

ESCUELA SUPERIOR POLITECNICA DEL LITORAL

Instituto de Ciencias Matemáticas

" Hoja Electrónica Estadística Inferencial "

TESIS DE GRADO

Previa a la obtención del Título de:

INGENIERO EN ESTADISTICA INFORMATICA

Presentada por:

Alex Michael Moreno Salazar



GUAYAQUIL - ECUADOR

AÑO

2000

TRIBUNAL DE GRADUACION

Ing. Pablo Alvarez
SUBDIRECTOR DEL ICM

Ing. Luis Rodríguez
DIRECTOR DE TESIS

Ing. Carlos Villafuerte
VOCAL

Ing. Guillermo Gallegos
VOCAL

AGRADECIMIENTO

A todas las personas que de una u otra manera colaboraron en la realización de este trabajo, y de manera muy especial al Ing. Luis Rodríguez Ojeda, Director de Tesis, por su invaluable e incondicional ayuda.

DEDICATORIA

A mis padres, Alejandro y Teresa

A mis hermanas, Jéssica y Beatriz

DECLARACION EXPRESA

"La responsabilidad del contenido de esta tesis de grado, me corresponde exclusivamente; y el patrimonio intelectual de la misma a la ESCUELA SUPERIOR POLITECNICA DEL LITORAL"



ALEX MORENO SALAZAR

RESUMEN

La Hoja Electrónica Estadística, la cual hemos llamado HOELEST, es un sistema orientado al análisis de datos y cálculo estadístico inferencial. Esta Tesis contiene los fundamentos teóricos informáticos y estadísticos sobre los que se sustenta la creación de Hoelest.

Para la implementación de Hoelest se utilizó tecnología de desarrollo de excelente calidad, flexibilidad y facilidad de aprendizaje, considerando además las tendencias tecnológicas que permiten aprovechar al máximo las capacidades operacionales ofrecidas por dichas tecnologías.

La aplicación Hoelest es un claro indicio de nuestra capacidad para crear aplicaciones robustas. Aunque no se trate de una edición comercial, sino más bien estudiantil, facilita el camino a los investigadores de las nuevas tecnologías de desarrollo, ofreciéndoles una idea de la aplicabilidad y potencia de las mismas.

INDICE GENERAL

Pág.

RESUMEN II

NDICE GENERAL III

NDICE DE FIGURAS IV

NDICE DE TABLAS V

NTRODUCCION 1

UNA HOJA ELECTRONICA ESTADISTICA 2

1.1. ¿Qué es una Hoja Electrónica? 2

1.1.1. Celdas 4

1.1.2. Introducir datos en celdas 5

1.1.3. Formato de datos 5

1.1.4. Formato de celdas 5

1.1.5. Fórmulas 6

1.1.6. Sintaxis de fórmulas 7

1.1.7. Referencias de celda 8

1.1.8. Funciones para hojas de cálculo 8

1.1.9. Gráficos 8

1.2. ¿Qué es un Paquete Estadístico? 10

1.2.1. Análisis Univariado 10

1.2.2. Análisis Multivariado 11

1.3. La Simbiosis: Una Hoja Electrónica Estadística. 12

1.4. Justificación 13

1.5. Alcance 14

1.6. Ventajas 15

1.7. Desventajas 16

1.8. El Entorno de Desarrollo 16

1.9. Requisitos del Sistema para HOELEST 17

. EL MUNDO DE LA ORIENTACION A OBJETOS 19

2.1. La Orientación a Objetos 19

2.1.1. Un nuevo paradigma de programación 19

2.1.2. Razones fundamentales que están influyendo en la importancia de la POO. 21

2.1.3. Objetos 22

2.1.4. La Orientación a Objetos 23

2.1.5. Beneficios de las Tecnologías de Objetos (TO) 30

2.2. Conceptos de Programación Orientada a Objetos 32

2.2.1. La Programación Estructurada 32

2.2.2. La Programación Orientada a Objetos (POO) 36

2.2.3. Objetos 37

2.2.4. Clases 50

2.2.5. Herencia 54

2.2.6. Anulación/Sustitución 60

2.2.7. Sobrecarga 61

2.2.8. Ligadura dinámica 62

2.2.9. Reutilización con Orientación a Objetos 65

2.2.10. Polimorfismo 67

2.3. Lenguajes de Programación Orientados a Objetos (LPOO) 69

2.3.1. Evolución de los Lenguajes de Programación Orientados a Objetos 69

2.3.2. Clasificación de los Lenguajes Orientados a Objetos 70

2.3.3. Puros frente a híbridos 72

II. TECNOLOGIAS DE DESARROLLO 74

3.1. Introducción a Visual Basic 74

3.1.1. Conceptos de Visual Basic 74

3.1.2. Funcionamiento de Windows: ventanas, eventos y mensajes 74

3.1.3. Descripción de la programación controlada por eventos 75

3.1.4. Palabras claves y fundamentos de la POO en Visual Basic 77

3.1.5. Módulos de código 77

3.1.6. Procedimientos 79

3.1.7. Alcance del código 81

3.1.8. Usar variables, constantes y tipos de datos 81

3.1.9. Alcance de las variables 83

3.1.10. Duración de las variables 85
3.1.11. Variables de procedimiento 85
3.1.12. Tratar los errores en tiempo de ejecución 86
3.1.13. Programar con Objetos 88
3.2. El Modelo de Objetos Componentes 90
3.2.1. Introducción a COM 90
3.2.2. Ventajas del uso de COM 92
3.2.3. La especificación COM 93
3.2.4. Utilizar interfaces 95
3.2.5. La interfaz IUnkown 96
3.2.6. Comunicación entre cliente y servidor 97
3.2.7. COM, GUID y el registro del sistema 99
3.3. Componentes de código 100
3.3.1. Introducción a los componentes ActiveX 100
3.3.2. Ventajas de utilizar componentes de código ActiveX 102
3.4. Creación de objetos en Visual Basic 103
3.4.1. Módulos de clase 104
3.4.2. Crear métodos, propiedades y eventos 104
3.4.3. Utilizar procedimientos de propiedad para crear propiedades ... 105
3.4.4. La propiedad predeterminada 105
3.4.5. Establecer las propiedades de un módulo de clase 105
3.4.6. Crear métodos Friend 108

3.4.7. Crear un modelo de objetos 109

3.4.8. Almacenar datos en una colección 110

DISEÑO DEL COMPONENTE DLL ACTIVE X 111

4.1. Referencias del Proyecto 111

4.2. Diseño del Proyecto DistribucionesProy 112

4.2.1. Objetos del Componente DistribucionesProy 112

4.2.2. Módulos de clase 113

4.2.3. Descripción del Objeto Distribuciones 113

4.2.4. Módulos de código 117

4.3. Diseño del proyecto Matrices 117

4.3.1. Objetos del Componente Matrices 118

4.3.2. Módulos de clase 118

4.3.3. Descripción del Objeto Matrices 118

4.3.4. Módulos de código 130

4.3.5. Constantes con nombre 131

4.4. Diseño del modelo de objetos de EstadísticaInferencialDLL 131

4.4.1. Módulos de clase 132

4.4.2. Descripción del Objeto EstadísticaInferencial 132

4.5. Módulos de Código 143

4.5.1. Módulo FuncionesBase 143

4.5.2. Módulo mEstInferencial 143

4.6. Constantes con nombre 144

V. DISEÑO DEL COMPONENTE EXE ACTIVE X 145

5.1. Referencias del Proyecto 146

5.2. Diseño del Modelo de Objetos del Componente Exe ActiveX 146

5.2.1. Módulos de clase 146

5.2.2. Descripción del modelo de objetos 148

5.2.3. Objeto Origen 149

5.2.4. Objeto de conjunto Hojas 159

5.2.5. Objeto Hoja 161

5.2.6. Objeto Rango 169

5.2.7. Objeto ConfiguraciónPágina 176

5.2.8. Objeto Fuente 179

5.2.9. Objeto FuncionesHoja 181

5.2.10. Objeto de conjunto *Formulas* 182

5.2.11. Objeto *Formula* 183

5.2.12. Objeto de conjunto Variables 184

5.2.13. Objeto *Variable* 185

5.3. Diseño de la interfaz visual 186

5.3.1. Estilos de interfaz 186

5.3.2. La Ventana Principal 188

5.3.3. El formulario de la Hoja electrónica 197

5.3.4. Buscar o sustituir datos 198

5.3.5. Insertar celdas, filas o columnas 199

5.3.6. Ocultar y Mostrar hojas ocultas 200
5.3.7. Insertar variables 201
5.3.8. Administración de la Configuración de Hoelest 201
5.3.9. Prueba de Hipótesis Simple 204
5.3.10. Prueba de Hipótesis Dos Grupos 204
5.3.11. Correlación 205
5.3.12. Regresión Lineal 206
5.3.13. Diseño de Experimentos 206
5.3.14. Análisis de Varianza ANOVA 207
5.3.15. Kruskal Wallis 208
5.3.16. Prueba de Wilcoxon 209
5.4. Módulos de Código 210
5.4.1. Módulo Búsquedas 210
5.4.2. FuncionesBase 210
5.4.3. Módulo Principal 211
5.4.4. Módulo TrabajoConArchivos 212
5.4.5. EvalúaFórmulas 212
5.4.6. Funciones Internas 212
5.5. Constantes de Hoelest 213
. EN TIEMPO DE EJECUCION 215
6.1. Nociones Básicas de Hoelest 216
6.1.1. Cómo ejecutar Hoelest 217

6.1.2. Cómo ejecutar Comandos 218
6.1.3. Trabajo con archivos 219
6.1.4. Definición de variables de estudio 219
6.2. Correlación Lineal 221
6.2.1. Introducción 221
6.2.2. La covarianza de dos variables aleatorias 221
6.2.3. El Coeficiente de Correlación Lineal Simple de Pearson 224
6.2.4. Correlación lineal simple de Pearson en Hoelest 226
6.3. Pruebas de Hipótesis 229
6.3.1. Introducción 229
6.3.2. La Hipótesis Estadística 231
6.3.3. Riesgo Límite 233
6.3.4. Elementos de una Prueba Estadística 234
6.3.5. Errores en la prueba de hipótesis 234
6.3.6. Prueba de hipótesis para una media poblacional 235
6.3.7. Prueba de hipótesis para comparar dos medias poblacionales 239
6.4. Regresión Lineal 243
6.4.1. Introducción 243
6.4.2. La idea de Regresión Lineal 246
6.4.3. La Proporción de Variación Explicada 253
6.4.4. Regresión Lineal en Hoelest 258
6.5. Diseño de Experimentos 264

6.5.1. Introducción. 264
6.5.2. ¿Qué es el Diseño Experimental? 267
6.5.3. Procedimiento del Diseño de un Experimento 268
6.5.4. Diseño de Experimentos en Hoelest 268
6.6. Análisis de Varianza 273
6.6.1. Introducción 273
6.6.2. Procedimiento de Análisis de Varianza 273
6.6.3. Experimentos con un solo factor 275
6.6.4. Bloques Aleatorizados 284
6.6.5. Cuadrados Latinos 292
6.6.6. Experimentos con dos o más factores 300
6.7. Estadística No Paramétrica 311
6.7.1. Introducción 311
6.7.2. Prueba de Wilcoxon de Rangos con Signo para un Experimento Aparejado 312
6.7.3. Prueba H de Kruskal-Wallis para un Diseño Completamente Aleatorizado 318
ONCLUSIONES Y RECOMENDACIONES
IBLIOGRAFIA

INDICE DE FIGURAS

Pág.

igura 1. 1. Hoja Electrónica de Microsoft Excel. 4
igura 1. 2. Formato de Celdas 6
igura 1. 3. Gráfico de pastel. 9
igura 1. 4. Histograma de frecuencias de la aplicación Systat 11
igura 1. 5. Matriz de Correlación entre variables Mes y Enfermedad. 11
igura 2. 1. Principios básicos de la orientación a objetos 23
igura 2. 2. El objeto como caja negra 30
igura 2. 3. Programa Estructurado 33
igura 2. 4. Variables globales y locales 35
igura 2. 5. Descomposición de un programa en módulos (funciones) 36
igura 2. 6. Diagrama orientado a objetos 40
igura 2. 7. Mensajes entre objetos 41
igura 2. 8. Envío de un mensaje 42
igura 2. 9. Objetos emisor y receptor de un trabajo 43
igura 2. 10. Notación gráfica OMT de una clase y de un objeto 46
igura 2. 11. Objetos en Notación Yourdon/Coad 46
igura 2. 12. Invocación de un método 47
igura 2. 13. Objeto Avión 49
igura 2. 14. Diagrama de un objeto 53
igura 2. 15. El objeto Saludo1 53
igura 2. 16. Herencia Simple 56
igura 2. 17. Tipos de herencia 58
igura 2. 18. Anulación de atributos y métodos en clases derivadas 60
igura 2. 19. Métodos o funciones virtuales 64
igura 2. 20. Clasificación de lenguajes OO de Wegner 70
igura 2. 21. Lenguajes OO puros/híbridos 73
igura 3. 1. Formulario Buscar de la aplicación Hoelest 45
igura 3. 2. Comunicación Cliente-Servidor 45
igura 3. 3. Comunicación mediante código Proxy 45
igura 4. 1. Modelo sencillo de objetos. Componente DistribucionesProy .. 113
igura 4. 2. Modelo sencillo de objetos. Componente Matrices 118
igura 4. 3. Modelo de objetos. Componente EstadisticaInferencialDLL 132
igura 5. 1. Modelo de objetos de Hoelest 148
igura 5. 2. WordPad, una aplicación con interfaz de documento único 187

igura 5. 3. Excel, una aplicación con interfaz de documentos múltiples. ... 188
igura 5. 4 Formulario Principal de Hoelest. Modo Diseño ... 189
igura 5. 5. Menú Archivo. Diseño ... 190
igura 5. 6. Menú Edición. Diseño ... 191
igura 5. 7. Menú Ver. Diseño ... 191
igura 5. 8. Menú Estadísticas. Diseño ... 192
igura 5. 9. Menú Insertar. Diseño ... 193
igura 5. 10. Menú Formato. Diseño ... 193
igura 5. 11. Menú Ventana. Diseño ... 194
igura 5. 12. Menú Ayuda. Diseño ... 195
igura 5. 13. Barra Estándar. Diseño ... 195
igura 5. 14. Barra de formato. Diseño ... 195
igura 5. 15. Barra de fórmulas. Diseño ... 196
igura 5. 16. Barra de Estado. Diseño ... 196
igura 5. 17. Interfaz visual Hoja Electrónica. Formulario frmhoja ... 198
igura 5. 18. Formulario frmBuscar. Diseño ... 199
igura 5. 19. Formulario frmInsertarCeldas. Diseño ... 199
igura 5. 20. Formulario frmMostrar. Diseño ... 200
igura 5. 21. Formulario frmVariables. Diseño ... 201
igura 5. 22. Formulario frmOpciones. Diseño ... 202
igura 5. 23. Formulario PHSimple. Diseño ... 204
igura 5. 24. Formulario PHDosGrupos. Diseño ... 205
igura 5. 25. Formulario frmCorrelación. Diseño ... 205
igura 5. 26. Formulario frmRegresión. Diseño ... 206
igura 5. 27. Formulario frmDiseñoDeExperimentos. Diseño ... 207
igura 5. 28. Formulario frmANOVA. Diseño ... 208
igura 5. 29. Formulario frmKruskalWallis. Diseño ... 209
igura 5. 30. Formulario frmPruebaDeWilcoxon. Diseño ... 209
igura 6. 1. Aplicación Hoelest ... 217
igura 6. 2. Ventana para la Definición de Variables de Estudio ... 220
igura 6. 3. Observaciones dependientes e independientes para Y_1 y Y_2 ... 222
igura 6. 4. Ventana para el cálculo de Correlación Lineal ... 226
igura 6. 5. Ventana para efectuar una Prueba de Hipótesis Simple ... 237
igura 6. 6. Ventana para una Prueba de Hipótesis de Dos Medias ... 241
igura 6. 7. Representación gráfica de los datos. ... 245
igura 6. 8. Nube de puntos o diagrama de dispersión ... 248
igura 6. 9. Representación de una nube de puntos a través de una línea recta. ... 248
igura 6. 10. Ajuste de una línea recta a través de un conjunto de puntos ... 250
igura 6. 11. Variación observada de la variable Y ... 253
igura 6. 12. Variación de la variable Y que es explicada por la variable X ... 254
igura 6. 13. Significado gráfico de los coeficientes de determinación y alienación. ... 257
igura 6. 14. Ventana para el cálculo de Correlación Lineal ... 259

Figura 6. 15. Ventana para generar Matrices de Diseño..... 269
Figura 6. 16. Porción de la Suma Total de Cuadrados de las desviaciones..... 274
Figura 6. 17. Ventana ANOVA de Hoelest..... 280
Figura 6. 18. Ventana para la prueba de Wilcoxon..... 315
Figura 6. 19. Ventana para la prueba de Kruskal-Wallis..... 321

INDICE DE TABLAS

		Pág.
Tabla I	Estilos de programación	20
Tabla II	Alcance de las variables	83
Tabla III	Tipos de instrucciones Resume. Tratamiento de errores	88
Tabla IV	Valores de la propiedad Instancing de los módulos de clase de Hoelest	147
Tabla V	Configuración de Hoelest en el Registro del Sistema	203
Tabla VI	Constantes globales de Hoelest	211
Tabla VII	Constantes con nombre de Hoelest	214
Tabla VIII	Datos para el ejemplo de Correlación Lineal	228
Tabla IX	Tabla de Decisiones	235
Tabla X	Datos para el ejemplo de Regresión Lineal	260
Tabla XI	Datos para el experimento de Prueba de Dureza	271
Tabla XII	Datos típicos para un Experimento Unifactorial	276
Tabla XIII	Tabla de Análisis de Varianza de un modelo unifactorial	279
Tabla XIV	Datos del experimento de Resistencia a la Tensión	282
Tabla XV	Tabla de Análisis de Varianza para un Diseño Aleatorizado por Bloques Completo	289
Tabla XVI	Datos del Experimento de Prueba de Dureza	290
Tabla XVII	Tabla de Análisis de Varianza para un Diseño de Cuadrado Latino	296
Tabla XVIII	Datos del experimento de la Fórmula de Dinamita	298
Tabla XIX	Disposición General para un Diseño Bifactorial	302
Tabla XX	Tabla de Análisis de Varianza para un Diseño Bifactorial	307
Tabla XXI	Datos del experimento de Diseño de una Bateria	309
Tabla XXII	Datos para el ejemplo de la Prueba de Wilcoxon	316
Tabla XXIII	Datos para el ejemplo de la Prueba de Kruskal-Wallis	322

INTRODUCCION

La Hoja Electrónica Estadística, HOELEST, es un sistema integrado por dos componentes ejecutables de código. Uno de ellos se llama Hoelest.exe, proporciona la interfaz visual de la hoja electrónica, con un conjunto de funciones para la manipulación de datos. Debido a este componente se justifica la parte "Hoja Electrónica". El otro componente se llama EstadisticaInferencialDLL, proporciona todas las funciones para el análisis estadístico inferencial, de aquí se justifica la parte "Estadística" de Hoelest.

Hoelest combina características de aplicaciones de hojas electrónicas con las de las aplicaciones orientadas exclusivamente al análisis estadístico. Estas características, hacen de él un programa atractivo al usuario común (el que lo utiliza como una aplicación de escritorio) y, además enormemente flexible para los usuarios experimentados (el que lo utiliza mediante código externo).

En este documento se detalla el diseño de los dos componentes de Hoelest por separado, su interacción y su aplicabilidad, mediante ejercicios prácticos que demuestran lo fácil de su utilización.

CAPITULO 1

1. UNA HOJA ELECTRONICA ESTADISTICA

En este capítulo revisaremos las bases para la creación de HOELEST, una hoja electrónica estadística. HOELEST reúne características de programas de hojas electrónicas así como de programas estadísticos. Necesitaremos, por consiguiente, definir cada uno de ellos para posteriormente justificar su creación e indicar cómo se implementó.

Además se especifica el alcance de HOELEST, en lo que respecta a su parte relacionada con *Estadística Inferencial*.

1.1. ¿Qué es una Hoja Electrónica?

Las aplicaciones de Hojas Electrónicas usualmente tienen las siguientes características en común: Una hoja electrónica es una matriz rectangular de celdas, organizadas en columnas y filas. El monitor o pantalla actúa como una ventana que muestra una sección de la hoja electrónica

(usualmente una hoja electrónica es demasiado grande para poder verse en el monitor). Nos podemos mover dentro de la hoja por medio de las teclas de dirección. El usuario del programa "escribe" en la hoja electrónica, ubicándose en la celda deseada y tipeando información. Una celda puede contener caracteres, números, fechas, horas o fórmulas (cálculo con resultado numérico).

Básicamente, es una aplicación que nos permite manipular grandes cantidades de datos de acuerdo a nuestras necesidades. En la actualidad hay varias aplicaciones que proporcionan esta funcionalidad, entre ellas se encuentran Microsoft Excel, Lotus 1-2-3, etc.

Las hojas de cálculo –hojas electrónicas- se utilizan para mostrar y analizar datos. Pueden introducirse y modificarse los datos en las hojas de cálculo y pueden ejecutarse los cálculos basándose en los datos contenidos en ellas. Además, es posible crear gráficos basados en los datos de la hoja de cálculo.

Las hojas electrónicas se utilizan para solucionar problemas como elaboración de roles de pago, elaboración de estados de cuentas, presupuestos empresariales, etc., y todo aquello que pueda resolverse con una calculadora y papel.

La figura 1.1 presenta una hoja electrónica de Microsoft Excel.

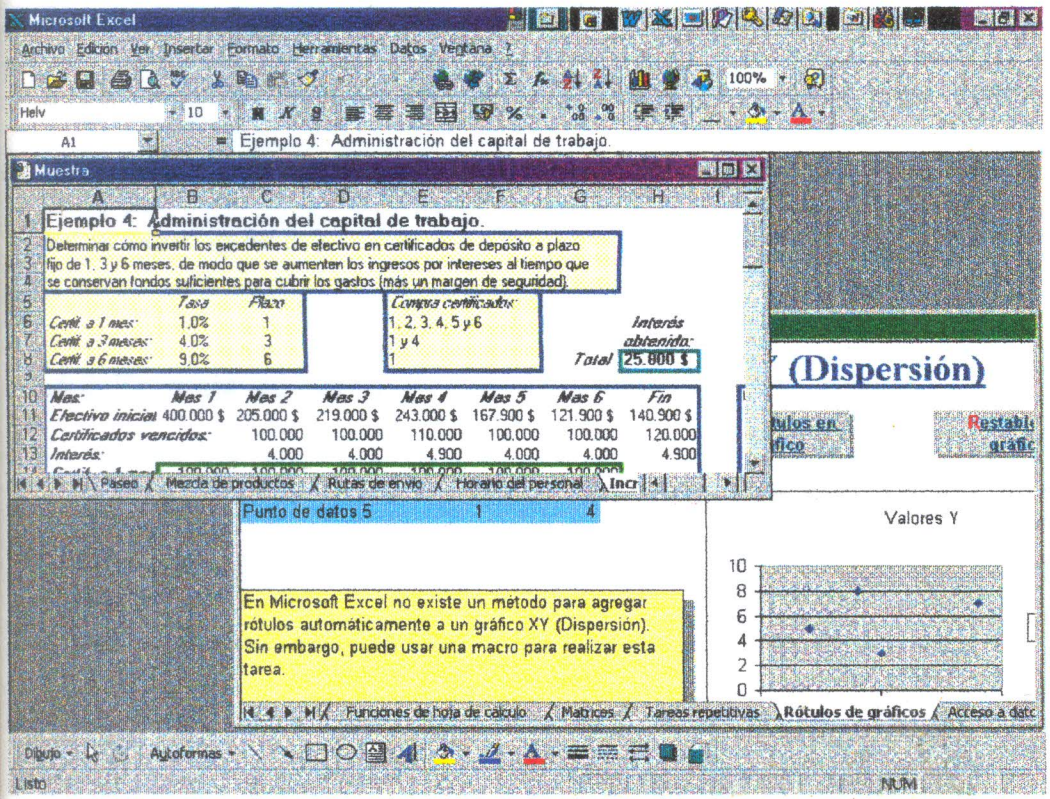


Figura 1. 1. Hoja Electrónica de Microsoft Excel.

1.1.1. Celdas

Una celda es un espacio definido dentro de la hoja, en la que puede introducirse información como datos, fórmulas.

Se identifica por una letra de columna y un número de fila, por ejemplo, podemos referirnos a la celda que se encuentra en la quinta fila y cuarta columna como: D5.

1.1.2. Introducir datos en celdas

Una celda puede contener números, texto, fechas, horas y fórmulas. Para introducir cualquiera de ellos basta con posicionarse en la celda en la que se desea insertar, por medio del ratón o con teclas direccionales, y escribir normalmente. Para introducir una fórmula el primer carácter debe ser el signo igual (=).

1.1.3. Formato de datos

Otra característica de las hojas electrónicas es que son capaces de dar un formato especial a sus datos, así por ejemplo, el número 2789787 puede presentarse de las siguientes formas:

Separado por miles	:	2.789.787
Con lugares decimales:		2789787,00

Aquellas celdas que contienen números, fechas u horas, pueden ser "formateados" de acuerdo a los requerimientos del usuario.

1.1.4. Formato de celdas

También es posible aplicar formatos a las celdas de la hoja electrónica, para que la misma tenga una apariencia más amigable, o para resaltar

ciertas partes que son de importancia crítica para el usuario.

Así, si existe una tabla de ventas trimestrales de 1999 por región (Costa, Sierra; Oriente, Insular), es posible que la misma tenga una apariencia más vistosa si se otorga un color de fondo a los títulos de Región. La figura 1.2 ilustra dicha tabla.

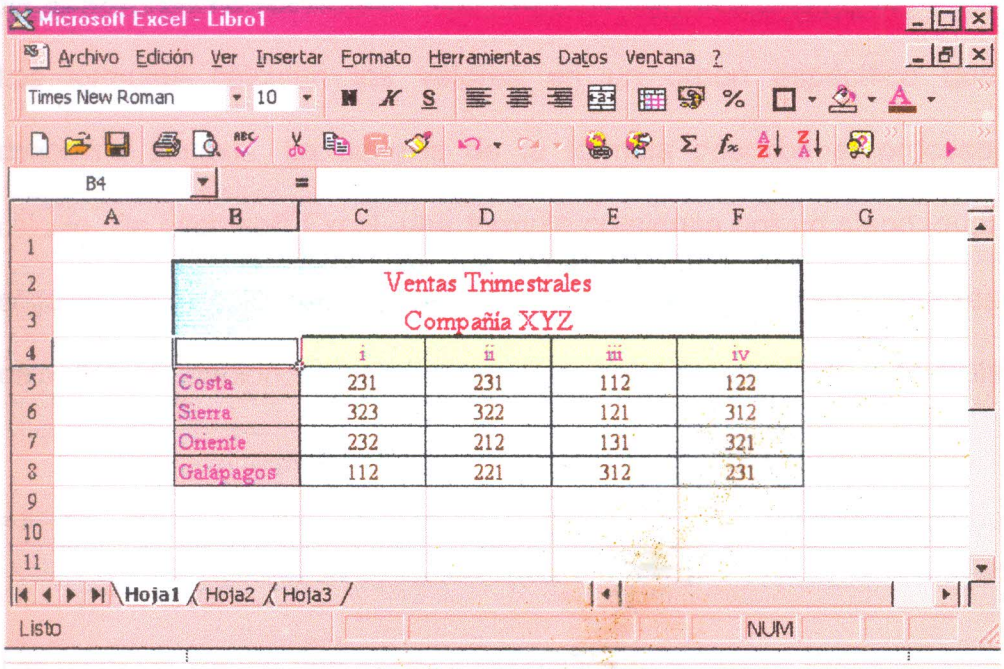


Figura 1. 2. Formato de Celdas

1.1.5. Fórmulas

Una fórmula es una ecuación que analiza los datos en una hoja de cálculo. La fórmulas realizan operaciones con los valores de la hoja de cálculo, como suma, multiplicación y comparación; también pueden

combinar los valores. Las fórmulas pueden hacer referencia a otras celdas en la misma hoja de cálculo. En el siguiente ejemplo, en Excel, se suma el valor de las celda B4 y 25 y, a continuación, el resultado se divide entre la suma de las celdas D5, E5 y F5.

=(B4+25)/SUMA(D5:F5)

1.1.6. Sintaxis de fórmulas

Las fórmulas calculan los valores en un orden específico que se denomina sintaxis. La sintaxis de la fórmula describe el proceso del cálculo. En Microsoft Excel, una fórmula comienza por un signo igual (=), seguido del cálculo que realiza. Por ejemplo, la siguiente fórmula resta 1 de 5. A continuación, se presenta en la celda el resultado de la fórmula.

=5 - 1

Las fórmulas siguen una sintaxis específica que incluye un signo igual (=) seguido de los elementos que van a calcularse (los operandos) y los operadores del cálculo. En Microsoft Excel, cada operando puede ser un valor que no cambie (un valor constante), una referencia de celda o de rango, un rótulo, un nombre o una función de la hoja de cálculo.

1.1.7. Referencias de celda

Una fórmula puede hacer referencia a una celda. La celda que contiene la fórmula se denomina celda dependiente ya que su valor depende del valor en la otra celda. Siempre que se cambie la celda a la que hace referencia la fórmula, cambiará también la celda que contiene la fórmula. Las fórmulas pueden hacer referencia a celdas, o rangos de celdas. La siguiente fórmula multiplica el valor en la celda B15 por 5. Cada vez que se cambie el valor en la celda B15 se volverá a calcular la fórmula.

*=B15*5*

1.1.8. Funciones para hojas de cálculo

Algunas aplicaciones, como Microsoft Excel, contienen muchas fórmulas predefinidas o integradas, denominadas funciones. Las funciones pueden utilizarse para ejecutar operaciones simples o complejas. La función más común en las hojas de cálculo es la función SUMA, que se utiliza para sumar rangos de celdas. Aunque puede crearse una fórmula para calcular el valor total de unas pocas celdas que contengan valores, la función de la hoja de cálculo SUMA calcula varios rangos de celdas.

1.1.9. Gráficos

Una característica muy atractiva es la capacidad de crear gráficos a partir

de los datos contenidos en las hojas electrónicas. Es de mucha utilidad presentar los datos mediante gráficas, ya que mediante un vistazo es posible detectar el comportamiento de los datos. Así por ejemplo, si realizamos una encuesta sobre la popularidad de dos candidatos presidenciales, podemos obtener datos que indiquen el número de votantes a favor de uno o del otro, así como el número de votantes indecisos. Los resultados pueden organizarse en una tabla, los mismos que se pueden representar mediante un gráfico de pastel, como lo muestra la figura 1.3.

