden programarse las siguiente macros para ser evaluadas en cada columna:

```
Function AproxDeRaiz(x)
   AproxDeRaiz = (1 / 2) * (x + 2 / x)
End Function

Function CalculoElError(Aproximacion, ValorExacto)
   CalculoElError = Abs(Aproximacion - ValorExacto)
End Function
```

	A	B	C	D
1			Tolerancia:	0,000000001
2			**Error**	
3			**absoluto**	**Se aproximó**
4	**n**	x_n	E_n	**a la tolerancia**
5	0	1,000000000000000	0,414213562	FRACASO
6	1	1,500000000000000	0,085786438	FRACASO
7	2	1,416666666666670	0,002453104	FRACASO
8	3	1,414215686274510	2,1239E-06	FRACASO
9	4	1,414213562374690	1,59472E-12	FRACASO
10	5	1,414213562373090	2,22045E-16	EXITO

Figure 1.16: Resultado de la aproximación de $\sqrt{2}$.

```
Function verificaTol(Error, Tol)
   If (Error < Tol) Then
     verificaTol = "EXITO"
   Else
     verificaTol = "FRACASO"
   End If
End Function
```

El diagrama siguiente ilustra la forma en que esta última función de verificación actúa con base en el valor de sus dos parámetros de entrada:

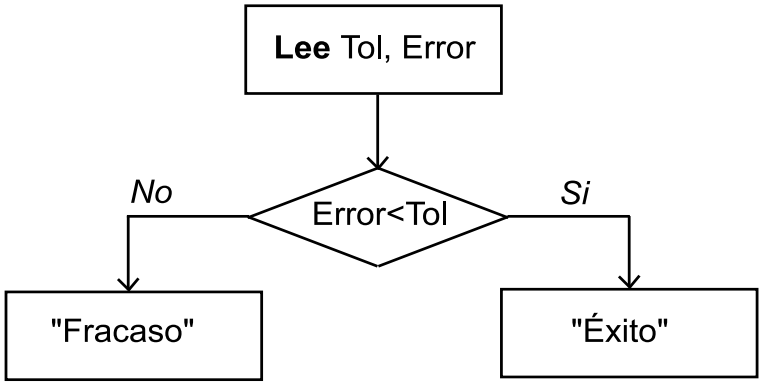


Figure 1.17: Diagrama de flujo condicional para verificar si se alcanzó la tolerancia.

1.5.3 Flujo repetitivo (`For-Next`, `While-Wend`, `Do While-Loop`)

El flujo repetitivo se presenta en un algoritmo cuando se requiere la ejecución de un proceso o parte de un proceso sucesivamente, hasta que ocurra una condición que permita terminar.

Este tipo de flujos repetitivos se presentan en tres formas que obedecen a maneras diferentes de razonarlos pero que en el fondo hacen lo mismo:

- Utilizar un contador que empiece en un número y termine en otro, ejecutando el proceso cada vez que el contador tome un valor distinto.

- Mientras una condición sea verdadera, ejecutar un proceso y regresar a la condición.
- Ejecutar un proceso, hasta que una condición deje de cumplirse.

En VBA tenemos las siguientes instrucciones para realizar procesos iterativos:

1. `For ... Next`

   Repite un grupo de instrucciones un número especificado de veces.

   Sintaxis (las instrucciones entre '[ ]' son instrucciones adicionales)

   `For` *contador* = *inicio* `To` *fin* [`Step` *incremento*]
   instrucciones
   `[Exit For]`
   instrucciones
   `Next` *contador*

2. `While...Wend`

   Ejecuta una serie de instrucciones mientras una condición dada sea True.

   Sintaxis

   `While` *condición*
   intrucciones
   `Wend`

   Nota: No hay un `Exit While`. En una subrutina, si fuera necesario, se podría usar `Exit Sub`

3. Una instrucción muy parecida a `While` pero más eficiente es `Do`

   Sintaxis

   `Do while` *condición*
   instrucciones
   `[Exit Do]`
   `Loop`

■ Ejemplo 3

Para ilustrar estas formas de realizar un flujo repetitivo, vamos a aproximar la suma de una serie alternada con un error estimado menor que una cantidad `tol` dada.

Consideremos la serie alternada

$$\sum_{k=1}^{\infty}(-1)^k\frac{1}{k^2} = -1 + \frac{1}{4} - \frac{1}{9} + \frac{1}{16} - \dots$$

La suma parcial N-ésima viene dada por

$$S_N = \sum_{k=1}^{N}(-1)^k\frac{1}{k^2} = -1 + \frac{1}{4} - \frac{1}{9} + \frac{1}{16} - \dots + (-1)^N\frac{1}{N^2}$$

es decir

$$\begin{array}{rcll} S_1 & = & -1 & = & -1 \\ S_2 & = & -1+1/4 & = & -0.75 \\ S_3 & = & -1+1/4-1/9 & = & -0.861111... \\ S_3 & = & -1+1/4-1/9+1/16 & = & -0.798611... \\ & & \vdots & & \end{array}$$

De acuerdo con la teoría de series alternadas, la serie $\sum_{k=1}^{\infty}(-1)^k\frac{1}{k^2}$ es convergente. Si su suma es S, al aproximarla con la suma parcial S_N, el error de la aproximación es menor que $\frac{1}{(N+1)^2}$, es decir

$$|S - S_N| \leq \frac{1}{(N+1)^2}$$

Primer problema

Dada una tolerancia `TOL`, calcular cada una de las sumas parciales hasta que el error de aproximación sea menor que TOL

Solución

	A	B	C	D	E	F	G	H
31				**Suma de la Serie**				
32		TOL	N	Suma-Parcial N	Error estimado	ya?		
33		0,01	1	-1	0,25	Error estimado > .01		
34			2	-0,75	0,111111111	Error estimado > .01		
35			3	-0,861111111	0,0625	Error estimado > .01		
36			4	-0,798611111	0,04	Error estimado > .01		
37			5	-0,838611111	0,027777778	Error estimado > .01		
38			6	-0,810833333	0,020408163	Error estimado > .01		
39			7	-0,831241497	0,015625	Error estimado > .01		
40			8	-0,815616497	0,012345679	Error estimado > .01		
41			100	-0,822417533	9,80296E-05	OK, error estimado < .01		

Figure 1.18: Sumas parciales y estimación del error

Implementamos dos macros, una para el cálculo de las sumas parciales y otra para hacer la verificación del error estimado. En este caso, vamos a suponer que `TOL` está en la celda `B33`

```
Function sumaParcial(hastaN) Dim Acum, signo As Integer Acum = 0
signo = -1

For k = 1 To hastaN
   Acum = Acum + signo * 1 / k ^ 2
    signo = -signo
Next k

sumaParcial = Acum End Function
'---------------------------------------------------------------------
Function verificaTol(elN, tol)
   If (1 / (elN + 1) ^ 2 > tol) Then
     verificaTol = "Error estimado > " + Str(tol)    'tol es un número
   Else                                              'no una String
     verificaTol = "OK, error estimado <" + Str(tol) 'por esto debemos usar 'Str'
   End If
End Function
```

En la primera llamada de las macros se usó `sumaParcial(C33)` y `verificaTol(C33;B$33)`

Segundo problema

Dada una lista de tolerancias TOL (donde TOL es una cantidad positiva, como 10^{-1},10^{-8}, etc.), aproximar la suma de la serie con una cantidad N de términos lo suficientemente grande de tal manera que $\frac{1}{(N+1)^2} < TOL$. La cantidad $\frac{1}{(N+1)^2}$ juega en este problema el papel de una cota del error, al aproximar la serie correspondiente hasta el término N-ésimo

1. **Primera solución**:

   Dado que hay que sumar hasta el término N-ésimo tal que $\frac{1}{(N+1)^2} < TOL$, en este caso es posible despejar el entero positivo N, quedando:

   $$N > \sqrt{\frac{1}{TOL}} - 1$$

   Tomamos N como la parte entera superior de $\sqrt{\frac{1}{TOL}} - 1$, por lo que se calcula:

   $$N = \left\lfloor \sqrt{\frac{1}{TOL}} \right\rfloor \quad \text{(parte entera)}$$

   Los pasos a seguir para programar la suma a partir de la tolerancia dada, son los siguientes:

   - Inicio: Ingresa la tolerancia con que se hará la aproximación.

- Paso 1: Calcular $N = \left\lfloor \sqrt{\frac{1}{TOL}} \right\rfloor$
- Paso 2: Acum = 0 (Se iniciliza el acumulador para la suma).
- Paso 3: Para k = 1 ... N:

  $$\text{Acum} = \text{Acum} + (-1)^k \frac{1}{k^2}$$
- Paso 4: Retorne Acum.
- Fin.

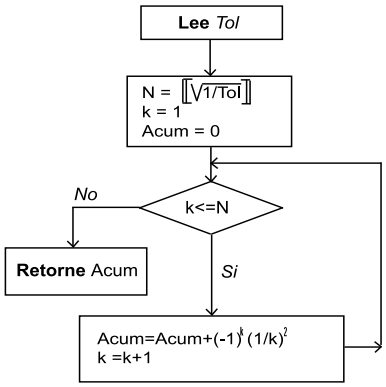


Figure 1.19: Diagrama de flujo correspondiente a la primera solución.

Observe que en cada sumando, se incluye el factor $(-1)^k$, que en la práctica, más que una potencia, lo que indica es el cambio de signo en los términos sucesivos. Para evitarle al programa cálculos innecesarios, podemos inicializar una variable *signo* en -1 y en cada paso de la instrucción repetitiva se cambia al signo contrario. La macro correspondiente a este programa puede escribirse como sigue:

```
Function SumaParcial(Tol)
    Acum = 0 signo = -1
    N = Int(1 / Sqr(Tol))
    For k = 1 To N
        Acum = Acum + signo * 1 / k ^ 2
        signo = -signo
    Next k
    SumaParcial = Acum
End Function
```

La siguiente figura muestra la evaluación de esta macro para algunos valores de la tolerancia.

2. **Segunda solución:**

En esta solución no es necesario calcular el valor de N, sino que se suman los términos mientras no se haya alcanzado la tolerancia. El programa en seudocódigo se puede escribir como sigue:

- Inicio: Ingresa la tolerancia con que se hará la aproximación.
- Paso 1: Iniciar con $N = 1$.
- Paso 2: Acum = -1 (Se iniciliza el acumulador para la suma con el primer término).
- Paso 3: Mientras $\frac{1}{(N+1)^2} > Tol$:

	A	B	C	D	E
1					
2		Aproximación de una serie alternada			
3	Primera solución: USANDO FOR K=1 TO N				
4		**Tol**	**N**	**Suma Parcial**	**Error estimado**
5		1	1	-1,00000000000000	0,25
6		0,1	3	-0,86111111111111	0,0625
7		0,01	10	-0,82796217561099	0,008264463
8		0,001	31	-0,82297055860543	0,000976563
9		0,0001	100	-0,82251753337413	9,80296E-05
10		0,00001	316	-0,82246204205917	9,95134E-06
11		0,000001	1000	-0,82246753392411	9,98003E-07
12		0,0000001	3162	-0,82246698343114	9,99543E-08
13		0,00000001	10000	-0,82246703842461	9,998E-09
14		1E-09	31622	-0,82246703292409	9,99986E-10
15		1E-10	100000	-0,82246703347410	9,9998E-11

Figure 1.20: Resultados calculados mediante la primera solución.

. $N = N + 1$

. $\text{Acum} = \text{Acum} + (-1)^N \dfrac{1}{N^2}$

- Paso 4: Retorne Acum.
- Fin.

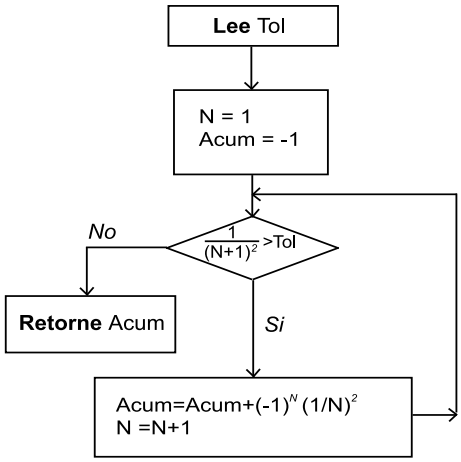


Figure 1.21: Diagrama de flujo correspondiente a la segunda solución.

El código es

```
Function SumaParcial2(Tol)
    N = 1 Acum = -1 signo = 1
    While (1 / (N + 1) ^ 2 > Tol)
          N = N + 1
          Acum = Acum + signo * 1 / N ^ 2
          signo = -signo
    Wend
    SumaParcial2 = Acum
End Function
```

1.5.4 Manejo de rangos

Una rango en Excel corresponde a una selección de celdas. Una *selección* de las celdas de una fila o una columna se maneja en Excel como una matriz de orden $1 \times n$ o de orden $n \times 1$ (un vector). La *selección* de un bloque de celdas se maneja como una matriz $n \times m$. Si una celda está en blanco, se lee un cero.

Ejemplo 4

Promedio simple. Consideremos una tabla con 5 notas, todas con igual peso.

	A	B	C	D	E	F	G	H
		Promedio Simple						
		Nombre	N1	N2	N3	N4	N5	Promedio
		Pedro Pérez	40	85	90	80	30	65
		Ana Pereira	55	66	77	89	100	77,4
		Victoria Ching	100	100	0	100	90	78

Figure 1.22: Promedio simple.

Para calcular el promedio simple, en cada fila, vamos a hacer una macro que recibe un rango, cuenta las notas, suma y divide entre el número de notas.

```
Function PromedioSimple(R As Range) As Double  'R es la variable
que recibe el rango
    Dim n As Integer
    Dim sump As Double

    sump = 0
    n = R.EntireColumn.Count                   'cantidad de notas en el rango
    For Each x In R                            'suma de las notas
        sump = sump + x
    Next x
    Sume = sump / n                              'promedio simple
End Function
```

En primera celda de la columna `Promedio`, llamamos a la macro con: `PROMEDIO(C52:G52)` pues en este caso el rango es `C52:G52`.

Ejemplo 5

El Promedio eliminando las dos notas más bajas. En este caso, a un conjunto de notas les calculamos el promedio simple pero eliminando las dos notas más bajas. El programa `PromedioQ` suma las n notas de una fila (rango), localiza la posición (en el vector `R`) de las dos notas más bajas y luego le resta a la suma estas dos notas para luego dividir entre $n-2$. En este caso, el rango `R` es una matriz $1 \times n$, o sea, se puede ver como un vector de n componentes.

```
Function PromedioQ(R As Range) As Double

    Dim n, i, Imin1, Imin2 As Integer
```

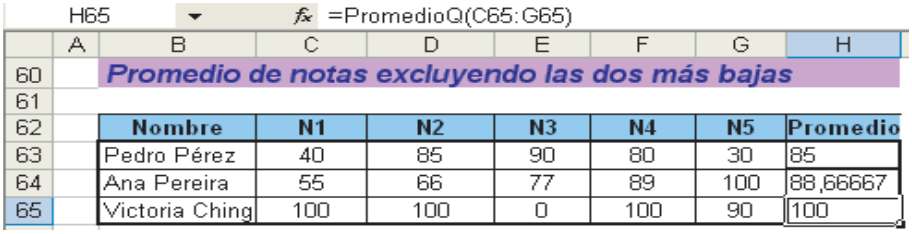


Figure 1.23: Promedio de notas, eliminando las dos más bajas.

```
    Dim suma As Double

    suma = 0
    n = R.EntireColumn.Count 'número de elementos de la selección

    For i = 1 To n
      suma = suma + R(1, i)  'R es una matriz 1xn (o sea, un vector)
    Next i                   'En R no se hace referencia a la celda

    Imin1 = 1                'POSICION de la 1ra nota mínima en R
    For i = 1 To n
      If R(1, i) < R(1, Imin1) Then
        Imin1 = i
      End If
    Next i

    Imin2 = 1                'POSICION de la segunda nota mínima en R
    If Imin1 = 1 Then
       Imin2 = 2
    End If

    'comparar con todos excepto Imini1
    For i = 1 To n
    If (R(1, i) < R(1, Imin2)) And (i <> Imin1) Then
        Imin2 = i
    End If
    Next i

    PromedioQ = (suma - R(1, Imin1) - R(1, Imin2)) / (n - 2)
End Function
```

Nota: También podríamos resolver este problema usando `Selection.Sort` pero la programación es un poco más compleja.