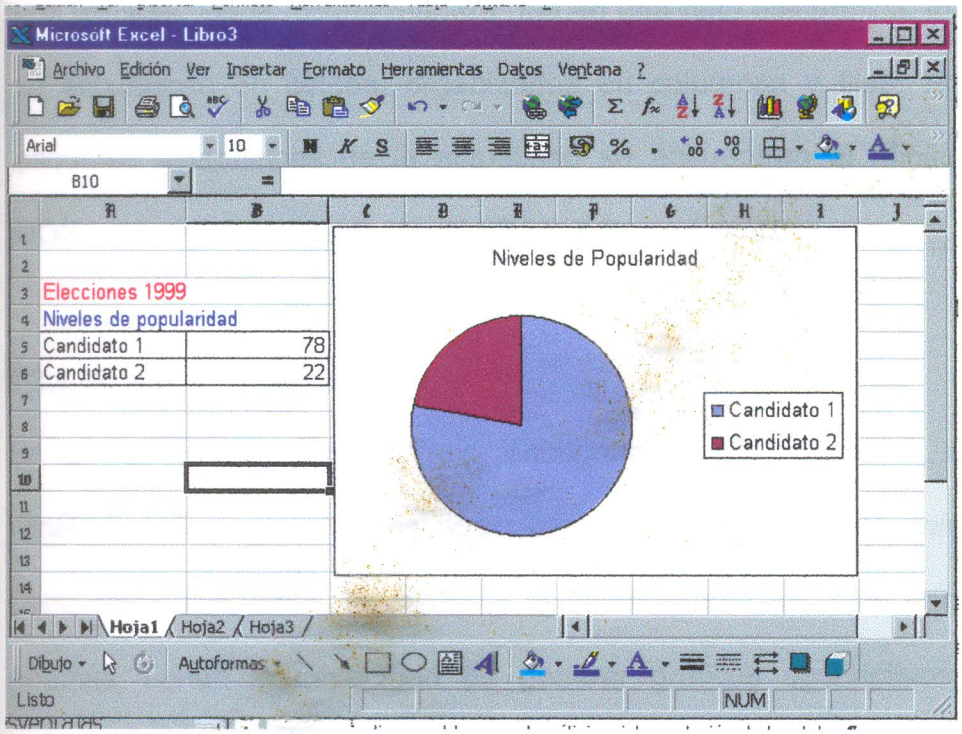


Figura 1. 3. Gráfico de pastel

Evidentemente, revisar el gráfico e interpretarlo, es más rápido que revisar la tabla. Si la cantidad de datos es mayor, o se presentan organizados con menor claridad en una tabla, entonces la utilidad de crear gráficos se vuelve indispensable para el análisis e interpretación de los datos.

1.2. ¿Qué es un Paquete Estadístico?

Un *paquete estadístico* es aquel que cuenta con características funcionales que nos permiten hacer un análisis estadístico de los datos. Cada columna de la hoja electrónica - que contenga datos - representa a una variable. De la misma manera que en una hoja electrónica, los datos se encuentran en celdas, organizados en columnas y filas. Entre los paquetes estadísticos de la actualidad se encuentran el Minitab para DOS y Windows, Systat X.X y el SPSS X.X entre otros.

1.2.1. Análisis Univariado

En un paquete estadístico se encuentran funciones para analizar variables individualmente, es decir, para realizar un análisis *univariado*. Entre estas funciones, la *media* y la *varianza* son las más conocidas. Además, tienen la capacidad de generar gráficos descriptivos, como histogramas de frecuencias, gráficos de barras, etc. La figura 1.4 muestra un histograma de frecuencias generado en el paquete estadístico Systat.

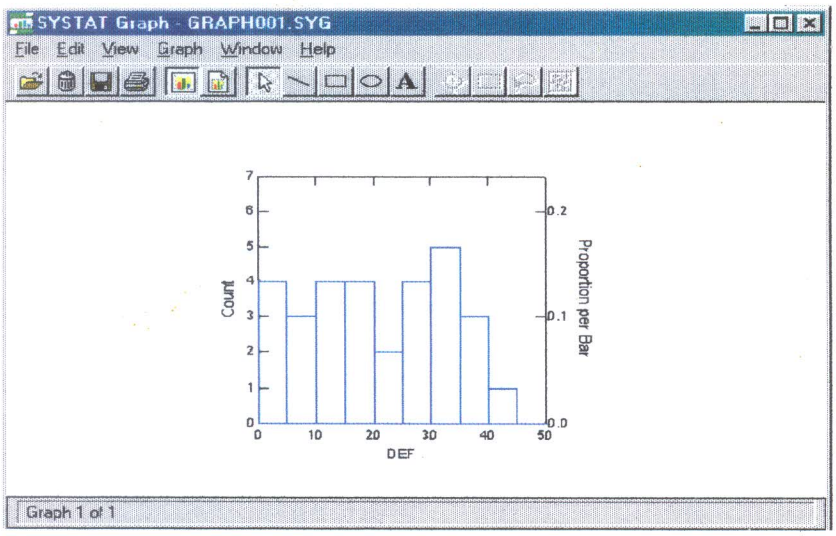


Figura 1. 4. Histograma de frecuencias de la aplicación Systat

1.2.2. Análisis Multivariado

De la misma manera, se encuentran funciones que permiten analizar un grupo de variables al mismo momento, o dicho de otra forma, permiten realizar un análisis *multivariado* de las mismas.

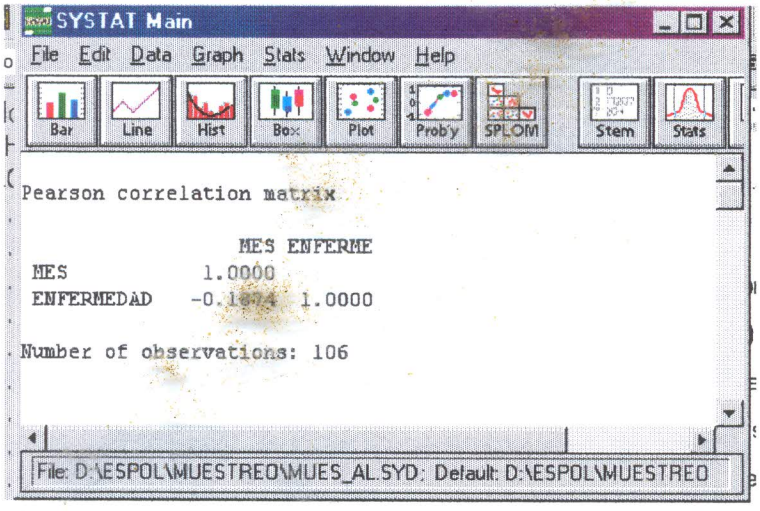


Figura 1. 5. Matriz de Correlación entre variables Mes y Enfermedad.

Así, es posible construir una matriz de covarianzas o una matriz de correlaciones. También se puede realizar un análisis factorial, un análisis de series de tiempo, de regresión lineal, etc.

1.3. La Simbiosis: Una Hoja Electrónica Estadística.

La creación de un programa que contenga características tanto de una *Hoja Electrónica* como las de un *Paquete Estadístico*, es el tema fundamental de la Tesis.

Por un lado, un programa que sólo maneja hojas electrónicas, no contiene funciones predefinidas especializadas en análisis estadístico de datos. En caso de que contara con algunas de ellas, el propósito principal de estos programas no es el análisis estadístico, sino prestar sus características para trabajos contables o la capacidad de automatizar cálculos sencillos

En contraste, los programas o paquetes estadísticos no proporcionan al usuario cierta funcionalidad, como la capacidad de editar fórmulas directamente en las celdas o la posibilidad de presentar sus datos en forma atractiva (dar formato), sea por medio visual o impreso. Usualmente se trabaja en forma alterna con un procesador de palabras o alguna aplicación de hojas electrónicas, para obtener la apariencia deseada cuando se necesita presentar un reporte.

La ***hoja electrónica estadística*** pretende reunir las características de estas dos clases de aplicaciones, siendo orientada al análisis estadístico de sus datos, al contar con funciones especializadas para ello, y con la capacidad de las hojas electrónicas para formatear los datos y la edición de fórmulas directamente, sin necesidad de cuadros de diálogos especiales. El usuario de la hoja electrónica estadística puede trabajar con ella del modo en que desee, por ejemplo, solo aplicar la funcionalidad como si fuera una hoja electrónica pura para realizar cálculos sencillos.

1.4. Justificación

Para justificar la creación de HOELEST debemos formularnos preguntas como: ¿Para qué crear la ***Hoja Electrónica Estadística***? ¿Quién la va a utilizar? La principal razón para la creación de la Hoja Electrónica Estadística es su flexibilidad, es decir, es modificable. El programador puede añadir, eliminar o modificar ciertas características de acuerdo a sus necesidades. Adicionalmente se trata de demostrar que podemos desarrollar localmente herramientas estadísticas comparables a otras existentes.

El programa estará disponible en los laboratorios del Instituto de Ciencias Matemáticas (ICM) de la Escuela Superior Politécnica del Litoral, y los usuarios evidentemente serán los estudiantes. Aquellos estudiantes que

deseen ir más allá, podrán modificar las características internas de HOELEST a fin de satisfacer sus necesidades, por ejemplo pueden añadir funciones financieras.

Además, el lenguaje utilizado para su desarrollo es Microsoft Visual Basic. Este lenguaje es muy fácil de aprender por programadores con experiencia en lenguajes estructurados, ya que sólo necesitarán dominar las extensiones orientadas a objetos. En esencia, HOELEST será una herramienta que podrá ser "moldeada" de acuerdo a los requerimientos de un usuario en particular.

1.5. Alcance

Para la implementación de HOELEST se dividieron responsabilidades. El tema que discutiremos en esta Tesis es la creación de la hoja electrónica y de las funciones relacionadas con la ***Estadística Inferencial*** (inferencia o predicción de un suceso). Recordemos que el objetivo de la Estadística es hacer inferencias con respecto a parámetros poblaciones desconocidos, basadas en la información obtenida mediante datos muestrales. De esta manera, con respecto a *Estadística Inferencial*, proporciona facilidades para:

a) Pruebas de Hipótesis (Simples y entre dos poblaciones)

b) Modelos Lineales (Regresión Lineal Simple y Múltiple)

c) Diseño de Experimentos (Generación de matrices de Diseño para Experimentos)

d) Análisis de Varianza (Unifactorial, Bloques, Cuadrados Latinos y MultiFactorial)

e) Estadística No Paramétrica (Pruebas de Kruskal-Wallis y de Wilcoxon)

f) Covarianzas y Coeficientes de Correlación Lineal Simple

1.6. Ventajas

Al ser un programa que reúne las características principales de aplicaciones de hojas electrónicas y de paquetes estadísticos puros, HOELEST es más amigable para el usuario común que un paquete estadístico tradicional. Recordemos que el propósito principal de HOELEST es el *Análisis Estadístico*. Las ventajas que ofrece son:

a) Es posible editar fórmulas sin necesidad de ingresar a un menú o a un cuadro de diálogo especial. (esta característica no suele presentarse en paquetes estadísticos)

b) Se presentan los datos con mayor vistosidad.

c) Es un programa flexible.

d) Esta a disposición de los estudiantes del ICM.

e) Proporciona un modelo de objetos a programadores experimentados.

f) Reutilización del código.

g) Obtiene las ventajas del Modelo de Componentes (ver capítulo 3).

Al ser desarrollado en Visual Basic 5.0, recibe beneficios como:

h) Desarrollo más rápido.

i) Calidad más alta.

j) Mantenimiento más fácil.

1.7. Desventajas

a) HOELEST no cuenta con algunas características de los programas de hojas electrónicas, como por ejemplo la revisión ortográfica.

b) Incompatibilidad con otras aplicaciones, como Microsoft Excel por ejemplo.

c) No existe una ventana de comandos propia. Para utilizarlo mediante código, se necesita que el usuario tenga Visual Basic 5.0 o versión superior instalado en su equipo.

1.8. El Entorno de Desarrollo

HOELEST fue desarrollado en **Visual Basic 5.0**, un lenguaje de

programación basado en objetos en el entorno Windows. Para su implementación se utilizaron *controles estándar* y *ActiveX*. Básicamente, el control *MSFlexGrid* de Microsoft proporcionó la interfaz gráfica de la hoja, así como características de *scroll* (movimiento a través de la hoja). Otros controles tales como el *TextBox* se utilizaron para edición de texto, números y fórmulas.

HOELEST es una *aplicación MDI*, es decir, permite que el usuario vea varios documentos al mismo tiempo y cada documento se presenta en su propia ventana. Los documentos o ventanas secundarias están contenidos en una ventana primaria, que proporciona un espacio de trabajo para todas las ventanas secundarias de la aplicación. Por ejemplo, HOELEST le permite crear y presentar ventanas de documentos múltiples de tipos diferentes. Cada ventana individual está confinada al área de la ventana primaria de HOELEST.

Cuando minimiza HOELEST, también minimiza todas las ventanas de documentos; sólo el icono de la ventana primaria aparece en la barra de tareas.

1.9. Requisitos del Sistema para HOELEST

Para la aplicación HOELEST se requiere el siguiente hardware y software:

a) Microsoft Windows NT 3.51 o posterior, o Microsoft Windows 95 o posterior.

b) Microprocesador 80486 o superior.

c) Pantalla VGA o de resolución superior compatible con Microsoft Windows.

d) 8 MB de RAM para aplicaciones

e) 16 MB de RAM para el entorno de desarrollo de Visual Basic (en caso de que el usuario desee modificar la estructura de HOELEST o se desee manipularlo mediante código externo).

CAPITULO 2

2. EL MUNDO DE LA ORIENTACION A OBJETOS

2.1. La Orientación a Objetos

2.1.1. Un nuevo paradigma de programación

La programación orientada a objetos (POO) se suele conocer como un nuevo paradigma de programación. Otros paradigmas conocidos son: el paradigma de programación imperativa (con lenguajes tales como Pascal o C), el paradigma de la programación lógica (PROLOG) y el paradigma de la programación funcional (Lisp).

Se entiende por paradigma al "Conjunto de teorías, estándares y métodos que juntos representan un medio de visualizar el mundo". En este sentido, la POO es un nuevo paradigma. La orientación a objetos fuerza a reconsiderar nuestro pensamiento sobre la computación, sobre lo que significa realizar computación y sobre cómo se estructura la información dentro del computador.

Jenkins y Glasgow sostienen que la mayoría de los programadores están familiarizados con un lenguaje y utilizan sólo un estilo de programación, por lo que les resulta difícil buscar métodos alternativos de resolución de un problema (4). Un estilo de programación, según la definición de Bobrow y Stefik es "Un medio de organización de programas sobre la base de algún modelo conceptual de programación y un lenguaje apropiado para hacer programas en un estilo claro" (4). Ellos sugieren que existen cuatro clases de estilos de programación:

Tabla I

Estilos de Programación

Orientados a procedimientos	Algoritmos
Orientados a objetos	Clases y objetos
Orientados a lógica	Expresado en cálculo de predicados
Orientados a reglas	Reglas If-then

No existe ningún estilo de programación idóneo para todas las clases de programación. La orientación a objetos se acopla a la simulación de situaciones del mundo real. En POO, las entidades centrales son los ***objetos,*** que son tipos de datos que encapsulan con el mismo nombre estructuras de datos y las operaciones o algoritmos que manipulan éstos.

2.1.2. Razones fundamentales que están influyendo en la importancia de la POO.

Algunas de las causas que están influyendo considerablemente en el notable desarrollo de las técnicas orientadas a objetos son:

a) La OO (Orientación a Objetos) es especialmente adecuada para realizar determinadas aplicaciones, sobre todo realización de prototipos y simulaciones de programas.

b) Los mecanismos de encapsulación de POO soportan un alto grado de reutilización de código.

c) Aumento espectacular de Lenguajes de Programación Orientados a Objetos (LPOO)

d) Interfaces de usuario gráficos (por iconos) y visuales. Las interfaces de usuario de una aplicación manipulan la entrada y salida del usuario. Por consiguiente, su función principal es la comunicación con el usuario final. La entrada al sistema se puede controlar a través de líneas de órdenes (enfoque utilizado por DOS), o alternativamente el usuario puede interactuar con el sistema, con construcciones de programación visuales.

Estas razones hacen fundamentales que dentro de las tendencias actuales de la ingeniería de software, el marco de POO se revela como el más adecuado para la elaboración del diseño y desarrollo de aplicaciones. Este marco se caracteriza por la utilización del diseño modular OO y la reutilización del software.

Los objetos pasan a ser los elementos fundamentales en este nuevo marco, en constrate con el de los subprogramas que lo han sido en los marcos tradicionales.

2.1.3. Objetos

Un objeto es sencillamente un tipo abstracto de datos al que se añaden importantes innovaciones en compartición de código y reutilización. Los mecanismos básicos de orientación a objetos son: objetos, mensajes y métodos, clases e instancias y herencia.

Una idea fundamental es la comunicación de los objetos a través del paso de mensajes. Además, se añaden dos mecanismos: 1. La *herencia* permite diferentes tipos de datos para compartir el mismo código, permitiendo una reducción en el tamaño del código y un incremento en la funcionalidad y, 2. El *polimorfismo* permite que un mismo mensaje pueda actuar sobre objetos diferentes y comportarse de modo distinto.

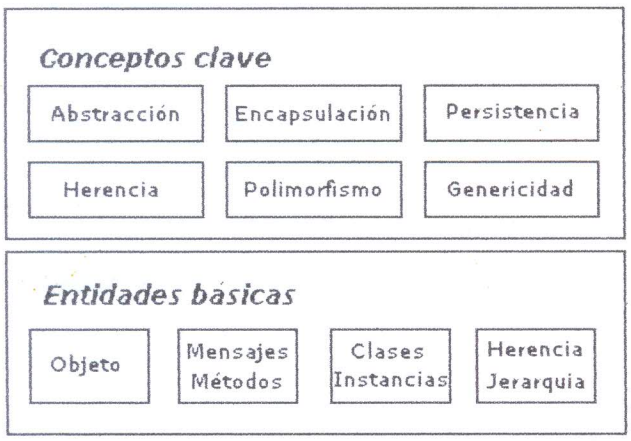


Figura 2. 1. Principios básicos de la orientación a objetos

La persistencia se refiere a la permanencia de un objeto, esto es la cantidad de tiempo para el cual se asigna espacio y permanece accesible en la memoria del computador. La figura 2.1 ilustra los conceptos clave en la POO.

2.1.4. La Orientación a Objetos

La orientación a objetos puede describirse como el conjunto de disciplinas (ingeniería) que desarrollan y modelizan software que facilitan la construcción de sistemas complejos a partir de componentes.

El atractivo intuitivo de la orientación a objetos es que proporciona conceptos y herramientas con las cuales se modela y representa el mundo real tan fielmente como sea posible.

Los conceptos y herramientas orientadas a objetos son tecnologías que permiten que los problemas del mundo real sean expresados de modo fácil y natural. Las técnicas orientadas a objetos proporcionan mejoras y metodologías para construir sistemas de software complejos a partir de unidades de software modularizado y reutilizable.

La tecnología orientada a objetos puede cubrir estos cambios y algunos otros más en el futuro. La orientación a objetos trata de cumplir las necesidades de los usuarios finales, así como las propias de los desarrolladores de productos software. Estas tareas se realizan mediante la modelización del mundo real.

El soporte fundamental es el ***modelo objeto***. Los cuatro elementos (propiedades) más importantes de este modelo son:

a) Abstracción
b) Encapsulación
c) Modularidad
d) Jerarquía

Si alguno de estos elementos no existe, se dice que el modelo no es orientado a objetos.

2.1.4.1. Abstracción

La ***abstracción*** fue un concepto introducido en programación estructurada. Se define como la capacidad para examinar algo sin preocuparse de sus detalles internos. En un programa estructurado es suficiente conocer la tarea específica que hace un procedimiento dado, y no es tan importante cómo se realiza esa tarea. Esta propiedad se conoce como *abstracción funcional.*

La *abstracción de datos* es a los datos lo que la *abstracción funcional* es a las operaciones. Con abstracción de datos, las estructuras y los ítems de datos se pueden utilizar sin tener que preocuparse sobre los detalles exactos de su realización (implementación). Por ejemplo, los números en coma flotante son una abstracción en los lenguajes de programación; no tiene que preocuparse de la representación binaria exacta de un número en coma flotante cuando se le asigna un valor.

Lo más importante es que los números en coma flotante actúen de una manera correcta y comprensible. La abstracción de datos libera al programador de preocuparse sobre los detalles no esenciales. La abstracción de datos ha existido formalmente en todos los lenguajes de programación para elementos complicados, tales como los citados números de coma flotante. Sin embargo, han sido los lenguajes modernos

como Modula-2, Ada y los lenguajes orientados a objetos (puros como Smalltalk o híbridos como C++, Turbo Pascal 5.5/6.0/7.0 y Visual Basic 4.0/5.0/6.0) quienes han permitido definir sus propios tipos abstractos de datos.

Tipos abstractos de datos (TAD) es un tipo de datos definido por el programador que se puede manipular de un modo similar a los tipos de datos incorporados al sistema. Al igual que con los tipos incorporados al lenguaje, un TAD pertenece a un conjunto de valores legales de datos (rango) y se pueden realizar sobre esos valores un número de operaciones primitivas. Los usuarios pueden crear variables con valores cuyos rangos están en el conjunto de valores legales, y puede operar sobre esos valores las operaciones definidas. Por ejemplo, un programador experimentado puede definir una pila (*stack*) como un tipo abstracto de datos, y las operaciones típicas en la pila (*meter, sacar y limpiar*) como las operaciones o funciones (métodos) del tipo de dato.

El elemento clave de la programación orientada a objetos es la *clase*. Una *clase* se puede definir como una descripción abstracta de un grupo de objetos, cada uno de los cuales se diferencia por su *estado* específico y por la posibilidad de realizar una serie de *operaciones*.

2.1.4.2. Encapsulación.

La *abstracción* y el *encapsulamiento* son conceptos complementarios: la abstracción se enfoca a la vista exterior del cuerpo y el encapsulamiento, también conocido como *ocultación de la información,* previene a otros objetos ver su interior, donde el comportamiento de la abstracción se ha realizado (implementado).

La encapsulación o encapsulamiento permite la división de un programa en módulos. Estos módulos se implementan mediante *clases*, de forma que una clase representa la encapsulación de una abstracción.

Un concepto fundamental en las clases es el ocultamiento de los detalles de su *implementación*. En la práctica, esto significa que cada clase debe tener dos partes: una ***interfaz*** y una ***implementación***. La *interfaz* de una clase se dedica sólo a la vista exterior y la *implementación* comprende la representación de la abstracción, así como los mecanismos que realizan los comportamientos deseados.

Por ejemplo, consideremos una biblioteca de clases gráficas que puede tener diferentes *implementaciones* para diferentes tipos de pantallas de video. Un programa que utilice clases gráficas se puede escribir de tal modo que sea compilable, sin modificaciones, en cualquier computadora

a la que se transporta su *implementación* de la biblioteca. De este modo, las clases suelen tener dos partes: parte pública (accesible fuera de la clase) y parte privada (sólo accesible dentro de la clase). Los datos y métodos se puede definir indistintamente en cada parte.

2.1.4.3. Modularidad

La *modularidad* es la propiedad que permite subdividir una aplicación en partes pequeñas (llamadas módulos), cada una de las cuales debe ser tan independiente como sea posible de la aplicación en sí y de las restantes partes.

La modularización, consiste en dividir un programa en módulos que se puedan compilar por separado, pero que tienen conexiones con otros módulos.

2.1.4.4. Jerarquía

La *jerarquía* es una propiedad que permite una ordenación de las abstracciones. Las dos jerarquías más importantes de un sistema complejo son: estructura de clases (jerarquía <<es-un>>: generalización /especialización) y estructura de objetos (jerarquía <<parte-de>>: agregación).

Las jerarquías de generalización/especialización se conocen como herencia. Básicamente, la herencia define una relación entre clases, en donde una clase comparte la estructura o comportamiento definido en una o más clases.

2.1.4.5. Polimorfismo

El polimorfismo permite referirse a objetos de clases diferentes mediante el mismo elemento de programa y realizar la misma operación de diferentes formas, según sea el objeto que es referenciado en ese momento.

Por ejemplo, cuando se describe la clase *mamíferos* se puede observar que la operación *comer* es una operación fundamental en la vida de los mamíferos, de modo que cada tipo de mamífero debe poder realizar la operación o función comer. Por otra parte, una cabra que pasta en un campo, un niño que se come un bombón y un león que devora a otro animal, son diferentes formas que utilizan los distintos mamíferos para realizar la misma operación (comer).

El polimorfismo implica la posibilidad de tomar un objeto de un tipo (mamífero) e indicarle que ejecute *comer*, esta acción se ejecutará de diferente forma, según sea el objeto mamífero al que se aplica.

2.1.5. Beneficios de las Tecnologías de Objetos (TO)

La principal razón para introducir métodos de Tecnología de Objetos en los procesos de desarrollo es, sin lugar a dudas, los beneficios de dichas TO: aumento de la fiabilidad y productividad del desarrollador. La fiabilidad se puede mejorar, debido a que cada objeto simplemente es una caja negra (figura 2.2) con respecto a objetos externos con los que debe comunicarse. Las estructuras de datos internos y métodos se pueden mejorar sin afectar a otras partes de un sistema.

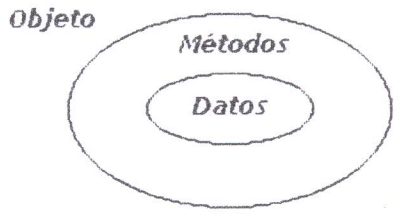


Figura 2. 2. El objeto como caja negra

La productividad del desarrollador se puede mejorar, debido a que las clases de objetos se pueden hacer reutilizables de modo que cada subclase o instancia de un objeto puede utilizar el mismo código de programa para la clase. Por otra parte, esta productividad también aumenta, debido a que existe una asociación más natural entre objetos del sistema y objetos del mundo real.

Taylor considera que los beneficios del modelado y desarrollo de objetos son los que se detallan a continuación (4):

a) Desarrollo más rápido
b) Calidad más alta
c) Mantenimiento más fácil
d) Costo reducido
e) Incremento con escalabilidad
f) Mejores estructuras de información
g) Incremento de adaptabilidad

Los beneficios genéricos de las tecnologías de objetos son:

a) Reutilización: las clases se construyen a partir de otras clases
b) Sistemas más fiables
c) Proceso de desarrollo más rápido
d) Desarrollo más flexible
e) Modelos que reflejan mejor la realidad
f) Mejor independencia e interoperatividad de la tecnología
g) Mejor informática distribuida y cliente-servidor
h) Bibliotecas de clases comerciales disponibles
i) Mejores relaciones con los clientes

j) Mejora la calidad del producto software terminado.

Sin embargo, Taylor también considera algunos inconvenientes, aunque algunos ya han sido superados o al menos reducido su impacto (4).

a) Inmadurez de la tecnología (hoy en día no se considera así)
b) Necesidades de estándares (el grupo OMG es una realidad)
c) Necesidad de mejores herramientas.
d) Velocidad de ejecución
e) Disponibilidad de personal cualificado
f) Costo de conversión.
g) Soporte para modularidad a gran escala.

2.2. Conceptos de Programación Orientada a Objetos

2.2.1. La Programación Estructurada

La idea fundamental de la programación estructurada es romper un programa en unidades más pequeñas, denominadas funciones (subrutinas, subprogramas o procedimientos) que realizan tareas menos complejas.

Cada una de las funciones tiene un propósito claramente definido, así como una interfaz a las otras funciones del programa. La información se

pasa entre funciones utilizando parámetros, y las funciones pueden tener datos locales a los cuales no se puede acceder fuera del ámbito del procedimiento.

La programación estructurada (figura 2.3) utiliza el método descendente o el refinamiento sucesivo, y cada una de las funciones se invocan sucesivamente. Los programas estructurados son más fáciles de escribir y mantener que los programas no estructurados.

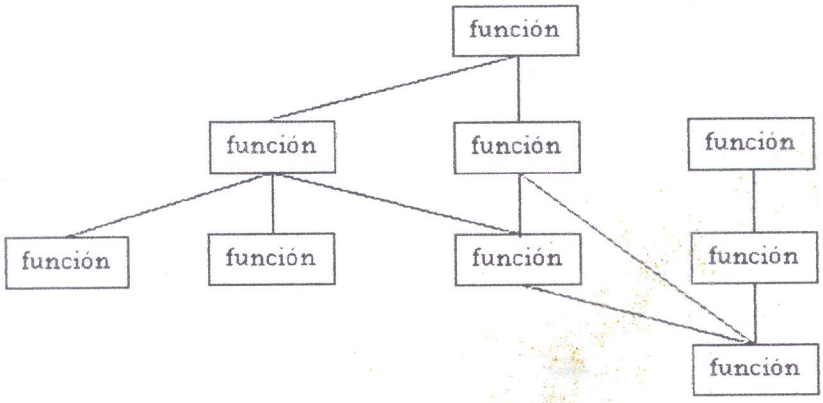


Figura 2. 3. Programa Estructurado

En un programa estructurado es suficiente conocer que un procedimiento dado realiza una tarea específica. El cómo se realiza esta tarea no es importante, sino conocer cómo se utiliza correctamente la función y lo que hace. Esto es la *abstracción*.

Sin embargo, a medida que el tamaño de los programas aumenta y se hacen muy grandes y más complejos, el énfasis se pone en los tipos de datos que se procesan. Las estructuras de datos en un programa se hacen tan importantes como las operaciones realizadas por ellos.

Los tipos de datos se procesan en muchas funciones, dentro de un programa estructurado y, cuando se producen cambios en esos tipos de datos, las modificaciones se deben hacer en cada posición que actúa sobre esos tipos de datos dentro del programa. Esta tarea, consume enormemente el tiempo del programador.

Esta circunstancia se produce esencialmente porque muchas funciones comparten datos (variables globales). Así por ejemplo, si se añaden nuevos datos, se necesitarán modificar todas las funciones que acceden a los datos, de modo que ellas también puedan acceder. Se necesita, por consiguiente, un método para restringir el acceso a los datos, *ocultarlos* de funciones no deseadas. La figura 2.4 ilustra el tipo de acceso a los datos.

Los programas basados en funciones son difíciles de diseñar, básicamente porque sus componentes principales –funciones y estructuras de datos- no modelan bien el mundo real. Existen otros

problemas, como la dificultad de crear nuevos tipos de datos (pese a que la mayoría de los lenguajes soportan tipos de datos definidos por el usuario).

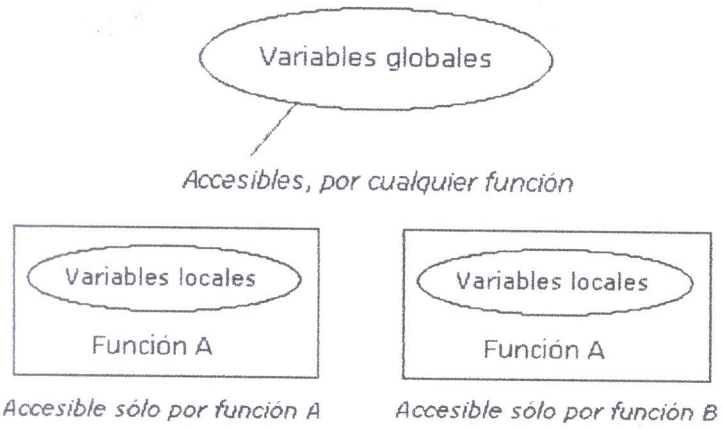


Figura 2. 4. Variables globales y locales

Los lenguajes de programación típicamente tienen tipos de datos incorporados: enteros, caracteres, bytes, etc. Los tipos definidos por el usuario y la facilidad para crear tipos abstractos de datos en determinados lenguajes es la propiedad conocida como *extensibilidad*. Los lenguajes tradicionales no son normalmente extensibles, y eso hace que los *programas tradicionales* sean más complejos de escribir y mantener.

El último inconveniente que consideramos se produce cuando un equipo de programadores trabajan en equipo en una aplicación. En un programa

estructurado, a cada programador se le asigna la construcción de una tarea específica de funciones y tipos de datos (figura 2.5).

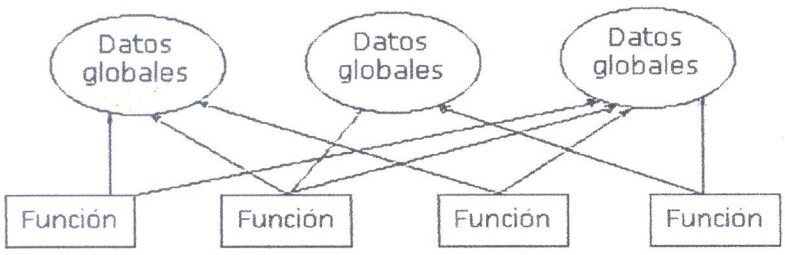


Figura 2. 5. Descomposición de un programa en módulos (funciones)

Dado que diferentes programadores manipulan funciones independientes que comparten datos, los cambios que un programador hace a los datos deben ser reflejados en el trabajo del resto del equipo. Los problemas y los errores de comunicación se reflejan en el diseño final.

2.2.2. La Programación Orientada a Objetos (POO)

La Programación orientada a objetos enfatiza en los datos, al contrario que la programación estructurada que enfatiza en los algoritmos. Adoptemos la definición de la POO que hizo Grady Booch (4), autor del método de diseño orientado a objetos, así:

Un método de implementación en el que los programas se organizan como colecciones cooperativas de objetos, cada uno de los cuales representan una instancia de una clase, y cuyas clases son todas miembros de una jerarquía de clases unidas mediante relaciones de herencia.

Existen tres importantes partes en la definición: la programación orientada a objetos 1) utiliza objetos, no algoritmos, como bloques de construcción lógicos (jerarquía de objetos); 2) cada objeto es una instancia de una clase, y 3) las clases se relacionan unas con otras por medio de relaciones de herencia. La potencia real de los objetos, es la capacidad para crear otros objetos, esto es la herencia, y es el mecanismo que ayuda a construir programas que se modifican fácilmente y se adaptan a aplicaciones diferentes.

Los conceptos fundamentales de programación son: *objetos, clases, herencia, mensajes y polimorfismo.* Los programas orientados a objetos constan de objetos.

2.2.3. Objetos

La combinación de datos y funciones que operan sobre esos datos, en una sola entidad, se denomina objeto. Por ello, un objeto contiene los datos de los lenguajes de programación tradicionales, tales como números, cadenas, etc., así como funciones o subrutinas que operan sobre ellos.

Las funciones de un objeto (llamados métodos en Visual Basic, funciones miembro en C++) son el único medio de acceder a los datos privados de

un objeto. Si se desea leer un elemento dato de un objeto se llama a la función miembro del objeto, se lee el elemento y se devuelve el valor. No se puede acceder a los datos directamente, debido a que están ocultos, y eso asegura que no se puedan modificar accidentalmente por otras funciones externas al objeto.

Los datos y los métodos asociados se dicen que están encapsulados en una única entidad o módulo. La encapsulación de datos y ocultación de datos son términos importantes en la descripción de lenguajes orientados a objetos. Si se desea modificar los datos de un objeto, se conoce exactamente cuales son las funciones que interactúan con el mismo. Ninguna otra función puede acceder a los datos. Esta característica simplifica la escritura, depuración y mantenimiento del programa.

2.2.3.1. Ejemplos de Objetos

El término objeto tiene el mismo significado que un nombre o frase nominal. Es una persona, un lugar o una cosa. La respuesta sólo está limitada a nuestra imaginación, así: objetos del mundo real son: personas, muebles, autos, planetas, animales. Elementos de interfaces gráficas de usuario: ventanas, menús, cuadros de diálogo, ratón. Tipos de datos definidos por el usuario: puntos de sistemas de coordenadas, datos complejos.

2.2.3.2. Métodos y mensajes

Un programa orientado a objetos consiste en un número de objetos que se comunican unos con otros llamando a funciones miembro o métodos, los cuales residen en los objetos y determinan como actúan cuando reciben un mensaje. Un ***mensaje*** es la acción que hace el objeto. Un ***método*** es el procedimiento o función que se invoca para actuar sobre el objeto. Un método especifica *cómo* se ejecuta un mensaje.

El conjunto de mensajes a los cuales puede responder un objeto se denomina protocolo del objeto. Por ejemplo, el protocolo de un icono puede constar de mensajes invocados por el click de un botón del ratón cuando el usuario localiza un puntero sobre un icono.

La estructura interna de un objeto está oculta a los usuarios y programadores. Los mensajes que recibe el objeto son los únicos conductos que conectan al objeto con el mundo externo. Los datos de un objeto están disponibles para ser manipulados sólo por los métodos del propio objeto.

Cuando se ejecuta un programa orientado a objetos ocurren tres sucesos. Primero, los objetos se crean a medida que se necesitan. Segundo, los mensajes se mueven de un objeto a otro (o desde el usuario a un objeto)

a medida que el programa procesa información internamente o responde a la entrada del usuario. Tercero, cuando los objetos ya no son necesarios, se borran y se libera la memoria. La figura 2.6 representa un diagrama orientado a objetos; cada objeto tiene sus propios datos y solamente sus funciones miembro (métodos) pueden manipularlos.

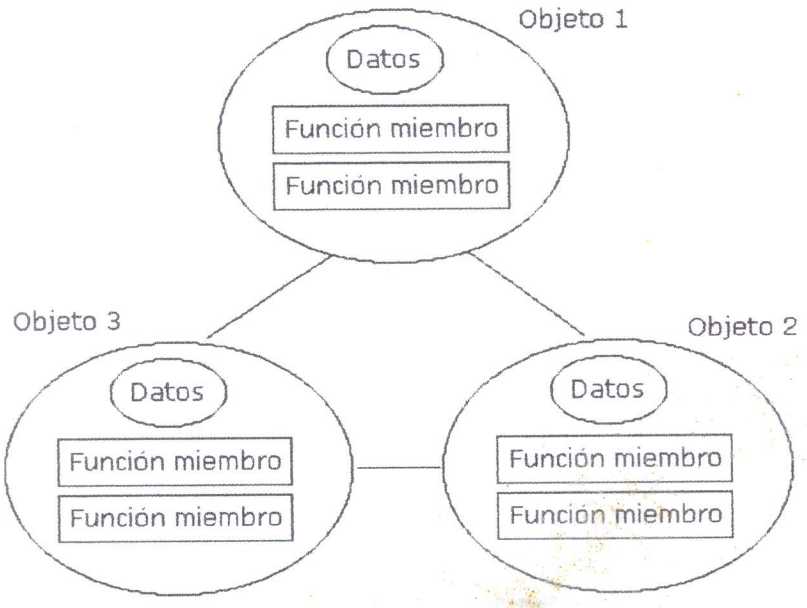


Figura 2. 6. Diagrama orientado a objetos

2.2.3.3. Comunicación entre objetos: Los Mensajes

El mensaje es esencialmente una orden que se envía a un objeto para indicarle que realice alguna acción. Los objetos se comunican entre sí enviando mensajes, al igual que sucede con las personas. Cuando un objeto recibe un mensaje entonces se invocan métodos para tratar la

acción solicitada. En terminología de programación tradicional, un mensaje es una llamada a una función (figura 2.7). Los mensajes pueden venir de otros objetos o desde fuentes externas, tales como un ratón o un teclado. Los mensajes juegan un papel crítico en programación orientada a objetos; sin ellos los objetos que se definen no se podrán comunicar entre sí.

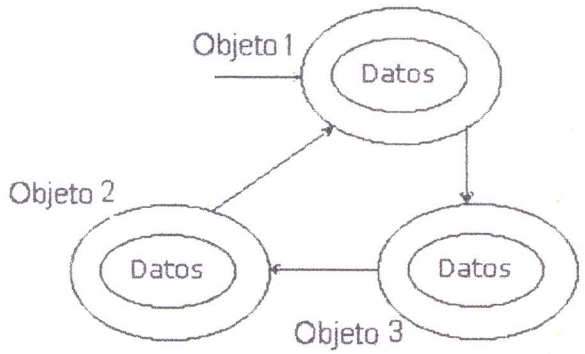


Figura 2. 7. Mensajes entre objetos

Una aplicación windows es un buen ejemplo de cómo se emplean los mensajes para comunicarse entre objetos. El usuario pulsa un botón para enviar mensajes a otros objetos que realizan una función específica.

Si se pulsa el botón derecho del ratón sobre un objeto, se despliega entonces un menú popup, entonces el usuario puede escoger de entre cualquiera de ellos, y de acuerdo al elemento seleccionado se efectúa una operación específica, como copiar o pegar del portapapeles.

2.2.3.3.1. Activación de objetos

A los objetos sólo se puede acceder a través de su interfaz pública. ¿Cómo se permite el acceso a un objeto?. Un objeto accede a otro objeto enviándole un mensaje. La figura 2.8 ilustra el envío de un mensaje al objeto carro.

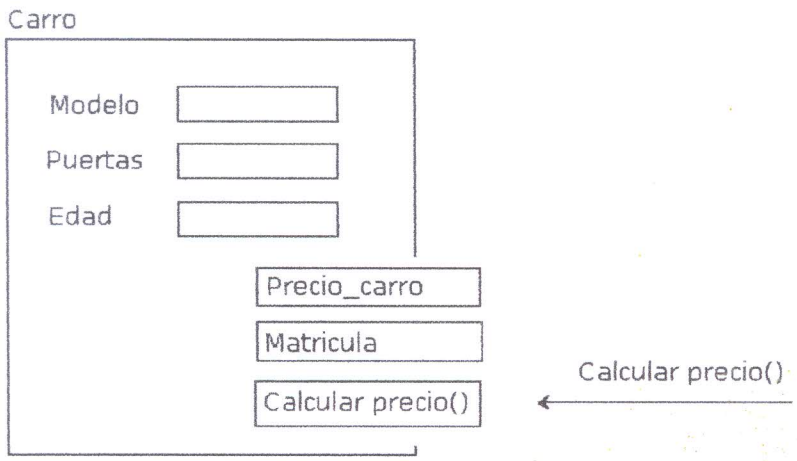


Figura 2. 8. Envío de un mensaje

2.2.3.3.2. Mensajes

Un mensaje es una petición de un objeto a otro objeto al que le solicita ejecutar uno de sus métodos. Por convenio, el objeto que envía el mensaje se denomina emisor y el objeto que recibe la petición se denomina receptor. La figura 2.9 ilustra la relación entre un objeto emisor y otro receptor.

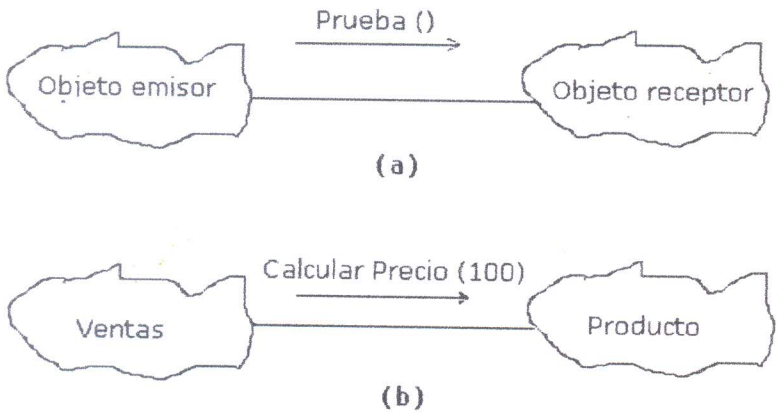


Figura 2. 9. Objetos emisor y receptor de un trabajo

Estructuralmente un mensaje consta de tres partes:

1. *Identidad* del receptor. 2.El *método* que se ha de ejecutar y 3. *Información especial* necesaria para realizar el método invocado (argumentos o parámetros requeridos)

Cuando un objeto está inactivo y recibe un mensaje se hace activo. El mensaje enviado por otros objetos o fuentes tiene asociado un método que se activará cuando el receptor recibe dicho mensaje. La petición no especifica *cómo* se realiza la operación. Tal información se oculta siempre al emisor. El conjunto de mensajes a los que responde un objeto se denomina *comportamiento del objeto*. No todos los mensajes de un objeto responden; es preciso que pertenezcan al interfaz accesible.

La secuencia de acontecimientos es que el emisor envía su mensaje; el receptor ejecuta el método apropiado, a continuación, el receptor devuelve algún tipo de respuesta al emisor para reconocer el mensaje y devolver cualquier información que se haya solicitado.

2.2.3.3.3. Paso de mensajes

Esencialmente el protocolo de un mensaje implica dos partes: el emisor y el receptor. Cuando un objeto emisor envía un mensaje a un objeto receptor, tiene que especificar lo siguiente: Un receptor, un nombre de mensaje y argumentos o parámetros (si se necesita).

Nombre de un mensaje: Un mensaje incluye el nombre de una operación y cualquier argumento requerido por esa operación. Con frecuencia, es útil referirse a una operación por nombre, sin considerar sus argumentos.

Métodos: Cuando un objeto recibe un mensaje, se realiza la operación solicitada ejecutando un método. Un método es el algoritmo ejecutado en respuesta a la recepción de un mensaje cuyo nombre se corresponde con el nombre del método.

Los objetos no especificados por el emisor no responderán, ya que el receptor trata de concordar el nombre del mensaje con los mensajes que

él reconoce. Si el mensaje no se entiende o no es reconocido, el receptor no se activará. Si el mensaje se entiende por el objeto receptor, el receptor aceptará y responderá al mensaje invocando el método asociado. Los parámetros o argumentos pueden ser: Datos utilizados por el método invocado o un mensaje propiamente dicho. La estructura de un mensaje es:

<objeto>.<método 1 (parámetro1,....,parámetro n)>

Veamos algunos mensajes sobre un objeto "gato". El primero de ellos invoca al método *dormir* y no tiene argumentos, el segundo invoca al método *comer,* y pasa como argumento *ratón.* Ejemplo:

Gato.dormir()
Gato.comer(ratón)

2.2.3.4. Estructura interna de un objeto

Un objeto consta de dos componentes básicos: atributos y métodos (operaciones o servicios).

Las figuras 2.10 y 2.11 ilustran dos tipos diferentes de representar un objeto en forma gráfica.

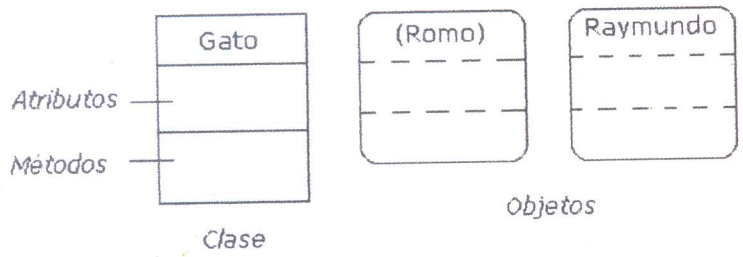


Figura 2. 10. Notación gráfica OMT de una clase y de un objeto

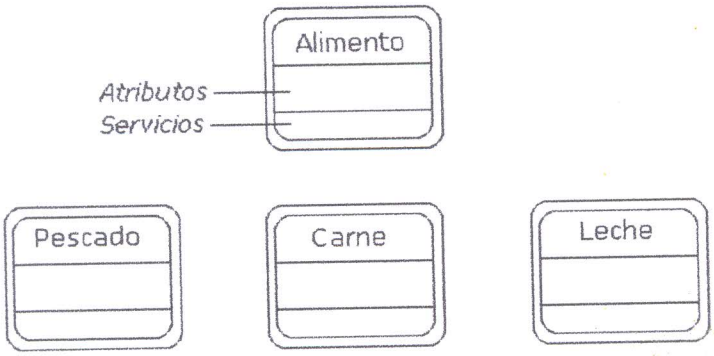


Figura 2. 11. Objetos en Notación Yourdon/Coad

2.2.3.4.1. Atributos

Los atributos describen el estado de un objeto. Un atributo consta de dos partes: un nombre de atributo y un valor de atributo.

Los objetos simples pueden constar de tipos primitivos, tales como enteros, carácter, boolean, reales, o tipos simples definidos por el usuario. Los objetos complejos pueden constar de pilas, conjuntos, listas, array,

etc., o incluso estructuras recursivas de alguno o todos los elementos.

2.2.3.4.2. Métodos

Los métodos (operaciones o servicios) describen el comportamiento asociado a un objeto. Representan las acciones que pueden realizarse por un objeto o sobre un conjunto. La ejecución de un método puede conducir a cambiar el estado del objeto o dato local del objeto.

Cada método tiene un nombre y un cuerpo que realiza la acción o comportamiento asociado con el nombre del método. En un Lenguaje Orientado a Objetos, el cuerpo de un método consta de un bloque de código procedimental que ejecuta la acción requerida. Todos los métodos que acceden a los datos de un objeto se definen dentro del objeto.

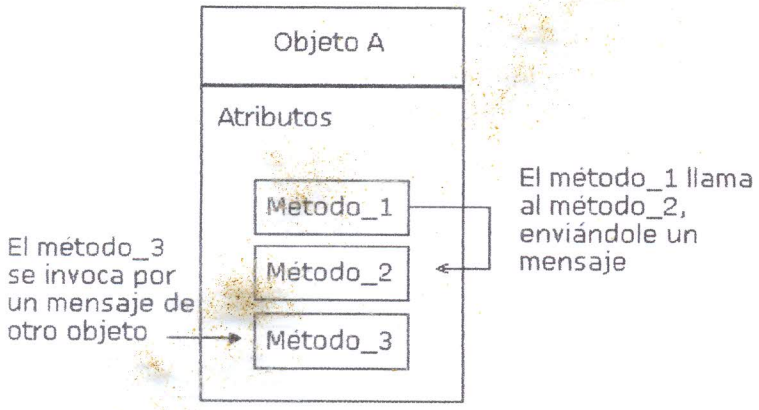


Figura 2. 12. Invocación de un método

Un objeto puede modificar directamente o acceder a los datos de otros objetos. Un método dentro de un objeto se activa por un mensaje que se envía por otro objeto al objeto que contiene el método. De modo alternativo, se puede llamar por otro método dentro del mismo objeto por un mensaje local enviado de un método a otro método dentro del objeto. La figura 2.12 ilustra esta característica.

2.2.3.5. Identificación de objetos

El primer problema que surge al querer desarrollar un programa orientado a objetos es *identificarlos*. La identificación se obtiene examinando la descripción del problema (análisis gramatical) y localizando los nombres o cláusulas nominales.

Los objetos, según Shaler, Mellor y Coad/Yourdon (4), pueden caer dentro de las siguientes categorías: Cosas tangibles (avión, casa, etc), Roles o papeles representados por personas (gerente, empleado, etc.), Organizaciones (empresa, compañía, etc.), Incidentes, es decir, un suceso, evento u ocurrencia (un accidente, una llamada, etc.), Interacciones, las que implican una transacción y relacionan dos objetos del modelo (matrimonio – esposo, esposa, fecha de boda), Especificaciones, las que muestran aplicaciones de inventario o fabricación (refrigeradora, nevera, etc.), y Lugares (comedor, oficina, etc.).

Una vez identificados los objetos, será necesario identificar los atributos y las operaciones que actúan sobre ellos. Los atributos describen la abstracción de características individuales que poseen todos los objetos.

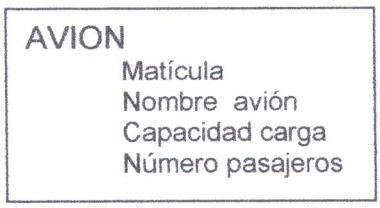


Figura 2. 13. Objeto Avión

Las operaciones cambian valores de uno o más atributos contenidos en el objeto. La figura 2.13 ejemplifica un objeto *avión* con 4 atributos. Aunque hay un gran número de operaciones que se pueden realizar sobre un objeto, generalmente se dividen en tres grandes grupos:

1. Operaciones que manipulan los datos de alguna forma específica (añadir, borrar, cambiar formato...). 2. Operaciones que realizan un cálculo o proceso y 3. Operaciones que comprueban (monitorizan) un objeto frente a la ocurrencia de algún suceso de control.

2.2.3.6. Duración de los objetos

Los objetos son entidades que existen en el tiempo, por ello deben ser creados o instanciados (normalmente a través de otros objetos). Esta

operación se hace a través de operaciones especiales llamadas constructores en C++ o inicializadores. Estas operaciones se ejecutarán implícitamente por el compilador o explícitamente por el programador, mediante invocación a los citados constructores.

2.2.3.7. Ocultación de datos

Con el fin de mantener las características de caja negra de POO, se debe considerar cómo se accede a un objeto en el diseño del mismo. Normalmente es una buena práctica restringir el acceso a las variables estado de un objeto y a otra información interna que se utiliza para definir el objeto. Cuando se utiliza un objeto no necesitamos conocer todos los detalles de la implementación. Esta práctica de limitación del acceso a cierta información interna se llama *ocultación de datos*.

2.2.4. Clases

Una clase es la descripción de un conjunto de objetos; consta de métodos y datos que resumen características comunes de un conjunto de objetos. Se puede definir muchos objetos de la misma clase. Las clases son similares a los tipos de datos y equivalen a modelos o plantillas que describen cómo se construyen ciertos tipos de objetos. Cada vez que se construye un objeto a partir de una clase, estamos creando lo que se llama una instancia de una clase. Por consiguiente, los objetos no son

más que una ***instancia*** de una clase. En general, instancia de una clase y objeto son términos intercambiables. Los objetos se crean cuando un mensaje de petición de creación se recibe por la clase base.

Un objeto es una instancia (ejemplar, caso u ocurrencia) de una clase. La clase es el bloque de construcción fundamental de un lenguaje de programación orientado a objetos. Una clase es un tipo abstracto de datos junto con un conjunto de transformaciones permitidas de dicho tipo abstracto de datos; puede definir también su interfaz a otras clases o funciones, descubriendo para ello que parte de su descripción interna de datos o conjunto de transformaciones permitidas puedan hacerse públicos.

Una interfaz es un grupo de procedimientos y métodos de propiedad que una clase expone para su uso por parte de los clientes. Una interfaz define el comportamiento que se espera de un objeto creado a partir de la clase. Sin embargo, no define cómo se implementa este comportamiento.

No se debe confundir clases y objetos de la misma clase: un coche rojo y un coche azul no son objetos de clases diferentes, sino objetos de la misma clase con atributos diferentes. En este caso la clase se denomina *coches* y el atributo es el *color.*

2.2.4.1. ¿Cuál es la diferencia entre clase y objeto?

Un objeto es un simple elemento, no importa lo complejo que pueda ser. Una clase, por el contrario, describe una familia de elementos similares. En la práctica, una clase es como un esquema o un plantilla que se utiliza para definir o crear objetos.

A partir de una clase se puede definir un número determinado de objetos. Cada uno de estos objetos generalmente tendrá un estado particular propio y otras características, aunque compartan algunas operaciones comunes. Los objetos tienen las siguientes características:

a) Se agrupan en tipos llamados *clases*.
b) Tienen *datos internos* que definen su estado actual
c) Soportan *ocultación de datos*.
d) Pueden *heredar* propiedades de otros objetos.
e) Pueden comunicarse con otros objetos pasando *mensajes*.
f) Tienen *métodos* que definen su comportamiento.

Es útil ilustrar objetos y clases con diagramas. La figura 2.14 muestra el esquema general de un diagrama objeto. Un objeto se dibuja como una caja, la misma que se etiqueta con el nombre del objeto y representa el límite o frontera entre el interior y el exterior de un objeto.

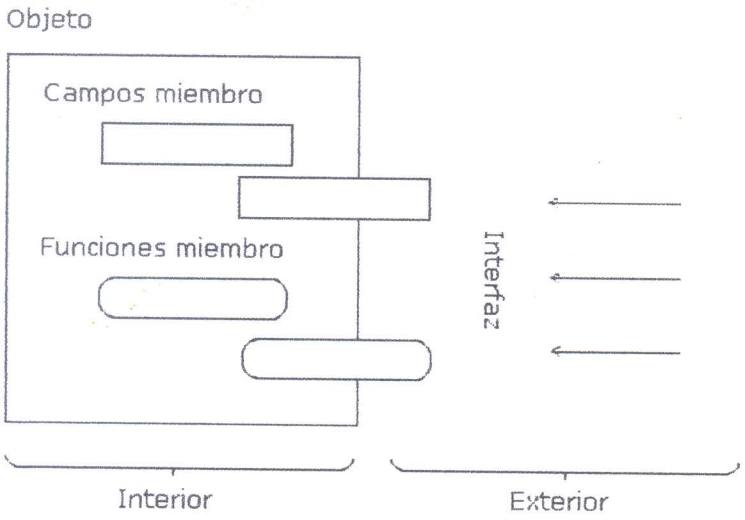


Figura 2. 14. Diagrama de un objeto

Un atributo se dibuja con una caja rectangular, una función por un hexágono largo. Los campos y funciones se etiquetan con sus nombres. Los campos y funciones en el interior de la caja están ocultos al exterior, que significa estar encapsulados; en cambio, aquellos que sobresalen son visibles para otros objetos, como el método *Leer* en la figura 2.15.

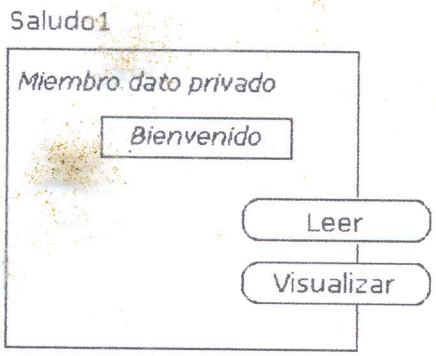


Figura 2. 15. El objeto Saludo1

Cada vez que se construye un objeto de una clase, estamos creando lo que se llama una instancia de una clase y la operación correspondiente se llama ***instanciación*** (creación de instancias).

Los objetos ocupan espacio en memoria, y en consecuencia existen en el tiempo, y deberán crearse o instanciarse (probablemente a partir de otros objetos). Por la misma razón, se debe liberar el espacio en memoria ocupado por los objetos.

Dos operaciones comunes típicas en cualquier clase son: *Constructor*: una operación que crea un objeto y/o inicializa su estado y *Destructor*: una operación que libera el estado de un objeto y/o destruye el propio objeto.

Cuando se desea crear una nueva instancia de una clase, se llama a un método de la propia clase para realizar el proceso de construcción. De modo similar, los métodos empleados para destruir objetos y liberar memoria ocupada también se definen dentro de la clase.

2.2.5. Herencia

El concepto de clases conduce a la idea de herencia. En nuestra vida diaria, el concepto de clases se divide en subclases. Así, las clases de

animales se dividen en: mamíferos, anfibios, insectos, etc.; las clases de bebidas se dividen en: cítricas, dietéticas, agri-dulces, etc. El principio en que se basan las sucesivas subclases es que cada subclase comparte características comunes con la clase de la que se deriva o desciende; por ejemplo, todas las clases vehículos tienen un motor, ruedas y un volante. Además de las características comunes, cada subclase tiene sus propias características particulares; por ejemplo, número de ruedas, número de asientos, diferentes portaequipajes, etc.

En POO las clases se pueden dividir en ***subclases***; dicho de otro modo, una clase se puede deducir o derivar de otra clase. La clase original se denomina ***clase base*** o ***superclase*** y, merced a la propiedad de la herencia una clase derivada o descendiente puede heredar las estructuras de datos y los métodos de su clase base. Además, la nueva clase puede añadir nuevos elementos datos y métodos a los que hereda de su clase base. Cualquier clase base (incluida una descendiente) puede tener cualquier número de clases descendientes.

Mediante el mecanismo de la herencia se puede construir una jerarquía de clases que adoptará la forma típica de un árbol, donde la clase base se llamará clase padre y, la clase descendiente o derivada, clase hija. La figura 2.16 ilustra la relación jerárquica entre clases y subclases.

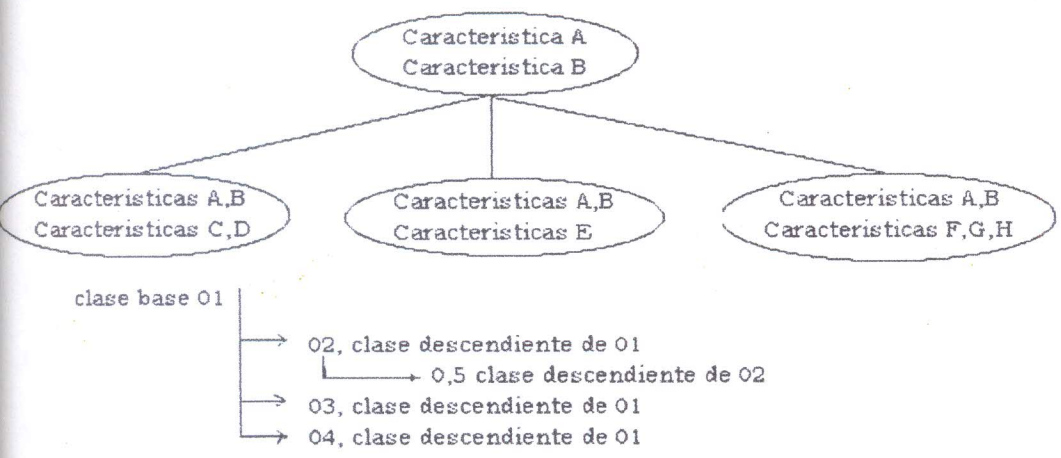


Figura 2. 16. Herencia Simple

Supongamos, por ejemplo, que se desea construir un conjunto de clases que describa una biblioteca de publicaciones. Consideremos dos tipos de publicaciones: periódicos y libros.

Se puede crear una clase base publicación con los datos: título, número de páginas, número de catálogo, año de publicación, precio y editor; los métodos de las publicaciones pueden ser: leer, almacenar y buscar o recuperar del depósito o biblioteca.

A continuación se crean dos clases descendientes llamadas periódico y libro. Un periódico tiene una fecha de publicación (además del año) y un número de orden y una serie de artículos: son los miembros dato de periódico. El periódico utilizará métodos: la suscripción o la compra en el

quiosco de prensa. El libro incluirá como datos el nombre de su autor, clase de libro (novela, poesía, etc.). Los métodos del libro pueden ser comprar en librería y regalar.

En resumen, un libro y un periódico comparten las características comunes de una publicación y, además tiene sus propios atributos. Así, vemos que la clase base define métodos de almacenamiento y recuperación que serán distintos en cada caso: los periódicos se almacenan en depósitos y hemerotecas; los libros, en estantes y bibliotecas. Los periódicos se almacenarán con reseñas de sus artículos, fecha de publicación y temas; los libros se registran en catálogos generales o específicos, editoriales, etc.

La encapsulación es una característica muy potente, y junto con el ocultamiento, representan el concepto avanzado de objeto, que adquiere su mayor relevancia cuando encapsula e integra datos, más las operaciones que manipulan los datos de dicha entidad. El objetivo final es la reutilización, es decir utilizar el código anteriormente desarrollado.

2.2.5.1. Tipos de herencia

La figura 2.17 ilustra los dos tipos de herencia que existen: **simple** (gráfico izquierda) y **múltiple** (gráfico derecha).