1.5.5 Subrutinas. Edición y ejecución de una subrutina

La subrutinas o procedimientos es otro de los tipos básicos de programas en Visual Basic. Una descripción de la sintaxis de una subrutina que no es completa, pero sí suficiente para los alcances de este material es la siguiente[2]:

[2]Aparte de la posibilidad de declarar la subrutina como *Private* o *Public*, también se puede declarar como *Friend*, pero esto tiene que ver más con la *programación orientada a objetos*, que no se aborda en el presente material

Sintaxis:

`Sub` Nombre-de-Subrutina(*lista-argumentos*)

instrucciones

`End Sub`

o también

`[Private | Public] [Static] Sub` Nombre-de-Subrutina(*lista-argumentos*)

instrucciones

`End Sub`

Lass partes entre corchetes indican que son opcionales. Además:

Public. Es opcional. Indica que la subrutina puede ser llamada por todas las demás subrutinas sin importar donde se encuentre.

Private. Es opcional. Indica que la subrutina puede ser llamada solamente por otras subrutinas que se encuentren en el mismo módulo.

Static. Es opcional. Indica que las variables locales de la subrutina se mantienen constantes de una llamada a otra. El ámbito de acción de esta declaración no incluye a variables declaradas fuera de la subrutina.

Nombre-De-Subrutina. Es requerido. Indica el nombre de la subrutina.

lista-argumentos. Es opcional e indica las variables que conforman los argumentos con que una subrutina es llamada. Para separar una variable de otra se escribe una coma.

instrucciones. Es opcional y conforma el conjunto de instrucciones que son ejecutadas a lo largo de la subrutina.

■ Ejemplo 6

Elevar al cuadrado los valores de una selección (ejecutar desde la ventana de ejecución de macros).

Podemos implementar una subrutina en una hoja, que recorra una *selección* hecha con el mouse y que vaya elevando al cuadrado el valor de cada celda.

	A	B	C
1			
2	-1	2	9
3	-2	3	10
4	-3	4	11
5	-4	5	12
6	-5	6	
7	-6	7	
8	-7	8	
9	-9		
10	-10		

→

	A	B	C
1			
2	1	4	81
3	4	9	100
4	9	16	121
5	16	25	144
6	25	36	0
7	36	49	0
8	49	64	0
9	81	0	0
10	100	0	0

Figure 1.24: Elevar al cuadrado los elementos de la selección

```
Sub elevAlCuadado()
    For Each cell In Selection.Cells  'para cada celda de la selección
        cell.Value = cell.Value ^ 2   'recalcula el valor de la celda
    Next cell
End Sub
```

Nota: La macro se aplica a los datos que están actualmente seleccionados

- Para editar la subrutina, vamos al editor VB (`Alt-F11`) y hacemos doble-clic sobre `(Hoja1)`

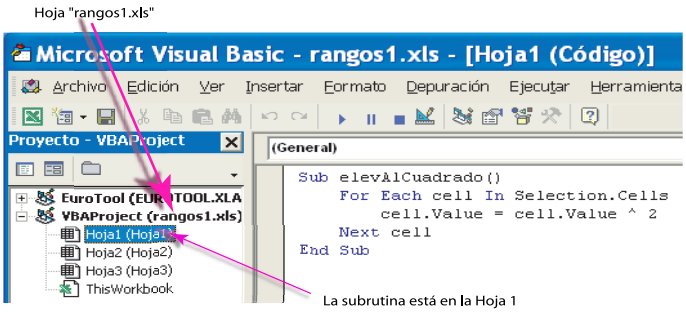


Figure 1.25: Edición de la subrutina en Hoja 1

Escribimos el código, compilamos (en menú `Depuración`), guardamos y nos devolvemos a la hoja.

- Para ejecutar la macro seleccionamos la tabla con el mouse y levantamos la ventana de ejecución de macros (`Alt-F8`)y damos clic en 'Ejecutar'

Nota: Esta subrutina también se puede editar en un módulo. Para ejecutarla se procede de la misma forma.

1.5.6 Ejecución de una subrutina mediante un botón

Otra posibilidad bastante práctica para ejecutar un programa o subrutina como los presentados en la sección precedente es mediante un botón de comando.

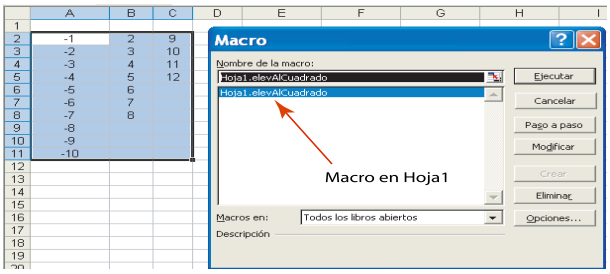


Figure 1.26: Ejecutar macro

Ejemplo 7

Elevar al cuadrado los valores de una selección.

1. Primero digitarmos la tabla de valores. Luego insertamos un botón. Para esto seleccionamos un botón del cuadro de controles (si la barra no está disponible, puede habilitarla con `Ver - Barra de herramientas - Cuadro de Controles`).

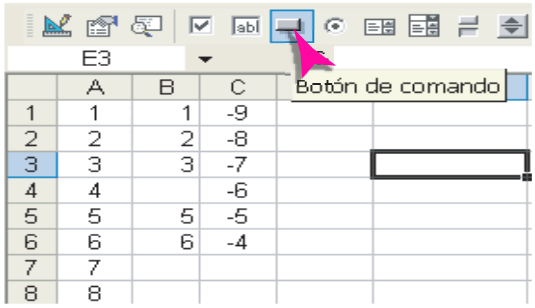


Figure 1.27: Insertar un botón

2. Luego *hacemos clic en el lugar de la hoja donde queremos el botón.* Una vez que tenemos el botón, podemos agregar algunas propiedades como etiqueta, color de fondo, etc., en el menú de contexto. Este menú se abre con `clic derecho + propiedades`. Luego cerramos el menú

3. Para editar el código que deberá ejecutar el botón, le damos un par de clics al botón (que todavía está en modo diseño). En este caso, si es la primera vez, nos aparece el código

```
Private Sub CommandButton1_Click() End Sub
```

Aquí tenemos dos opciones

a.) Implementar la subrutina por separado y luego llamarla desde la subrutina del botón

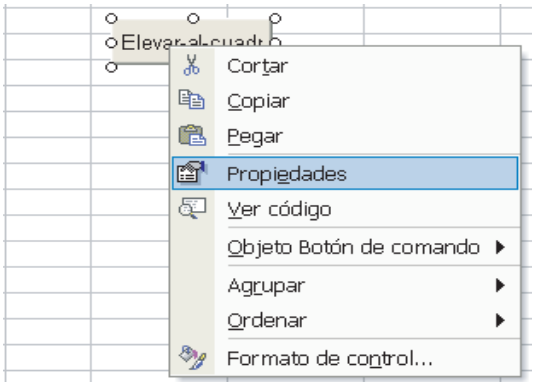


Figure 1.28: Menú de contexto para un botón

	A	B	C	D	E	F
1	1	1	-9			
2	2	2	-8			
3	3	3	-7			
4	4		-6			
5	5	5	-5			
6	6	6	-4			
7	7					
8	8					

Elevar-al-cuadrado

Figure 1.29: Tabla y botón

```
 Sub elevAlCuadado()
    For Each cell In Selection.Cells  'para cada celda de la selección
        cell.Value = cell.Value ^ 2   'recalcula el valor de la celda
    Next cell
End Sub

'------------------------------------------------------------------------
Private Sub CommandButton1_Click()
    elevAlCuadado    'sin paréntesis
End Sub
'------------------------------------------------------------------------
```

b.) Incluir en la subrutina del botón lo que vamos a querer que se ejecute cuando se haga clic sobre él

```
Private Sub CommandButton1_Click()
     For Each cell In Selection.Cells  'para cada celda de la selección
        cell.Value = cell.Value ^ 2    'recalcula el valor de la celda
    Next cell
End Sub
```

4. Una vez que escogemos alguna de las dos opciones anteriores, compilamos (en menú `Depurar`), guardamos y nos devolvemos a la hoja.

5. Para habilitar el botón debemos deshabiltar el ícono de diseño  .

6. Ahora solo resta *seleccionar* la tabla y hacer clic sobre el botón.

   Observe que al dar un clic sobre el botón, el programa opera sobre lo que *tengamos seleccionado* previamente

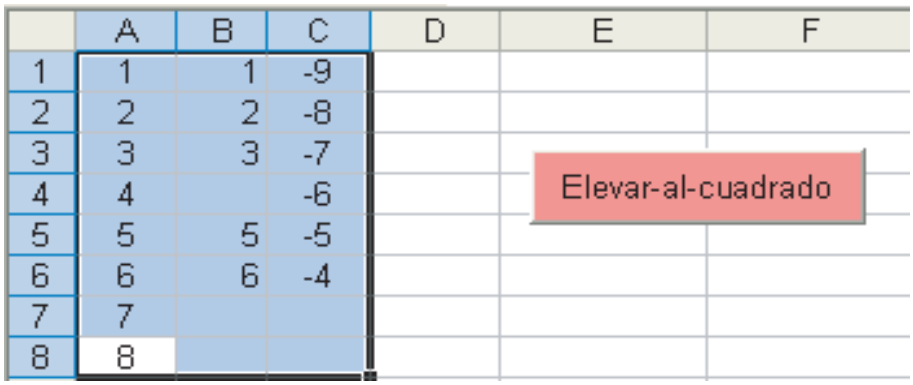


Figure 1.30: Correr una subrutina desde un botón con la tabla seleccionada

Nota: Puede ser importante conocer la primera o la última celda de una selección. Para esto podemos usar la propiedad `Address`

```
Set R = Selection
  adr = R(1, 1).Address  'primera celda de la selección, por ejemplo $A$1
  num = Right(adr, 1)    'primera posición a la derecha de adr (1 en este caso)
```

1.5.7 Matrices dinámicas

Cuando hacemos una *selección* con el mouse, es conveniente entrar los valores seleccionados en una matriz dinámica, es decir, una matriz que se ajuste a la cantidad de datos seleccionada y que, eventualmente, se pueda recortar o hacer más grande.

Una matriz dinámica `mtr1` de entradas enteras se declara así:

```
Dim mtr1() As Integer    ' Declara una matriz dinámica.
```

Las instrucciones siguientes cambian el tamaño de la matriz `mtr1` y la inicializa. Observe el uso de `Redim` para cambiar el tamaño de la matriz dinámica.

```
Dim   mtr1() As Integer    ' Declara una matriz dinámica.
Dim   r() as Double
Redim  mtr1(10)            ' Cambia el tamaño a 10 elementos, 1x10.
Redim r(n,m)               ' Cambia tamaño a   n x m

For i = 1 To 10          ' Bucle 10 veces.
    mtr1(i) = i          ' Inicializa la matriz.
Next i
```

Usando `Preserve` se puede cambiar el tamaño de la matriz `mtr1` pero sin borrar los elementos anteriores.

```
Redim Preserve mtr1(15)    ' Cambia el tamaño a 15 elementos.
```

Ejemplo 8

Centro de gravedad de un conjunto de puntos en $\mathbb{R}^2$

Consideremos un conjunto $\Omega = \{(x_i, y_i) \ / \ i = 1, 2, \cdots, n, \ \ x_i, y_i \in \mathbb{R}\}$. Supongamos que a cada punto (x_i, y_i) se le asigna un peso p_i tal que $\sum_{i=1}^{n} p_i = 1$. El centro de gravedad G_Ω de Ω se define así

$$G_\Omega = \sum_{i=1}^{n} p_i \, (x_i, y_i)$$

Por ejemplo, si tenemos dos puntos $A, B \in \mathbb{R}^2$ con igual peso (este deberá ser 1/2 para cada punto), entonces el centro de gravedad es el punto medio del segmento que une A con B

$$G_\Omega = \frac{A+B}{2}$$

La subrutina que calcula el centro de gravedad de un conjunto de puntos $\Omega \subset \mathbb{R}^2$ actúa sobre un rango de tres columnas en el que se han escrito las coordenadas (x, y) de dichos puntos, así como sus pesos. Podemos correr el programa una vez que se ha *seleccionado* el rango completo en el que se ubican los puntos y sus pesos, haciendo clic sobre un botón "Centro Gravedad". El gráfico es un trabajo adicional.

Como el programa necesita que el usuario haya seleccionado al menos dos filas y exactamente tres columnas, incluimos un fragmento de código adicional para controlar la selección.

```
Set R = Selection n = R.Rows.Count     ' Número de filas m =
R.Columns.Count  ' Número de columnas

If n > 1 And m = 3 Then
       'todo está bien, el programa continúa
Else
      MsgBox ("Debe seleccionar los datos")
      Exit Sub           ' salimos de la subrutina
End If
```

Observemos que si no se cumple el requisito, se envía un mensaje y la subrutina se deja de ejecutar. Puede causar curiosidad que que antes del `Else` no haya código. A veces se usa esta construcción por comodidad. Si no hay código el programa continúa.

Veamos el código completo en la subrutina del botón

```
Private Sub CommandButton2_Click()
    Dim R As Range
    Dim n,m, i As Integer
    Dim x() As Double   ' matriz dinámica
    Dim y() As Double   ' se ajustará a la selección de datos
    Dim p() As Double
    Dim Sumapesos, Gx, Gy As Double

        'En el rango R guardamos el rango seleccionado:
    Set R = Selection
    n = R.Rows.Count     ' Número de filas
    m = R.Columns.Count  ' Número de columnas
                         ' chequear que se hayan seleccionado los datos de la tabla
    If n > 1 And m = 3 Then
       'nada pasa, todo bien
    Else
      MsgBox ("Debe seleccionar los datos")
```

```
      Exit Sub           ' salimos de la subrutina
    End If

    ReDim x(n)        ' vector x tiene ahora n campos
    ReDim y(n)
    ReDim p(n)

    Sumapesos = 0     'inicializa para acumular
    Gx = 0
    Gy = 0
                      'inicializa las matrices con los datos
    If n > 1 Then     'si hay datos seleccionados, n > 1

        For i = 1 To n
            x(i) = R(i, 1) 'entra los datos de columna de los x's
            y(i) = R(i, 2)
            p(i) = R(i, 3)
                           'calcula centro de gravedad
            Sumapesos = Sumapesos + p(i)
            Gx = Gx + x(i) * p(i)
            Gy = Gy + y(i) * p(i)
        Next i
        If Abs(Sumapesos - 1) < 0.001 Then  'la suma los pesos debe ser 1
            Gx = Gx / Sumapesos
            Gy = Gy / Sumapesos
                               'escribe G en la celda D14
            Cells(14, 4) = "G = (" + Str(Gx) + "," + Str(Gy) + ")"
            Cells(14, 5) = ""  'limpia E14 de mensajes previos
        Else
                               'mensaje de error
            Cells(14, 5) = "Error, los pesos  suman " + Str(Sumapesos)
            Cells(14, 4) = ""  'limpia D14 de valores previos
        End If
    Else
        Cells(14, 4) = ""
                               'ventana de advertencia: seleccione datos!
        MsgBox ("Debe seleccionar los datos")
        Exit Sub               ' aborta la subrutina si no hay datos
    End If
End Sub
```

Nota: Por cuestiones de redondeo, la instrucción `If Sumapesos = 1` se cambió por `If Abs(Sumapesos - 1) < 0.001` .