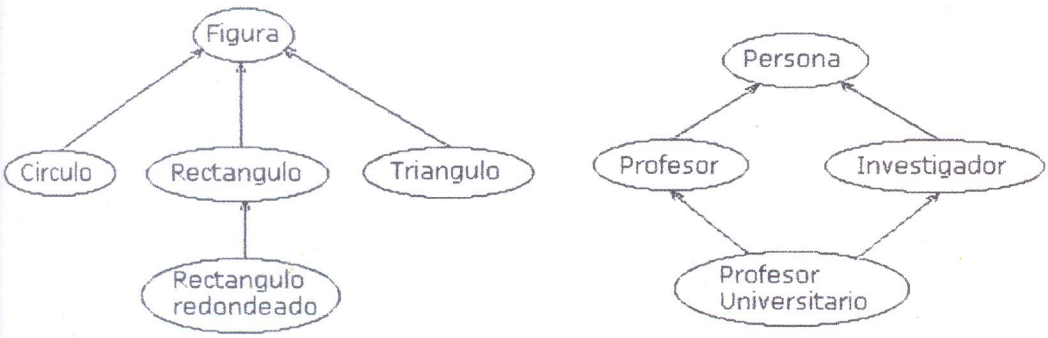


Figura 2. 17. Tipos de herencia

2.2.5.1.1. Herencia simple

La herencia simple (herencia jerárquica) es aquella donde una clase sólo puede tener un ascendiente. La herencia simple permite que una clase herede las propiedades de su superclase en una cadena jerárquica.

2.2.5.1.2. Herencia múltiple (herencia en malla)

Es aquella en que un objeto puede tener más de un ascendiente, o lo que es igual, adquirir datos y métodos de más de una clase.

La mayoría de los lenguajes soportan herencia simple. Sólo algunos incorporan la propiedad de herencia múltiple: C++(3.0) y Eiffel. Un caso típico de herencia múltiple es el popular término multimedia. Supongamos dos tipos de clases: *Sonido y Gráfico*. Si se combinan ambos, puede dar lugar a una nueva clase *multimedia* con propiedades de ambas clases.

Sin embargo, es preciso advertir que en herencia múltiple se plantean dificultades diversas, sobre todo cuando se combinan tipos de clases, cada una de las cuales define los mismos campos o métodos. Pueden presentarse dos problemas: 1. La propiedad referida solo está en una de las subclases padre y 2. La propiedad concreta existe en más de una superclase.

2.2.5.2. Clases abstractas

Con frecuencia, cuando se diseña un modelo orientado a objetos es útil introducir clases a cierto nivel, que pueden no existir en la realidad pero que son construcciones conceptuales útiles. Estas clases se conocen como clases abstractas. Una clase abstracta normalmente ocupa una posición adecuada en la jerarquía de clases que le permite actuar como un depósito de métodos y atributos compartidos para las subclases de nivel inmediatamente inferior.

Las clases abstractas no tienen instancias directamente. Se utiliza para agrupar otras clases y capturar información que es común al grupo. Sin embargo, las subclases de clases abstractas que corresponden a objetos del mundo real pueden tener instancias. Una clase abstracta es una clase que sirve como una base común, pero no tendrá instancias. Las clases derivadas de una clase base se conocen como clases concretas, que ya

pueden instanciarse (es decir, pueden tener instancias o pueden crearse objetos a partir de ellas)

2.2.6. Anulación/Sustitución

Como se ha comentado anteriormente, los atributos y métodos definidos en la superclase se heredan por las subclases. Sin embargo, si la propiedad se define nuevamente en la subclase, aunque se haya definido anteriormente a nivel de superclase, la definición realizada en la subclase es la utilizada en esa subclase. Entonces se dice que se anulan las correspondientes propiedades de la superclase. Esta propiedad se denomina *anulación* o *sustitución*.

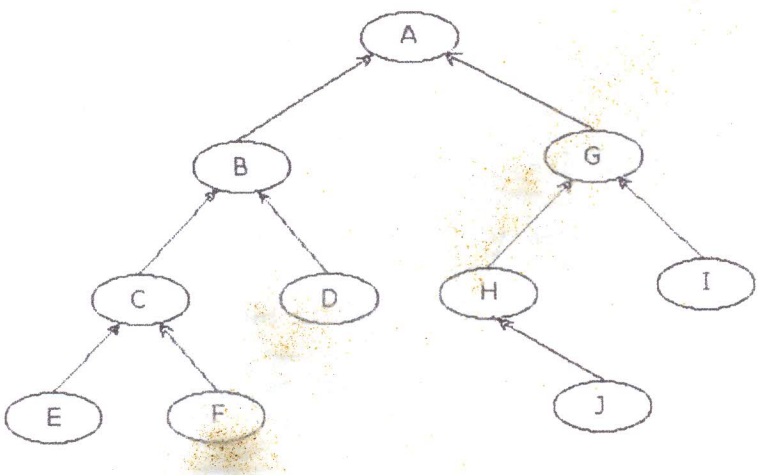


Figura 2. 18. Anulación de atributos y métodos en clases derivadas

Supongamos que ciertos atributos y métodos definidos en la clase A se redefinen en la clase C (figura 2.18). Entonces las clases E y F heredan

stos atributos y métodos. El convenio adoptado es que una vez que un tributo o método se redefine en un nivel de clase específico entonces ualquier hijo de esa clase, o sus hijos en cualquier profundidad utilizan ste método o atributo redefinido.

.2.7. Sobrecarga

a sobrecarga es una propiedad que describe una característica decuada que utiliza el mismo nombre de operación para representar peraciones similares que se comportan de modo diferente cuando se plican a clases diferentes. Por consiguiente, los nombres de las peraciones se pueden sobrecargar, esto es, las operaciones se definen n clases diferentes y pueden tener nombres idénticos aunque su código rograma pueda diferir.

La sobrecarga no es una propiedad específica de los lenguajes orientados a objetos. Lenguajes tales como C y Pascal soportan funciones sobrecargadas. Algunos de los ejemplos son los operadores aritméticos, peraciones de E/S y operaciones en la asignación de valores.

Así, por ejemplo, las operaciones de E/S (Entrada/Salida) se utilizan con frecuencia para leer números enteros, caracteres o reales. En Pascal se puede utilizar *read*(x), siendo x un entero, un carácter o un real.

Naturalmente, el código de máquina real ejecutado para leer una cadena de caracteres es muy diferente del código máquina para leer enteros. *Read*(x) es una operación sobrecargada que soporta tipos diferentes. Otros operadores tales como los de asignación son sobrecargados, ya que los mismos operadores se utilizan para variables de diferentes tipos.

Los lenguajes de programación convencionales soportan sobrecarga para algunas de las operaciones sobre algunos tipos de datos, como enteros, reales y caracteres. Los sistemas orientados a objetos dan un poco más en la sobrecarga y la hacen disponible para operaciones sobre cualquier tipo de objeto.

2.2.8. Ligadura dinámica

Los lenguajes OO tienen la característica de poder ejecutar ligadura tardía (dinámica), al contrario que los lenguajes imperativos, que emplean ligadura temprana (estática).

La ligadura estática permite un tiempo de ejecución más rápido que la ligadura dinámica, que necesita resolver enlaces en tiempo de ejecución. Sin embargo, en ligadura estática se ha de especificar en tiempo de compilación las operaciones exactas a que responderá una invocación del método o función específica, así como conocer sus tipos.

Por el contrario, en la ligadura dinámica simplemente se especifica un método en un mensaje, y las operaciones reales que realiza este método se determinan en tiempo de ejecución. Esto permite definir funciones o métodos virtuales. Revisemos el siguiente código de ejemplo:

```
Método ALIMENTARSE ( objeto 1, objeto 2 )
inicio
    objeto1.comer(objeto2)
fin
```

Al método ALIMENTARSE pasan dos objetos como argumentos. Este código ejemplifica lo que es enlace en tiempo de ejecución, ya que sólo cuando se esté ejecutando el programa se podrá determinar que objetos pasan como argumentos (*pueden pasar coche1 y coche2*) y, si son aceptados por el método, se ejecutará el código del mismo. Más precisamente se llamará al método *comer* del *objeto1*. Ahora veamos un ejemplo de la ligadura estática:

```
Método ALIMENTARSE ( gato, ratón )
inicio
    gato.comer(ratón)
fin
```

El código de arriba, en cambio, nos muestra como es el enlace en tiempo de diseño o compilación. El lenguaje detecta los objetos, así como las

propiedades y métodos del mismo. De esta manera determina el enlace entre el mensaje y el método asociado. En este ejemplo, se llama al método *comer* del objeto *gato*.

2.2.8.1. Funciones o métodos virtuales

Las funciones virtuales permiten especificar un método como virtual en la definición de una clase particular. La implementación real del método se realiza en las subclases. En este caso, por consiguiente, la selección del método se hace en tiempo de compilación, pero el código real del método utilizado se determina utilizando ligadura dinámica o tardía en tiempo de ejecución.

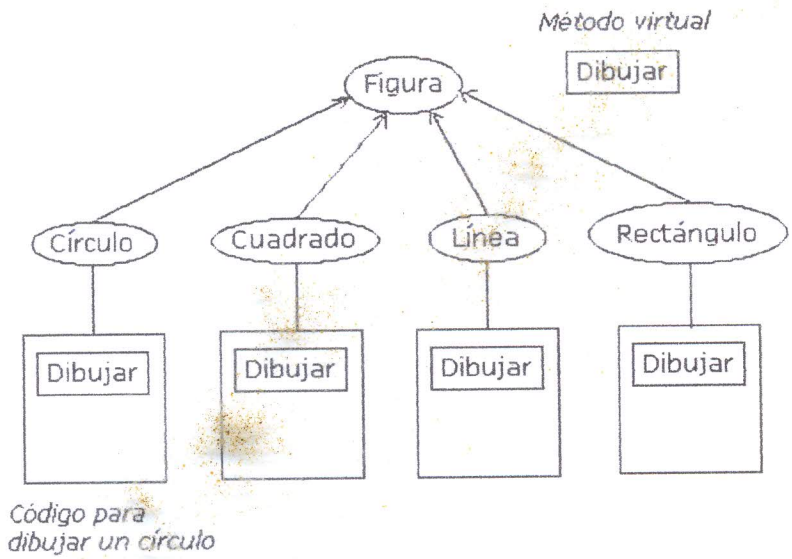


Figura 2. 19. Métodos o funciones virtuales

Esto permite definir el método de un número de formas diferentes para cada una de las diferentes clases derivadas.

En la figura 2.19, el método virtual se define en la clase FIGURA y el código procedimiento real utilizado se define en cada una de las subclases CIRCULO, CUADRADO, RECTANGULO y LINEA. Ahora, si un mensaje se envía a una clase específica, se ejecuta el código asociado con ella. Esto contrasta con un enfoque mas convencional que requiere definir los procedimientos por defecto, con nombres diferentes, tales como Dibujar_Circulo, Dibuja_cuadrado, etc. También se requerirá utilizar una llamada al nombre de la función específica cuando sea necesario.

La capacidad de utilizar funciones virtuales y ejecutar sobrecarga conduce a una característica importante de los sistemas OO, conocida como *polimorfismo*, que esencialmente permite desarrollar sistemas en los que objetos diferentes puedan responder de modo diferente al mismo mensaje. En el caso de un método virtual se puede tener especialización incremental de, o adición incremental a, un método definido anteriormente en la jerarquía.

2.2.9. Reutilización con Orientación a Objetos

Reutilización es la propiedad por la que el software desarrollado puede

ser utilizado cuantas veces sea necesario en más de un programa. Así por ejemplo, si se necesita una función que calcule el cuadrado o el cubo de un número, se puede crear la función que realice la tarea que el programa necesita. Con un esfuerzo suplementario se puede crear una función que pueda elevar cualquier número a cualquier potencia. Esta función se debe guardar para utilizar como herramientas de propósito general en cuantas ocasiones sea necesario.

Las ventajas de la reutilización son evidentes. El ahorro de tiempo es, sin duda, una de las ventajas más considerables, y otra la facilidad para intercambiar software desarrollado por diferentes programadores. En la programación tradicional, las bibliotecas de funciones evitan tener que ser escritas cada vez que se necesita su uso.

2.2.9.1. Objetos y reutilización

La POO proporciona el marco idóneo para la reutilización de las clases. Los conceptos de encapsulamiento y herencia son las bases que facilitan la reutilización. Un programador puede utilizar una clase existente, y sin modificarla, añadirle nuevas características y datos. Esta operación se consigue derivando una clase a partir de la clase base existente. La nueva clase hereda las propiedades de la antigua, pero se pueden añadir nuevas propiedades.

La facilidad de reutilizar clases (y en consecuencia objetos) es una de las propiedades fundamentales que justifican el uso de la POO. Por esta razón, los sistemas y en particular los lenguajes orientados a objetos suelen venir provistos de un conjunto (biblioteca) de clases predefinidas, que permite ahorrar tiempo y esfuerzo en el desarrollo de cualquier aplicación.

2.2.10. Polimorfismo

Polimorfismo es la propiedad que permite a una operación tener diferente comportamiento en objetos diferentes. En otras palabras, objetos diferentes reaccionan de modo diferente al mismo mensaje. Por ejemplo, supongamos un determinado número de figuras geométricas, en las que todas ellas comprenden el mensaje *Dibujar*. Cada objeto reacciona a este mensaje, visualizándose a sí mismo en la pantalla de video. Evidentemente, un objeto rectángulo se dibuja de un modo diferente que un objeto círculo.

El polimorfismo juega un papel importante en la simplificación de la sintaxis al realizar la misma operación sobre una colección de objetos. Por ejemplo, el polimorfismo permite dibujar todas las figuras geométricas pertenecientes a un array de figuras, mediante un bucle similar a este: Para cada figura perteneciente al array de figuras, enviar mensaje dibujar

a las figuras. Esto es posible debido a que, con independencia de la figura geométrica exacta, cada objeto comprende el mensaje *Dibujar* y reaccione a él de un modo apropiado a esa figura.

El polimorfismo requiere herencia; simplifica la tarea del programador, generalizando la sintaxis de comunicación que permite tratar objetos de un tipo diferente de modo similar. El polimorfismo se aplica únicamente a los métodos heredados de una clase base. El polimorfismo significa que un método puede tener un nombre que es compartido a lo largo de una jerarquía de clases. Cada clase puede tener una implementación diferente para ese método. El nombre del método es el mismo para cada objeto; sin embargo, lo que hace es diferente.

El polimorfismo depende de la ligadura, que es el proceso por el cual un método se asocia con una función real. Cuando los métodos polimórficos se utilizan, el compilador no puede determinar que función llamar. La función específica llamada depende de la clase del elemento a la cual se envía el mensaje; en consecuencia, la función a llamar se determina en tiempo de ejecución. Esta operación se conoce como *ligadura tardía o postergada,* ya que ocurre cuando el programa se está ejecutando. La ligadura temprana, anticipada o previa ocurre para métodos no polimórficos; el compilador conoce exactamente cuál es el método que se

invoca, de modo que se puede construir una llamada directa en tiempo de compilación.

Aunque la ligadura previa es muy eficiente, la mayoría de los lenguajes orientados a objetos utilizan ligadura postergadas para todos los métodos.

2.3. Lenguajes de Programación Orientados a Objetos (LPOO)

2.3.1. Evolución de los Lenguajes de Programación Orientados a Objetos

El primer lenguaje de programación que introdujo el concepto de clase fue Simula-67, como entidad que contenía datos y las operaciones que manipulaban los datos. Así mismo, introdujo también el concepto de herencia. El siguiente lenguaje orientado a objetos, y seguramente el más popular desde un enfoque conceptual exclusivamente de objetos, es Smalltalk, cuya primer versión comercial se desarrolló en 1976.

Entre los lenguajes orientados a objetos que se han desarrollado a partir de los ochenta destacan extensiones de lenguajes tradicionales tales como C++ y Objetive-C (extensiones de C), modula-2 y Object Pascal (extensión de Pascal) y recientemente Object Cobol, que a lo largo de 1994 han aparecido sus primeras versiones comerciales, y Java, el nuevo lenguaje para programación en Internet.

Otro lenguaje orientado a objetos puros es Eiffel, creado por Bertrand Meyer y que soporta todas las propiedades fundamentales de objetos. Ada ha sido también un lenguaje –en este caso basado en objetos- que soporta la mayoría de las propiedades orientadas a objetos.

En los últimos años han aparecido lenguajes con soporte de objetos que cada vez se popularizan más: Clipper 5-2, Visual Basic, etc.

2.3.2. Clasificación de los Lenguajes Orientados a Objetos

Existen varias clasificaciones de lenguajes orientados a objetos, atendiendo a criterios de construcción o características específicas de los mismos. Una clasificación ampliamente aceptada y difundida es la dada por Wegner (4), y que se ilustra en la figura 2.20.

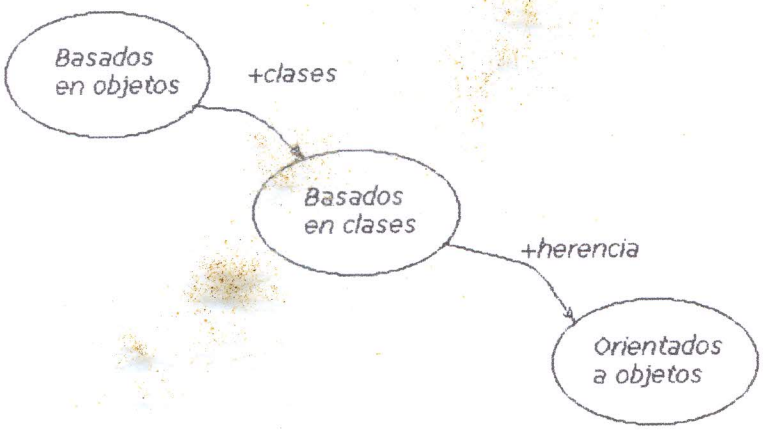


Figura 2. 20. Clasificación de lenguajes OO de Wegner

La clasificación de Wegner divide los lenguajes en tres categorías:

Lenguajes ***basados en objetos*** que soportan objetos. Es decir, disponen de componentes caracterizados por un conjunto de operaciones (comportamiento) y un estado. Ejemplos: Ada-83, Actor, Clipper 5.X, Visual Basic 4.0/5.0/6.0.

Lenguajes ***basados en clases*** que implican objetos y clases. Es decir, disponen de componentes tipo clase con operaciones y estado común. Una clase de un objeto se construye con un <<interfaz>> que especifica las operaciones posibles y un <<cuerpo>> que implementa dichas operaciones. Ejemplo: Clu.

Lenguajes **orientados a objetos** que además de objetos y clases ofrecen mecanismos de herencia entre clases. Esto es, la posibilidad de derivar operaciones y atributos de una clase (superclase) a sus subclases. Ejemplos: Smalltalk, C++, Prolog++, Ada-95, Eiffel, Overon, Delphi.

La definición anterior, pese a su antigüedad, sigue teniendo vigencia. Existen otras clasificaciones similares, pero con la inclusión de la propiedad de *polimorfismo* en la categoría 3, como requisito para ser lenguaje orientado a objetos. De cualquier forma hoy en día es posible

ampliar esa clasificación de acuerdo a criterios puramente técnicos y hacer una nueva clasificación de la categoría 3.

Lenguajes **orientados a objetos puros**. Soportan en su totalidad el paradigma de orientación a objetos. Ejemplos: Smalltalk, Eiffel, Simula.

Lenguajes **orientados a objetos híbridos**. Soportan en su totalidad el paradigma de la orientación a objetos sobre un núcleo de lenguaje híbrido. Ejemplos: C++, Object-COBOL, Object PASCAL, Java.

2.3.3. Puros frente a híbridos

Un LPOO puro es un lenguaje diseñado para soportar únicamente el paradigma orientado a objetos, en el que todo consta de objetos, métodos y clases. Los POO más populares son Smalltalk y Eiffel.

Un LPOO híbrido, por otra parte, soporta otros paradigmas de programación (tales como el tradicional –estructurado-, funcional, etc.), además del paradigma orientado a objetos. Los lenguajes híbridos se construyen a partir de otros lenguajes existentes, tales como C, Pascal o Basic, de los cuales se derivan; es posible utilizar el LPOO de un modo no orientado a objetos y también como orientado a objetos utilizando objetos, clases y métodos. Visual Basic es un lenguaje híbrido, basado en objetos.

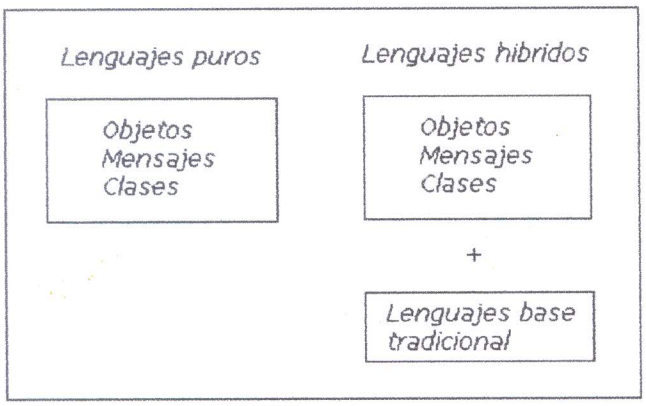


Figura 2. 21. Lenguajes OO puros/híbridos

Cada tipo de lenguaje tiene sus ventajas e inconvenientes. Los LPOO puros pueden ser más potentes porque utilizan todas las ventajas de la tecnología y, proporcionan máxima flexibilidad para cambiar los aspectos esenciales del lenguaje, dado que todo el lenguaje se construye sobre la base de los objetos.

Sin embargo, la mayoría de los lenguajes puros no son tan rápidos como los híbridos y son difíciles de codificar en toda clase de operaciones fundamentales. Esto les hace perder eficiencia en tiempo de ejecución. En contraste, los lenguajes híbridos pierden con frecuencia algunas de las características de los lenguajes puros y normalmente no son tan flexibles como los puros, debido a que no permiten modificar características de construcción del lenguaje base. Sin embargo, suelen ser más rápidos que los puros.

CAPITULO 3

3. TECNOLOGIAS DE DESARROLLO

3.1. Introducción a Visual Basic

3.1.1. Conceptos de Visual Basic

Para entender el proceso de desarrollo de HOELEST, es útil comprender algunos de los conceptos clave alrededor de los cuales está construido Visual Basic. Puesto que Visual Basic (VB) es un lenguaje de desarrollo para Windows, es necesario familiarizarse con el entorno Windows.

3.1.2. Funcionamiento de Windows: ventanas, eventos y mensajes

Una **ventana** es simplemente una región rectangular con sus propios límites. Hay varios tipos de ventanas: una ventana Explorador en Windows 95, una ventana de documento dentro de su procesador de textos o un cuadro de diálogo que emerge para recordar una cita. Aunque éstos son los ejemplos más comunes, realmente hay otros muchos tipos de ventanas como los iconos, cuadros de texto, barras de menús,etc.

El sistema operativo Microsoft Windows administra todas estas ventanas. El sistema controla continuamente cada una de estas ventanas para ver si existen signos de actividad o **eventos**. Los eventos pueden producirse mediante acciones del usuario, como hacer clic con el *mouse* (ratón) o presionar una tecla, mediante programación o incluso como resultado de acciones de otras ventanas.

Cada vez que se produce un evento se envía un **mensaje** al sistema operativo. El sistema procesa el mensaje y lo transmite a las demás ventanas. Entonces, cada ventana puede realizar la acción apropiada, basándose en sus propias instrucciones para tratar ese mensaje en particular (por ejemplo, volverse a dibujar cuando otra ventana la ha dejado al descubierto).

3.1.3. Descripción de la programación controlada por eventos

Visual Basic se deriva del lenguaje Basic, que es un lenguaje de programación estructurado. No obstante, Visual Basic utiliza un modelo de programación controlado por eventos.

3.1.3.1. Aplicaciones basadas en procedimientos

En las aplicaciones tradicionales o basadas en procedimientos, es la aplicación quien controla qué fragmentos del código se ejecutan y la

secuencia en que se ejecutan. La ejecución de la aplicación comienza con la primera línea de código y sigue una ruta predefinida por la aplicación, llamando a procedimientos a medida que son necesarios.

3.1.3.2. Aplicaciones controladas por eventos

En una aplicación controlada por eventos, la ejecución no sigue una ruta predeterminada, sino que ejecuta distintas secciones de código como respuesta a eventos. Los eventos pueden desencadenarse por acciones del usuario, mensajes del sistema o de otras aplicaciones, o incluso desde la propia aplicación. La secuencia de estos eventos determina la secuencia en la que se ejecuta el código. Por tanto, la ruta a través del código de la aplicación difiere cada vez que se ejecuta el programa. La figura 3.1 ilustra el formulario "Buscar" de HOELEST, muestra algunas acciones que generan eventos, por ejemplo, dar un click sobre el botón *Buscar*. Estos eventos pueden producirse en cualquier orden.

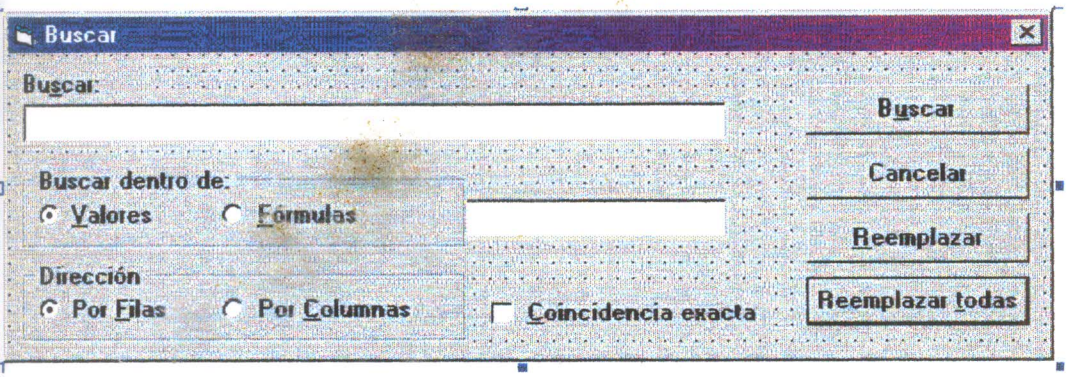


Figura 3. 1. Formulario Buscar de la aplicación Hoelest

Existen 3 ediciones de Visual Basic 5.0: la edición de aprendizaje, la profesional y la empresarial. HOELEST fue desarrollado en la Edición profesional de Visual Basic 5.0, debido a que éste ofrece un conjunto completo de herramientas para el desarrollo de aplicaciones.

3.1.4. Palabras claves y fundamentos de la POO en Visual Basic

Revisar conceptos tales como módulos de código, procedimientos o tratamiento de errores, será de gran utilidad al lector para interpretar los siguientes capítulos. Esta sección sirve de referencia y podrá regresar a ella en caso de necesitar recordar algún *tecnicismo* utilizado en la Tesis.

En *tiempo de diseño*, que es cualquier momento mientras se está desarrollando una aplicación en el entorno de Visual Basic, se trabaja con formularios y controles, se establecen propiedades y se escribe código para los eventos. *Tiempo de ejecución* es cualquier momento mientras se ejecuta realmente la aplicación y se interactúa con ella como lo haría un usuario.

3.1.5. Módulos de código

El código en Visual Basic se almacena en módulos. Hay tres tipos de módulos: de formulario, estándar y de clase. Cada módulo estándar, de clase y de formulario puede contener lo siguiente:

Declaraciones. Se pueden colocar declaraciones de constantes, tipos, variables y procedimientos de bibliotecas de vínculos dinámicos (DLL) al nivel de módulo de formulario, de clase o estándar.

Procedimientos. Un procedimiento **Sub**, **Function** o **Property** contiene partes de código que se pueden ejecutar como una unidad.

3.1.5.1. Módulos de formulario

Los módulos de formulario pueden contener procedimientos que controlen eventos, procedimientos generales y declaraciones a nivel de formulario de variables, constantes, tipos personalizados y procedimientos externos. El código que se escribe en un módulo de formulario es específico de la aplicación a la que pertenece el formulario y puede hacer referencia a otros formularios u objetos de la aplicación.

3.1.5.2. Módulos estándar

Los módulos estándar son contenedores de los procedimientos y declaraciones a los que tienen acceso otros módulos de la aplicación. Pueden contener declaraciones globales (disponibles para toda la aplicación) o a nivel de módulo de variables, constantes, tipos personalizados, procedimientos externos y procedimientos globales. El código que se escribe en un módulo estándar no está ligado

necesariamente a una aplicación determinada, es decir, se pueden reutilizar en distintas aplicaciones.

3.1.5.3. Módulos de clase

Los módulos de clase son la base de la programación orientada a objetos en Visual Basic. Se puede escribir código en módulos de clase para crear nuevos objetos. Estos objetos nuevos pueden incluir propiedades y métodos personalizados. Los módulos de clase de HOELEST proporcionan objetos a programadores de Automatización. Más adelante se explicara qué es Automatización.

3.1.6. Procedimientos

En Visual Basic encontramos dos tipos de procedimientos: procedimientos de evento y procedimientos generales.

3.1.6.1. Procedimientos de evento

Visual Basic llama automáticamente a los procedimientos de evento como respuesta a una acción del teclado, del *mouse* o del sistema. Por ejemplo, los botones de comando tienen un procedimiento de evento *Click*.

Cualquiera que sea el código colocado en un procedimiento de evento *Click*, se ejecutará cuando el usuario haga click en un botón de comando.

Cada control posee un conjunto fijo de procedimientos de evento. El siguiente ejemplo de código muestra el procedimiento de evento *Click* de un botón de comando llamado cmdOK:

```
Sub cmdOK_Click()
    MsgBox "Hola"
End Sub
```

3.1.6.2. Procedimientos generales

Los procedimientos generales son procedimientos **Sub** o **Function** que se crean para realizar determinadas tareas.

3.1.6.2.1. Procedimientos Sub

Los procedimientos **Sub** no devuelven valores. Por ejemplo:

```
Private Sub ActualizarNombre(S as String) 'Inicia el procedimiento
'Código para actualizar el nombre...
End Sub                                   'Fin del procedimiento
```

3.1.6.2.2. Procedimientos Function

Los procedimientos **Function** devuelven valores. En el siguiente código, el procedimiento recibe un número y lo devuelve elevado al cubo:

```
Function Cubo(I As Integer) As Integer      'Inicia el procedimiento
        Cubo = I * I * I                    'Ejecuta la operación
End Function                                'Fin del procedimiento
```

3.1.7. Alcance del código

Los procedimientos pueden declararse como **Private** o como **Public**.

A los procedimientos **Private** sólo se les puede llamar desde otros procedimientos ubicados en ese formulario, módulo o clase. Los procedimientos **Public** de un formulario se convierten en métodos del formulario. Se puede llamar al procedimiento desde cualquier lugar de la aplicación si especifica los nombres del formulario y del procedimiento. Los procedimientos públicos de un módulo están disponibles en toda la aplicación y se les puede llamar con sólo especificar el nombre del procedimiento.

3.1.8. Usar variables, constantes y tipos de datos

3.1.8.1. Declarar variables

Para declarar una variable, se utiliza la instrucción **Dim.** La palabra clave **As** permite especificar el tipo de dato a crear. En el siguiente ejemplo de código se declara la variable **iCounter** como una variable **Integer**:

Dim iCounter As Integer

3.1.8.2. Declarar constantes

Se utiliza la instrucción **Const** para declarar una constante. El valor de

una constante no se puede cambiar. El siguiente ejemplo de código declara la constante PI como una variable de tipo **Single**:

```
Const PI As Single = 3.14
```

En Visual Basic existen numerosas constantes incorporadas que resultan de gran utilidad con diversas instrucciones, ya que ayudan a escribir código más legible. En el siguiente ejemplo de código se utilizan las constantes predefinidas **vbYesNo** y **vbYes** con la instrucción **MsgBox**:

```
iResult = MsgBox("¿Desea continuar?", vbYesNo)
If iResult = vbYes Then
    'Hace algo.
End If
```

3.1.8.3. Usar Tipos de Datos

El tipo de datos de una variable determina el tipo de información que puede almacenar una variable y el intervalo de valores posibles.

Si no especifica ningún tipo de datos al declarar una variable, ésta recibirá el tipo de datos **Variant**. El tipo de datos **Variant** puede almacenar datos nulos, numéricos, de fecha y hora o de cadena. Sin embargo, el tipo de datos **Variant** requiere más memoria que otros. Por ejemplo, *sFname* es de tipo Variant y *sLname* es de tipo String.

Dim sFname As Variant
Dim sLname As String

3.1.9. Alcance de las variables

El alcance de una variable define qué partes del código son conscientes de su existencia. Cuando declara una variable en un procedimiento, sólo el código de dicho procedimiento puede modificar el valor de la variable; tiene un alcance que es local al procedimiento.

Como a veces se necesita utilizar una variable con un alcance más general, como aquella cuyo valor está disponible para todos los procedimientos del mismo módulo o incluso para todos los procedimientos de toda la aplicación, Visual Basic permite especificar el alcance de una variable cuando la declara. El lugar en que se declara la variable (en un procedimiento, formulario o módulo) y cómo se declara (como Public o Private) determinan el alcance de la variable, como se indica en la Tabla:

Tabla II

Alcance de las variables

Alcance	Privado	Público
Nivel de procedimiento	Las variables son privadas para el procedimiento en el que aparecen.	No se puede declarar variables públicas dentro de un procedimiento.
Nivel de módulo	Las variables son privadas para el módulo en el que aparecen.	Las variables están disponibles para todos los módulos.

3.1.9.1. Variables de procedimiento

Una variable declarada en un procedimiento sólo se reconoce dentro del procedimiento en el que se declara.

3.1.9.2. Variables de formulario

Las variables declaradas dentro de la sección de declaraciones generales de un formulario (código escrito fuera de los procedimientos) pueden ser **Private** o **Public**.

Las *variables privadas* **(Private)** están disponibles en todos los procedimientos del formulario, pero no son visibles para ningún procedimiento fuera del formulario. Las *variables públicas* **(Public)** están disponibles en toda la aplicación. Por ejemplo:

```
Public fTotal As Integer 'Declarada en el formulario Form1
Form2.fTotal = 4         'Se usa fuera del formulario Form1.
```

3.1.9.3. Variables de módulo estándar

Las variables declaradas dentro de la sección de declaraciones generales de un módulo estándar pueden ser **Private** o **Public**.

Las *variables privadas* **(Private)** sólo están disponibles en el módulo. Las *variables públicas* **(Public)** están disponibles en toda la aplicación.

3.1.10. Duración de las variables

La duración de una variable se refiere al tiempo que dicha variable está disponible. Dicha duración está determinada por la ubicación en la que se ha declarado (procedimiento o módulo) y cómo se ha realizado la declaración (**Static** o no **Static**).

3.1.11. Variables de procedimiento

Una variable de procedimiento tiene una duración igual a la del procedimiento. Cuando vuelve a llamarse al procedimiento, la variable se reinicializa.

Cuando se desarrolló HOELEST, se vió la necesidad de impedir que ciertas variables se reinicialicen, por lo que las declaramos como **Static**. Las variables estáticas siguen estando disponibles sólo para el procedimiento; sin embargo, conservan sus valores durante todo el tiempo en que se ejecuta HOELEST.

3.1.11.1. Variables de formulario

Las variables declaradas en formularios se mantienen hasta que el valor del formulario se establece a **Nothing**. Las variables de formulario conservan su valor cuando se descarga el formulario. Cuando asigna a una variable de formulario el valor **Nothing**, las variables se borran y se

ejecuta el evento Terminate del formulario, es decir se borra de la memoria.

3.1.11.2. Variables de módulo estándar

Las variables declaradas en un módulo están disponibles durante todo el tiempo en que se ejecuta HOELEST.

3.1.12. Tratar los errores en tiempo de ejecución

En tiempo de ejecución, es decir cuando un usuario está utilizando HOELEST, se generan errores; por ejemplo, los usuarios olvidan insertar discos en las unidades de disco, los sistemas se quedan sin memoria y los archivos no están donde espera encontrarlos. HOELEST intercepta estos errores gracias a su código de tratamiento de errores, por lo que se convierte en una aplicación más robusta.

Visual Basic proporciona el objeto *Err* para el código de tratamiento de errores. La instrucción *OnError GoTo* activa una interceptación de errores y especifica a dónde saltará la ejecución cuando se produzca un error.

Si se produce un error de tiempo de ejecución, la ejecución continuará en la etiqueta indicada por la instrucción *OnError GoTo*. El controlador de errores de VB ejecuta el código de tratamiento de errores, seguido de una

instrucción *Resume* que indica dónde debe continuar el proceso.

3.1.12.1. Propiedades del objeto Err

La propiedad *Number* es un entero que indica el último error que ocurrió. En algunos casos se puede corregir un error y permitir que continúe el proceso sin interrumpir al usuario. De lo contrario, se notifica al usuario acerca de la existencia de un error y se toma una acción basándose en la respuesta del usuario.

La propiedad *Description* es una cadena que contiene una descripción del error.

La propiedad *Source* contiene el nombre de la aplicación del objeto que generó el error. Esto es útil cuando se utiliza Automatización. Por ejemplo, si inicia HOELEST y éste genera un error, HOELEST asignará a *Err.Number* el código de error correcto y *Hoelest.Origen* a *Err.Source*.

3.1.12.2. Opciones de Resume

Para especificar dónde continuará el proceso su aplicación después de tratar el error, se utiliza la instrucción *Resume*. La tabla siguiente muestra los tres tipos de instrucciones *Resume* disponibles en Visual Basic y utilizadas para desarrollar HOELEST.

Tabla III

Tipos de instrucciones Resume. Tratamiento de errores

Instrucción	Descripción
Resume	Vuelve a la instrucción que produjo el error. Se utiliza **Resume** para repetir una operación después de corregir el error.
Resume Next	Vuelve a la instrucción que sigue inmediatamente a la instrucción que produjo el error.
Resume línea o etiqueta	Si no hay ninguna instrucción **Resume**, el procedimiento sale.

3.1.13. Programar con Objetos

Como se explicó en el capítulo anterior, los objetos están encapsulados; es decir, contienen su propio código y sus propios datos, lo cual facilita su mantenimiento más que los métodos de escritura de código tradicionales.

Los objetos de Visual Basic tienen propiedades, métodos y eventos. Las propiedades son los datos que describen un objeto. Los métodos son lo que puede hacer con el objeto. Los eventos son tareas que realiza el objeto; se puede escribir código para ejecutar cuando se produzcan eventos.

Los objetos de Visual Basic se crean a partir de clases; así, un objeto se dice que es una instancia de una clase. La clase define las interfaces de un objeto, si el objeto es público y en qué circunstancias se puede crear.

Las descripciones de las clases se almacenan en bibliotecas de tipos y se pueden ver con examinadores de objetos.

Para utilizar un objeto tiene que mantener una referencia a él en una variable de objeto. El tipo de enlace determina la velocidad con la que se tiene acceso a los métodos de un objeto mediante la variable de objeto. Una variable de objeto puede ser un enlace en tiempo de compilación (el más lento) o un enlace en tiempo de diseño. Las variables enlazadas en tiempo de diseño pueden ser enlaces DispID o enlaces vtable (los más rápidos).

Al conjunto de propiedades y métodos se le llama interfaz. La interfaz predeterminada de un objeto de Visual Basic es una interfaz dual que admite las tres formas de enlace. Si una variable de objeto está correctamente definida (es decir, Dim ... As nombre_clase), utilizará la forma más rápida de enlace.

Además de su interfaz predeterminada, los objetos de Visual Basic pueden implementar interfaces adicionales para proporcionar polimorfismo. El polimorfismo le permite manipular muchos tipos diferentes de objetos sin preocuparse de su tipo. Las interfaces múltiples son una característica del Modelo de Objetos Componentes (COM);

permiten que los programas evolucionen con el tiempo, agregando nueva funcionalidad sin afectar al código existente.

3.2. El Modelo de Objetos Componentes

El *Modelo de Objetos Componentes (COM)* es una arquitectura orientada a objetos para generar aplicaciones y la base sobre la que se generan los *componentes ActiveX*. Los componentes ActiveX permiten a los programadores combinar fragmentos de código reutilizables para crear aplicaciones y servicios.

COM tiene dos aspectos diferentes. El primero es que la especificación COM proporciona una definición (o modelo) de lo que es un objeto. El segundo es que COM proporciona servicios para la creación de objetos y para las comunicaciones entre el cliente y el servidor.

3.2.1. Introducción a COM

Para llegar a comprender COM, debe familiarizarse con los siguientes términos y sus definiciones:

3.2.1.1. Componente

Una unidad de código ejecutable que proporciona funcionalidad. Los componentes los proporcionan los servidores que son archivos .exe, .dll u

.ocx. Los servidores pueden contener más de un componente y los componentes proporcionan las plantillas a partir de las cuales se crean los objetos.

El Modelo de Objetos Componentes (COM) especifica cómo se crean los componentes y cómo las aplicaciones cliente se conectan con dichos componentes. COM también procesa las solicitudes de las aplicaciones cliente para crear objetos.

3.2.1.2. Objeto

Una combinación de código y datos que se puede tratar como una unidad. Los objetos tienen una duración; es decir, se crean y se destruyen.

3.2.1.3. Automatización

La parte de la especificación COM que define un método estándar para la creación de componentes y el uso de los objetos. Los objetos de Automatización también se conocen como *objetos programables*.

Para un programador de aplicaciones cliente, un objeto recibe una entrada y proporciona una salida, pero el programador no puede tener acceso a su funcionamiento interno. Los programadores sólo necesitan saber cómo pueden utilizar la funcionalidad ofrecida por los objetos. Dicha

funcionalidad se ofrece a través de una o varias interfaces publicadas. Estas interfaces son el medio por el que las aplicaciones cliente se comunican con el componente.

3.2.2. Ventajas del uso de COM

La tecnología de componentes facilita la creación de aplicaciones más eficientes al proporcionar ventajas como la compatibilidad y reutilización del código y el control de versiones. HOELEST tiene estas características.

3.2.2.1. Compatibilidad binaria y desarrollo multiplataforma

Como se han convertido en un estándar del sector, las aplicaciones cliente y los servidores compatibles con COM son independientes del lenguaje de desarrollo y del sistema operativo.

COM asegura la total compatibilidad binaria entre el cliente desarrollado en Microsoft Visual Basic y los componentes desarrollados en Microsoft Visual C++ o Microsoft Visual J++, o los componentes que se ejecuten en un servidor Macintosh o UNIX.

En otras palabras, un programador de Visual C++ puede utilizar la funcionalidad de análisis de datos que proporciona HOELEST, que es un componente desarrollado en Visual Basic.

3.2.2.2. Reutilización del código

Los componentes COM exponen grupos de métodos, conocidos como interfaces, a través de los cuales los clientes interactúan con los objetos. Puesto que dichas interfaces están documentadas, muchos clientes pueden reutilizar el código que crea dichos objetos para diversos propósitos.

3.2.2.3. Control de versiones

Se dice que los componentes COM generan sus propias versiones. Esto significa que se puede agregar nueva funcionalidad a un componente (al agregar o modificar la implementación de una interfaz) sin afectar por ello a los clientes que ya utilicen el componente.

Nunca se pierde funcionalidad cuando se actualizan los componentes; siempre se mejora o se amplía. Esto significa que es posible crear una nueva versión de HOELEST y aquellas aplicaciones cliente, que utilizaban las versión anterior no generarán problemas

3.2.3. La especificación COM

La especificación COM define el esquema para la creación de los componentes. La especificación dicta las responsabilidades de un componente, pero no especifica cómo se implementan dichas

responsabilidades. En vez de esto, la especificación COM define:

1. Cómo se crea un objeto a partir de un componente. 2. Cómo un cliente tiene acceso a las características de un objeto. 3. La responsabilidad del objeto de destruirse a sí mismo cuando ya no se utilice.

Todas estas actividades se controlan mediante interfaces, lo que hace que el servidor sea mucho más fácil de utilizar que antes. Antes del desarrollo de las tecnologías de componentes, los servidores solían proporcionar servicios a los clientes a través de una amplia variedad de funciones poco organizadas. La documentación detallaba las distintas funciones ofrecidas por el servidor y los clientes llamaban a esas funciones de la forma apropiada. Llevaba mucho tiempo descubrir qué funciones había que utilizar. En este escenario existen tres problemas:

1. Había que escribir los clientes especialmente para que pudieran aprovechar las ventajas de un servidor. 2. Cuando el servidor cambiaba, había que modificar los clientes para adaptarlos a los cambios. 3. Los servidores eran difíciles de comprender porque las funciones que ofrecían no tenían una estructura lógica.

Mediante el uso de interfaces, COM resuelve todos estos problemas.

3.2.4. Utilizar interfaces

Las interfaces son grupos de funciones que proporcionan puntos de conexión mediante los cuales se comunican los componentes de cliente y servidor. Las interfaces proporcionan un acceso estándar a los métodos y las propiedades (funcionalidad) disponibles en los servidores.

Además, estructuran el acceso para que el uso de los servidores sea más sencillo. La figura 3.2 muestra cómo los componentes de cliente y servidor se comunican mediante interfaces.

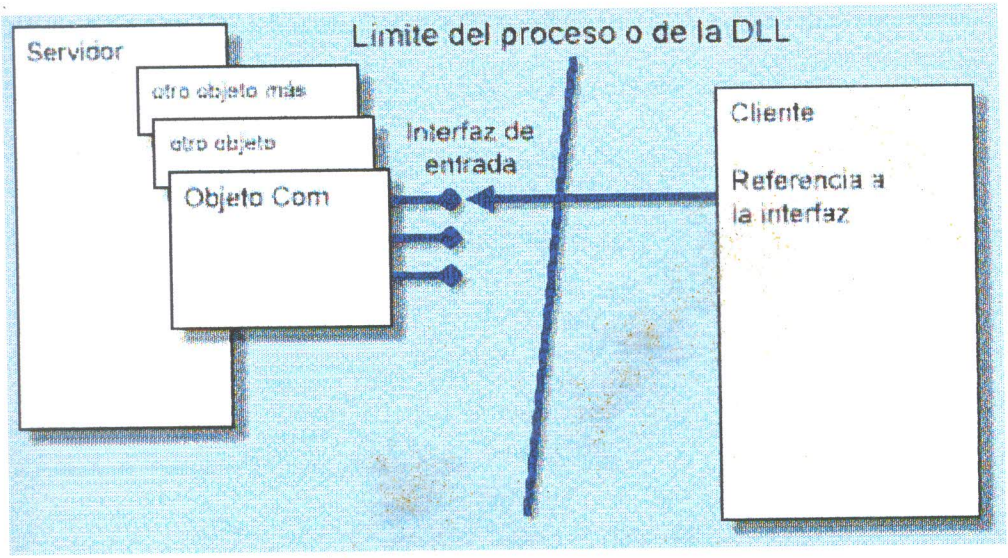


Figura 3. 2 Comunicación Cliente-Servidor

Un objeto puede tener varias interfaces que proporcionen acceso a funcionalidades diferentes para distintos clientes.

3.2.5. La interfaz IUnkown

La especificación COM requiere la siguiente funcionalidad en todos los objetos:

1. El objeto debe ser capaz de efectuar el seguimiento del número de conexiones establecidas con él. Cuando ya no se esté utilizando, tiene que ser capaz de destruirse a sí mismo y, 2. El objeto debe ser capaz de recibir solicitudes por parte de un cliente de interfaces adicionales que puede aceptar.

Para poder proporcionar esta funcionalidad, todos los objetos tienen una interfaz incorporada llamada *IUnknown*. *IUnknown* es una interfaz que todos los objetos COM tienen que aceptar. Además, todas las demás interfaces del objeto tienen que incluir la funcionalidad proporcionada por *IUnknown*.

3.2.5.1. Funciones de la interfaz IUnknown

La interfaz *IUnknown* tiene tres funciones: *AddRef*, *Release* y *QueryInterface*. Las funciones *AddRef* y *Release* efectúan el seguimiento de la creación y la destrucción del objeto, y *QueryInterface* permite que los clientes consulten otras interfaces proporcionadas por el objeto.

La función *AddRef* incrementa el contador de uso del objeto cuando éste asigna un *puntero* de interfaz. La función *Release* disminuye el contador de uso cuando alguna variable que señala al objeto pierde su alcance.

3.2.5.2. Consultar las interfaces adicionales

Cuando un cliente contiene una variable de objeto que señala a una interfaz válida de COM, el cliente puede consultar las interfaces adicionales proporcionadas por el objeto mediante el método **QueryInterface**.

3.2.6. Comunicación entre cliente y servidor

Puesto que COM está integrado en el sistema operativo, implementa la comunicación entre las aplicaciones cliente y los servidores que proporcionan componentes. Como es un estándar, COM está incorporado a varios sistemas operativos y a varios compiladores.

3.2.6.1. Comunicación entre un cliente y un servidor en proceso

A nivel de sistema operativo, la velocidad de la comunicación entre un cliente y los componentes servidores varía si el servidor es un componente en proceso (una biblioteca DLL) o si es un componente fuera de proceso (un archivo .exe). La comunicación entre un cliente y un componente servidor en proceso proporciona el intercambio de

información más eficiente. Como el cliente y el componente servidor comparten el mismo espacio de direcciones, las llamadas entre ellos se hacen directamente.

3.2.6.2. Comunicación entre un cliente y un servidor fuera de proceso

La comunicación entre un cliente y un componente servidor fuera de proceso es más compleja que con un servidor en proceso, ya que tiene que pasar los límites de dos procesos. La comunicación entre un cliente y un archivo ejecutable de un equipo remoto es aún más compleja.

Para administrar esta complejidad, los compiladores compatibles con COM insertan código proxy en los clientes con el fin de tratar las llamadas a los servidores y sus valores de retorno. Cuando un autor compila un servidor de componentes compatibles con COM, se inserta código de etiqueta para tratar las llamadas desde el código proxy de los clientes y los valores devueltos a los clientes.

A este proceso de tratamiento de los parámetros y sus valores de retorno se denomina encolado. El *encolado* es el empaquetamiento de todos los parámetros y sus valores de retorno, y su envío en ambos sentidos a través de los límites de los procesos. La figura 3.3 muestra el flujo de información desde un cliente a un servidor.

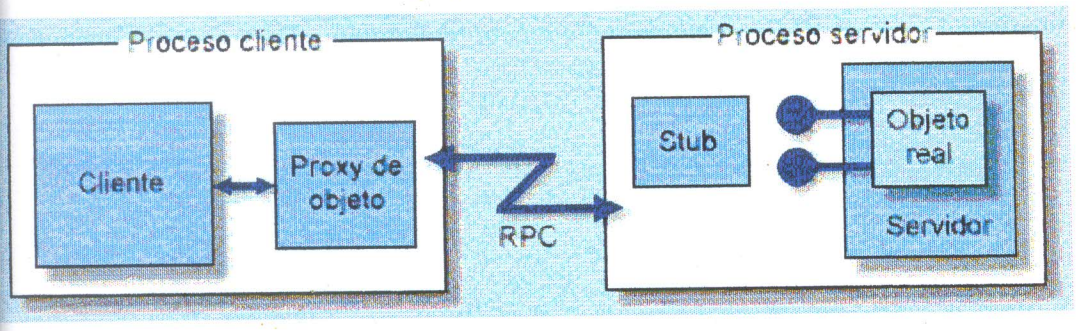


Figura 3. 3. Comunicación mediante código Proxy

Esta implementación de la comunicación entre procesos es independiente de la plataforma ya que se basa en la especificación Llamada a Procedimiento Remoto de Entorno de Computación Distribuida (DCE-RPC). Es posible realizar llamadas entre componentes compatibles con COM entre entornos Windows 95, Windows NT, UNIX y Macintosh, así como en otros entornos que acepten DCE-RPC.

3.2.7. COM, GUID y el registro del sistema

Cuando se compila un componente, el compilador crea identificadores únicos globales (GUID) para cada clase e interfaz pública del componente. Estos GUID son números de 128 bits generados mediante un algoritmo diseñado para asegurar su unicidad en todo el mundo.

Para que un componente esté disponible para los clientes, debe registrarse a sí mismo en el sistema operativo. Para hacer esto, actualiza la sección HKEY_CLASSES_ROOT del registro del sistema con

información sobre él mismo, que incluye su Id. de programa (ProgID), que es el nombre del componente en formato legible, su Id. de clase (ClassID), su ubicación y la ubicación de su *biblioteca de tipos*.

Cuando se compila una aplicación cliente que hace referencia al componente servidor, los Id. de clase y los Id. de interfaz de los objetos creados por el componente se incluyen en el archivo ejecutable. En tiempo de ejecución, el cliente (junto con COM) utiliza la información del registro para llamar al componente servidor y solicitar a dicho componente que cree un objeto que devuelva al cliente una interfaz predeterminada.

3.3. Componentes de código

Un componente de código ActiveX es una aplicación servidora que expone funcionalidad que otras aplicaciones pueden utilizar y reutilizar mediante Automatización. Los componentes de código ofrecen una completa funcionalidad.

3.3.1. Introducción a los componentes ActiveX

La tecnología ActiveX permite ensamblar componentes reutilizables en aplicaciones y servicios. En esta sección se definen los tres tipos principales de *componentes ActiveX* y se repasan las ventajas de utilizar la tecnología ActiveX.

Un componente ActiveX es una unidad de código ejecutable, como un archivo .exe, .dll u .ocx, que sigue las especificaciones de ActiveX para proporcionar objetos. Un componente ActiveX expone objetos que pueden utilizar otras aplicaciones.

Con Visual Basic es posible crear tres tipos de componentes ActiveX: Controles ActiveX, Documentos ActiveX y Componentes de Código ActiveX.

3.3.1.1. Controles ActiveX

Los controles ActiveX (antes llamados controles OLE) son elementos estándar de la interfaz de usuario que permiten ensamblar rápidamente formularios y cuadros de diálogo reutilizables. Por ejemplo, un control *Calendar* tiene una apariencia de Calendario, es ideal para pedir el ingreso de fechas a los usuarios.

3.3.1.2. Documentos ActiveX

Los documentos ActiveX (antes llamados objetos documentos) son componentes ActiveX que deben almacenarse y activarse desde una aplicación cliente. Expande la funcionalidad de la modificación visual de los objetos incrustados y permite que las aplicaciones genéricas de shell, como Internet Explorer, puedan albergar distintos tipos de documentos.

3.3.1.3. Componentes de código ActiveX

Los componentes de código son bibliotecas de objetos. Las aplicaciones cliente utilizan los componentes de código mediante la creación de objetos a partir de las clases proporcionadas por el componente. Los clientes llaman a las propiedades, métodos y eventos proporcionados por el objeto.

Cualquier aplicación que admita Automatización estándar puede utilizar los componentes de código ActiveX creados en Visual Basic. Esto incluye Visual Basic, Microsoft Excel, Microsoft Access, Microsoft Project, Microsoft Visual FoxPro, Microsoft Visual C++ e incluso HOELEST.

3.3.2. Ventajas de utilizar componentes de código ActiveX

Las ventajas de utilizar componentes de código ActiveX son:

3.3.2.1. Interfaces claramente definidas

A la funcionalidad implementada en un componente de código ActiveX se tiene acceso mediante propiedades, métodos y eventos. Estas interfaces se documentan a sí mismas en cuanto a que el componente de código las expone automáticamente. Los usuarios de Automatización pueden examinar la información descriptiva de cada interfaz mediante el Examinador de objetos u otras herramientas.

3.3.2.2. Menor complejidad

Se oculta ante otros programadores la complejidad del programa. De esta forma, los demás programadores sólo necesitarán saber qué información deben proporcionar al objeto y qué información deben recuperar.

3.3.2.3. Actualización más sencilla

Es posible crear componentes de código ActiveX que encapsulen reglas de negocios. Si cambian estas reglas, basta con actualizar el componente de código, no todas las aplicaciones que utilicen dicho componente.

3.3.2.4. Uso compartido de la información

Se pueden crear componentes de código ActiveX que muestren formularios. Después, otras aplicaciones podrán llamar a un método del componente para mostrar un formulario y recuperar información.

3.4. Creación de objetos en Visual Basic

La creación de componentes de código ActiveX implica trabajar con módulos de clase para definir los objetos proporcionados por el componente. Las clases encapsulan la funcionalidad interna de HOELEST y los clientes crean objetos a partir de las instancias de las clases en tiempo de ejecución.

3.4.1. Módulos de clase

Un módulo de clase es un tipo de módulo de código proporcionado por Visual Basic. Cada módulo de clase actúa como un plano de un objeto. Es decir, cada módulo de clase define un tipo de objeto. Se pueden tener varios módulos de clase en una misma aplicación. Para crear un objeto en tiempo de ejecución hay que crear una instancia de una clase.

Los módulos de clase representan un objeto lógico del componente. Por ejemplo, HOELEST tiene una clase *Variable* que tiene algunas propiedades, entre ellas *Variable.Nombre* y *Variable.TipoDeVariable*. Además existe una colección *Variables* que contiene a objetos *Variable*.

3.4.2. Crear métodos, propiedades y eventos

En HOELEST se crearon procedimientos de propiedad públicos dentro del módulo de clase para definir las propiedades de los objetos, puesto que permiten ejecutar código cuando se establece o se recupera el valor de una propiedad.

Los métodos son procedimientos que proporciona el objeto, por ejemplo el objeto *Hojas* tiene un procedimiento *Añadir* que añade objetos *Hoja* a la colección. Los eventos que proporciona el objeto pueden ser utilizados por un cliente para escribir código cuando se produzca algún suceso.

3.4.3. Utilizar procedimientos de propiedad para crear propiedades

Con los procedimientos de propiedad, es posible realizar las siguientes tareas cuando se establece o se recupera el valor de una propiedad:

1. Ejecutar un procedimiento cuando cambia o se lee el valor de una propiedad. 2. Restringir una propiedad a un conjunto reducido de valores válidos. 3.Exponer una propiedad que es de sólo lectura.

3.4.4. La propiedad predeterminada

Es posible establecer el valor de una propiedad predeterminada para un objeto sin hacer referencia explícita al objeto propiamente dicho. Por ejemplo, *Item* es la propiedad predeterminada de un objeto *Rango* de HOELEST; en las siguientes líneas de código se muestra un ejemplo de ello:

```
Rango.Item("A1") = "hola"
Rango("A1") = "hola"
```

3.4.5. Establecer las propiedades de un módulo de clase

Después de establecer las propiedades del proyecto HOELEST, establecimos las propiedades de cada módulo de clase del componente para determinar cómo actuará cada clase.

3.4.5.1. Propiedad Name

Las aplicaciones cliente utilizan la propiedad *Name* cuando crean una instancia de la clase. Por ejemplo, si la clase se llama MiClase y está en un proyecto denominado Proyecto1, un cliente podría utilizar el código siguiente para crear una instancia de esa clase:

```
Dim x as Proyecto1.MiClase
Set x = New Proyecto1.MiClase
```

3.4.5.2. Propiedad Instancing

La propiedad *Instancing* determina si las aplicaciones externas al proyecto pueden crear nuevas instancias de la clase y, en caso afirmativo, la forma en que se crearán dichas instancias. A continuación se definen los valores de la propiedad *Instancing*:

3.4.5.2.1. Private

No se permite a otras aplicaciones tener acceso a la información de la biblioteca de tipos acerca de la clase y dichas aplicaciones no pueden crear instancias de la misma.

Los objetos privados sólo pueden utilizarse dentro del componente. Esto se aplica a los proyectos siguientes: EXE ActiveX, DLL ActiveX, controles ActiveX y EXE estándar.

3.4.5.2.2. PublicNotCreatable

Otras aplicaciones sólo pueden utilizar los objetos de esta clase si el componente crea primero los objetos. Las demás aplicaciones no pueden utilizar la función *CreateObject* o el operador *New* para crear objetos de esta clase. Se asigna a la propiedad *Instancing* este valor cuando se desea crear *objetos dependientes*. Sólo se aplica a los proyectos siguientes: EXE ActiveX, DLL ActiveX y controles ActiveX.

3.4.5.2.3. SingleUse

Permite que otras aplicaciones creen objetos a partir de la clase, pero cada objeto de esta clase creado por un cliente inicia una nueva instancia del componente. Sólo se aplica a los proyectos EXE ActiveX

3.4.5.2.4. GlobalSingleUse

Similar a *SingleUse*, excepto en que las propiedades y los métodos de la clase se pueden llamar como si fueran funciones globales. Sólo se aplica a los proyectos EXE ActiveX.

3.4.5.2.5. MultiUse

Permite que otras aplicaciones creen objetos a partir de la clase. Una instancia del componente puede proporcionar cualquier número de objetos creados de esta forma, con independencia del número de

aplicaciones que los soliciten. Sólo se aplica a los proyectos EXE ActiveX y DLL ActiveX.

3.4.5.2.6. GlobalMultiUse

Similar a MultiUse, excepto en que las propiedades y los métodos de la clase se pueden llamar como si fueran funciones globales. No es necesario crear primero explícitamente una instancia de la clase porque se creará una automáticamente. Sólo se aplica a los proyectos EXE ActiveX y DLL ActiveX.

3.4.6. Crear métodos Friend

Hay ocasiones en las que se desea que los objetos de un componente de código puedan comunicarse entre sí, pero no desee que estén disponibles para otras aplicaciones cliente. En estos casos, se puede utilizar la palabra clave *Friend* para declarar un método de forma que su alcance sea sólo aplicable a los demás objetos del componente.

Es posible que otros objetos llamen a los métodos *Public* de una clase, pero también es posible que los llamen los clientes. No se puede llamar a los métodos *Private* desde fuera de un componente ni son visibles para los restantes objetos del componente.

Los métodos declarados con la palabra clave *Friend* son visibles para los restantes objetos del componente, pero no son visibles para los clientes porque los métodos *Friend* no se agregan a la biblioteca de tipos ni a la interfaz pública.

3.4.7. Crear un modelo de objetos

Un modelo de objetos define una relación entre los objetos de un componente de código. Algunos objetos pueden ser dependientes de otros objetos. Es decir, estos objetos dependientes sólo existen dentro del contexto de otro objeto.

Por ejemplo, en HOELEST, un objeto *Variables* no puede existir por sí mismo. Sólo existe dentro de un objeto *Hoja*. Una aplicación cliente no puede crear directamente un objeto *Variables*. El objeto *Hoja* ofrece un método *Añadir* que crea un nuevo objeto *Variables*.

Al definir las relaciones entre los objetos utilizados en el programa, un modelo de objetos organiza los objetos de forma que sea más fácil programar.

Para crear un modelo de objetos para un componente ActiveX, se siguen los pasos siguientes:

1.Se crea un objeto de nivel superior; para ello, se asigna a la propiedad *Instancing* de la clase el valor *SingleUse* o *MultiUse* ,y 2. Se crean objetos dependientes; para ello, se asigna a la propiedad *Instancing* de la clase el valor *PublicNotCreatable*.