Un ejemplo de corrida se ve en la figura que sigue

Ejercicios 1

1. Usando la notación del último ejemplo, se define la *inercia total* de Ω como

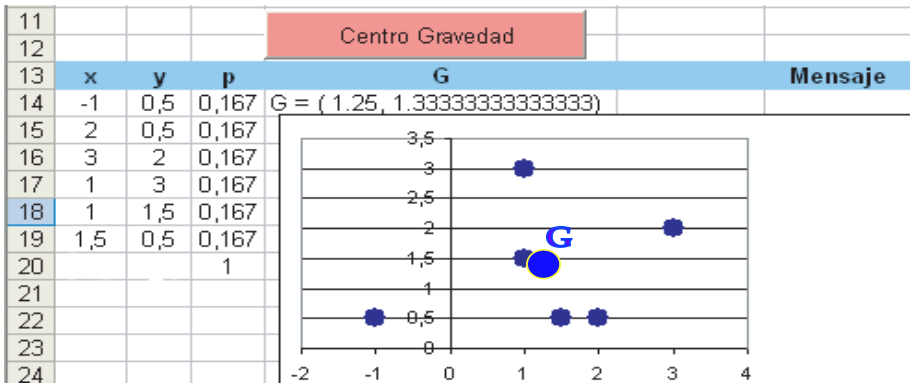


Figure 1.31: Correr una subrutina desde un botón

$$I(\Omega) = \sum_{i=1}^{n} p_i \,||\,(x_i, y_i) - G_\Omega\,|\,|^2$$

donde $||(a,b)|| = \sqrt{a^2+b^2}$ es la norma usual.

Implemente una subrutina que calcula la Inercia Total de Ω y aplíquela a la tabla del ejemplo 8.

2. Si tenemos $n+1$ puntos $\{(x_0,y_0)\,,\,(x_1,y_1)\,,\,(x_2,y_2)\,,\,\cdots\,(x_n,y_n)\,\}$, se define

$$L_{in}(x) = \frac{(x-x_0)(x-x_1)\cdots(x-x_{i-1})(x-x_{i+1})\cdots(x-x_n)}{(x_i-x_0)(x_i-x_1)\cdots(x_i-x_{i-1})(x_i-x_{i+1})\cdots(x_i-x_n)}$$

O sea, en $L_{in}(x)$ se elimina el factor $(x-x_i)$ en el numerador y el factor (x_i-x_i) en el denominador

Implemente una hoja, como se ve en la figura, en la que el usuario hace una selección de la tabla y al hacer clic en el botón, se calcula $L_{2n}(2.56)$

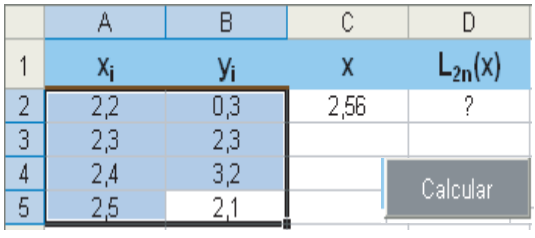


Figure 1.32: $L_{2n}(2.56)$

Parte del código sería

```
...
Set R = Selection
n = R.Rows.Count  ' Número de filas
ReDim x(n)        ' vector x tiene ahora n campos
ReDim y(n)
valorX = Cells(2, 3) ' valorX está en celda C2
If n > 1 Then        'si hay datos seleccionados, n > 1
    For i = 1 To n
        x(i) = R(i, 1) 'entra los datos de columna de los xi's y los yi's, inicia en 1
        y(i) = R(i, 2) 'aquí, iniciamos desde x1, es decir el x0 de la teoría, es x1
    Next i

        L2n= 1        'inicia cálculo de L2n(variable x)
        For j = 1 To n 'calculamos L2n(valorX)
            If j <> 2 Then
                L2n = L2n * (valorX - x(j)) / (x(k) - x(j))  'L2n evaluado en valorX
            End If
        Next j
    Cells(2, 4) = L2n
Else
    MsgBox ("Debe seleccionar los datos")
    Exit Sub               ' aborta la subrutina si no hay datos seleccionados
End If
...
```

3. Si tenemos $n+1$ puntos $\{(x_0, y_0), (x_1, y_1), (x_2, y_2), \cdots (x_n, y_n)\}$, un polinomio que pasa por todos estos puntos se llama *Polinomio Interpolante*. Un polinomio interpolante muy conocido es el Polinomio de Lagrange

$$P(x) = \sum_{i=0}^{n} \prod_{j=0, j\neq i}^{n} \frac{(x - x_j)}{(x_i - x_1)} y_i$$

o sea

$$P(x) = \sum_{i=0}^{n} \frac{(x - x_0)(x - x_1)\cdots(x - x_{i-1})(x - x_{i+1})\cdots(x - x_n)}{(x_i - x_0)(x_i - x_1)\cdots(x_i - x_{i-1})(x_i - x_{i+1})\cdots(x_i - x_n)} y_i$$

O sea, para cada valor de i se elimina el factor $(x - x_i)$ en el numerador y el factor $(x_i - x_i)$ en el denominador

Este polinomio cumple $P(x_i) = y_i \quad i = 1, 2, ...n$.

Por ejemplo, para el caso de tres puntos (x_0, y_0), (x_1, y_1) y (x_2, y_2), el Polinomio de Lagrange es

$$P(x) = \frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)}y_0 + \frac{(x-x_0)(x-x_2)}{(x_1-x_0)(x_1-x_2)}y_1 + \frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)}y_2$$

(a) Implemente una subrutina que, a partir de una *selección* de puntos (x_i, y_i) (en dos columnas) en una hoja de Excel, *evalúa* el polinomio interpolante en una valor dado de antemano en una celda, o sea, calcula $P(a)$ para un valor a dado.

(b) Aplique la implementación anterior a una tabla como la que se presenta en la figura

Polinomio de Lagrange				
x_i	**y_i**	a	P(a)	
-1	0,5	3,5	?	
2	0,5			
3	2			
4	3			
5	1,5			
6	4,6			

Figure 1.33: Polinomio de Lagrange

Parte del código sería

```
...
suma = 0          'inicializa para acumular
If n > 1 Then     'si hay datos seleccionados, n > 1
    For i = 1 To n
        x(i) = R(i, 1) 'entra los datos de columna de los xi's y los yi's, inicia en 1
        y(i) = R(i, 2) 'aquí, iniciamos desde x1, es decir el x0 de la teoría, es x1
    Next i

    For k = 1 To n
        Lkn = 1         'inicia cálculo de Lkn
        For j = 1 To n 'calculamos Lkn(valorX)
            If j <> k Then
                Lkn = Lkn * (valorX - x(j)) / (x(k) - x(j)) 'Lkn evaluado en valorX
            End If
        Next j
        suma = suma + Lkn * y(k)
    Next k
    Cells(5, 4) = suma
Else
                          'ventana de advertencia: seleccione datos!
    MsgBox ("Debe seleccionar los datos")
    Exit Sub              ' aborta la subrutina si no hay datos seleccionados
End If
...
```

1.5.8 Inclusión de procedimientos de borrado

En ocasiones es necesario borrar alguna información que ha sido escrita en una hoja electrónica, por lo que es importante conocer una forma de incorporar en la aplicación un procedimiento de borrado.

Ejemplo 9

Presentamos a continuación un programa que, a partir de un número N construye el triángulo de Pascal de N niveles. También se incluye un programa que funciona como borrador o destructor del triángulo.

En cada nivel del triángulo hay un uno en los extremos y, a partir del tercer nivel, cada número, salvo los extremos, es la suma de los dos de arriba. Concretamente, el triángulo de Pascal es un arreglo triangular de números de la forma:

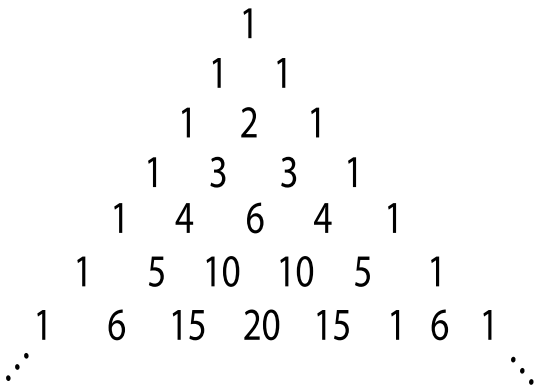


Figure 1.34: Triángulo de Pascal.

Por simplicidad, presentamos un programa que lee el número de niveles del triángulo de Pascal en la celda E1 y lo despliega de la forma:

```
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
...
```

El procedimiento para borrar el triángulo también lee el número de niveles del triángulo de la celda E1.

Tanto la construcción del triángulo como su destrucción pueden ser activados mediante botones. Las instrucciones que realizan estos procedimientos también pueden ser incluidas directamente en el código de los botones, tal y como se detalla a continuación.

El código para la construcción del triángulo quedaría así:

```
Private Sub EjecucionDePascal_Click()
```

```
   ' Lectura de la cantidad de niveles:
     N = Cells(1,5)
   ' Llenar unos:
     For i = 1 To N
         Cells(i,1)= 1
         Cells(i,i)= 1
     Next i
   ' Llenar el resto:
     If N > 2 Then
         For i=3 To N
              For j=2 To i-1
                  Cells(i,j)= Cells(i-1,j) + Cells(i-1,j-1)
              Next j
         Next i
     End If
End Sub
```

El procedimiento para borrar el triángulo también lee el número de niveles y hace el mismo recorrido de celdas que hizo el constructor y en cada celda escribe un valor nulo.

```
Private Sub Borrador_Click()
     N = Cells(1, 5).Value
     For i = 1 To N
         For j = 1 To i
              Cells(i, j).Value = Null
         Next j
     Next i
End Sub
```

	A	B	C	D	E	F	G	H	I	J
1	1				9					
2	1	1								
3	1	2	1							
4	1	3	3	1						
5	1	4	6	4	1					
6	1	5	10	10	5	1				
7	1	6	15	20	15	6	1			
8	1	7	21	35	35	21	7	1		
9	1	8	28	56	70	56	28	8	1	
10										

Construir

Borrar

Figure 1.35: Triángulo de Pascal construido con 10 niveles.

Ejercicios 2

1. **Triángulo de Pascal.** Haga un programa que, al activarlo desde un botón de comando, lea un un número N de la celda A1 y construya el triángulo de Pascal en la forma:

```
             1
           1 1
         1 2 1
       1 3 3 1
     1 4 6 4 1
   1 5 10 10 5 1 ...
```

Además, incluya un botón que active un destructor especializado para este triángulo.

2. **Aritmética entera**. En VBA hay dos tipos de división: `a/b` es la división corriente (en punto flotante) y `a\b` que es la división entera. Por ejemplo, `3/2 = 1.5` mientras que `3\2 = 1`.

   Hay muchos algoritmos en los que se usa exclusivamente divisón entera. Veamos un ejemplo.

   Si N es un entero positivo, vamos a calcular el entero más grande que es menor o igual a $\sqrt{N}$, es decir $\lfloor\sqrt{N}\rfloor$ (' $\lfloor\ \rfloor$ ' es la parte entera). El algoritmo, que opera en aritmética entera, es el siguiente

   a.) Inicio: $s_0 = \frac{N}{2}$

   b.) $s_{i+1} = \dfrac{s_i + \dfrac{N}{s_i}}{2}, \quad i = 0, 1, 2, ...$

   c.) iterar hasta que $s_{i+1} \geq s_i$

   d.) el último s_i es $\lfloor\sqrt{N}\rfloor$

   Por ejemplo si $N = 10$, $\sqrt{N} \approx 3,16227766$ y el último s_i sería 3

   Implemente el algoritmo.

3. **Cálculo de la raíz cuadrada**. Para calcular $\sqrt{x}$ con $x \in \mathbb{R}^+$, se pueden usar varios algoritmos dependiendo de la precisión y rapidez que se busca. La calculadoras usualmente tienen implementadas muy buenas subrutinas para calcular exponenciales y logaritmos por lo que para calcular $\sqrt{x}$ usan la identidad $\sqrt{x} = e^{\frac{1}{2}\ln(x)}$

   En nuestro caso, por simplicidad, vamos a usar un método iterativo: el método de Newton. Bajo ciertas hipótesis, si x_0 es una buena aproximación a una solución r de la ecuación $P(x) = 0$ entonces el esquema iterativo

   $$x_{i+1} = x_i - \frac{P(x_i)}{P'(x_i)}$$

   converge a r con error $\leq |x_i - x_{i+1}|$.

   Para hallar $\sqrt{U}$ con $U > 0$, vamos a resolver la ecuación $\dfrac{1}{x^2} - \dfrac{1}{U} = 0$ con el método de Newton. Esto nos dará una aproximación a $\sqrt{U}$.

   Nota: Como U es una constante, el esquema iterativo se podría simplificar un poco más en el código

   Nota: También se pudo haber usado la ecuación $x^2 - U = 0$ pero el proceso de aproximación es un poco más lento.

Nota: Para una aproximación inicial U_0 se podría usar $\lfloor\sqrt{U}\rfloor$ usando la solución del problema anterior. En la práctica se usa una aproximación inicial basada en la representación binaria del número.

4. **Números primos.** Obtenga un programa para hallar el número primo más cercano a un número N, siendo N un número entero positivo mayor que uno.[3]

   Por ejemplo, los números primos menores que 50 son:

   $$2,\ 3,\ 5,\ 7,\ 19,\ 23,\ 29,\ 31,\ 37,\ 41,\ 43.$$

   Si $N = 50$, el programa debería retornar el número 43.

   Un procedimiento clásico para hallar todos los números primos menores que un entero positivo N, es la llamada *criba de Eratóstenes*. Lo que se hace es colocar en una lista todos los números del 2 al N e ir eliminando de esta lista todos los múltiplos de 2 (4, 6, 8, ...), todos los múltiplos de 3 (6, 9, 12, ...), y así sucesivamente, hasta eliminar todos los múltiplos de los primos que han ido quedando en la lista menores o iguales que $\sqrt{N}$. Para decidir si un número es múltiplo de otro, usamos la función `mod`. Esta función devuelve el resto de una división (entera). Así, si un número es múltiplo de otro, la división es exacta, o sea, el resto es cero. En código sería así

```
If m mod n = 0 Then  'si m es múltiplo de n .... End If
```

1.6 Evaluando expresiones matemáticas escritas en lenguaje matemático común

Un evaluador de expresiones matemáticas o un "parser", es un programa que determina la estructura gramatical de una frase en un lenguaje. Este es el primer paso para determinar el significado de una frase tal como `"x^2+y"`, que para un lenguaje de programación significa traducirla en lenguaje de máquina.

En [9] encontrará un "parser" para evaluar expresiones matemática en Visual Basic. Sin embargo funciona bien con Excel. Además nos da la posibilidad de agregar funciones propias más complejas además de la que el parser trae implementadas tales como `BesselJ(x,n)`, `HypGeom(x,a,b,c)` o `WAVE_RING()`. Por ejemplo, la función a trozos. Este parser fue desarrollado por Leonardo Volpi [9].

1.6.1 Usando clsMathParser. Sintaxis

Primero descargamos `clsMathParser.zip`. En la carpeta `clsMathParser`, además de la documentación en pdf, vienen dos archivos: `clsMathParser.cls` y `mMathSpecFun.bas`. El primero es el parser y el segundo es una biblioteca con funciones especiales ya implementadas.

Para implementar un ejemplo de uso del parser, implementamos una hoja excel como la que se ve en la figura

[3] Un número natural $p > 1$, es *primo* si sus únicos divisores positivos son 1 y p. En caso contrario, es un número *compuesto.*

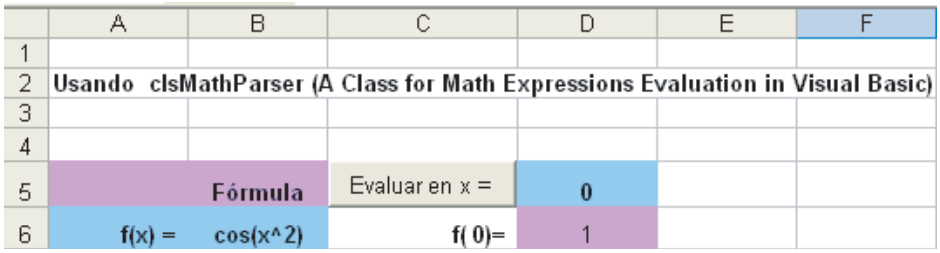


Figure 1.36: Usando clsMathParser.