3.4.8. Almacenar datos en una colección

Crear una colección de objetos permite almacenar juntos los elementos relacionados. En Visual Basic, el objeto *Collection* es un objeto predefinido y se puede crear de la misma forma que otros objetos, como se muestra en el código siguiente:

```
Dim Employees As New Collection
```

El tipo de datos *Collection* tiene predefinidos los métodos *Add* y *Remove* y una propiedad *Count*, para añadir, eliminar y contar el número de elementos en la colección.

Almacenar los objetos relacionados en una colección, ayuda a trabajar fácilmente con esos objetos como un grupo.

CAPITULO 4

4. DISEÑO DEL COMPONENTE DLL ACTIVE X

El nombre del componente DLL ActiveX es EstadísticaInferencialDLL. Este componente contiene funciones para el análisis estadístico de datos, todos ellas accesibles desde un único objeto *EstadísticaInferencial* proporcionado por el mismo.

Por medio del objeto *EstadísticaInferencial*, permite que usuarios experimentados, tales como programadores de clientes ActiveX, utilicen sus métodos para crear aplicaciones personalizadas. De hecho, este componente se diseñó para que actúe como servidor para el proyecto Hoelest (Exe ActiveX) que revisaremos en el capítulo 5.

4.1. Referencias del Proyecto

El proyecto DLL ActiveX mantiene referencias a dos componentes: DistribucionesProy y Matriz. EL componente EstadísticaInferencialDLL

tiene acceso a las funciones (métodos) que proporcionan ambos componentes para poder realizar los cálculos estadísticos que proporciona.

Se tiene entonces que EstadísticaInferencialDLL es un cliente de los componentes DistribucionesProy y Matriz y, al mismo tiempo, es un servidor para el componente Hoelest.

4.2. Diseño del Proyecto DistribucionesProy

DistribucionesProy es un componente de código DLL Active X. El propósito de este componente, es brindar a sus clientes un conjunto de funciones que devuelven valores y probabilidades de las principales funciones de probabilidad, como por ejemplo proporciona un método para calcular la probabilidad de una variable aleatoria Normal dado un punto en la recta real.

4.2.1. Objetos del Componente DistribucionesProy.

El componente DistribucionesProy proporciona un solo objeto, cuya única función es permitir el acceso a los métodos proporcionados por el componente. El nombre del objeto es Distribuciones. No tiene interfaz visual.

4.2.2. Módulos de clase

DistribucionesProy sólo cuenta con un módulo de clase, cuya propiedad Instancing está establecida a MultiUse, de forma que las aplicaciones cliente podrán crear objetos a partir de la clase y el componente podrá tratar múltiples objetos a partir de cada clase.

En la figura 4.1 se ilustra este modelo de objetos:

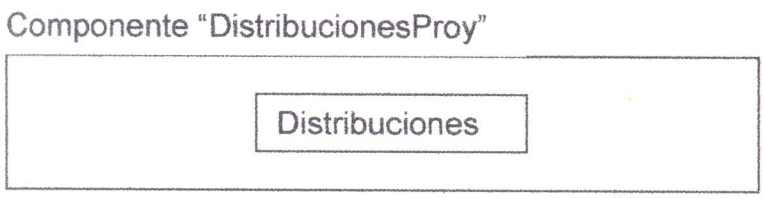


Figura 4. 1. Modelo sencillo de objetos. Componente DistribucionesProy

4.2.3. Descripción del Objeto Distribuciones

Su único propósito es permitir el acceso a los métodos del componente.

4.2.3.1. Método GammaEnteroPositivo

Devuelve el valor de la función Gamma de un entero positivo. Sintaxis:

GammaEnteroPositivo(N)

N Integer requerido. El número entero positivo que servirá como argumento de la función gamma. En este caso, el valor devuelto será el mismo que si se aplicara la función factorial(N-1).

4.2.3.2. Método GammaUnMedioMasN

Devuelve el valor de la función Gamma de un argumento ½ + N. Sintaxis:

GammaUnMedioMasN(N)

N Integer requerido. El número entero positivo o negativo que sumado a ½ servirá como argumento para evaluar la función gamma.

4.2.3.3. Método ProbabilidadFdeFisher

Devuelve una probabilidad de la distribución F de Fisher. Sintaxis:

ProbabilidadFdeFisher(Punto, GL1, GL2)

Punto Double requerido. Calcula la probabilidad acumulada hasta Punto.

GL1 Integer requerido. Grados de libertad del numerador.

GL2 Integer requerido. Grados de libertad del denominador

4.2.3.4. Método ProbabilidadJiCuadrada

Devuelve una probabilidad de la distribución Ji Cuadrada. Sintaxis:

ProbabilidadJiCuadrada(Punto, GL)

Punto Double requerido. Calcula la probabilidad acumulada hasta Punto.

GL1 Integer requerido. Grados de libertad.

4.2.3.5. Método ProbabilidadNormal

Devuelve una probabilidad de la Normal Estándar. Sintaxis:

ProbabilidadNormal(Punto)

Punto Double requerido. Calcula la probabilidad acumulada hasta Punto.

4.2.3.6. Método ProbabilidadTdeStudent

Devuelve una probabilidad de la distribución T de Student. Sintaxis:

ProbabilidadTdeStudent(Punto, GL)

Punto Double requerido. Calcula la probabilidad acumulada hasta Punto.

GL1 Integer requerido. Grados de libertad.

4.2.3.7. Método ValorNormal

Devuelve el percentil de una distribución Normal Estándar dada una probabilidad. Sintaxis:

ValorNormal(Prob)

Prob Double requerido. Probabilidad que utilizará para encontrar el percentil correspondiente.

4.2.3.8. Método ValorTdeStudent

Devuelve el percentil de una distribución T de Student dada una probabilidad y los grados de libertad. Sintaxis:

ValorTdeStudent(Prob,GL)

Prob Double requerido. Probabilidad que utilizará para encontrar el percentil correspondiente.

GL Integer requerido. Grados de libertad.

4.2.4. Módulos de código

Solo existe un módulo de código estándar. Este módulo contiene funciones privadas necesarias para el cálculo de las probabilidades y los percentiles. Estas funciones son extraídas de los diversos métodos numéricos que fueron investigados para poder realizar estos cálculos que son complejos.

Las funciones utilizadas para el cálculo de la probabilidad Normal son: erf, erfc, R1, R2, R3 y ProbabilidadNormal. Las funciones utilizadas para el cálculo de puntos porcentuales Normales son Z2 y ValorNormal. Las funciones utilizadas para el cálculo de la probabilidad Ji Cuadrada son F1, F2 y ProbabilidadJiCuadrada. Para la probabilidad F de Fisher se utilizaron F4 y ProbabilidadFdeFisher. Para la probabilidad T de Student F3 y ProbabilidadTdeStudent. Para los puntos porcentuales de la T de Student se utilizaron: AproxT1paraTdeStudent, Fi1, Fi1prima, W y ValorTdeStudent.

4.3. Diseño del proyecto Matrices

Matrices es un componente de código DLL Active X. El propósito de este componente, es brindar a sus clientes un conjunto de funciones y procedimientos que manipulan matrices; por ejemplo, se proporciona un método para obtener la inversa de una matriz cuadrada.

4.3.1. Objetos del Componente Matrices.

El componente Matrices proporciona un solo objeto, cuya única función es permitir el acceso a los métodos proporcionados por el componente. El nombre del objeto es Matrices. No tiene interfaz visual.

4.3.2. Módulos de clase

Matrices sólo cuenta con un módulo de clase, cuya propiedad Instancing está establecida a MultiUse, de forma que las aplicaciones cliente podrán crear objetos a partir de la clase y el componente podrá tratar múltiples objetos a partir de cada clase. En la figura 4.2 se ilustra este modelo de objetos:

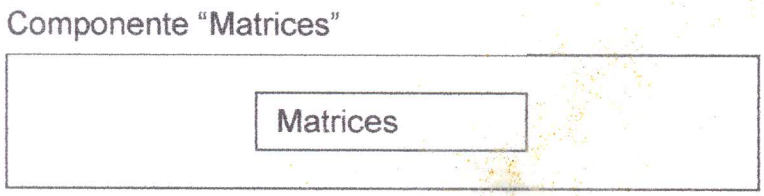


Figura 4. 2. Modelo sencillo de objetos. Componente Matrices

4.3.3. Descripción del Objeto Matrices

Su único propósito es permitir el acceso a los métodos del componente.

4.3.3.1. Método MatAumentaMatriz

Devuelve True si aumenta una matriz con éxito. Sintaxis:

MatAumentaMAtriz(MatInicial, MatNueva,MatResult)

MatInicial Matriz requerida. Matriz a la que se unirá una matriz nueva

MatNueva Matriz requerida. Matriz que se aumentará a la derecha de MatInicial

MatResult Matriz requerida. Matriz resultado de la unión de MatInicial y MatNueva.

4.3.3.2. Método MatCombLinealColumna

Multiplica un escalar1 por columna1 y escalar2 (1 por defecto) por columna2, luego sumamos a columna 3 si se especifica, sino a la columna 2. Sintaxis

MatCombLinealColumna(Matriz, columna1, columna2, escalar1, escalar2, columna 3)

Matriz Requerido. Matriz con la que se va a operar

Columna 1 Long requerido. Indice de columna que se multiplicará con escalar 1.

Columna 2 Long requerido. Indice de columna que se multiplicará con escalar 2.

Escalar 1 Single requerido. Valor que multiplicará todos los elementos de columna1

Escalar 2 Single opcional. Valor que multiplicará todos los elementos de columna2. Por defecto tiene el valor de 1.

Columna 3 Long opcional. Indice de columna a la que se sumará la columna resultado. Si no se especifica, se utilizará la columna2.

4.3.3.3. Método MatCombLinealFila

Multiplica un escalar1 por fila1 y escalar2 (1 por defecto) por fila2, luego sumamos a fila 3 si se especifica, sino a la fila 2. Sintaxis

MatCombLinealFila(Matriz, fila1, fila2, escalar1, escalar2, fila 3)

Matriz Requerido. Matriz con la que se va a operar

fila 1 Long requerido. Indice de fila que se multiplicará con escalar 1.

fila 2 Long requerido. Indice de fila que se multiplicará con escalar 2.

Escalar 1 Single requerido. Valor que multiplicará todos los elementos de fila1

Escalar 2 Single opcional. Valor que multiplicará todos los elementos de fila2. Por defecto tiene el valor de 1.

fila 3 Long opcional. Indice de columna a la que se sumará la fila resultado. Si no se especifica, se utilizará la fila2.

4.3.3.4. Método MatConstante

Genera una matriz con todos los elementos iguales a un escalar. Sintaxis:

MatConstante(Matriz, Filas, Columnas, Escalar)

Matriz Requerido. Matriz que se generará.

Filas Long requerido. Número de filas que tendrá Matriz

Columnas Long requerido. Número de columnas que tendrá Matriz

Escalar Single requerido. Valor que se asignará a cada entrada.

4.3.3.5. Método MatCopiar

Hace una copia de una matriz. Sintaxis:

MatCopiar (MatOriginal, MatResult)

MatOriginal Requerido. Matriz de la que se creará una copia

MatResult Requerido. Matriz Resultado, igual a MatOriginal

4.3.3.6. Método MatExtraerDiagonal

Extrae sólo los elementos de la diagonal de una matriz cuadrada. Sintaxis:

MatExtraerDiagonal(Matriz, Diag)

Matriz Requerido. Matriz de la que se extraerá la diagonal.

Diag Requerido. Matriz que contendrá los elementos de la diagonal.

4.3.3.7. Método MatIntercambiarColumnas

Intercambia dos columnas de una matriz. Sintaxis:

MatIntercambiarColumnas(Matriz,Col1, Col2)

Matriz	Requerido. Matriz con la que se operará
Col1, Col2	Long requeridos. Indices de columnas a intercambiar

4.3.3.8. Método MatIntercambiarFilas

Intercambia dos filas de una matriz. Sintaxis:

MatIntercambiarFilas(Matriz,Fila1,Fila2)

Matriz	Requerido. Matriz con la que se operará
Fila1, Fila2	Long requeridos. Indices de las filas a intercambiar

4.3.3.9. Método MatInversa

Devuelve True si se obtuvo la inversa de una matriz. Sintaxis:

MatInversa(Matriz)

Matriz Requerido. Matriz que se va a invertir

4.3.3.10. Método MatInvertirColumnas

Invierte el orden de las columnas de una matriz. Sintaxis:

MatInvertirColumnas(Matriz)

Matriz Requerido. Matriz a la que se invertirá el orden de las columnas

4.3.3.11. Método MatInvertirFilas

Invierte el orden de las filas de una matriz. Sintaxis:

MatInvertirFilas(Matriz)

Matriz Requerido. Matriz a la que se invertirá el orden de las filas

4.3.3.12. Método MatMultiplicar

Devuelve true si se multiplicó dos matrices con éxito. Sintaxis:

MatMultiplicar(MatA, MatB, MatResult)

MatA, MatB Requeridos. Matrices que se multiplicarán.

MatResult Requerido. Matriz que tendrá el resultado de la multiplicación.

4.3.3.13. Método MatMultiplicarEscalar

Multiplica una matriz por un escalar. Sintaxis:

MatMultiplicarEscalar(Matriz, Escalar)

Matriz Requerido. Matriz con la que se operará

Escalar Single requerido. Valor que multiplicará todos los elementos de Matriz.

4.3.3.14. Método MatMultiplicarEscalarColumna

Multiplica una determinada columna de una matriz por un escalar. Sintaxis:

MatMultiplicarEscalarColumna(Matriz, Escalar, Columna)

Matriz Requerido. Matriz con la que se operará

Escalar Single requerido. Escalar que multiplicará los elementos de la columna especificada.

Columna Long requerido. Indice de la columna con la que se desea operar.

4.3.3.15. Método MatMultiplicarEscalarFila

Multiplica una determinada fila de una matriz por un escalar. Sintaxis:

MatMultiplicarEscalarFila(Matriz, Escalar, Fila)

Matriz Requerido. Matriz con la que se operará

Escalar Single requerido. Escalar que multiplicará los elementos de la fila especificada.

Fila Long requerido. Indice de la fila con la que se desea operar.

4.3.3.16. Método MatOrdenar

Ordena todas las filas de una matriz según el orden de una columna especificada. Sintaxis:

MatOrdenar (Matriz, Columna)

Matriz requerido. Matriz que se desea ordenar

Columna Long requerido. Indice de columna que se ordenará ascendentemente.

4.3.3.17. Método MatPartición

Obtiene una nueva matriz extrayéndola de otra. Sintaxis:

MatPartición (Matriz, MatResult, FilaSup, ColIzq, FilaInf, ColDer)

Matriz Requerido. Matriz de la que se extraerá otra.

MatResult Requerido. Matriz extraída de la original

FilaSup, FilaInf Long requerido. Indices superior e inferior de filas que indican el inicio y fin de la extracción por filas.

ColIzq, ColDer Long requerido. Indices izquierdo y derecho de columnas que indican inicio y fin de la extracción por columnas.

4.3.3.18. Método MatRaizDeElementos

Calcula la raíz cuadrada de los elementos de una matriz. Sintaxis:

MatRaizDeElementos(Matriz)

Matriz Requerido. Matriz, a cuyos elementos se les calculará la raíz cuadrada.

4.3.3.19. Método MatRestar

Devuelve True si se restan dos matrices con éxito Sintaxis:

MatRestar(Matriz1,Matriz2,MarResult)

Matriz1, Matriz2 Requeridos. Matrices que se restarán.

MatResult Requerido. Matriz que contendrá el resultado de la resta

4.3.3.20. Método MatSumar

Devuelve True si se suman dos matrices con éxito Sintaxis:

MatSumar(Matriz1,Matriz2,MarResult)

Matriz1, Matriz2 Requeridos. Matrices que se sumarán.

MatResult Requerido. Matriz que contendrá el resultado de la suma

4.3.3.21. Método MatTranspuesta

Obtiene la Transpuesta de una matriz. Sintaxis:

MatTranspuesta (Matriz)

Matriz Requerido. Matriz con la que se operará

4.3.3.22. Método MElementoMáximo

Devuelve el máximo valor de una columna especifica de una matriz. Sintaxis:

MElementoMáximo(Matriz,Col)

Matriz Requerido. Matriz con la que se operará

Col Long requerido. Indice de columna de la que se obtendrá el elemento máximo.

4.3.3.23. Método MElementoMínimo

Devuelve el mínimo valor de una columna específica de una matriz. Sintaxis:

MElementoMínimo(Matriz,Col)

Matriz Requerido. Matriz con la que se operará

Col Long requerido. Indice de columna de la que se obtendrá el elemento mínimo.

4.3.3.24. Método MTriangularizar

Devuelve true si se logró triangularizar una matriz con éxito. Sintaxis:

MTriangularizar(MatMiembroIzq, MatMiembroDer, Tipo)

MatMiembroIzq, MatMiembroDer Requeridos. Miembros izquierdo y derecho de un sistema. Hace la triangularización del miembro izquierdo.

Tipo Integer opcional. Puede ser TipoTriangInferior o TipoTriangSuperior.

4.3.4. Módulos de código

Solo existe un módulo de código estándar. Este módulo contiene realmente todas los métodos que manejan matrices. Cuando se invoca un método, el objeto Matrices simplemente pasa los argumentos a estas

funciones internas y luego las recupera y pasa al cliente que se lo solicitó.

4.3.5. Constantes con nombre

EL componente Matrices proporciona sólo una constante con nombre que es TipoTriangularización y, que puede acoger los valores de TipoTriangInferior y TipoTriangSuperior.

4.4. Diseño del modelo de objetos de EstadísticaInferencialDLL

El propósito de este componente, es brindar a sus clientes un conjunto de funciones y procedimientos para el análisis estadístico de datos; por ejemplo, se proporciona un método ANOVA para realizar una análisis de varianza. El componente EstadísticaInferencialDLL proporciona un solo objeto, cuya única función es permitir el acceso a los métodos proporcionados por el componente. EL nombre del objeto es *EstadísticaInferencial*. No tiene interfaz visual.

Esta sección detalla los métodos que expone el componente, más no considera como interpretar los resultados. El Capítulo 6 se encarga de introducir al lector sobre los conceptos de la estadística inferencial que están dentro del alcance de Hoelest, además de describir cómo se tiene acceso a ellos y cómo deben interpretarse sus resultados.

4.4.1. Módulos de clase

EstadísticaInferencialDLL sólo cuenta con un módulo de clase, cuya propiedad Instancing está establecida a MultiUse, de forma que las aplicaciones cliente podrán crear objetos a partir de la clase y el componente podrá tratar múltiples objetos a partir de cada clase. En la figura 4.3 se ilustra este sencillo modelo de objetos:

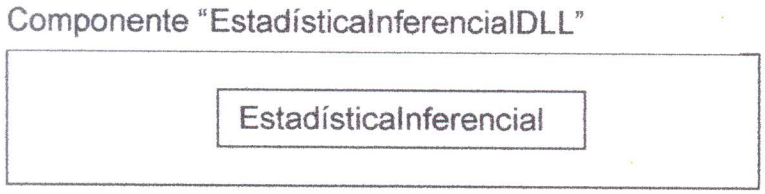


Figura 4. 3. Modelo de objetos. Componente EstadisticaInferencialDLL

4.4.2. Descripción del Objeto EstadísticaInferencial

Su único propósito es permitir el acceso a los métodos del componente.

4.4.2.1. Método AleatorioExponencial

Devuelve un número aleatorio exponencial. Sintaxis:

AleatorioExponencial(Beta)

Beta Single requerido. Media de la distribución exponencial.

4.4.2.2. Método AleatorioNormal

Devuelve un número aleatorio normal. Sintaxis:

AleatorioNormal(Media, Desviación)

Media Single requerido. Media de la distribución normal

Desviación Single requerido. Desviación estándar de la distribución normal

4.4.2.3. Método AleatorioNormalEstándar

Devuelve un número aleatorio normal estándar. Sintaxis:

AleatorioNormalEstándar

4.4.2.4. Método Aleatorio Uniforme

Devuelve un número aleatorio uniforme entre a y b. Sintaxis:

AleatorioUniforme(a,b)

a,b Single requeridos. Límites inferior y superior de la distribución uniforme.

4.4.2.5. Método ANOVA

Devuelve True si se logró realizar un análisis de varianza con éxito. Sintaxis:

ANOVA(Observaciones, Factores, TablaAnovaResultado, Nombres, varBloques , varLatinos)

Observaciones Requerido. Matriz con los valores observados del experimento

Factores Requerido. Matriz que contiene los factores.

TablaAnovaResultado Requerido. Matriz que contendrá organizados los resultados en una tabla.

Nombres Requerido. Matriz con los nombres de los factores.

VarBloques Boolean opcional. True si se trata de un diseño por bloques.

VarLatinos Boolean opcional. True si se trata de un diseño de cuadrados latinos.

4.4.2.6. Método Combinaciones

Devuelve el número de combinaciones de k objetos tomados n a la vez, es decir, el número de subconjuntos, cada uno de tamaño n, que se pueden formar a partir de los k objetos. Sintaxis:

Combinaciones(n,k)

N Tamaño de los subconjuntos, N es menor o igual que K

K Número total de objetos.

4.4.2.7. Método Factorial

Devuelve el factorial de un número entero positivo. Sintaxis:

Factorial(N)

N Long requerido. Entero sobre el cual se calculará el factorial.

4.4.2.8. Método GeneraMatrizDiseño

Devuelve True si se logró generar una matriz de diseño con éxito. Genera matrices de diseño que guiarán al usuario tanto para efectuar las mediciones en un experimento como para ingresar los datos del mismo en

una hoja electrónica. Sintaxis:

GeneraMatrizDeDiseño(TablaResultado, TipoDiseño, NumNiveles, NumFactores, NumReplicas, NumBloques)

TablaResultado Requerido. Matriz que contenga los resultados con un formato de salida preestablecido.

TipoDiseño Long requerido. Puede ser una de las siguientes constantes: mHLTTipoDiseño: mHLTFactorial, mHLTBloques o mHLTLatinos.

NumNiveles Long requerido. Número de niveles o tratamientos por cada factor.

NumFactores Long requerido. Número de factores en el diseño.

NumReplicas Long requerido. Número de réplicas o de repeticiones de muestreo.

NumBloques Long opcional. Número de bloques del diseño, sólo se toma en cuenta cuando TipoDiseño es mHLTBloques.

4.4.2.9. Método KruskalWallis

Devuelve True si se logró realizar un análisis de varianza por medio del método desarrollado por Kruskal-Wallis con éxito. Sintaxis:

KruskalWallis(Observaciones, Factores, TablaResultado, Nombres)

Observaciones Requerido. Matriz con los valores observados del experimento

Factores Requerido. Matriz que contiene los factores.

TablaResultado Requerido. Matriz que contendrá organizados los resultados en una tabla.

Nombres Requerido. Matriz con las etiquetas de los factores que intervienen en el análisis.

4.4.2.10. Método MatrizDeCoeficientesCorrelacion

Devuelve True si se logró realizar con éxito el cálculo de los coeficientes de correlación lineal de un conjunto de variables. Sintaxis:

MatrizDeCoeficientesCorrelacion(matriz, S, TablaResultado, Nombres)

Matriz Requerido. Matriz cuyas columnas se tratarán como variables para el cálculo de las correlaciones.

S Requerido. Matriz que contiene solamente los coeficientes.

TablaResultado Requerido. Matriz que contendrá organizados los resultados en una tabla.

Nombres Requerido. Matriz con las etiquetas de las variables que intervienen en el análisis.

4.4.2.11. Método MatrizDeCovarianzas

Devuelve True si se logró realizar con éxito el cálculo de las varianzas y covarianzas de un conjunto de variables. Sintaxis:

MatrizDeVarianzas(matriz, S, TablaResultado, Nombres)

Matriz Requerido. Matriz con los valores observados del experimento.

S Requerido. Matriz que contiene las varianzas y covarianzas de las variables.

TablaResultado Requerido. Matriz que contendrá organizados los resultados en una tabla.

Nombres Requerido. Matriz con las etiquetas de las variables que intervienen en el análisis.

4.4.2.12. Método PHDosMuestras

Devuelve True si se realizó con éxito el cálculo de pruebas de hipótesis para comparar las medias de dos poblaciones. Sintaxis:

PHDosMuestras(MatVariables, ConfidenciaHP, TablaResultado, Nombres)

MatVariables Requerido. Matriz cuyas columnas representan las variables.

ConfidenciaHP Single requerido. Nivel de confianza utilizado para elaborar el intervalo de confianza.

TablaResultado Requerido. Matriz que contendrá organizados los resultados en una tabla.

Nombres Requerido. Matriz con los nombres de los factores.

4.4.2.13. Método PHSimple

Devuelve True si se realizó con éxito el cálculo de prueba de hipótesis para comparar la media de una población con un valor y nivel de confianza dados. Sintaxis:

PHSimple(MatVariables, MediaHP, ConfidenciaHP, TablaResultado, Nombres)

MatVariables Requerido. Matriz cuyas columnas representan las variables.

MediaHP Single requerido. Valor que corresponde a la hipótesis nula.

ConfidenciaHP Single requerido. Nivel de confianza utilizado para elaborar el intervalo de confianza.

TablaResultado Requerido. Matriz que contendrá organizados los resultados en una tabla.

Nombres Requerido. Matriz con los nombres de los factores.

4.4.2.14. Método RegresiónLS

Devuelve True si se realizó con éxito el cálculo de un análisis de Regresión Lineal. Sintaxis:

RegresiónLS(X, y, beta, sce, TablaResultado, Nombres, constante)

X Requerido. Matriz cuyas columnas representan las variables dependientes del modelo.

Y Requerido. Matriz de una sola columna que contiene los datos de la variable independiente.

TablaResultado Requerido. Matriz que contendrá organizados los resultados en una tabla.

Nombres Requerido. Matriz con los nombres de los factores.

Constante Boolean opcional. Especifica si se añade la constante en el modelo de regresión lineal.

4.4.2.15. Método VectorMedia

Calcula la media muestral de un conjunto de variables. Sintaxis:

VectorMedia (Matriz, Media)

Matriz Requerido. Matriz cuyas N columnas representan N variables.

Media Requerido. Matriz con N filas y una columna. El elemento de la fila i es la media calculada de la columna i de Matriz.

4.4.2.16. Método Wilcoxon

Devuelve True si se logró realizar con éxito una prueba de Wilcoxon de rangos con signo. Sintaxis:

Wilcoxon(MatrizVariables, TablaResultado, Nombres)

MatVariables Requerido. Matriz cuyas columnas representan variables.

TablaResultado Requerido. Matriz que contendrá organizados los resultados en una tabla.

Nombres Requerido. Matriz con los nombres de los factores.

4.5. Módulos de Código

EstadísticaInferencialDLL tiene dos módulos de código estándar: FuncionesBase y mEstInferencial.

4.5.1. Módulo FuncionesBase

Solamente contiene dos funciones que son útiles para sumar los elementos de una matriz. Estas funciones son utilizadas en el cálculo de las sumas cuadráticas en el análisis de varianza.

Las funciones son: *Suma* y *SumaDeCuadrados*

4.5.2. Módulo mEstInferencial

En este módulo se encuentra el código que realiza los cálculos estadísticos. Los métodos que expone el objeto EstadísticaInferencial invocan a las funciones contenidas dentro del módulo para realizar las operaciones de cálculo.

Los procedimientos que tiene son: VectorMedia, IncluirConstanteRegresión.

Las funciones que tiene son: Aleatorio, AleatorioExponencial, AleatorioNormal, AleatorioNormalEstándar, AleatorioUniforme, ANOVA,

fbloques, fLatinos, UnFactor, VariosFactores, Combinación, Factorial, GeneraMatrizDiseño, KruskalWallis, MatrizDeCoeficientesCorrelación, MatrizDeCovarianzas, PHDosMuestras, PHSimple, RegresiónLS, y Wilcoxon.

4.6. Constantes con nombre

El componente EstadísticaInferencialDLL proporciona sólo una constante con nombre que es mHLTTipoDiseño que puede tomar los valores: mHLTBloques, mHLTFactorial y mHLTLatino.

CAPITULO 5

5. DISEÑO DEL COMPONENTE EXE ACTIVE X

El nombre del componente EXE ActiveX es HOELEST. Este componente tiene como función principal proporcionar la interfaz visual de la aplicación HOELEST, utiliza el componente de código DLL ActiveX EstadísticaInferencialDLL para el análisis estadístico de datos y, se encarga de la presentación visual e impresa de datos y resultados.

La interfaz visual de HOELEST se desarrolló pensando en el usuario común; es decir, aquel que ejecuta HOELEST como una aplicación de escritorio como Microsoft Word o Excel. En realidad este componente no es más que "el rostro de Hoelest", puesto que el "cerebro" que realiza los cálculos es el componente EstadísticaInferencialDLL.

Además HOELEST proporciona un modelo de objetos. Este modelo permite que usuarios experimentados, tales como programadores de

clientes ActiveX utilicen los objetos de HOELEST para crear aplicaciones personalizadas.

5.1. Referencias del Proyecto

El proyecto Exe ActiveX tiene una única referencia y, es precisamente el componente EstadísticaInferencialDLL. Por medio de esta referencia, Hoelest tiene acceso a las funciones (métodos) que proporciona dicho componente por medio de su objeto EstadísticaInferencial.

Por lo expresado, se tiene entonces que Hoelest se convierte en un cliente ActiveX del componente EstadísticaInferencialDLL, ya que éste actúa como servidor de funciones de análisis estadístico.

5.2. Diseño del Modelo de Objetos del Componente Exe ActiveX

El modelo de objetos de Hoelest define la relación entre los objetos del componente. Describiremos los módulos de clase y posteriormente cada uno de los objetos que se crean a partir de éstos.

5.2.1. Módulos de clase

Los módulos de clase son las plantillas para la creación de objetos. Hoelest expone 7 objetos en su biblioteca de tipos; sin embargo, es capaz de crear 11 objetos. Los cuatro objetos (Formulas, Formulas, Variables y

Variable) que no expone son de uso interno.

La propiedad Instancing de los módulos de clase establece los lineamientos de la creación de objetos de Hoelest desde una aplicación externa, como los detalla la tabla IV:

Tabla IV

Valores de la propiedad Instancing de los módulos de clase de Hoelest

Objeto	Valor propiedad Instancing
Origen	SingleUse
Hojas	PublicNotCreatable
Hoja	MultiUse
Variables	Private
Variable	Private
Formulas	Private
Formula	Private
Rango	PublicNotCreatable
Fuente	PublicNotCreatable
ConfiguraciónPágina	PublicNotCreatable
FuncionesHoja	PublicNotCreatable

5.2.2. Descripción del modelo de objetos

El signo asterisco (*) al final del nombre del objeto indica que se trata de un objeto de creación pública, todos los demás son objetos dependientes. Además, el signo de número (#) indica que se trata de un objeto privado.

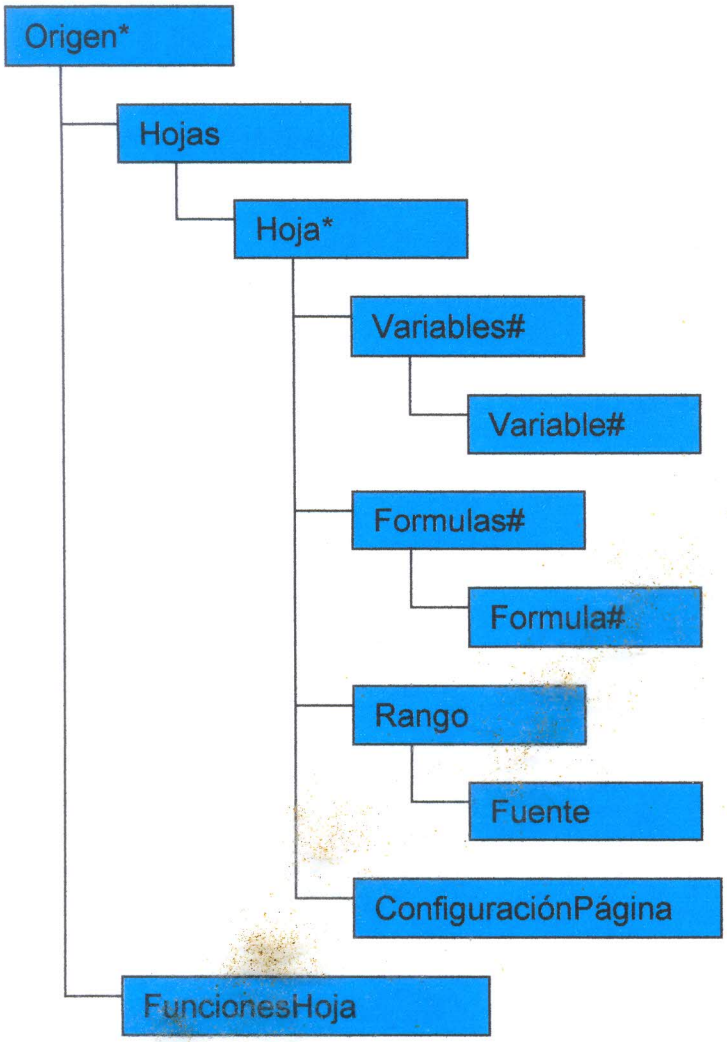


Figura 5. 1. Modelo de objetos de Hoelest

5.2.3. Objeto Origen

Representa la aplicación completa Hoelest. El objeto *Origen* contiene valores y opciones de toda la aplicación (por ejemplo, muchas de las opciones del cuadro de diálogo Opciones, en el menú Herramientas), además de métodos que devuelven objetos de nivel superior, como *CeldaActiva*, *HojaActiva*, etc.

5.2.3.1. Propiedades del objeto Origen

5.2.3.1.1. Propiedad CeldaActiva

Devuelve un objeto **Rango** que representa la celda activa de la ventana activa (la ventana superior) o de la ventana especificada. Es de sólo lectura.

Celda activa no es lo mismo que selección. La celda activa es una sola celda de la selección.

La selección puede contener más de una celda, pero sólo una es la celda activa.

5.2.3.1.2. Propiedad ImpresoraActiva

Devuelve o establece el nombre de la impresora activa. String de Lectura/Escritura.

5.2.3.1.3. Propiedad HojaActiva

Devuelve un objeto **Hoja** que representa la hoja activa (la ventana superior). Devuelve Nothing si no hay ventanas abiertas. Es de sólo lectura.

5.2.3.1.4. Propiedad CalcularAntesGuardar

Verdadero si las hojas se calculan antes de guardarlas en el disco (si la propiedad ModoCálculo se establece como mHLTManual). Esta propiedad se conserva incluso cuando cambia la propiedad ModoCálculo. Boolean de Lectura/Escritura.

5.2.3.1.5. Propiedad ModoCálculo

Devuelve o establece el modo de cálculo. Puede ser una de las siguientes constantes mHLTModoCálculo: mHLTAutomatico, mHLTManual. Long de Lectura/Escritura.

5.2.3.1.6. Propiedad Título

Devuelve el nombre que aparece en la barra de título de la ventana principal de HOELEST. Si no establece un nombre o si establece el nombre como el valor Empty, esta propiedad devolverá "HOELEST". String de Lectura/Escritura.

5.2.3.1.7. Propiedad Celdas

Devuelve un objeto **Rango** que representa todas las celdas de la hoja de cálculo activa. Es de sólo lectura.

5.2.3.1.8. Propiedad Columnas

Devuelve un objeto **Rango** que representa todas las columnas de la hoja de cálculo activa. Es de sólo lectura.

5.2.3.1.9. Propiedad MostrarBarraFórmulas

Verdadero si está visible la barra de fórmulas. Boolean de Lectura/Escritura.

5.2.3.1.10. Propiedad PantallaCompleta

Verdadero si HOELEST está en el modo de pantalla completa. Boolean de Lectura/Escritura. El modo de pantalla completa maximiza la ventana de la aplicación, para que ocupe toda la pantalla y oculte la barra de título de la aplicación.

Las barras de herramientas, la barra de estado y la barra de fórmulas mantienen configuraciones de presentación independientes en los modos de pantalla completa y normal.

5.2.3.1.11. Propiedad MostrarArchivosRecientes

Verdadero si se muestra la lista de archivos de uso reciente en el menú Archivo. Boolean de Lectura/Escritura.

5.2.3.1.12. Propiedad MostrarBarraEstado

Verdadero si está visible la barra de estado. Boolean de Lectura/Escritura.

5.2.3.1.13. Propiedad Alto

Devuelve o establece el alto de la ventana principal de la aplicación en puntos. Si la ventana está minimizada, esta propiedad es de sólo lectura y se refiere al alto del icono. Si la ventana está maximizada, no se puede establecer la propiedad. Long de Lectura/Escritura.

5.2.3.1.14. Propiedad Izquierda

Devuelve o establece la distancia desde el borde izquierdo de la pantalla hasta el borde izquierdo de la ventana principal de HOELEST. Long de Lectura/Escritura.

5.2.3.1.15. Propiedad DirectorioBiblioteca

Devuelve la ruta del directorio BIBLIOTECA, sin incluir el separador final. String de sólo lectura.

5.2.3.1.16. Propiedad DirectorioTrabajo

Devuelve la ruta completa del objeto (ruta de la aplicación HOELEST), a excepción del separador final y el nombre del objeto. String de sólo lectura.

5.2.3.1.17. Propiedad Rango

Devuelve un objeto **Rango** que representa una celda o un **Rango** de celdas.

5.2.3.1.18. Propiedad Filas

Devuelve un objeto **Rango** que representa todas las filas de la hoja de cálculo activa. Si no hay documento activo, esta propiedad fallará. Es de sólo lectura.

5.2.3.1.19. Propiedad Selección

Devuelve el objeto seleccionado en la ventana activa. (por ejemplo, si está seleccionada una celda, la propiedad devolverá un objeto **Rango**).

Esta propiedad puede extender su uso en una futura versión de Hoelest, puesto que actualmente el único objeto que puede devolver esta propiedad es un objeto Rango.

5.2.3.1.20. Propiedad FuenteEstándar

Devuelve o establece el nombre de la fuente estándar. String de Lectura/Escritura.

Si se modifica la fuente estándar mediante esta propiedad, el cambio no será efectivo hasta que se vuelva a abrir HOELEST.

5.2.3.1.21. Propiedad TamañoFuenteEstándar

Devuelve o establece el tamaño de fuente estándar en puntos. Long de Lectura/Escritura.

Si se modifica el tamaño de la fuente estándar mediante esta propiedad, el cambio no será efectivo hasta que se vuelva a abrir HOELEST.

5.2.3.1.22. Propiedad Tope

Devuelve o establece la distancia desde el borde superior de la pantalla hasta el borde superior de la ventana principal de HOELEST. Si la ventana de la aplicación está minimizada, esta propiedad controla la posición del icono de la ventana.

5.2.3.1.23. Propiedad Valor

Siempre devuelve "HOELEST". Es de sólo lectura.

5.2.3.1.24. Propiedad Versión

Devuelve el número de versión de HOELEST. String de sólo lectura.

5.2.3.1.25. Propiedad Visible

Verdadero si el objeto Origen está visible. Boolean o Long de Lectura/Escritura.

5.2.3.1.26. Propiedad Ancho

Devuelve o establece el ancho de un objeto, en este caso la distancia desde el borde izquierdo de la ventana de la aplicación hasta el borde derecho de la ventana. Long de Lectura/Escritura. Si la ventana está minimizada, la propiedad es de sólo lectura y devuelve al ancho del icono de la ventana.

5.2.3.1.27. Propiedad EstadoVentana

Devuelve o establece el estado de la ventana. Puede ser una de las siguientes constantes mHLTEstadoVentana: mHLTMaximizado, mHLTMinimizado o mHLTNormal. Long de Lectura/Escritura.

5.2.3.1.28. Propiedad Hojas

Devuelve un conjunto Hojas que representa todos las hojas abiertas. Es de sólo lectura.

5.2.3.1.29. Propiedad FuncionesHoja

Devuelve el objeto **FuncionesHoja**. Es de sólo lectura.

5.2.3.2. Métodos del Objeto Origen

5.2.3.2.1. Método Calcular

Calcula todos las hojas abiertas en la aplicación.

5.2.3.2.2. Método CentímetrosToPuntos

Convierte una medida de centímetros a puntos (un punto equivale a 0,035 cms). Este método es útil para permitir a los usuarios especificar márgenes de página en centímetros y luego realizar la conversión antes de la impresión.Sintaxis:

CentímetrosToPuntos(Centímetros)

Centímetros Double requerido. Especifica el valor en centímetros que se convertirá a puntos.

5.2.3.2.3. Método ObtenerNombreArchivoParaAbrir

Presenta el cuadro de diálogo estándar Abrir y obtiene un nombre de archivo que selecciona el usuario sin que en realidad se abra ningún archivo. Sintaxis:

ObtenerNombreArchivoParaAbrir

Este método devuelve el nombre del archivo seleccionado o el nombre introducido por el usuario. El nombre que se devuelve puede incluir una especificación de ruta de acceso. Devuelve False si el usuario cancela el cuadro de diálogo. Este método puede cambiar la unidad de disco o carpeta activas.

5.2.3.2.4. Método ObtenerNombreArchivoGuardarComo

Presenta el cuadro de diálogo estándar *Guardar como* y obtiene el nombre de archivo seleccionado por el usuario sin guardar realmente ningún archivo. Sintaxis:

ObtenerNombreArchivoGuardarComo(NombreInicial, FiltroArchivo, IndiceFiltro)

NombreInicial: Variant opcional. Especifica el nombre de archivo sugerido. Si este argumento se omite, HOELEST usará el nombre de la hoja activa.

FiltroArchivo: Variant opcional. Una cadena que especifica los criterios de filtrado de archivos.

En Microsoft Windows, esta cadena consta de pares de cadenas de filtros de archivos seguidas de la especificación de filtro de archivo comodín de MS-DOS, separados cada par y cada parte por comas. Se enumera cada par individual en el cuadro de lista desplegable Tipo de archivo. Por ejemplo, la siguiente cadena especifica dos filtros de archivos, texto y macros automáticas: "Archivos de texto (*.txt), *.txt, Archivos de macro automática (*.xla), *.xla". Si se omite, este parámetro tomará el valor predeterminado de "Todos los archivos (*.*),*.".

IndiceFiltro: Variant opcional. Especifica el número de índice de los criterios predeterminados de filtrado de archivos, que puede ser desde 1 hasta el número de filtros especificado en FiltroArchivo. Si este argumento se omite o es mayor que el número de filtros dados, se usará el 1er filtro.

Este método devuelve el nombre del archivo seleccionado o el nombre introducido por el usuario. El nombre que se devuelve puede incluir una especificación de ruta de acceso. Devuelve False si el usuario cancela el cuadro de diálogo. Este método puede cambiar la unidad de disco o carpeta activas.

5.2.3.2.5. Método Ayuda

Muestra un tema de Ayuda. Sintaxis:

Ayuda(NombreArchivo, IDContenido)

NombreArchivo Variant opcional. El nombre del archivo de Ayuda en pantalla que desee mostrar. Si este argumento no se especifica, se usará el archivo de la Ayuda de HOELEST.

IDContenido Variant opcional. Especifica el número de identificación de contexto para el tema de Ayuda. Si este argumento no se especifica, se mostrará el cuadro de diálogo de Temas de Ayuda.

5.2.3.2.6. Método Salir

Sale de HOELEST. Cuando se utiliza este método, si hay algún libro abierto, HOELEST mostrará un cuadro de diálogo para preguntar si se desea guardar los cambios.

5.2.4. Objeto de conjunto Hojas

Un conjunto de todos los objetos **Hoja** abiertos actualmente en la aplicación HOELEST.

Por medio de este objeto se pueden crear nuevas hojas electrónicas. También es posible abrir un archivo, creando una hoja nueva para el archivo abierto.

5.2.4.1. Propiedades del objeto de conjunto Hojas

5.2.4.1.1. Propiedad Contar

Devuelve el número de objetos del conjunto. Long de sólo lectura.

5.2.4.1.2. Propiedad Item

Devuelve un único objeto Hoja de un conjunto Hojas. Sintaxis:

Item(Index)

Index Variant requerida. El nombre o número de índice de la hoja.

5.2.4.1.3. Propiedad Padre

Devuelve el objeto maestro (jerárquicamente superior, en este caso un objeto Origen) del objeto especificado. Es de sólo lectura.

5.2.4.2. Métodos del objeto de conjunto Hojas

5.2.4.2.1. Método Añadir

Crea una hoja nueva. La nueva hoja se convertirá en la hoja activa.

5.2.4.2.2. Método Cerrar

Cierra todos los objetos Hoja abiertos en la aplicación. Si alguno tiene cambios y no ha sido guardado se presentará un mensaje que le

preguntará si desea guardar los cambios. Esto puede evitarse, por ejemplo, estableciendo la propiedad HaSidoGrabada de cada hoja a Verdadero.

5.2.4.2.3. Método Abrir

Abre una hoja electrónica. Sintaxis:

Abrir(NombreArchivo, FormatoArchivo)

NombreArchivo String requerida. El nombre de archivo del libro que se va a abrir.

FormatoArchivo Variant opcional. Al abrir un archivo, este argumento especifica el tipo de formato del mismo. HOELEST sólo pueden abrir dos tipos de archivos: Archivos de texto delimitado por tabulaciones y Archivos de HOELEST (extensión .est). Si se omite este argumento, se utilizará el formato de archivos de HOELEST (*.est).

5.2.5. Objeto Hoja

Representa una hoja de cálculo. El objeto Hoja es un elemento del conjunto Hojas. El conjunto Hojas contiene todos los objetos Hoja de la aplicación.

5.2.5.1. Propiedades del objeto Hoja

5.2.5.1.1. Propiedad Celdas

Devuelve un objeto **Rango** que representa las celdas de la hoja de cálculo (no sólo las celdas que se usan actualmente). Es de sólo lectura.

5.2.5.1.2. Propiedad Columnas

Devuelve un objeto **Rango** que representa todas las columnas de la hoja de cálculo especificada. Es de sólo lectura.

5.2.5.1.3. Propiedad HabilitaCálculo

Verdadero para recalcular automáticamente la hoja de cálculo cuando sea necesario. False para que ninguna hoja se recalcule de manera automática. Boolean de Lectura/Escritura. Al cambiar la propiedad de False a Verdadero, la hoja se recalcula.

5.2.5.1.4. Propiedad Nombre

Devuelve el nombre del objeto. String de sólo lectura.

5.2.5.1.5. Propiedad ConfiguraciónPágina

Devuelve un objeto ConfiguraciónPágina que contiene todos los valores de configuración de página del objeto seleccionado. Es de sólo lectura.

5.2.5.1.6. Propiedad Padre

Devuelve el objeto maestro (jerárquicamente superior, en este caso el objeto de conjunto Hojas) del objeto especificado. Es de sólo lectura.

5.2.5.1.7. Propiedad Rango

Devuelve un objeto **Rango** que representa una celda o un **Rango** de celdas.

Sintaxis 1: **Rango**(Celda1)

Sintaxis 2: **Rango**(Celda1, Celda2)

Celda1 Sintaxis 1. El nombre del **Rango**. Debe ser una referencia con estilo A1.

Sintaxis 2: Variant requerido. Puede incluir un operador de **Rango** (dos puntos).

Variant opcionales *Celda1*, *Celda2* La celda de la esquina superior izquierda e inferior derecha del **Rango**. Puede ser un objeto **Rango** que contenga una sola celda, una columna o fila entera o una cadena que hace referencia a una sola celda.

5.2.5.1.8. Propiedad Filas

Devuelve un objeto **Rango** que representa todas las filas de la hoja de cálculo especificada. Es de sólo lectura.

5.2.5.1.9. Propiedad AltoEstándar

Devuelve el alto estándar (predeterminado) de todas las filas de la hoja, en puntos. Double de sólo lectura.

5.2.5.1.10. Propiedad AnchoEstándar

Devuelve o establece el ancho estándar (predeterminado) de todas las columnas de la hoja. Double de Lectura/Escritura.

5.2.5.1.11. Propiedad RangoUsado

Devuelve un objeto **Rango** que representa el **Rango** utilizado en la hoja de cálculo especificada. Es de sólo lectura.

5.2.5.1.12. Propiedad Visible

Verdadero si el objeto está visible. Boolean o Long de Lectura/Escritura.

5.2.5.1.13. Propiedad FormatoArchivo

Devuelve el formato de archivo. Long de sólo lectura. Puede ser una de las siguientes constantes mHLTFormatoArchivo: mHLTTexto o

mHLTHoelest.

5.2.5.1.14. Propiedad NombreLargo

Devuelve el nombre del objeto, incluida su ruta de acceso en el disco, en forma de cadena. String de sólo lectura. Esta propiedad es equivalente a **DirectorioTrabajo**, seguida del separador de sistema (\) del archivo actual y de la propiedad **Nombre**.

5.2.5.1.15. Propiedad DirectorioTrabajo

Devuelve la ruta completa del objeto, a excepción del separador final y el nombre del objeto. String de sólo lectura. Si una hoja no se ha guardado nunca, su propiedad **DirectorioTrabajo** devolverá una cadena vacía ("").

5.2.5.1.16. Propiedad HaSidoGrabada

Verdadero si no se han efectuado cambios en la hoja especificada desde la última vez que se guardó. Boolean de Lectura/Escritura.

5.2.5.2. Métodos del objeto Hoja

5.2.5.2.1. Método Activar

Convierte esta hoja en la hoja activa. Equivale a hacer click en cualquier lugar de una hoja inactiva, o escoger una hoja inactiva de la lista de documentos abiertos del menú Ventana. Sintaxis:

Activar

5.2.5.2.2. Método Cerrar

Cierra el objeto. Sintaxis:

Cerrar(GrabarCambios, NombreArchivo)

GrabarCambios Variant opcional. Si la hoja no contiene cambios, se ignorará este argumento. Si la hoja contiene cambios, este argumento especifica si los cambios deben guardarse, como se indica a continuación:

Si es Verdadero guarda los cambios de la hoja. Si aún no tiene asociado un nombre de archivo, se usará NombreArchivo. Si NombreArchivo se omite, se pedirá al usuario que especifique un nombre de archivo.

Si es Falso no guarda los cambios del archivo. Por último, si se omite, muestra un cuadro de diálogo en el que se pregunta al usuario si desea guardar los cambios.

NombreArchivo Variant opcional. Guarda los cambios con el nombre de archivo especificado.

5.2.5.2.3. Método Imprimir

Imprime el objeto. Sintaxis:

Imprimir(DesdePágina, HastaPágina, NúmeroCopias)

DesdePágina Variant opcional. El número de la página en que se empezará a imprimir. Si este argumento se omite, la impresión comenzará por el principio.

HastaPágina Variant opcional. El número de la última página a imprimir. Si este argumento se omite, la impresión finalizará en la última página.

NúmeroCopias Variant opcional. El número de copias que se imprimirán. Si este argumento se omite, se imprimirá una sola copia.

5.2.5.2.4. Método Grabar

Guarda los cambios de la hoja especificada. La primera vez que se guarda un archivo, se utiliza el método GrabarComo para especificar su nombre de archivo. Sintaxis:

Grabar

5.2.5.2.5. Método GrabarComo

Guarda los cambios de la hoja en un archivo diferente. Sintaxis:

GrabarComo(NombreArchivo, FormatoArchivo)

NombreArchivo Variant opcional. Una cadena que indica el nombre del archivo que se va a guardar. Puede especificar la ruta completa del archivo. Si sólo especifica el nombre, el archivo se guardará en la carpeta actual.

FormatoArchivo Variant opcional. El formato con el que se guardará el archivo. Revise la propiedad FormatoArchivo para obtener una lista de los tipos válidos.

5.2.5.2.6. Método Calcular

Calcula una hoja de cálculo específica.

5.2.5.2.7. Método Pegar

Pega el contenido del Portapapeles en la hoja. Sintaxis:

Pegar

5.2.6. Objeto Rango

Representa una celda, una fila, una columna, o una selección de celdas que contienen uno o más bloques contiguos de celdas.

5.2.6.1. Propiedades del objeto Rango

5.2.6.1.1. Propiedad Dirección

Devuelve la referencia del **Rango** en el estilo A1. String de sólo lectura.

5.2.6.1.2. Propiedad Celdas

Devuelve un objeto **Rango** que representa las celdas del **Rango** especificado (es decir, no hace nada). Es de sólo lectura.

5.2.6.1.3. Propiedad Columnas

Devuelve un objeto **Rango** que representa las columnas del **Rango** especificado. Es de sólo lectura.

5.2.6.1.4. Propiedad AnchoColumna

Devuelve o establece el ancho de las columnas del **Rango** especificado. Variant de Lectura/Escritura. Si todas las columnas en el **Rango** tienen el mismo ancho, lo devolverá la propiedad *AnchoColumna*. Si las columnas del **Rango** tienen anchos diferentes, la propiedad devuelve Null.

5.2.6.1.5. Propiedad Contar

Devuelve el número de celdas que contiene el rango. Long de sólo lectura.

5.2.6.1.6. Propiedad ColumnaEntera

Devuelve un objeto **Rango** que representa toda la columna (o columnas) que contiene el **Rango** especificado. Es de sólo lectura.

5.2.6.1.7. Propiedad FilaEntera

Devuelve un objeto **Rango** que representa toda la fila (o filas) que contiene el **Rango** especificado. Es de sólo lectura.

5.2.6.1.8. Propiedad Fuente

Devuelve un objeto Fuente que representa la fuente del objeto especificado. Es de sólo lectura.

5.2.6.1.9. Propiedad Alto

Devuelve el alto del **Rango**, en puntos. Long de sólo lectura.

5.2.6.1.10. Propiedad Alineamiento

Devuelve o establece la alineación horizontal del objeto. Puede ser una de las siguientes constantes mHLTAlineamiento: mHLTAlinIzquierda,

mHLTAlinCentro, mHLTAlinDerecha. Long de Lectura/Escritura.

5.2.6.1.11. Propiedad Izquierda

Devuelve la distancia desde el borde izquierdo de la columna, en puntos, al borde izquierdo del **Rango**.

Si el **Rango** tiene más de una columna de ancho, se usará la columna del extremo izquierdo del **Rango**. Long de sólo lectura.

5.2.6.1.12. Propiedad Padre

Devuelve el objeto maestro (jerárquicamente superior, en este caso el objeto Hoja que contiene al rango) del objeto especificado. Es de sólo lectura.

5.2.6.1.13. Propiedad Item

Devuelve un objeto **Rango** que representa una celda o un **Rango** de celdas.

Sintaxis 1: Item(Celda1)

Sintaxis 2 Item(Celda1, Celda2)

Celda1 Sintaxis 1. El nombre del **Rango**. Debe ser una referencia con estilo A1. Sintaxis 2: Variant requerido. Puede incluir un operador de **Rango** (dos puntos).

Variant opcionales *Celda1, Celda2* La celda de la esquina superior izquierda e inferior derecha del **Rango**. Puede ser un objeto **Rango** que contenga una sola celda, una columna o fila entera o una cadena que hace referencia a una sola celda.

5.2.6.1.14. Propiedad Modiftamaño

Redimensiona el **Rango** especificado. Devuelve un objeto **Rango** que representa el **Rango** redimensionado.Sintaxis:

Modiftamaño(TamFilas, TamColumnas)

TamFilas Variant opcional. El número de filas en el nuevo **Rango**. Si este argumento se omite, el número de filas del **Rango** no cambiará.

TamColumnas Variant opcional. El número de columnas del nuevo **Rango**. Si este argumento se omite, el número de columnas del **Rango** no cambiará.

5.2.6.1.15. Propiedad AltoFila

Devuelve el alto de todas las filas del **Rango** especificado, medido en puntos. Devuelve Null si no todas las filas del **Rango** tienen el mismo alto. Variant de Lectura/Escritura.

5.2.6.1.16. Propiedad Filas

Devuelve un objeto **Rango** que representa las filas del **Rango** especificado. Es de sólo lectura.

5.2.6.1.17. Propiedad Tope

Devuelve la distancia desde el borde superior de la fila 1 hasta el borde superior del Rango, en puntos. Si el Rango tiene más de una fila de alto, se usará la fila superior (con el menor número) del Rango. Long de sólo lectura.

5.2.6.1.18. Propiedad Valor

Devuelve o establece el valor de la celda especificada. Si la celda está vacía, **Valor** devuelve el valor Empty. Si el objeto **Rango** contiene más de una celda, devuelve una matriz de valores.

5.2.6.1.19. Propiedad Ancho

Devuelve el ancho del Rango, en puntos. Long de sólo lectura.

5.2.6.2. Métodos del objeto Rango

5.2.6.2.1. Método Limpiar

Quita todo el objeto.

5.2.6.2.2. Método Copiar

Copia el **Rango** en el Portapapeles. Sintaxis:

Copiar

5.2.6.2.3. Método Cortar

Corta el objeto al Portapapeles. El **Rango** cortado debe estar compuesto por celdas adyacentes. Sintaxis:

Cortar

5.2.6.2.4. Método Imprimir

Imprime el objeto. Sintaxis:

Imprimir(DesdePágina, HastaPágina, NúmeroCopias)

DesdePágina Variant opcional. El número de la página en que se empezará a imprimir. Si se omite, la impresión comenzará por el principio.

HastaPágina Variant opcional. El número de la última página a imprimir. Si se omite, la impresión finalizará en la última página.

NúmeroCopias Variant opcional. El número de copias que se imprimirán. Si este argumento se omite, se imprimirá una sola copia.

5.2.6.2.5. Método Ordenar

Ordena un **Rango.** Sintaxis:

Ordenar(Clave, Orden)

Clave Variant opcional. El campo de ordenación, que es un objeto Rango.

Orden Variant opcional. Puede ser una de las siguientes constantes mHLTOrdenarOrden: mHLTOrdenAscendente o mHLTOrdenDescendente. El valor predeterminado es mHLTAscendente.

5.2.6.2.6. Método Seleccionar

Selecciona el objeto. Sintaxis:

Seleccionar

5.2.7. Objeto ConfiguraciónPágina

Representa la descripción de la configuración de página. El objeto ConfiguraciónPágina contiene todos los atributos de configuración de página (margen izquierdo, margen inferior, tamaño de papel, etc.) en forma de propiedades.

5.2.7.1. Propiedades del objeto ConfiguraciónPágina

5.2.7.1.1. Propiedad BlancoyNegro

Verdadero si los elementos del documento se imprimen en blanco y negro. Boolean de Lectura/Escritura.

5.2.7.1.2. Propiedad MargenInferior

Devuelve o establece el tamaño del margen inferior, en puntos. Double de Lectura/Escritura.

5.2.7.1.3. Propiedad CentrarHorizontalmente

Verdadero si la hoja se centra horizontalmente en la página cuando se imprima. Boolean de Lectura/Escritura.

5.2.7.1.4. Propiedad CentrarVerticalmente

Verdadero si la hoja se centrará verticalmente en la página cuando se imprima. Boolean de Lectura/Escritura.

5.2.7.1.5. Propiedad PrimerNúmeroPágina

Devuelve o establece el primer número de página que se usará para la impresión. Si es mHLTAutomático, HOELEST elegirá el primer número de página. El valor predeterminado es mHLTAutomático. Long de Lectura/Escritura.

5.2.7.1.6. Propiedad MargenPieDePágina

Devuelve o establece la distancia en puntos desde el borde inferior de la página hasta el pie de página.

5.2.7.1.7. Propiedad MargenEncabezado

Devuelve o establece la distancia en puntos desde el borde superior de la página hasta el encabezado. Double de Lectura/Escritura.

5.2.7.1.8. Propiedad Orden

Devuelve o establece el orden que HOELEST usará para numerar las páginas al imprimir una hoja de cálculo extensa. Puede ser una de las siguientes constantes mHLTOrden: mHLTAbajoLuegoDerecha o mHLTDerechaLuegoAbajo. Long de Lectura/Escritura.

5.2.7.1.9. Propiedad Orientación

Devuelve o establece el modo de impresión vertical u horizontal: Puede

ser cualquiera de las siguientes constantes mHLTPagOrientación: mHLTPagOrientaciónVertical o mHLTPagOrientaciónHorizontal. Long de Lectura/Escritura.

5.2.7.1.10. Propiedad TamañoPapel

Devuelve o establece el tamaño del papel. Long de Lectura/Escritura. Puede ser una de las siguientes constantes mHLTTamañoPapel: mHLTPapelCarta (8 1/2 x 11 pda), mHLTPapelOficio (8 1/2 x 14 pda) o mHLTPapelUsuario. Definido por el usuario. (es posible que algunas impresoras no acepten algunos de los tamaños de papel):

5.2.7.1.11. Propiedad Padre

Devuelve el objeto maestro (jerárquicamente superior, en este caso un objeto Hoja) del objeto especificado. Es de sólo lectura.

5.2.7.1.12. Propiedad ImprimirLineasDivisión

Verdadero si las líneas de división de las celdas se imprimen en la página. Boolean de Lectura/Escritura.

5.2.7.1.13. Propiedad ImprimirTituloColumnas

Verdadero si se imprimen encabezados de columna en esta página. Boolean de Lectura/Escritura. Si la hoja electrónica contiene variables, el

valor de esta propiedad siempre será verdadera.

5.2.7.1.14. Propiedad MargenTope

Devuelve o establece el tamaño del margen superior, en puntos. Double de Lectura/Escritura.

5.2.7.1.15. Propiedad Zoom

Devuelve o establece un porcentaje (entre 10 y 400%) según el cual HOELEST ajustará la escala de la hoja de cálculo para la impresión. Variant de Lectura/Escritura. Los ajustes de escala conservarán la relación de aspecto del documento original.

5.2.8. Objeto Fuente

Contiene los atributos de fuente (nombre, tamaño, color, etc.) de un objeto

5.2.8.1. Propiedades del objeto Fuente

5.2.8.1.1. Propiedad Negrita

Verdadero si la fuente es negrita. Variant de Lectura/Escritura.

5.2.8.1.2. Propiedad Color

Devuelve o establece el color primario del objeto Fuente. Long de Lectura/Escritura.

5.2.8.1.3. Propiedad Itálica

Verdadero si el estilo de la fuente es cursiva. Boolean de Lectura/Escritura.

5.2.8.1.4. Propiedad Nombre

Devuelve o establece el nombre de la fuente. String de Lectura/Escritura

5.2.8.1.5. Propiedad Padre

Devuelve el objeto maestro (jerárquicamente superior, en este caso un objeto Rango) del objeto especificado. Es de sólo lectura.

5.2.8.1.6. Propiedad Tamaño

Devuelve o establece el tamaño de la fuente. Variant de Lectura/Escritura.

5.2.8.1.7. Propiedad Tachado

Verdadero si la fuente está tachada con una línea horizontal. Boolean de Lectura/Escritura.

5.2.8.1.8. Propiedad Subrayado

Verdadero si la fuente está subrayada. Boolean de Lectura/Escritura.