1. En el editor de VBA, seleccionamos la "Hoja1" y hacemos clic en el botón derecho del mouse. Ahí elegimos `Importar Archivo...`. Luego vamos a la carpeta `clsMathParser` y seleccionamos `clsMathParser.cls`

2. De la misma manera, importamos el archivo `mMathSpecFun.bas`.

   Comentario. Con esto ya podemos crear un objeto `Fun` con `Dim Fun As New clsMathParser1` en el código del botón. Las evaluaciones se llevan a cabo con `Fun.Eval1(xval)`

3. En el editor VBA deberá quedar algo parecido a

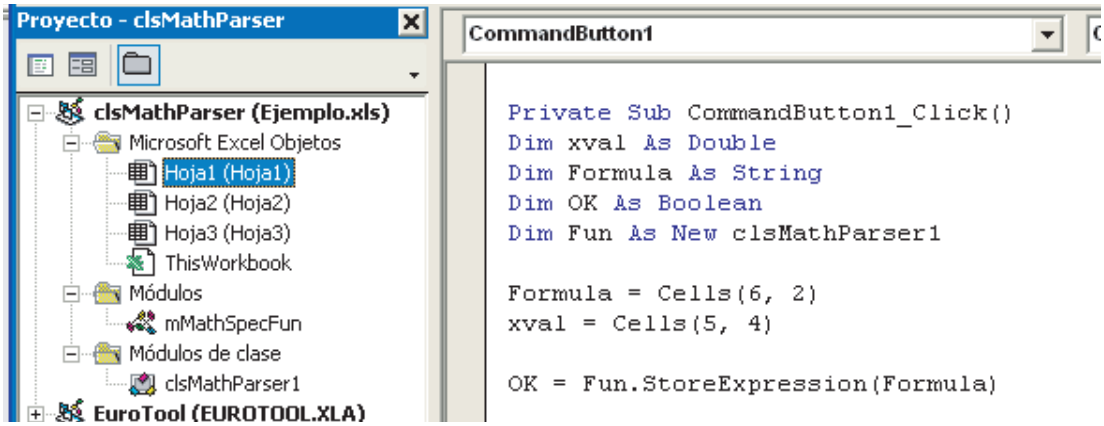


Figure 1.37: Editor VBA con los módulos.

   En este caso el módulo de clase quedó con el nombre `clsMathParser`. Así que el objeto `Fun` se construye con `Dim Fun As New clsMathParser`

4. Finalmente el código del botón es

```
Private Sub CommandButton1_Click()
    Dim xval As Double
    Dim Formula As String
    Dim OK As Boolean
    Dim Fun As New clsMathParser ' crear objeto Fun
```

```
    Formula = Cells(6, 2)
    xval = Cells(5, 4)

    OK = Fun.StoreExpression(Formula) ' leer 'Formula' y analizar sintaxis
    If Not OK Then GoTo Error_Handler ' si hay un error  va al manejador de errores

    Cells(6, 3) = "f(" + str(valx) + ")= "
    Cells(6, 4) = Fun.Eval1(xval)   'evaluar en la fórmula actual

    If Err Then GoTo Error_Handler  'si hay error...
    Error_Handler: Cells(1,1)= Fun.ErrorDescription 'Mensaje de error

End Sub
```

1. Las variables deben empezar con una letra, por ejemplo `t, xi, time`. Para evitar la ambigüedad, `xy` se interpreta como una variable. El producto requiere `*`, es decir `x*y` es el producto de x e y. Se permite multiplicación implícita solo para $x, y,$ y z, o sea, por ejemplo $2x$ se interpreta como `2*x`

2. Los argumentos de una función, si hay varios, se separan con una coma, por ejemplo `max(x,y), root(x,y).`

3. Las expresiones lógicas retornan 1 o 0. Como ejemplo de estas expresiones tenemos `(x<=1), (2<x<2)`. Una expresión como "$x < 0$ o $x > 3$" se introduce como `(x<0)+(x>3)`

4. La expresión

$$f(x) = \begin{cases} x^2 + x & \text{si} \quad x < 0 \\ \log(x) + \sqrt{x} & \text{si} \quad x > 2 \end{cases}$$

se introduce como `(x<0)*(x^2+x) + (x>2)*(log(x)+sqr(x))`

5. (Manejo de signos) Las siguientes expresiones *son equivalentes*

`x^-n`	`x^(-n)`
`x*-sin(a)`	`x*(-sin(a))`
`-5*-2`	`-5*(-2)`
`-x^2`	`(-x)^2, o sea, -3^2 = (-3)^2 = 9`

6. Constantes: π es `pi#` y e es `e#`. Por ejemplo

   (a) $\cos(x + \pi) + e^x$ se introduce como `cos(x+pi#)+e#^x`

   (b) e^{-x^2} se debe introducir como `1/e#^(x^2)` o como `e#^(-1*x^2)` pues `e#^(-x^2)` $= e^{x^2}$

7. Constantes físicas.

Constante de Planck	h#	`6.6260755e-34 J s`
Constante de Boltzmann	K#	`1.380658e-23 J/K`
Carga elemental	q#	`1.60217733e-19 C`
Número de Avogadro	A#	`6.0221367e23 particles/mol`
Velocidad de la luz	c#	`2.99792458e8 m/s`
Permeabilidad del vacio (m)	mu#	`12.566370614e-7 T2m3/J`
Permitividad del vacio (e)	eps#	`8.854187817e-12 C2/Jm`
Masa del Electron	me#	`9.1093897e-31 kg`
Masa del Proton	mp#	`1.6726231e-27 kg`
Masa del Neutron	mn#	`n 1.6749286e-27 kg`
Constante Gas	R#	`8.31451 m2kg/s2k mol`
Constante gravitacional	G#	`6.672e-11 m3/kg s2`
Aceleración de la gravedad	g#	`9.80665 m/s2`

Las constantes físicas deben ir seguidas por sus unidades de medida: `"1 s"` , `"200 m"` , `"25 kg"` , `"150 MHz"`, `"0.15 uF"`, `"3600 kohm"`. Se utilizan de manera análoga a las constante matemáticas. Por ejemplo `eps# * S/d#`, `sqr(m*h*g#)`, `s0+v*t+0.5*g#*t^2`

8. Símbolos y Funciones

```
+           adicción
-           resta
*           multiplicación
/           division              35/4 = 8.75
%           porcentaje            35% = 3.5
\           división entera       35\4 = 8
^           elevar a potencia     3^1.8 = 7.22467405584208 (°)
| |       valor absoluto         |-5|=5 !          factorial            5!=120
abs(x)      valor absoluto        abs(-5)= 5
atn(x)      atan(x)               x en radianes
cos(x)                            x en radianes
sin(x)
exp(x)      exponencial           exp(1) = 2.71828182845905 o sea e^1
fix(x)      parte entera          fix(-3.8) = 3
int(x)      parte entera          int(-3.8) = -4
dec(x)      parte decimal         dec(-3.8) = -0.8
ln(x),      logaritmo natural     x>0
log(x)      logaritmo natural
logN(x,n)   logaritmo base N      logN(16,2) = 4
rnd(x)      random                retorna un valor aleatorio entre  x y 0
sgn(x)      signo                 retorna 1 si x >0 , 0 si x=0, -1 si x<0
sqr(x)      raíz cuadrada         sqr(2) =1.4142135623731, también 2^(1/2)
cbr(x)      raíz cúbica
root(x,n)   raíz n-ésima          x^(1/n)
mod(a,b)                          mod(29,6) = 5,  mod(-29 ,6) = -5
comb(n,k)   combinaciones         comb(6,3) = 20 , comb(6,6) = 1
perm(n,k)   permutaciones         perm(8,4) = 1680
```

También, entre otras,

```
min(a,b),max(a,b), sech(x), coth(x), acsch(x), asech(x), acoth(x), round(x,d),
mcd(a,b), mcm(a,b,gcd(a,b),lcm(a,b), csc(x), sec(x), cot(x), acsc(x), asec(x),
acot(x), csch(x),tan(x), acos(x), asin(x), cosh(x), sinh(x), tanh(x), acosh(x),
asinh(x), atanh(x), and(a,b),or(a,b),not(a),xor(a,b), Psi(x), DNorm(), CNorm(),
DPoisson(),CPoisson(),DBinom(),CBinom(), Si(x) (SineIntegral), Ci(x),FresnelS(x),
etc.
```

9. Métodos

```
StoreExpression(f)          Almacena y revisa la sintaxis
Eval(x)                     Evalúa la expresión (que posiblemente tenga varias variables)
Eval1(x)                    Evalúa una expresión de una variable
```

`Eval()` se usa cuando vamos a evaluar expresiones con varios parámetros y/o variables. `Eval1()` es para funciones de una sola variable.

Vamos a ver algunos ejemplos en los que se aplican estos métodos.

1.6.2 Ejemplo: un graficador 2D

Podemos implementar de manera muy sencilla un graficador de funciones $y = f(x)$.

Para esto, leemos la ecuación de la función $y = f(x)$ en una celda y el intervalo de graficación $[a, b]$. Para hacer el gráfico necesitamos una tabla de valores (x_i, y_i). Lo que hacemos es partir el intervalo $[a, b]$ en $n+1$ puntos x_i, $i = 0, 1, ..., n$; y evaluamos la función en cada uno de estos x_i. Si los puntos son equidistantes, la distancia de separación es $h = (b-a)/n$ y este caso

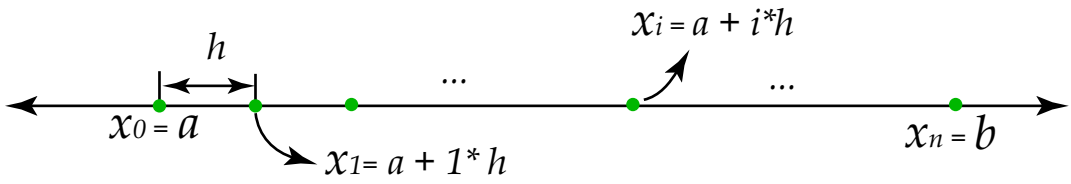


Figure 1.38: Partir el intervalo en n puntos.

$x_0 = a + 0 \cdot h$

$x_1 = a + 1 \cdot h$

$x_2 = a + 2 \cdot h$

$\vdots$

$$x_i = a + i \cdot h$$

$$x_n = a + n \cdot h$$

Entonces la tabla de puntos es el conjunto $\{(a + i \cdot h,\ f(a + i \cdot h)),\ \ i = 0, 1, ..., n\}$

Supongamos que el graficador va a quedar en una hoja como la que se ve en la figura

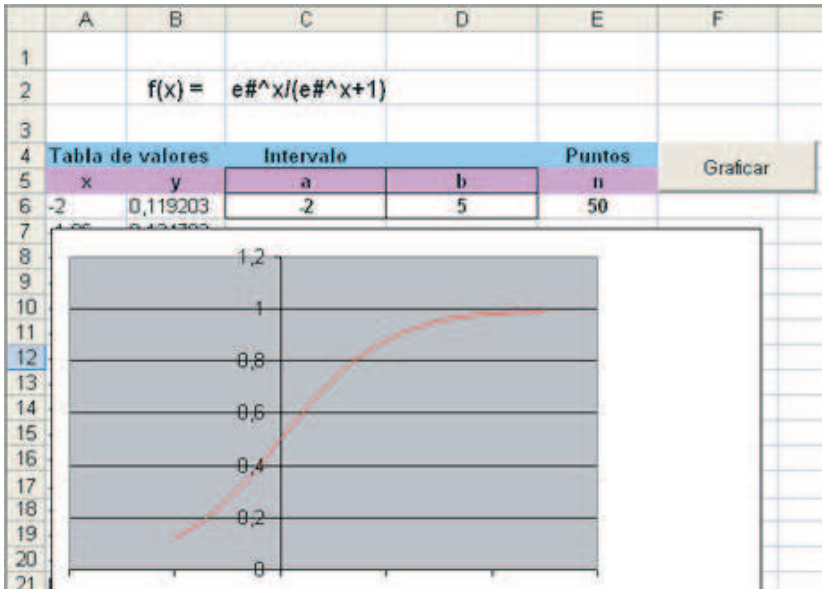


Figure 1.39: Graficador 2D.

Entonces la tabla de puntos la hacemos con el código

```
n = Cells(6, 5)
a = Cells(6, 3)
b = Cells(6, 4)
h = (b - a) / n
Formula = Cells(2, 3)

OK = Fun.StoreExpression(Formula)    'lectura de la fórmula
If Not OK Then GoTo Error_Handler

For i = 0 To n
    Cells(6 + i, 1) = a + i * h            'xi's
    Cells(6 + i, 2) = Fun.Eval1(a + i * h) 'yi's
Next i
```

Luego debemos seleccionar la tabla de valores y hacer un gráfico (chart). Si hay un gráfico previo lo podemos borrar (que no es necesario si uno quiere comparar gráficos por ejemplo). Estas dos cosas se hace con el código

```
'----------------------- eliminar gráficos anteriores----------------
  Set chartsTemp = ActiveSheet.ChartObjects
  If chartsTemp.Count > 0 Then
     chartsTemp(chartsTemp.Count).Delete
  End If

'------------------------ gráfico ----------------------------------
datos = Range(Cells(6, 1), Cells(6 + n, 2)).Address 'rango a graficar
Set graf = Charts.Add    'gráfico

With graf                'características
    .Name = "Gráfico"
    .ChartType = xlXYScatterSmoothNoMarkers  'tipo de gráfico  XY(Dispersión)
    .SetSourceData Source:=Sheets("Hoja1").Range(datos), PlotBy:=xlColumns
    .Location Where:=xlLocationAsObject, Name:="Hoja1"
  End With
```

El código completo del código del botón que levanta el gráfico es

```
Private Sub CommandButton2_Click()
    Dim n As Integer
    Dim h As Double
    Dim Formula As String
    Dim graf   As Chart
    Dim chartsTemp As ChartObjects 'contador de charts (gráficos) para eliminar el anterior
    Dim OK As Boolean
    Dim Fun As New clsMathParser

    n = Cells(6, 5)
    a = Cells(6, 3)
    b = Cells(6, 4)
    h = (b - a) / n
    Formula = Cells(2, 3)

    OK = Fun.StoreExpression(Formula)    'lectura de la fórmula
    If Not OK Then GoTo Error_Handler

    For i = 0 To n
    Cells(6 + i, 1) = a + i * h
    Cells(6 + i, 2) = Fun.Eval1(a + i * h)
    Next i


'----------------------- eliminar gráficos anteriores-------------
    Set chartsTemp = ActiveSheet.ChartObjects
    If chartsTemp.Count > 0 Then
         chartsTemp(chartsTemp.Count).Delete
      End If
'-----------------------------------------------------------------


    datos = Range(Cells(6, 1), Cells(6 + n, 2)).Address 'rango a graficar
```

```
    Set graf = Charts.Add    'gráfico y sus caraterísticas

    With graf
        .Name = "Gráfico"
        .ChartType = xlXYScatterSmoothNoMarkers
        .SetSourceData Source:=Sheets("Hoja1").Range(datos), PlotBy:=xlColumns
        .Location Where:=xlLocationAsObject, Name:="Hoja1"
      End With

'-----------------------------------------------------------------
    If Err Then GoTo Error_Handler
    Error_Handler: Cells(1, 1) = Fun.ErrorDescription 'imprimir mensaje error
'-----------------------------------------------------------------

End Sub
```

Observe que se pueden graficar funciones a trozos. Por ejemplo, para graficar

$$f(x) = \begin{cases} x^2 & \text{si} \quad x \in]-\infty, 0[\\ x & \text{si} \quad x \in]0, 2] \\ 4-x & \text{si} \quad x \in]2, \infty[\end{cases}$$

digitamos la función como

```
f(x) =(x<0)*(x^2)+(0<x<=2)*x+(x>2)*(4-x)
```

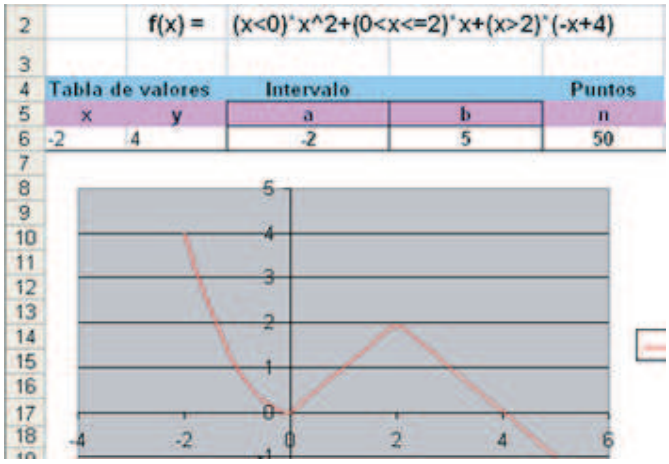


Figure 1.40: Función a trozos.

Ejercicios 3

Implemente un graficador para curvas definidas por una ecuación parámetrica.

Nota. Algunas curvas, como los círculos, las elipses, etc., se definen por medio de ecuaciones parámetricas

$$(x(t), y(t)), \quad t \in [a, b].$$

Por ejemplo

- Las ecuaciones paramétricas de un círculo de radio r y centro en (h, k) son

$$(h + r\cos(t),\ k + r\text{sen}(t)), \quad t \in [0, 2\pi].$$

- Las ecuaciones paramétricas de un cicloide son $(a > 0)$

$$x = a(\theta - \text{sen}\theta), \quad y = a(1 - \cos\theta), \quad \theta \in \mathbb{R}.$$

- Una curva con ecuación en polares $r = f(\theta), \quad \theta \in [a, b]$; tiene ecuación parámetrica

$$(f(\theta) \cdot \cos(\theta),\ f(\theta) \cdot \text{sen}(\theta)), \quad t \in [a, b].$$

1.6.3 Ejemplo: un graficador de superficies 3D

Podemos implementar de manera muy sencilla un graficador de superficies de ecuación $z = f(x, y)$.
Para esto solo hay que observar que en vez de una tabla de puntos $(x_i, y_i, z_i(x_i, y_i))$ en tres columnas, Excel necesita una tabla de la forma

	x0	x1	⋯	xn
y0	f(x0,y0)	f(x1,y0)	⋯	f(xn,y0)
y1	f(x0,y1)	f(x1,y1)		
⋮				
yn	f(x0,yn)	f(x1,yn)	⋯	f(xn,yn)

Los intervalos de graficación son $x \in [xmin, xmax]$ y $y \in [ymin, ymax]$. Cada intervalo lo partimos en n puntos (que puede puede ser modificado por el usuario).

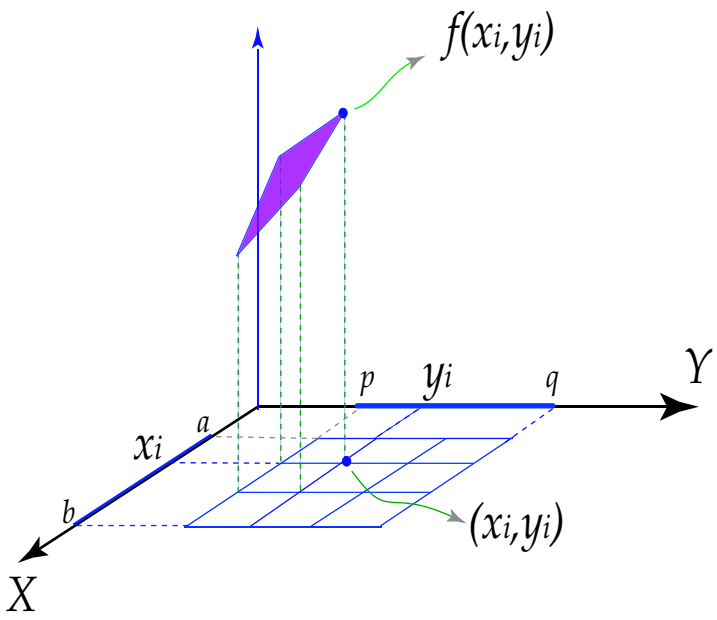


Figure 1.41: Partición.

Si `hx = (xmax-xmin)/n` y si `{hy = (ymax-ymin)/n`, entonces

```
xi = xmin + i*hx,      i = 0,1,...,n

yj = ymin + j*hy,      j = 0,1,...,n
```

Para que la tabla quede en el formato adecuado, debemos usar dos `For`

```
For i = 0 To n
    xi = xmin + i * hx
    Cells(7, 2 + i) = xi        'fila de las xi
    For j = 0 To n
        yi = ymin + j * hy
        Cells(8 + j, 1) = yi    'columna de la yi

        OK = Fun.StoreExpression(fxy)      'formula actual es 'f(x,y)'
        If Not OK Then GoTo Error_Handler
        Fun.Variable("x") = xi             ' asignación para evaluar
        Fun.Variable("y") = yi
        Cells(8 + j, 2 + i) = Fun.Eval()   'retorna f(xi,yi)
        Next j
Next i
```

Una vez que se genera la tabla de datos, la seleccionamos y la ocultamos (porque es muy grande y llena de decimales). Luego generamos el gráfico 3D.

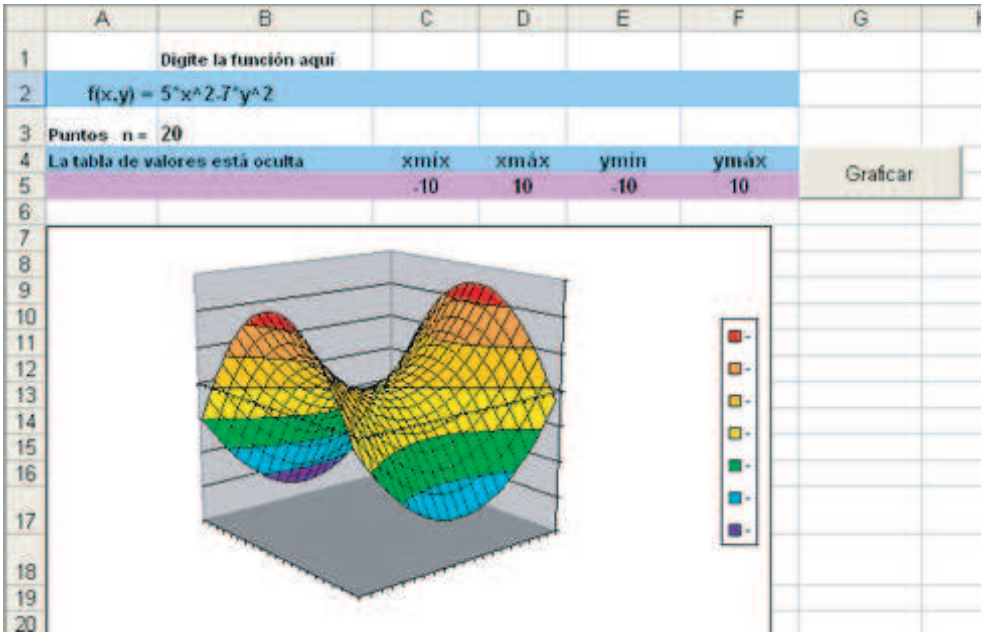


Figure 1.42: Graficador 3D.

El código completo es

```
Private Sub CommandButton2_Click()
    Dim xmin, xmax, ymin, ymax, hx, hy, xi, yi As Double
    Dim n As Integer
    Dim fxy As String 'función f(x,y)
    Dim graf   As Chart
    Dim OK As Boolean
    Dim Fun As New clsMathParser      ' así se llama el módulo de clase aquí

    fxy = Cells(2, 2)
    xmin = Cells(5, 3)
    xmax = Cells(5, 4)
    ymin = Cells(5, 5)
    ymax = Cells(5, 6)
    n = Cells(3, 2) ' número de puntos n x n
    hx = (xmax - xmin) / n
    hy = (ymax - ymin) / n

    If hx > 0 And hy > 0 And n > 0 Then
        For i = 0 To n
            xi = xmin + i * hx
            Cells(7, 2 + i) = xi
            For j = 0 To n
                yi = ymin + j * hy
                Cells(8 + j, 1) = yi

                OK = Fun.StoreExpression(fxy)      'formula actual es 'f(x,y)'
                If Not OK Then GoTo Error_Handler
```

```
                Fun.Variable("x") = xi
                Fun.Variable("y") = yi
                Cells(8 + j, 2 + i) = Fun.Eval() 'retorna f(xa,ya)
            Next j
        Next i

    End If
    '---------------------- eliminar gráficos anteriores-------------
    Set chartsTemp = ActiveSheet.ChartObjects
    If chartsTemp.Count > 0 Then
        chartsTemp(chartsTemp.Count).Delete
    End If
    '-----------------------------------------------------------------
    datos = Range(Cells(7, 1), Cells(7 + n, n + 2)).Address 'rango a graficar
    Range(datos).Select
    Selection.NumberFormat = ";;;" 'ocular celdas
    Charts.Add
    ActiveChart.ChartType = xlSurface
    ActiveChart.SetSourceData Source:=Sheets("Hoja1").Range(datos), PlotBy:=xlColumns
    ActiveChart.Location Where:=xlLocationAsObject, Name:="Hoja1"
    '-----------------------------------------------------------------
    If Err Then GoTo Error_Handler
    Error_Handler: Cells(1, 1) = Fun.ErrorDescription  'enviar un mensaje de error
    '-----------------------------------------------------------------

End Sub
```

Nota: para rotar la figura, seleccione el área del gráfico y haga clic con el botón derecho del mouse, seleccione en el menú la opción Vista en 3D...

1.6.4 Ejemplo: series numéricas y series de potencias

Ejemplo 10

Implementaremos una función que calcula la suma parcial $S_N = \sum_{k=k_0}^{N} f(k)$. Luego la aplicaremos a la serie $\sum_{k=1}^{\infty} \frac{2^k}{k!}$.

Para esto vamos a definir una función

$$\texttt{SumaParcial(formula, Kini, N)=} \sum_{k=Kini}^{N} f(k).$$

Esta función revisa la sintaxis de la fórmula y calcula la suma parcial.

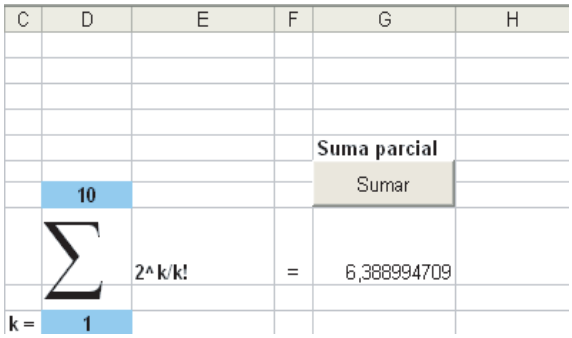


Figure 1.43: Suma parcial.

```
Function SumaParcial(laformula, Kini, N)
    Dim suma As Double
    Dim OK As Boolean
    Dim Fun As New clsMathParser         ' así se llama el módulo de clase aquí
    OK = Fun.StoreExpression(laformula)  ' formula actual es "laformula"

    If Not OK Then GoTo Error_Handler

    suma = 0
    For i = Kini To N
        suma = suma + Fun.Eval1(i) 'evalúa en "laformula"
    Next i
    '-----------------------------------------------------------------
    If Err Then GoTo Error_Handler
    Error_Handler: Cells(1,1)= Fun.ErrorDescription
    '-----------------------------------------------------------------
SumaParcial = suma
End Function
```

Ahora, podemos aplicar esta función con un botón

```
Private Sub CommandButton1_Click()
Dim formula1 As String
Dim N, kk As Integer

formula1 = Cells(8, 5)
N = Cells(7, 4)
kk = Cells(10, 4)
Cells(8, 7) = SumaParcial(formula1, kk, N)
End Sub
```

■ Ejemplo 11

Dada una serie numérica $\sum_{k=k_0}^{\infty} f(k)$, vamos a desplegar las primeras N sumas parciales $S_N = \sum_{k=k_0}^{N} f(k)$, de la celda A4 en adelante.

	A	B	C	D
1	**Sumas parciales de la serie**			
2		**Fórmula f(k)**	Sumas Parciales	**N**
3	**SN**	**2^k/k!**		20
4	S 1 =	2		
5	S 2 =	4		
6	S 3 =	5,333333333		
7	S 4 =	6		
8	S 5 =	6,266666667		
9	S 6 =	6,355555556		

Figure 1.44: Sumas parciales.

Usando la función `SumaParcial(formula, Kini, N)`, solo debemos calcular cada suma parcial por separado.

```
For i = 1 to N
    Cells(3+i,1)= SumaParcial(formula,Kini,i)
Next i
```

El código del botón es

```
Private Sub CommandButton1_Click()
    Dim x0 As Double
    Dim formula1 As String
    Dim N As Integer

    formula1 = Cells(3, 2)
    N = Cells(3, 4)

    For i = 1 To N   'sumas parciales S1, S2, ... SN
        Cells(3 + i, 1) = "S" + str(i) + " = "
        Cells(3 + i, 2) = sp(formula1, 1, i)
    Next i
End Sub
```

Ahora vamos a cambiar a series de potencias

■ Ejemplo 12

Usando Eval(). Dada una serie de potencias $\sum_{k=k_0}^{\infty} f(x,k)$, implementaremos una subrutina que calcula la suma parcial $S_N = \sum_{k=k_0}^{N} f(x_0,k)$. Luego la aplicaremos a la serie $\sum_{k=1}^{\infty} \frac{x^k}{k!}$, con $x = 2$.

Vamos a implementar una nueva función `SumaParcial(laformula, Kini, N, xx)`.

Aquí la situación es diferente, $f(x,k)$ tiene dos variables, por tanto no podemos usar `Eval1()` porque este método es para funciones de una sola variable.

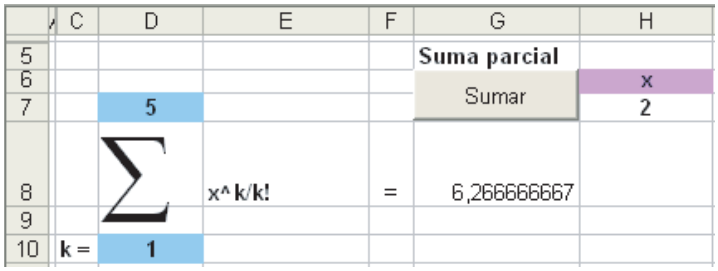


Figure 1.45: Suma parcial para una serie de potencias.

`Eval()` evalúa la expresión actual almacenada y retorna su valor. Para evaluar, se debe especificar el valor de cada variable con `Fun.Variable()`. En nuestro caso debemos hacer la especificación `Fun.Variable("x")=xx` y `Fun.Variable("k")=i`, antes de llamar a `Eval()`. Veamos el código de la función `SumaParcial(laformula, Kini, N, xx)`

```
Function SumaParcial(laformula, Kini, N, xx)
    Dim suma As Double
    Dim OK As Boolean
    Dim Fun As New clsMathParser         ' así se llama el módulo de clase aquí
    OK = Fun.StoreExpression(laformula)  ' formula actual es "laformula"

    If Not OK Then GoTo Error_Handler

    suma = 0
    Fun.Variable("x") = xx
    For i = Kini To N
        Fun.Variable("k") = i
        suma = suma + Fun.Eval() 'evalúa en "laformula" actual
    Next i
    '-----------------------------------------------------------------
    If Err Then GoTo Error_Handler
    Error_Handler:     Cells(1,1)= Fun.ErrorDescription
    '-----------------------------------------------------------------
SumaParcial = suma
End Function
```

Ahora, el código del botón sería

```
Private Sub CommandButton1_Click()
    Dim formula1 As String
    Dim N, kk As Integer
    Dim x0 As Double

    formula1 = Cells(8, 5)
    N = Cells(7, 4)
    kk = Cells(10, 4)
    x0 = Cells(7, 8)
    Cells(8, 7) = SumaParcial(formula1, kk, N, x0)
End Sub
```

Ejercicios 4

1. Implemente un función para series alternadas que además de dar la suma parcial pedida, indique la estimación del error. Recuerde que si $\sum_{k=1}^{\infty} f(k)$ es alternada entonces si el error cometido al aproximar la serie con la suma parcial $S_N = \sum_{k=1}^{N} f(k)$ es R_N, entonces $|R_N| \leq a_{N+1}$.

Chapter 2

Elementos de Análisis Numérico

2.1 Solución de ecuaciones de una variable

2.1.1 Método de Newton-Raphson

La intención no es hacer un desarrollo teórico de este método, sino centrarse en una implementación con código sencillo, ideal para experimentos. En general, no hay control de excepciones (como divisiones por cero, overflow, etc).

	A	B	C	D	E
1	**Digite la función aquí: g(x) = x - f(x) / f'(x)**				
2					
3	**g(x) =**	**x - (x-cos(x))/(1+sin(x))**			
4					Aplicar
5					
6	**Tol**	**N**	**x0**	**x_n**	**Error estimado**
7	0,00001	15	1	0,75036387	0,249636132
8				0,73911289	0,011250977
9				0,73908513	2,77575E-05
10				0,73908513	1,70123E-10
11				Ok	

Figure 2.1: Método de Newton.

Vamos a usar `clsMathParser` por ser un evaluador rápido y eficiente.

Si x_0 es una aproximación suficientemente cercana a una solución r de $f(x) = 0$, con $f'(r) \neq 0$, entonces la sucesión $x_{i+1} = x_i - \dfrac{f(x_i)}{f'(x_i)}$ aproxima r con error menor que $|x_{i+1} - x_i|$

Para implementar este método, leemos la función $g(x) = x - \dfrac{f(x)}{f'(x)}$, entonces el algoritmo sería muy sencillo

```
i=1
xi = x0
while i <= N
    xim1 =  g(xi)  'xim1 es x_{i+1}
```

```
    if  |xim1 - xi|< Tol then
        'parar
        Exit Sub
    end if
    i=i+1
    xi = xim1
wend
```

En este caso, debemos leer y almacenar la función g. Entonces, en el código, $g(x_i) =$ `Fun.Eval1(xi)`

En este ejemplo vamos a usar

```
Do while condición
instrucciones
Loop
```

Si ejecutamos el código desde un botón entonces el código sería algo como

```
Private Sub CommandButton1_Click()
    Dim x0, tol, xi, xim1, errorEst As Double
    Dim N As Integer
    Dim OK As Boolean
    Dim Fun As New clsMathParser      ' así se llama el módulo de clase aquí

    x0 = Cells(7, 3)
    N = Cells(7, 2)
    tol = Cells(7, 1)
    g = Cells(3, 2)
    Cells(6, 6) = " " 'limpia mensajes anteriores

    OK = Fun.StoreExpression(g)       'formula actual es g
    If Not OK Then GoTo Error_Handler

    xi = x0
    i = 1
    Do While i <= N
        xim1 = Fun.Eval1(xi)  'formula actual es g, ésta es la que está evaluando
        errorEst = Abs(xim1 - xi)
        Cells(6 + i, 4) = xim1
        Cells(6 + i, 5) = errorEst

        If errorEst < tol Then
            Cells(6 + i + 1, 4) = "Ok"
            Exit Sub
        End If
        i = i + 1
        xi = xim1
    Loop
    Cells(6, 6) = " No se tuvo éxito"  'si se terminó el Do sin éxito

    '-------------------------------------------------------------
    If Err Then GoTo Error_Handler
```

```
    Error_Handler: Cells(1,1)=Fun.ErrorDescription
    '------------------------------------------------------------
End Sub
```

2.2 Integración

2.2.1 Método de Romberg para integración

En el método de integración de Romberg se usa la regla compuesta del trapecio para obtener aproximaciones preliminares y luego el proceso de extrapolación de Richardson para mejorar las aproximaciones.

Para aproximar la integral $\int_a^b f(x)\,dx$ se calcula un arreglo

$$\begin{array}{llll} R_{11} & & & \\ R_{21} & R_{22} & & \\ R_{31} & R_{32} & R_{33} & \\ \vdots & & & \\ R_{n1} & R_{n2} & \ldots & R_{nn} \end{array}$$

y $\int_a^b f(x)\,dx \approx R_{nn}$. El criterio de parada es $|R_{nn} - R_{n-1\,n-1}| < Tol$ (a veces también se usa la desigualdad anterior y la desigualdad $|R_{n-1\,n-1} - R_{n-2\,n-2}| < Tol$ para evitar algunos casos no deseables).

Si $h_k = \dfrac{b-a}{2^{k-1}}$

a.) $R_{1,1} = \dfrac{h_1}{2}[f(a) + f(b)]$

b.) $R_{k,1} = \frac{1}{2}\left[R_{k-1,1} + h_{k-1}\displaystyle\sum_{i=1}^{2^{k-2}} f(\,a + h_k(2i-1)\,)\right], \quad k = 2, 3, ..., n$

c.) $R_{k,j} = R_{k,j-1} + \dfrac{R_{k,j-1} - R_{j-1,j-1}}{4^{j-1} - 1}, \;\; k = 2, 3, ..., n;\; j = 2, 3, ...k$

Este algoritmo es muy eficiente para funciones suficientemente suaves y sin singularidades en $[a, b]$. En general, para integrales impropias se usan otros algoritmos. Por ejemplo, Romberg no es adecuado para calcular la integral $\Gamma(z) = \int_0^\infty t^{z-1}e^{-t}dt$

En vez de implementar el método de Romberg de manera directa, vamos a implementar el siguiente algoritmo

1. $h = b - a$
2. $R_{1,1} = \dfrac{h_1}{2}[f(a) + f(b)]$

3. para $i = 1, 2, ..., n$ hacer

   (a) $R_{2,1} = \frac{1}{2}\left[R_{1,1} + h\sum_{i=1}^{2^{i-1}} f(\,a\, +\, h(k-0.5)\,)\right]$

   (b) para $j = 2, ..., i$
   $R_{2,j} = R_{2,j-1} + \dfrac{R_{2,j-1} - R_{1,j-1}}{4^{j-1} - 1}$

   (c) Salida: $R_{2,j}$, $j = 1, 2, ..., i$

   (d) $h = h/2$

   (e) para $j = 1, 2, ..., i$ tome $R_{1,j} = R_{2,j}$.

4. Parar

En el código de un botón, definimos una matriz `R(n,n)` con la cual vamos a implementar el algoritmo.

	A	B	C	D	E	F	G	H
1	f(x) =	1 e=^(x^2)	a	b	n	Romberg		
2			0,00	1,00	6			
3	0,73137025							
4	0,7429841	0,74685538						
5	0,74586561	0,74682612	0,74682417					
6	0,7465846	0,74682426	0,74682413	0,74682413				
7	0,74676425	0,74682414	0,74682413	0,74682413	0,74682413			
8	0,74680916	0,74682413	0,74682413	0,74682413	0,74682413	0,74682413		

Figure 2.2: Integración Romberg.

```
Private Sub CommandButton1_Click()
    Dim R() As Double
    Dim a, b, h, suma As Double
    Dim n As Integer
    Dim formula As String
    Dim OK As Boolean
    Dim Fun As New clsMathParser      ' así se llama el módulo de clase aquí

    formula = Cells(1, 2)
    a = Cells(2, 3)
    b = Cells(2, 4)
    n = Cells(2, 5)
    ReDim R(n, n)
    h = b - a

    OK = Fun.StoreExpression(formula)      'formula actual es 'formula'
    If Not OK Then GoTo Error_Handler
    '-------------------------------------------------------------------------
        For i = 1 To 20    'limpiar
            For j = 1 To 20
```

```
            Cells(2 + i, j) = Null
        Next j
    Next i
'-------------------------------------------------------------------
R(1, 1) = h / 2 * (Fun.Eval1(a) + Fun.Eval1(b))

'paso3 de algoritmo de Romberg
For i = 1 To n
    'paso 4
        suma = 0
        For k = 1 To 2 ^ (i - 1)
            suma = suma + Fun.Eval1(a + h * (k - 0.5)) 'evalúa en la fórmula actual
        Next k
        R(2, 1) = 0.5 * (R(1, 1) + h * suma)
    'paso5
        For j = 2 To i
            R(2, j) = R(2, j - 1) + (R(2, j - 1) - R(1, j - 1)) / (4 ^ (j - 1) - 1)
        Next j
    'paso 6 salida R(2,j)
    For j = 1 To i
        Cells(3 + i - 1, j) = R(2, j) 'columnas 2,3,...n
    Next j
    'paso 7
        h = h / 2
    'paso 8
    For j = 1 To i
        R(1, j) = R(2, j)
    Next j
Next i

'-------------------------------------------------------------
If Err Then GoTo Error_Handler
Error_Handler: Cells(1,1)=Fun.ErrorDescription
'-------------------------------------------------------------
End Sub
```

Nota: Observe la función que se integra en la figura anterior. Recordemos que en este evaluador, `-x^2 = (-x)^2`. Entonces, para aproximar $\int_a^b e^{-x^2}\,dx$ debemos introducir la función como `1/e#^(x^2)` o `e#^(-1*x^2)`.

2.2.2 La función Gamma

La función Gamma se define por:

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

Cuando el argumento z es entero la función Gamma se reduce al factorial $(z-1)!$. No obstante cuando z no es entero la integral impropia que se genera debe ser calculada en forma numérica.

Para este cálculo hemos recurrido a la aproximación siguiente [10].

Para ciertos valores enteros N y γ, y para ciertos coeficientes $c_1, c_2, \ldots, c_N$ la función Gamma está dada por

$$\Gamma(z+1) = \left(z+\gamma+\frac{1}{2}\right)^{z+0.5}e^{-\left(z+\gamma+\frac{1}{2}\right)} \times \sqrt{2\pi}\left[c_0 + \frac{c_1}{z+1} + \frac{c_2}{z+2} + \ldots + \frac{c_N}{z+N} + \epsilon\right]$$

Para implementar el programa se toman los valores de $N = 6$ y $\gamma = 5$ y en ese caso el error en la aproximación es menor que 2×10^{-10} en el semiplano $Re\, z > 0$

Los valores de los primeros 6 ci's son

```
c0 = 1
c1 = 36.18109133
c2 = -86.50532033
c3 =  24.51409822
c4 =-1.291739516
c5 = 0.120858003
c6 = -0.539082
```

Así

$$\int_0^\infty t^z e^{-t} dt \approx \texttt{(z+5.5)^(z+ 0.5*e\#^(-1*(z+5.5)))*sqr(2*pi\#)*(c0+c1/(z+1)+...+c6/(z+6))}$$

con error menor o igual a 2×10^{-10} en el semiplano $Re\, z > 0$

Este es un algoritmo muy eficiente, de hecho requiere muy pocos cálculos y tiene una alta precisión.

2.2.3 Cuadratura gaussiana e integral doble gaussiana.

Cuadratura gaussiana.

La cuadratura gaussiana selecciona $x_1, x_2, ..., x_n$ en $[a, b]$ y unos coeficientes $c_1, c_2, ..., c_n$ de tal manera que se reduzca "en lo posible" el error esperado en la aproximación

$$\int_a^b f(x)dx \approx \sum_{i=1}^{n} c_i\, f(x_i) \qquad (*)$$

La reducción "en lo posible" del error esperado en la aproximación se entiende en el sentido de que, por lo menos, $\int_{-1}^{1} P(x)\, dx = \sum_{i=1}^{n} c_i\, P(x_i)$, con $P(x)$ un polinomios de grado $< 2n$. En este caso, los valores adecuados para $x_1, ..., x_n$ son las raíces del polinomio de Legendre de grado n y los números c_i están definidos por

$$\int_{-1}^{1} \prod_{\substack{j=1 \\ j\neq i}}^{n} \frac{x - x_j}{x_i - x_j} dx$$

Los polinomios de Legendre se definen de manera recursiva

$$\begin{aligned} P_0(x) &= 1 \\ P_1(x) &= x \\ P_2(x) &= \tfrac{1}{2}(3x^2-1) \\ P_3(x) &= \tfrac{1}{2}(5x^3-3x) \\ &\vdots \quad\quad \vdots \\ P_{n+1}(x) &= \frac{2n+1}{n+1}\,x\,P_n(x) - \frac{n}{n+1}\,P_{n-1}(x), \quad \text{para } n=2,3,\ldots \end{aligned}$$

Estos valores x_i y c_i reducen "en lo posible", del error esperado en la aproximación

$$\int_{-1}^{1} f(x)\,dx \approx \sum_{i=1}^{n} c_{ni}\,f(x_{ni})$$

Ahora en vez de xi y ci ponemos x_{ni} y c_{ni} para indicar que son los valores que corresponden al polinomio de Legendre de grado n.

Para aproximar $\int_a^b f(x)\,dx$ se requiere de un cambio de variable $x = \frac{1}{2}[\,(b-a)t + a + b\,]$ con el cual

$$\int_a^b f(x)\,dx = \int_{-1}^{1} f\left(\frac{1}{2}\,(b-a)t + a + b\right)\frac{b-a}{2}\,dt$$

Ahora si $g(t) = \frac{b-a}{2}\,f\left(\frac{1}{2}\,(b-a)t + a + b\right)$ entonces

$$\int_{-1}^{1} g(t)\,dt \approx \sum_{i=1}^{n} c_{ni}\,g(x_{ni})$$

Así, para aproximar $\int_0^1 e^{-x^2}\,dx$ debemos aplicar la fórmula de aproximación a $\int_{-1}^{1} 0.5\,e^{-0.25\,(1+t)^2}\,dt$

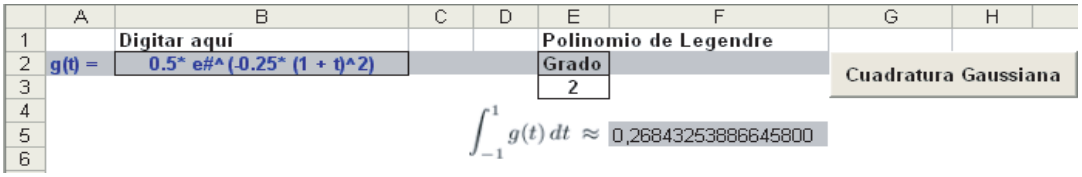


Figure 2.3: Cuadratura gausssina.

Las raíces x_i de los polinomios de Legrende de grado menor o igual a $n = 5$ y los c_i se pueden ver en la tabla que sigue

Grado de $P(x)$	c_i	x_i
2	$c_{21} = 1$	$x_{21} = 0.5773502692$
	$c_{22} = 1$	$x_{22} = -0.5773502692$
3	$c_{31} = 0.5555555556$	$x_{31} = -0.7745966692$
	$c_{32} = 0.8888888889$	$x_{32} = 0.0000000000$
	$c_{33} = 0.5555555556$	$x_{33} = 0.7745966692$
4	$c_{41} = 0.3478548451$	$x_{41} = -0.8611363116$
	$c_{42} = 0.6521451549$	$x_{42} = -0.3399810436$
	$c_{43} = 0.6521451549$	$x_{43} = 0.3399810436$
	$c_{44} = 0.3478548451$	$x_{44} = 0.8611363116$
5	$c_{51} = 0.2369268850$	$x_{51} = 0.9061798459$
	$c_{52} = 0.4786286705$	$x_{52} = 0.5384693101$
	$c_{53} = 0.5688888889$	$x_{53} = 0.0000000000$
	$c_{54} = 0.4786286705$	$x_{54} = -0.5384693101$
	$c_{55} = 0.2369268850$	$x_{55} = -0.9061798459$

Para hacer la implementación definimos dos funciones `xi(i,j)` y `ci(i,j)` donde `i` es el grado del polinomio de Legendre y `j` es el valor j.

```
Function xi(i, j)
    Dim xis(5, 5) As Double
    xis(2, 1) = 0.5773502692
    xis(2, 2) = -0.5773502692

    xis(3, 1) = 0.7745966692
    xis(3, 2) = 0
    xis(3, 3) = -0.7745966692

    xis(4, 1) = 0.8611363116
    xis(4, 2) = 0.3399810436
    xis(4, 3) = -0.3399810436
    xis(4, 4) = -0.8611363116

    xis(5, 1) = 0.9061798459
    xis(5, 2) = 0.5384693101
    xis(5, 3) = 0
    xis(5, 4) = -0.5384693101
    xis(5, 5) = -0.9061798459

 xi = xis(i, j)
End Function

Function ci(i, j)
    Dim cis(5, 5) As Double
    cis(2, 1) = 1
    cis(2, 1) = 1

    cis(3, 1) = 0.5555555556
    cis(3, 2) = 0.8888888889
```

```
    cis(3, 3) = 0.5555555556

    cis(4, 1) = 0.3478548451
    cis(4, 2) = 0.6521451549
    cis(4, 3) = 0.6521451549
    cis(4, 4) = 0.3478548451

    cis(5, 1) = 0.236926885
    cis(5, 2) = 0.4786286705
    cis(5, 3) = 0.5688888889
    cis(5, 4) = 0.4786286705
    cis(5, 5) = 0.236926885

ci = cis(i, j)
End Function
```

Además debemos incluir un mensaje en el caso de que se seleccione un grado menor que 2 mayor que 5.

El código del botón es

```
Private Sub CommandButton1_Click()
    Dim suma As Double
    Dim n As Integer
    Dim g As String
    Dim OK As Boolean
    Dim Fun As New clsMathParser  ' así se llama el módulo de clase aquí

    g = Cells(2, 2)
    n = Cells(3, 5)
    Cells(5, 6) = "" 'limpiar cálculos anteriores
    If n < 2 Or n > 5 Then
        MsgBox ("Solo tenemos datos para n entre 2 y 5")
        Exit Sub
    End If

    OK = Fun.StoreExpression(g)      'formula actual es 'formula'
    If Not OK Then GoTo Error_Handler

    suma = 0
    For i = 1 To n
    suma = suma + ci(n, i) * Fun.Eval1(xi(n, i))
    Next i

    Cells(5, 6) = suma

    '-----------------------------------------------------------------
    If Err Then GoTo Error_Handler
    Error_Handler: Cells(1,1)=Fun.ErrorDescription 'Mensaje de error
    '-----------------------------------------------------------------
End Sub
```

Integral doble gaussiana.

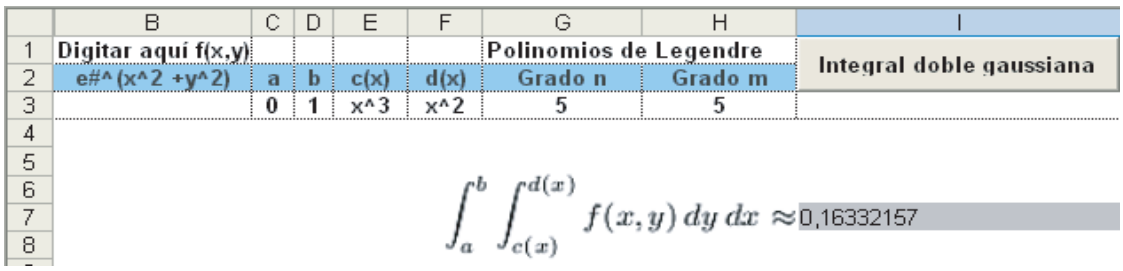


Figure 2.4: Integral doble gaussiana.

Si queremos aplicar la cuadratura gaussiana a $\int_a^b \int_{c(x)}^{d(x)} f(x,y)\,dy\,dx$ se debe hacer, primero que todo, un cambio de variable para cambiar $[c(x),\, d(x)]$ a $[-1, 1]$. Esto se hace de manera parecida al procedimiento numérico.

$$\int_a^b \int_{c(x)}^{d(x)} f(x,y)\,dy\,dx \approx \int_a^b 0.5(d(x)-c(x)) \sum_{j=1}^{n} c_{nj} f\left[x,\, 0.5\cdot(d(x)-c(x)\,x_{nj} \;+\; d(x)+c(x))\,dx\right]$$

y luego debemos volver hacer esto para $[a, b]$.

Cada Integral puede ser aproximada usando las raíces de un polinomio de Legendre de grado distinto.

El algoritmo es el siguiente

1. Entrada: a, b y los enteros m, n correspondientes al grado del polinomio de Legendre que se usa en cada integral. Deben ser enteros adecuados de acuerdo a los valores c_{ij}, x_{ij} disponibles.

2. Tome

   $h_1 = (b-a)/2$;
   $h_2 = (b+a)/2$

   (a) $J = 0$

3. Para

4. $i = 1, 2, ..., m$ hacer

   (a) Tome $JX = 0$
   $x = h_1 x_{mi} + h_2$
   $d_1 = d(x)$
   $c_1 = c(x)$
   $k_1 = (d_1 - c_1)/2$
   $k_2 = (d_1 + c_1)/2$

(b) para $j = 1, 2, ..., n$ hacer

$$y = k_1 x_{nj} + k_2$$
$$Q = f(x, y)$$
$$JX = JX + c_{nj} Q$$

(c) Tome $J = J + c_{mi} k_1 JX$

5. Tome $J = h_1 J$
6. Salida: la aproximación de la integral es J

En la implementación debemos evaluar funciones de una variable, $c(x)$ y $d(x)$,y una función de dos variables, $Q = f(x, y)$. Para hacer esto, almacenamos la expresión actual y procedemos a evaluar ya sea con `Eval1()` para funciones de una variable o `Eval()` para varias variables. En el código aparecen las siguientes líneas para este fin

```
OK = Fun.StoreExpression(dx)      'formula actual es 'd(x)'
d1 = Fun.Eval1(xa) 'd(xa)
  ...
OK = Fun.StoreExpression(cx)      'formula actual es 'c(x)'
c1 = Fun.Eval1(xa) 'c(xa)
  ...
OK = Fun.StoreExpression(fxy)      'formula actual es 'f(x,y)'
Fun.Variable("x") = xa
Fun.Variable("y") = ya
Q = Fun.Eval() 'retorna f(xa,ya)
  ...
```

El código completo del botón es (faltan las funciones `xi()` y `yi()`

```
Private Sub CommandButton1_Click()
Dim a, b, h1, h2, Jt, JX, xa, d1, c1, k1, k2, ya, Q As Double
Dim n, m As Integer
Dim fxy, cx, dx As String 'funciones
Dim OK As Boolean
Dim Fun As New clsMathParser      ' así se llama el módulo de clase aquí

fxy = Cells(2, 2)
cx = Cells(3, 5)
dx = Cells(3, 6)
a = Cells(3, 3)
b = Cells(3, 4)
n = Cells(3, 7)
m = Cells(3, 8)

Cells(7, 9) = "" 'limpiar cálculos anteriores

If n < 2 Or n > 5 Or m < 2 Or m > 5 Then
    MsgBox ("Solo tenemos datos para n y m entre 2 y 5")
    Exit Sub
End If
'paso 1
h1 = (b - a) / 2
```

```
h2 = (b + a) / 2
Jt = 0
'paso2
For i = 1 To m
    'paso 3
    JX = 0
    xa = h1 * xi(m, i) + h2
    OK = Fun.StoreExpression(dx)      'formula actual es 'd(x)'
    If Not OK Then GoTo Error_Handler

    d1 = Fun.Eval1(xa) 'd(xa)

    OK = Fun.StoreExpression(cx)      'formula actual es 'c(x)'
    If Not OK Then GoTo Error_Handler

    c1 = Fun.Eval1(xa) 'c(xa)

    k1 = (d1 - c1) / 2
    k2 = (d1 + c1) / 2

    For J = 1 To n
        ya = k1 * xi(n, J) + k2

        OK = Fun.StoreExpression(fxy)      'formula actual es 'f(x,y)'
        If Not OK Then GoTo Error_Handler
        Fun.Variable("x") = xa
        Fun.Variable("y") = ya
        Q = Fun.Eval() 'retorna f(xa,ya)

        JX = JX + ci(n, J) * Q
    Next J

    Jt = Jt + ci(m, i) * k1 * JX

Next i

Jt = h1 * Jt

Cells(7, 9) = Jt  ' aproxx de la integral

'-----------------------------------------------------------------
If Err Then GoTo Error_Handler
Error_Handler: Debug.Print Fun.ErrorDescription
'-----------------------------------------------------------------

End Sub
```

Nota: Observe que $\displaystyle\int_a^b \int_0^1 2\,y\,f(x)\,dy\,dx = \int_a^b f(x)\,dx,$ entonces esta implementación se podría usar para la cuadratura gaussiana en una variable.

Nota: Se puede aplicar la misma idea y aproximar una integral doble usando Simpson.

2.3 Problemas de valor inicial para ecuaciones diferenciales ordinarias

2.3.1 Existencia y unicidad

Consideremos el problema de valor inicial

$$\frac{dy}{dx} = f(t,y), \quad a \leq t \leq b, \quad y(a) = y_0 \qquad (*)$$

Una solución de este problema de valor inicial es una función derivable $y(t)$ que cumple las condiciones de $(*)$.

Definición 1

Consideremos un conjunto $D \subseteq \mathbb{R}$ y una función $f(t,y)$ definida en D. Si existe una constante $L > 0$ tal que

$$|f(t,y_1) - f(t,y_2)| \leq L|y_1 - y_2|, \quad \forall\,(t,y_1),\,(t,y_2) \in D$$

se dice que $f(t,y)$ cumple una condición de Lipschitz en la variable y en D. A L se le llama constante de Lipschitz para f.

Nota: Una condición *suficiente* para que $f(t,y)$ cumpla una condición de Lipschitz en D es que exista $L > 0$ tal que $\left|\dfrac{\partial f}{\partial y}(t,y)\right| \leq L, \quad \forall\,(t,y) \in D$

Definición 2

Un conjunto $D \subseteq \mathbb{R}$ se dice convexo si $\forall\,(t_1,y_1),\,(t_2,y_2) \in D$, el segmento $\{\,(1-\lambda)(t_1,y_1) + \lambda(t_2,y_2), \quad \lambda \in [0,1]\}$ está contenido en D.

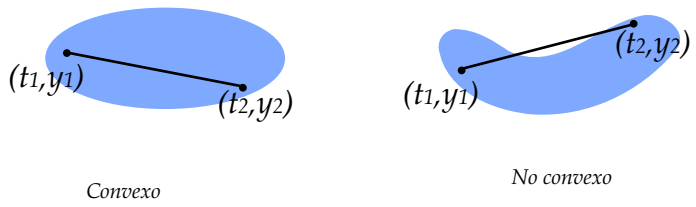


Figure 2.5: Convexidad

Nota: Observe que, cuando $\lambda = 0$ estamos en el punto inicial (t_1,y_1) y cuando $\lambda = 1$ estamos en el punto final (t_2,y_2). $\lambda = 1/2$ corresponde al punto medio del segmento.

Teorema 1

Si $D = \{(t, y) : a \leq t \leq b, -\infty \leq y \leq \infty\}$ y si $f(t, y)$ es continua en D y satisface una condición de Lipschitz respecto a y en D, entonces el problema $(*)$ tiene una solución única $y(t)$ para $a \leq t \leq b$.

———

Nota: Un problema de valor inicial está *bien planteado* si pequeños cambios o perturbaciones en el planteo del problema (debido a errores de redondeo en el problema inicial, por ejemplo), ocasiona cambios pequeños en la solución del problema. Si un problema cumple las hipótesis del teorema anterior, entonces está bien planteado.

Ejemplo 13

Consideremos el problema de valor inicial $y' = y - t^2 + 1,\ t \in [0, 4], \quad y(0) = 0.5$. Aquí $f(t, y) = y - t^2 + 1$. Si $D = \{(t, y) : 0 \leq t \leq 4, -\infty \leq y \leq \infty\}$, entonces como

$$\left|\frac{\partial f}{\partial y}(t, y)\right| = |1| \quad \forall\, (t, y) \in D$$

f cumple una condición de Lipschitz en y (en este caso podemos tomar $L = 1$). Además, como $f(t, y)$ es continua en D, el problema de valor inicial tiene una solución única. De hecho la única solución es $y(t) = (t + 1)^2 - 0.5e^t$

———

2.3.2 Método de Euler

Supongamos que tenemos un problema, bien planteado, de valor inicial

$$\frac{dy}{dx} = f(t, y), \quad a \leq t \leq b, \quad y(a) = y_0 \qquad (*)$$

Si hacemos una partición del intervalo $[a, b]$ en n subintervalos $[t_0, t_1], [t_1, t_2], ...[t_{n-1}, t_n]$ cada uno de longitud $h = (b - a)/n$, es decir $t_{i+1} = a + ht_i$, entonces usando un polinomio de Taylor alrededor de $t_0 = a$, podemos calcular n aproximaciones $w_i \approx y(t_i)$. Este procedimiento, conocido como método de Euler, es como sigue

Recordemos que, usando un el polinomio de Taylor de grado 1, podemos encontrar una aproximación (lineal) de la función f en los alrededores de a. El teorema de Taylor, formulado de manera conveniente, es

Teorema 2

Sea $h > 0$ y f una función tal que f y sus primeras k derivadas son continuas en el intervalo $[a, a + h]$ y la derivada f^{k+1} existe en $]a, a + h[$, entonces existe un número ξ ('xi'), en el intervalo $]a, a + h[$ tal que

$$f(a + h) = f(a) + \frac{f'(a)}{1!}\, h + \frac{f''(a)}{2!}\, h^2 + \cdots + \frac{f^{(k)}(a)}{k!}\, h^k + \frac{f^{(k+1)}(\xi)}{(k + 1)!}\, h^{k+1}$$

———

Ahora, si $y(t)$ es la solución de la ecuación $(*)$ y es dos veces derivable en $[a,b]$ entonces

$$y(t_{i+1}) = y(t_i + h) = y(t_i) + h\, y'(t_i) + \frac{y''(\xi_i)}{2!}\, h^2$$

con ξ en $]t_i, t_{i+1}[$

Despreciando el término de error $\dfrac{y''(\xi_i)}{2!}\, h^2$ y usando $(*)$ obtenemos

$$y(t_{i+1}) = y(t_i + h) \approx y(t_i) + h\, y'(t_i) = y(t_i) + h\, f\,(t_i, y(t_i))$$

Si w_i es la aproximación de $y(t_i)$, es decir $w_i = y(t_{i-1}) + h\, f\,(t_{i-1}, y(t_{i-1}))$, entonces la ecuación

$$\begin{cases} w_0 &= y_0 \\ w_{i+1} &= w_i + h\, f(t_i, w_i), \quad i = 0, 1, ..., n-1 \end{cases}$$

se llama ecuación de diferencia asociada al método de Euler.

Estimación del error

Teorema 3

Si $D = \{(t,y) : a \le t \le b, -\infty \le y \le \infty\}$ y si $f(t,y)$ es continua en D y satisface una condición de Lipschitz respecto a y en D con constante L entonces si existe una constante M tal que

$$|y''(t)| \le M, \ \ \forall\, t \in [a,b]$$

entonces para cada $i = 0, 1, 2, ..., n$,

$$|y(t_i) - w_i| \le \frac{hM}{2L}\left(e^{L(t_i - a)} - 1\right)$$

Nota: para calcular $|y''(t)|$ usamos regla de la cadena: $y''(t) = \dfrac{\partial f}{\partial t}(t,y) + \dfrac{\partial f}{\partial y}(t,y) \cdot y'(t)$. Posiblemente sea difícil obtener M dado que puede ser necesaria información acerca de $y(t)$.

Ejemplo 14

Consideremos el problema de valor inicial $y' = y - t^2 + 1,\ t \in [0,4], \quad y(0) = 0.5$.

Si $a = 0$, $b = 4$, $n = 10$ entonces $h = 0.4$ y $t_i = a + h\, i = 0.4\, i$

$$\begin{cases} w_0 &= 0.5 \\ w_{i+1} &= w_i + h\,(w_i - t_i^2 + 1) = w_i + 0.4\,\left(w_i - (0.4\,i)^2 + 1\right), \quad i = 0, 1, ..., n-1 \end{cases}$$

Nota: observe que, usando regla de la cadena, $y''(t) = \dfrac{\partial f}{\partial t} + \dfrac{\partial f}{\partial y} \cdot y'(t) = -2t + y(t) - t^2 + 1$. Así que no podemos usar el teorema anterior para estimar el error dado que no conocemos, en principio, $y(t)$.

En la siguiente figura se pueden apreciar las aproximaciones w_i para distintos valores de n

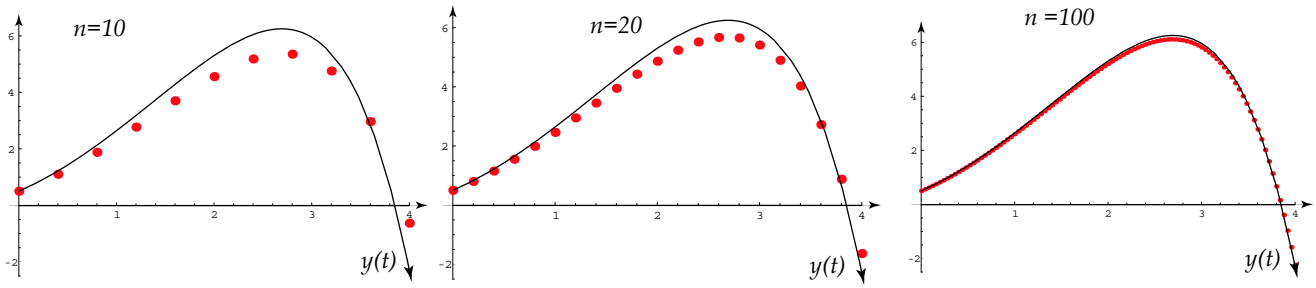


Figure 2.6: Aproximación con el método de Euler

Implementación con VBA

Como antes, usamos `clsMathParser` para leer $f(t, y)$. La implementación para ejecutar desde un botón es

	A	B	C	D	E	F	G
1							
2	y'(t) =	y - t^2+1	y(0)= 0,5	Valor inicial			Aplicar Euler
3	a	b	n	y0	t_i	w_i	y(ti)
4	0	4	100	0,5	0	0,5	0,5
5					0,04	0,56000000	0,56119461
6					0,08	0,62233600	0,62475647
7					0,12	0,68697344	0,69065157
8					0,16	0,75387638	0,75884456
9					0,2	0,82300743	0,82929862
10					0,24	0,89432773	0,90197542

Figure 2.7: Hoja de Excel

```
Private Sub CommandButton1_Click()
    Dim a, b, y0, yi, ti, ftiwi As Double
    Dim fty As String  'f(t,y)
    Dim ok As Boolean
    Dim Fun As New clsMathParser

    a = Cells(4, 1)
    b = Cells(4, 2)
```

```
n = Cells(4, 3)
y0 = Cells(4, 4) 'condición inicial
fty = Cells(2, 2)
ok = Fun.StoreExpression(fty)      'formula actual es 'f(t,y)'
If Not ok Then GoTo Error_Handler
h = (b - a) / n
yi = y0
ti = 0
For i = 0 To n
        Fun.Variable("y") = yi
        Fun.Variable("t") = ti
        ftiwi = Fun.Eval()
        Cells(4 + i, 6) = yi
        yi = yi + ftiwi * h
        Cells(4 + i, 5) = ti
        ti = a + (i + 1) * h
Next i

'-----------------------------------------------------------------
If Err Then GoTo Error_Handler
Error_Handler:        Cells(1, 1) = Fun.ErrorDescription
'-----------------------------------------------------------------
End Sub
```

Implementación con *Mathematica*

Usamos un `Array` para definir la lista {`w[1],w[2],...,w[n]`} de aproximaciones

```
n = 10;
a = 0; b = 4;
h = (b - a)/n;
wis = Array[w, n + 1] ; (* wis = {w[1], w[2], ..., w[n]}; wis[[i]] = w[i] *)
(* EDO *)
f[vt_, vw_] = vw - vt^2 + 1;
w[1] = 0.5;
ti = 0;
Do[
    w[i + 1] = w[i] + h*f[ti, w[i]];
    ti = a + i*h;
, {i, 1, n}];
wlista = Table[{a + i*h, wis[[i + 1]]}, {i, 0, n}];

(*---------------------- Imprimir -------------------------*)
y[vx_] = (vx + 1)^2 - 0.5*E^vx; (*Solución exacta*)
TableForm[Table[{a + i*h, wis[[i + 1]], y[a + i*h]}, {i, 0, n}],
          TableHeadings -> {Automatic, {"ti", "wi", "y(ti)"}}] // N

(*---------------------- Gráficos -------------------------*)
g1 = ListPlot[wlista, PlotStyle -> {PointSize[0.005], RGBColor[1, 0, 0]},
  DisplayFunction -> Identity];
g2 = Plot[{y[x], -0.01}, {x, a, b}, DisplayFunction -> Identity];
```

```
Show[{g1, g2}, PlotRange -> All, DisplayFunction -> $DisplayFunction]
```

La salida se puede ver en la figura que sigue

```
          ti        wi              y(ti)
1         0.         Q5             Q5
2         0.4       11              1.21409
3         0.8       1.876           2.12723
4         1.2       2.7704          3.17994
5         1.6       3.70256         4.28348
6         2.         455958         5.30547
7         2.4       5.18342         6.04841
8         2.8       5.35278         6.21768
9         3.2       4.7579          5.37373
10        36         2.96506        2.86088
11        4.       -0.632919       -2.29908
```

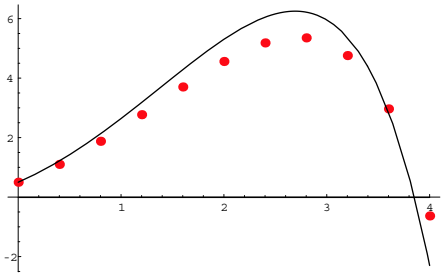


Figure 2.8: Método de Euler con $n = 10$

2.3.3 Métodos de Heun

De nuevo consideremos el problema

$$\frac{dy}{dx} = f(t, y), \quad a \leq t \leq b, \quad y(a) = y0 \qquad (*)$$

Para obtener una aproximación (t_1, y_1) de $(t_1, y(t_1))$ podemos usar el teorema fundamental del cálculo para integrar $y'(t)$ en $[t_0, t_1]$.

$$\int_{t_0}^{t_1} f(t, y(t))\, dt = \int_{t_0}^{t_1} y'(t)\, dt = y(t_1) - y(t_0)$$

de donde

$$y(t_1) = y(t_0) + \int_{t_0}^{t_1} f(t, y(t))\, dt$$

La integral la podemos aproximar usando regla del trapecio

$$y(t_1) \approx y(t_0) + \frac{h}{2}[\, f(t_0, y(t_0)) + f(t_1, y(t_1))\,]$$

ahora, cambiamos $y(t_1)$ por la aproximación que se obtiene con el método de Euler

$$y(t_1) \approx y(t_0) + \frac{h}{2}[\, f(t_0, y(t_0)) + f(t_1, y_0 + hf(t_0, y_0))\,]$$

prosiguiendo de esta manera obtenemos las sucesiones

$$\begin{cases} w_{k+1} = y_k + hf(t_k, y_k), \quad t_{k+1} = t_k + h \quad \text{(predictor)} \\ y_{k+1} = y_k + \dfrac{h}{2}[\, f(t_k, y_k) + f(t_{k+1}, w_{k+1}), \quad i = 0, 1, ..., n-1 \quad \text{(corrector)} \end{cases}$$

Nota: Aquí la aproximación a $y(t_k)$ es y_k .

■ Ejemplo 15

Consideremos el problema de valor inicial $y' = y - t^2 + 1,\ t \in [0, 4], \quad y(0) = 0.5$.

Los gráficos que siguen muestran las aproximaciones para distintos valores de n .

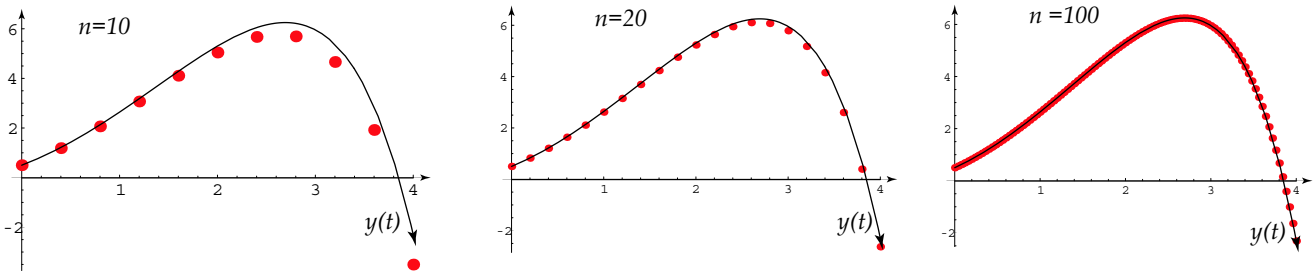


Figure 2.9: Método de Heun

Implementación con VBA

```
Private Sub CommandButton2_Click()
    Dim a, b, tk, y0, yk, ftkyk, ftkm1wkm1, wkm1 As Double
    Dim fty As String
    Dim ok As Boolean
    Dim Fun As New clsMathParser
    a = Cells(4, 1)
    b = Cells(4, 2)
    n = Cells(4, 3)
```

	A	B	C	D	E	F	G
1							Aplicar Heun
2	y'(t) =	y-t^2+1	y(0)= 0,5	Valor inicial			
3	a	b	n	y0	t_i	w_i	y(ti)
4	0	4	100	0,5	0	0,5	0,5
5					0,04	0,56116800	0,56119461
6					0,08	0,62470237	0,62475647
7					0,12	0,69056911	0,69065157
8					0,16	0,75873281	0,75884456
9					0,2	0,82915663	0,82929862

Figure 2.10: Implementación en Excel del método de Heun

```
    y0 = Cells(4, 4)
    fty = Cells(2, 2)
    ok = Fun.StoreExpression(fty)      'formula actual es 'f(t,y)'
    If Not ok Then GoTo Error_Handler
    h = (b - a) / n
    yk = y0
    tk = 0
    For k = 0 To n
        Fun.Variable("y") = yk
        Fun.Variable("t") = tk
        ftkyk = Fun.Eval()
        Cells(4 + k, 6) = yk
        wkm1 = yk + h * ftkyk     'w_{k+1}=yk+h*f(tk,yk)
        Cells(4 + k, 5) = tk
        tk = tk + h    't_{k+1}=tk+h

        Fun.Variable("y") = wkm1
        Fun.Variable("t") = tk
        ftkm1wkm1 = Fun.Eval()
            'y_{k+1}=yk + h/2[f(tk,yk) + f(t_{k+1},w_{k+1}) ]
        yk = yk + h / 2 * (ftkyk + ftkm1wkm1)
    Next k

    '---------------------------------------------------------------
    If Err Then GoTo Error_Handler
    Error_Handler:        Cells(1, 1) = Fun.ErrorDescription
    '---------------------------------------------------------------
End Sub
```

Implementación con *Mathematica*

Usamos dos `Array` para definir la lista {`w[1],w[2],...,w[n]`} y la lista {`yk[1],yk[2],...,yk[n]`} de aproximaciones

```
Clear[yk, w];
n = 20;
a = 0; b = 4;
h = (b - a)/n;
wks = Array[w, n + 1] ;  (* wis = {w[1], w[2], ..., w[n]};wks[[i]] = w[i] *)
```

```
yis = Array[yi, n + 1] ; (* yis = {yi[1], yi[2], ..., yi[n]};yis[[i]] = yi[i] *)

(* EDO *)
f[vt_, vy_] = vy - vt^2 + 1;
w[1] = 0.5;
yk = 0.5;
yi[1] = 0.5;
tk = 0;

Do[
    w[k + 1] = yk + h*f[tk, yk];
    tkm1 = tk + h;
    yk = yk + h/2*(f[tk, yk] + f[tkm1, w[k + 1]]);
    yi[k + 1] = yk; (*para imprimir*)
    tk = tkm1;
    , {k, 1, n}];

yilista = Table[{a + i*h, yi[i + 1]}, {i, 0, n}];

y[vx_] = (vx + 1)^2 - 0.5*E^vx; (*Solución exacta*)
TableForm[Table[{a + i*h, yi[i + 1], y[a + i*h]}, {i, 0, n}],

      TableHeadings -> {Automatic, {"tk", "yk", "y(tk)"}}] // N

g1 = ListPlot[yilista, PlotStyle -> {PointSize[0.01], RGBColor[1, 0, 0]},
              DisplayFunction -> Identity];
g2 = Plot[{y[x], -0.01}, {x, a, b}, DisplayFunction -> Identity];

Show[{g1, g2}, PlotRange -> All, DisplayFunction -> $DisplayFunction]
```

La salida se puede ver en la figura que sigue

	tk	yk	y(tk)
1	0.	05	05
2	0.4	1.188	1.21409
3	0.8	2.06544	2.12723
4	1.2	3.06965	3.17994
5	1.6	4.10788	4.28348
6	2.	504287	5.30547
7	2.4	5.67144	6.04841
8	2.8	5.69294	6.21768
9	3.2	4.66235	5.37373
10	36	1.92108	2.86088
11	4.	-3.50561	-2.29908

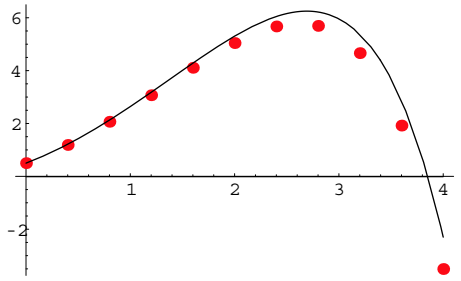


Figure 2.11: Método de Euler con $n = 10$

Bibliography

[1] Barrantes, H. et al *Introducción a la Teoría de Números*. EUNED, 1998.

[2] Bullen, S. et al *Excel 2002 VBA. Programmer's Reference*. Wiley Publishing, Inc., 2003.

[3] Burden, R.; Faires, J. *Análisis Numérico*. Thomson Learning, México, 2002.

[4] De Levie, R. *Advanced Excel for Scientific data analysis*. Oxford University Press. 2004

[5] Liengme, B. *A Guide to Microsoft Excel 2002 for Scientists and Engineers*. Butterworth Heinemann, 3rd, ed. 2002.

[6] Scheid, F. *Introducción a la Ciencia de las Computadoras*. Serie Schaum. Mc Graw Hill, 1982.

[7] Stewart, J. *Cálculo en una variable*. 4a. ed. Ed. Thomson Learning. 2001.

[8] Volpi, L. *Runtime Math Formulas Evaluation in Excel*. Foxes Team. 2003 (`http://digilander.libero.it/foxes/mathparser/Run_Time_%20Math_Formula_%20Excel.htm`).

[9] Volpi, L et al *"A Class for Math Expressions Evaluation in Visual Basic"*. Foxes Team. 2003 `http://digilander.libero.it/foxes/`

[10] Press, W. et al *Numerical recipes in C: the art of scientific computing*. 2nd ed. Cambridge Press. 1997

[11] Mora, W. Marín, M. *"Internet y educación matemática: Una exploración de diversas opciones computacionales para enseñar matemática a través de Internet."* ITCR. Tesis. 1992.