5.2.9. Objeto FuncionesHoja

Utilizado como contenedor de las funciones de hoja de cálculo de HOELEST que pueden llamarse desde Visual Basic.

5.2.9.1. Propiedades del objeto FuncionesHoja

5.2.9.1.1. Propiedad Padre

Devuelve el objeto maestro (jerárquicamente superior, en este caso un objeto origen) del objeto especificado. Es de sólo lectura.

5.2.9.2. Métodos del objeto FuncionesHoja

5.2.9.2.1. Método Suma

Suma los elementos de una matriz. Sintaxis:

Suma(matriz)

Matriz Variant requerida. Matriz con los elementos a sumar.

5.2.9.2.2. Método SumaCuadrados

Suma los elementos al cuadrado de una matriz. Sintaxis:

Suma(matriz), Donde *Matriz* es aquella cuyos elementos se sumarán al cuadrado. Variant requerida

5.2.10. Objeto de conjunto *Formulas*

Un conjunto de todos los objetos *Formula* contenidos en una hoja electrónica. Por medio de este objeto se pueden añadir, eliminar o editar fórmulas introducidas por el usuario en tiempo de ejecución. Este objeto es privado, es decir, Hoelest se encarga de manipularlo y no se permite el acceso desde aplicaciones externas a la aplicación.

5.2.10.1. Propiedades del objeto de conjunto Formulas

5.2.10.1.1. Propiedad Contar

Devuelve el número de objetos del conjunto. Long de sólo lectura.

5.2.10.1.2. Propiedad Item

Devuelve un único objeto *Formula* de un conjunto *Formulas*. Sintaxis:

Item(Index)

Index Variant requerida. El número de la fórmula en la colección.

5.2.10.1.3. Propiedad Padre

Devuelve el objeto maestro (jerárquicamente superior, en este caso un objeto Hoja) del objeto especificado. Es de sólo lectura.

5.2.10.2. Métodos del objeto de conjunto *Formulas*

5.2.10.2.1. Método Añadir

Añade una fórmual a la colección. Devuelve un objeto ***Formula***.

5.2.10.2.2. Método Eliminar

Elimina un objeto *Formula* de la colección.

5.2.11. Objeto *Formula*

Representa una fórmula contenida dentro de una hoja de cálculo. El objeto *Formula* es un elemento del conjunto Formulas. En tiempo de ejecución, el usuario puede escribir una fórmula en cualquier celda, Hoelest evaluará su sintaxis y devolverá el resultado de calcular la misma. Este objeto es privado, es decir, Hoelest se encarga de manipularlo y no se permite el acceso desde aplicaciones externas a la aplicación.

5.2.11.1. Propiedades del objeto *Formula*

5.2.11.1.1. Propiedad CadenaFormula

Representa la cadena de la fórmula introducida por el usuario. String de Lectura\Escritura.

5.2.11.1.2. Columna

Devuelve o establece el índice que identifica la posición de columna de la

fórmula. Long de Lectura|Escritura.

5.2.11.1.3. Fila

Devuelve o establece el índice que identifica la posición de fila de la fórmula. Long de Lectura|Escritura.

5.2.11.1.4. Valor

Contiene el resultado del cálculo realizado sobre la propiedad CadenaFormula. Variant de Lectura\Escritura.

5.2.12. Objeto de conjunto Variables

Un conjunto de todos los objetos *Variable* contenidos en una hoja electrónica. Por medio de este objeto se pueden añadir, eliminar o editar variables de manera interna exclusivamente. Existe un objeto de conjunto *Variables* por cada hoja electrónica de la aplicación.

5.2.12.1. Propiedades del objeto de conjunto Variables

5.2.12.1.1. Propiedad Contar

Devuelve el número de objetos del conjunto. Long de sólo lectura.

5.2.12.1.2. Propiedad Item

Devuelve un único objeto *Variable* de un conjunto Variables. Sintaxis:

Item(Index)

Index Variant requerida. El número o el nombre de la variable en la colección.

5.2.12.1.3. Propiedad Padre

Devuelve el objeto maestro (jerárquicamente superior, en este caso un objeto *Hoja*) del objeto especificado.

Es de sólo lectura.

5.2.12.2. Métodos del objeto de conjunto Variables

5.2.12.2.1. Método Añadir

Añade una *Variable* a la colección. Devuelve un objeto *Variables*.

5.2.12.2.2. Método Eliminar

Elimina un objeto *Variable* de la colección.

5.2.13. Objeto *Variable*

Representa una variable contenida dentro de una hoja de cálculo.

El objeto *Variable* es un elemento del conjunto Variables.

5.2.13.1. Propiedades de objeto *Variable*

5.2.13.1.1. Propiedad Nombre

Nombre de la variable. String de Lectura\Escritura.

5.2.13.1.2. Propiedad Tipo

Devuelve o establece el tipo de variable. Puede tomar los valores TipoVariable: TPNumérica o TPTexto. Long de Lectura|Escritura.

5.2.13.1.3. Propiedad Padre

Devuelve el objeto maestro (jerárquicamente superior, en este caso un objeto de conjunto *Variables*) del objeto especificado. Es de sólo lectura.

5.2.13.1.4. Propiedad Columna

Devuelve o establece el índice de columna en la cual se encuentra la variable. Integer de Lectura\Escritura.

5.3. Diseño de la interfaz visual

5.3.1. Estilos de interfaz

Hay dos estilos principales de interfaz de usuario: la interfaz de documento único (SDI) y la interfaz de documentos múltiples (MDI). Un ejemplo de la interfaz SDI es la aplicación WordPad incluida en Microsoft

Windows (figura 5.2). En WordPad, sólo se puede abrir un documento y se debe cerrar un documento para abrir otro.

Aplicaciones como Microsoft Excel y Microsoft Word para Windows son interfaces MDI; le permiten presentar varios documentos al mismo tiempo y cada documento se presenta en su propia ventana (figura 5.3). Los documentos o ventanas secundarias están contenidos en una ventana primaria, que proporciona un espacio de trabajo para todas las ventanas secundarias de la aplicación.

Las aplicaciones MDI se pueden reconocer por la inclusión de un menú Ventana con submenús para cambiar entre las distintas ventanas o documentos.

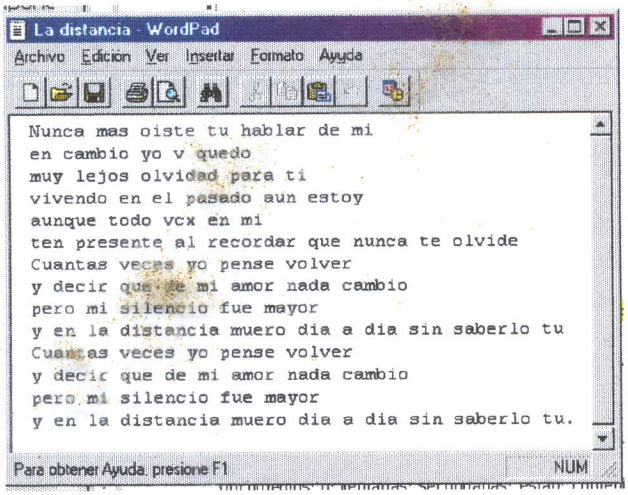


Figura 5. 2. WordPad, una aplicación con interfaz de documento único

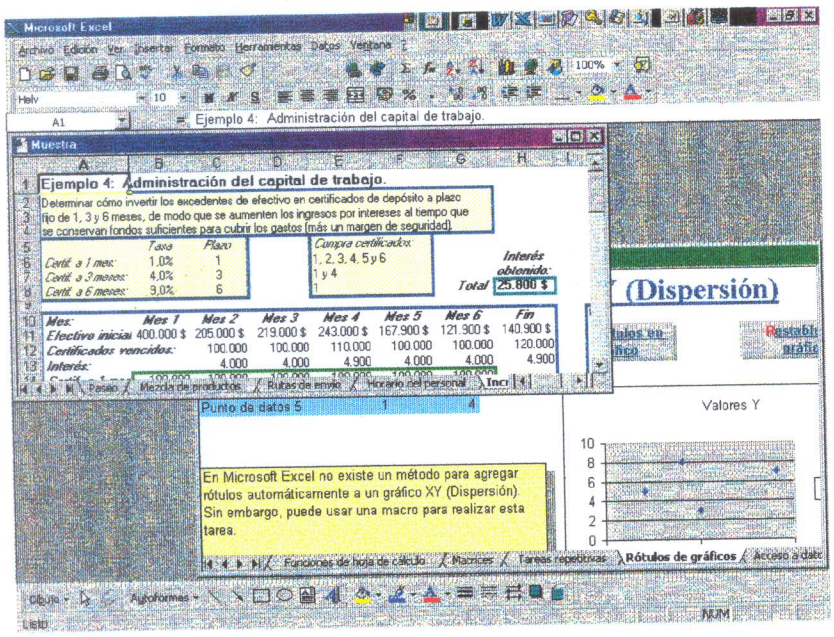


Figura 5. 3. Excel, una aplicación con interfaz de documentos múltiples.

5.3.2. La Ventana Principal.

Para determinar cuál es el mejor estilo de interfaz, nos basamos en el propósito de la aplicación. Decidimos que HOELEST debería tener el estilo de interfaz MDI, ya que es posible que el usuario necesite trabajar con más de una hoja electrónica a la vez, por ejemplo comparar Tablas de Análisis de Varianza de dos experimentos similares. La figura 5.4 corresponde al formulario MDI de HOELEST, en el que podemos apreciar la barra de menús, las barras de herramientas y de estado.

Como se expresó anteriormente, los formularios secundarios MDI se muestran siempre dentro del formulario principal MDI en tiempo de

ejecución. Cada ventana individual está confinada al área de la ventana primaria de HOELEST. Cuando minimiza HOELEST, también minimiza todas las ventanas de documentos; sólo el icono de la ventana primaria aparece en la barra de tareas.

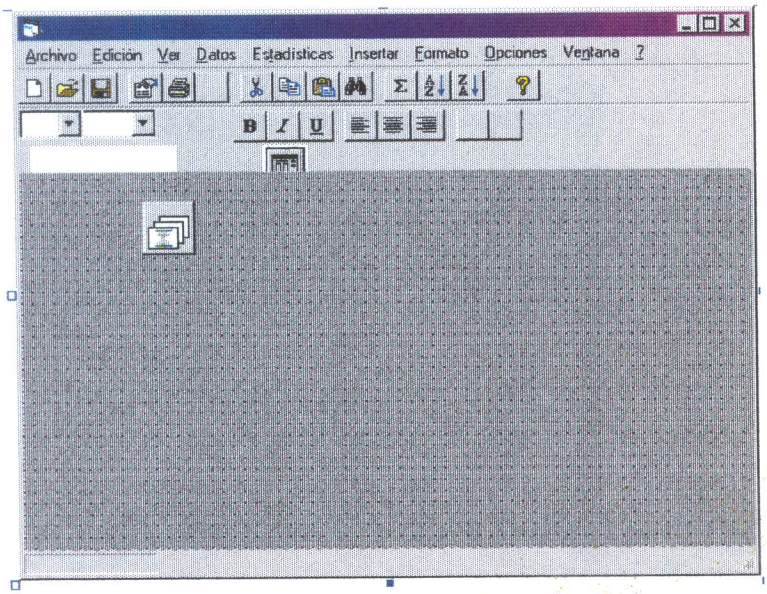


Figura 5. 4 Formulario Principal de Hoelest. Modo Diseño

5.3.2.1. Barra de menús

5.3.2.2. Trabajo con archivos

HOELEST posibilita al usuario el trabajo con archivos. En este menú, las opciones como Abrir utilizan llamadas a las DLL de Windows para presentar los cuadros de dialógo estándares.

La figura 5.5 muestra los submenús del menú Archivo y sus teclas de

acceso rápido. En este menú se encuentra la opción para salir de HOELEST; recordaremos que si salimos de la aplicación contenedora, todos los formularios secundarios, en nuestro caso las hojas electrónicas también se cerrarán, previo a cuadros de diálogo que permitan ser guardadas en disco.

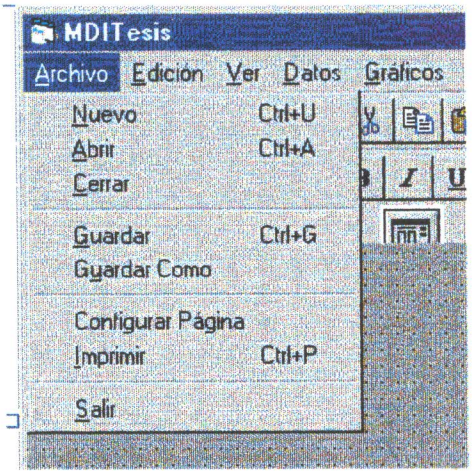


Figura 5. 5. Menú Archivo. Diseño

5.3.2.3. El menú Edición

En él aparecen opciones como Cortar, Copiar y Pegar, las cuales hacen uso del portapapeles de Windows. Se incluyen también las funciones de Buscar y Reemplazar. En tiempo de diseño, las funciones de buscar y reemplazar se implementaron en un mismo formulario. En tiempo de ejecución se activan los controles necesarios según la tarea exigida por el usuario, por ejemplo el usuario puede activar directamente los controles

para Reemplazar escogiendo la opción directamente del menú Edición o, en su defecto primero escoger la opción Buscar y dentro del formulario que se muestra escoger la opción Reemplazar.



Figura 5. 6. Menú Edición. Diseño

5.3.2.4. El menú Ver

Este menú nos permite controlar la presentación de las barras de herramientas. Es posible que un usuario desee ocultar alguna barra con el objeto de disponer de mayor campo de visión de la hoja electrónica.

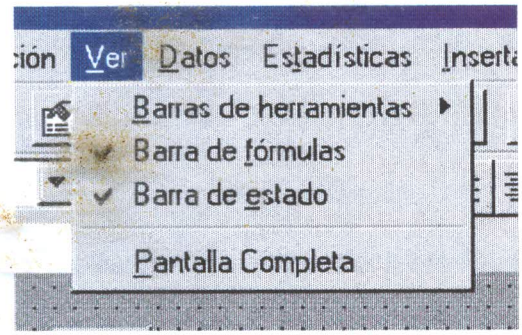


Figura 5. 7. Menú Ver. Diseño

5.3.2.5. El menú Estadísticas

El objeto de la creación de HOELEST se resume en este menú, específicamente la parte de la Estadística Inferencial.

Contiene las opciones para realizar los análisis estadísticos requeridos por el usuario, como pruebas de comparación de medias muestrales por ejemplo.

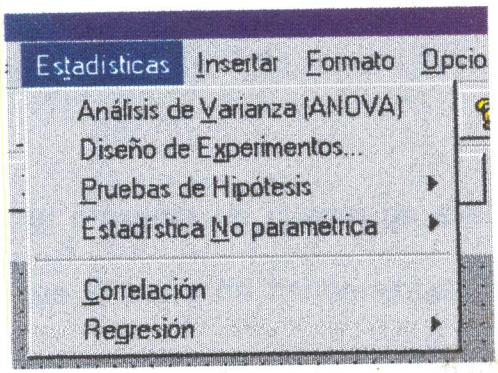


Figura 5. 8. Menú Estadísticas. Diseño

5.3.2.6. El menú Insertar

Contiene características propias de una hoja electrónica como la capacidad de insertar celdas, filas o columnas en el lugar deseado.

El submenú Celdas presenta una segunda pantalla en la que se pide se especifique como se va a mover el contenido de las celdas después de la inserción de las nuevas celdas.



Figura 5. 9. Menú Insertar. Diseño

5.3.2.7. El menú Formato

Dentro de él se encuentran opciones para mejorar la presentación de los datos en la hoja electrónica.

La subopción celdas presenta un cuadro de diálogo que presenta las mismas opciones que la barra de herramientas Formato. Las opciones Fila y Columna, permiten modificar atributos de dimensión y visibilidad de filas y columnas.

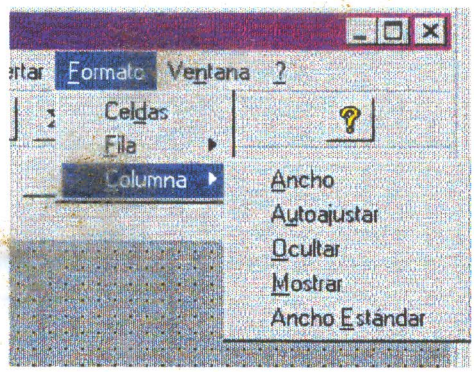


Figura 5. 10. Menú Formato. Diseño

5.3.2.8. EL menú Ventana

Dentro de él se encuentran los submenús que permiten administrar las hojas electrónicas (documentos secundarios) contenidos dentro del formulario MDI principal. Es posible presentarlos en cascada, en mosaicos, ocultarlos o mostrarlos; además nos posibilita transferir el enfoque de una a otra hoja por medio del listado de ventanas contenidas en la aplicación.

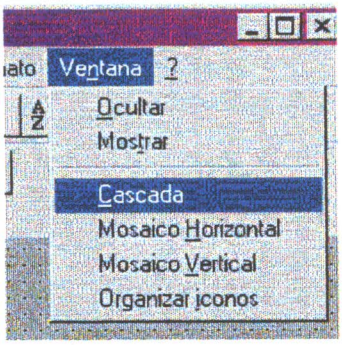


Figura 5. 11. Menú Ventana. Diseño

5.3.2.9. ¿Cómo obtener ayuda?

Es posible obtener ayuda presionando la tecla funcional F1, o ingresando al menú ?, Ayuda de.

Si se necesita algún tipo de información adicional, puede escoger la subopción Acerca de, que contiene los datos de los desarrolladores del paquete.

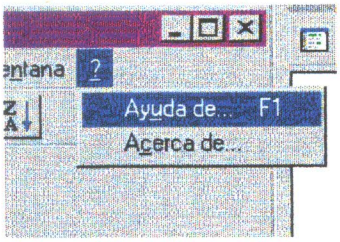


Figura 5. 12. Menú Ayuda. Diseño

5.3.2.10. Barra Estándar

La barra estándar de HOELEST contiene los botones para acciones comunes de la hoja electrónica, como abrir un nuevo archivo o imprimir el archivo actual por ejemplo. Esta barra puede visualizarse si se escoge del menú Ver, la opción Barras de herramientas, Estándar



Figura 5. 13. Barra Estándar. Diseño

5.3.2.11. Barra de Formato

Esta barra proporciona botones de acceso rápido para dar formato a la presentación de los datos en la hoja, tales como establecer el tipo de fuente, el color y alineación por ejemplo. Esta barra puede visualizarse si se escoge del menú Ver, la opción Barras de herramientas, Formato.



Figura 5. 14. Barra de formato. Diseño

5.3.2.12. Barra de Fórmulas

La barra de fórmulas es útil para la creación, edición o eliminación de fórmulas personalizadas. Está siempre relacionada con la celda activa de la hoja activa de la aplicación.

Por ejemplo, una celda puede mostrar el número 15, siendo este el resultado de una fórmula escrita por el usuario. Al posicionarse sobre dicha celda, la barra de fórmulas muestra la cadena de la fórmula correspondiente.



Figura 5. 15. Barra de fórmulas. Diseño

5.3.2.13. Barra de Estado

El propósito de la barra de estado es brindar información visual al usuario mientras se realiza alguna operación. Es de gran utilidad, debido a que el usuario se siente más cómodo cuando se siente retroalimentado por el programa.



Figura 5. 16. Barra de Estado. Diseño

5.3.3. El formulario de la Hoja electrónica

Cada hoja electrónica es un formulario que contiene dos controles: un MSFlexgrid y un TextBox. El control MSFlexGrid muestra datos de tablas y efectúa operaciones en ellos, proporciona una completa flexibilidad para ordenar, fusionar y dar formato a tablas que contienen cadenas e imágenes. Este control constituye la interfaz que da la apariencia de una hoja electrónica. El control TextBox se utiliza para presentar información escrita por el usuario en tiempo de ejecución o asignada a la propiedad Text del control en tiempo de diseño o de ejecución. La figura 5.17 muestra la posición de los controles en tiempo de diseño.

En tiempo de ejecución, nos podemos mover por la hoja electrónica con las teclas direccionales o con el ratón.

Cuando presionamos alguna tecla alfanumérica, es decir, cuando deseamos ingresar información en alguna celda, el control TextBox se posiciona en la celda requerida y posteriormente se adecua al tamaño de la misma, de esta manera realmente tipeamos información dentro del control TextBox. Cuando presionamos <enter>, una tecla direccional o damos un click en otra celda, la información contenida en el TextBox pasa al control MSFlexGrid. De esta manera se simula la interacción directa del usuario con la hoja.

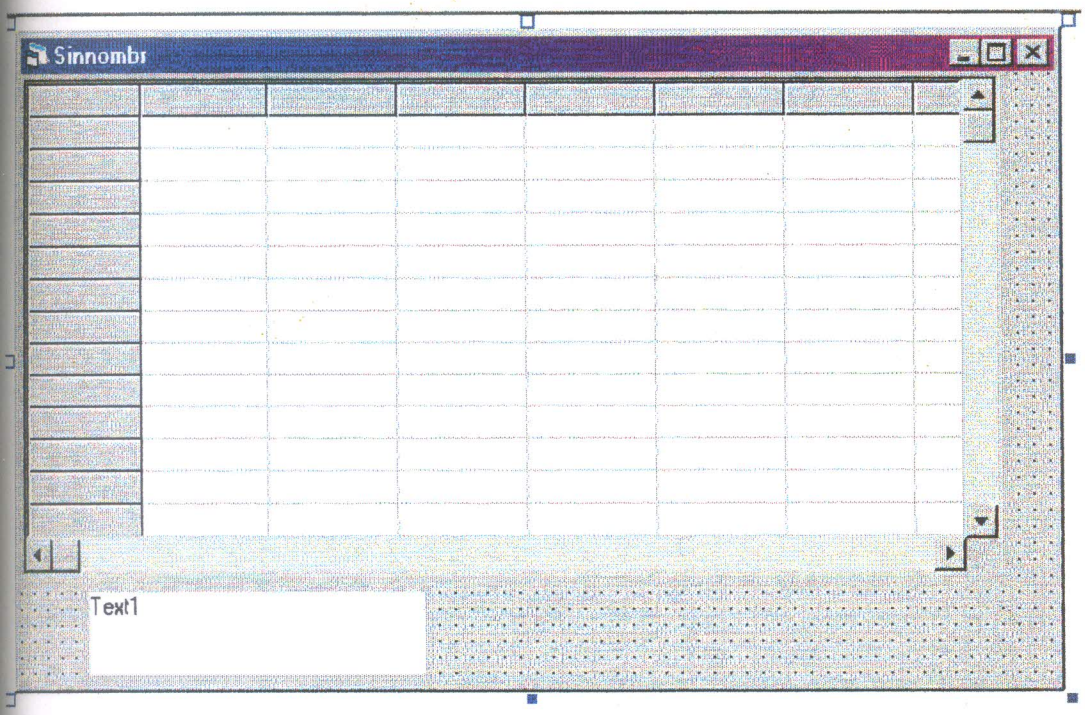


Figura 5. 17. Interfaz visual Hoja Electrónica. Formulario frmhoja

5.3.4. Buscar o sustituir datos

Puede hacerse un búsqueda de un texto o números determinados que se desean revisar o modificar y automáticamente se sustituirán los datos encontrados. Pueden buscarse todas las celdas que contengan el mismo tipo de datos, como fórmulas, así como las celdas cuyos datos no coincidan con la celda activa.

El formulario diseñado para llevar a cabo estas funciones se muestra en la figura 5.18. Como podemos observar contiene controles que se utilizan tanto para las funciones de búsqueda como para las de reemplazo.

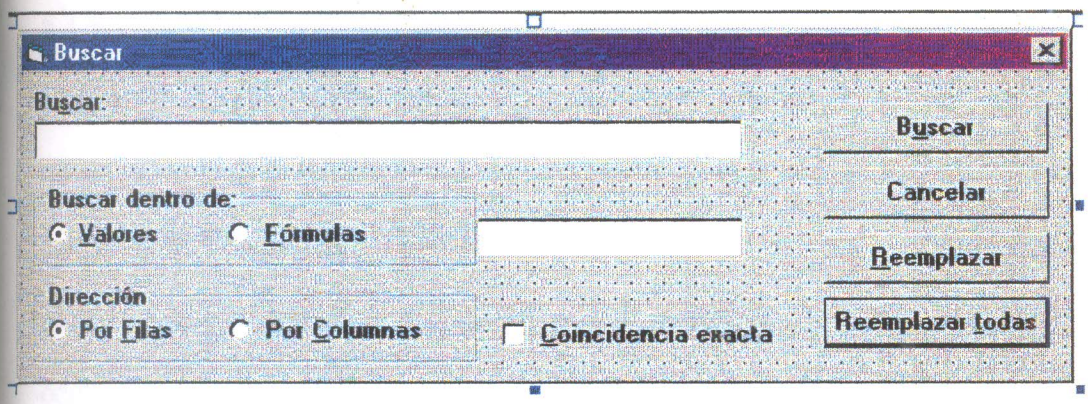


Figura 5. 18. Formulario frmBuscar. Diseño

5.3.5. Insertar celdas, filas o columnas

Pueden insertarse celdas, filas y columnas vacías e introducir datos en las mismas.

Si se mueven y se copian celdas, pueden insertarse entre las celdas actuales para evitar que se peguen encima de los datos. La figura 5.19 muestra el formulario para efectuar estas operaciones.

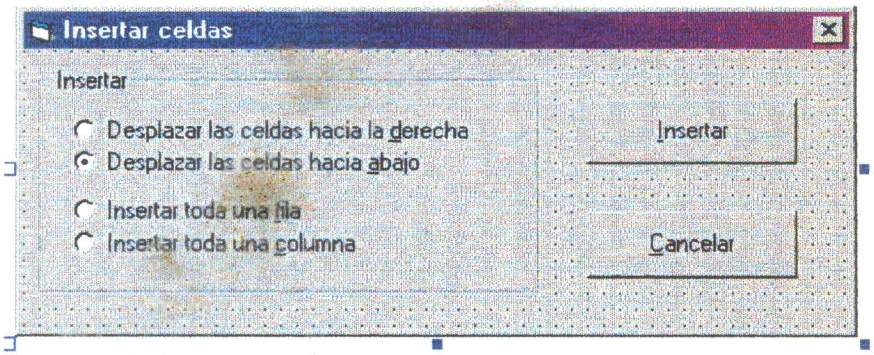


Figura 5. 19. Formulario frmInsertarCeldas. Diseño

5.3.6. Ocultar y Mostrar hojas ocultas

El usuario puede ocultar hojas en cualquier momento de acuerdo a sus necesidades, por ejemplo, para mantener organizada la información contenida en cada hoja y priorizar su utilización al mostrarla. Para ocultar una hoja bastará con ingresar al menú Ventana y escoger la opción ocultar (ver figura 5.11).

Como es posible ocultar varias hojas, se debe mantener un registro de todas ellas; cuando el usuario necesita mostrar alguna hoja se le presentará un listado de las hojas ocultar y se le permitirá escoger la que él requiera. Este listado se encuentra en un control ListBox, como lo muestra la figura 5.20. En tiempo de ejecución se podrá acceder a este listado si es que existen ventanas ocultas por medio de: menú Ventana, Mostrar.

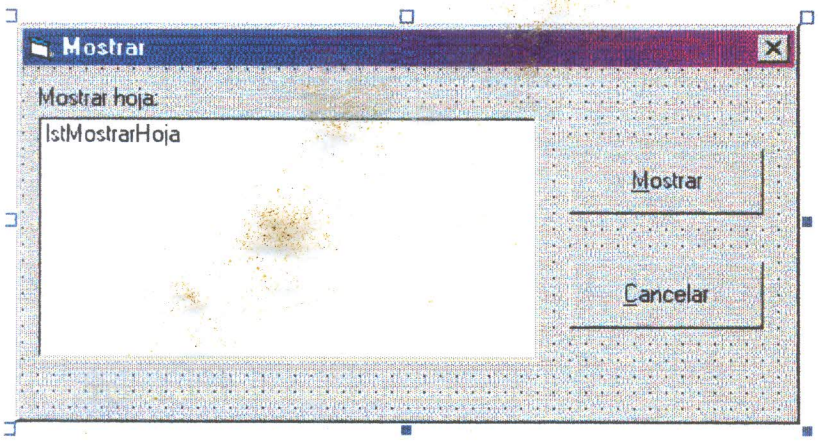


Figura 5. 20. Formulario frmMostrar. Diseño

5.3.7. Insertar variables

Como una característica propia de la Hoja Electrónica Estadística está la definición de variables. Básicamente, HOELEST trabajará con dos tipos de variables: numéricas y cadenas. Al ingresar una nueva variable, su nombre se etiqueta como el encabezado de una columna, lo que indica que los datos contenidos en la columna guardan estrecha relación entre sí, puesto que corresponden a distintas observaciones de la variable de estudio. El formulario diseñado para la definición de variables se muestra en la figura 5.21.

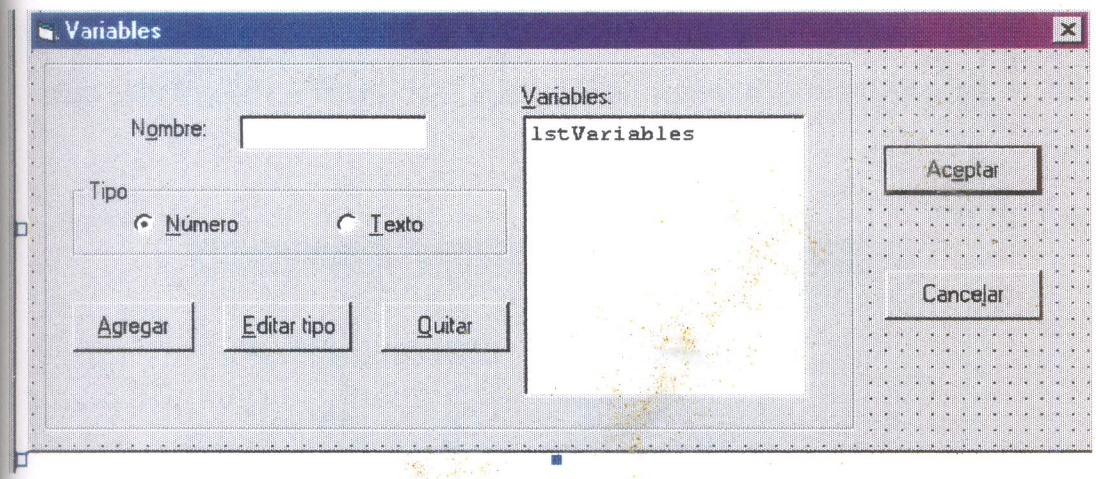


Figura 5. 21. Formulario frmVariables. Diseño

5.3.8. Administración de la Configuración de Hoelest

En Windows 95 y en las versiones posteriores de Windows, la configuración de los programas se almacena en el registro del sistema.

Visual Basic proporciona una ubicación estándar del registro para almacenar la información de las aplicaciones creadas con Visual Basic:

HKEY_CURRENT_USER\Software\VB and VBA Program Settings\
nombre_aplicación\ sección\clave

Visual Basic también proporciona cuatro instrucciones y funciones para manipular la configuración almacenada en la ubicación de la aplicación dentro del registro del sistema. De esta manera se diseñó la configuración que Hoelest debe mantener en el registro, con el fin de guardar las preferencias de sus usuarios. Hoelest proporciona un formulario al usuario para esta tarea. La figura 5.22 ilustra dicho formulario frmOpciones.

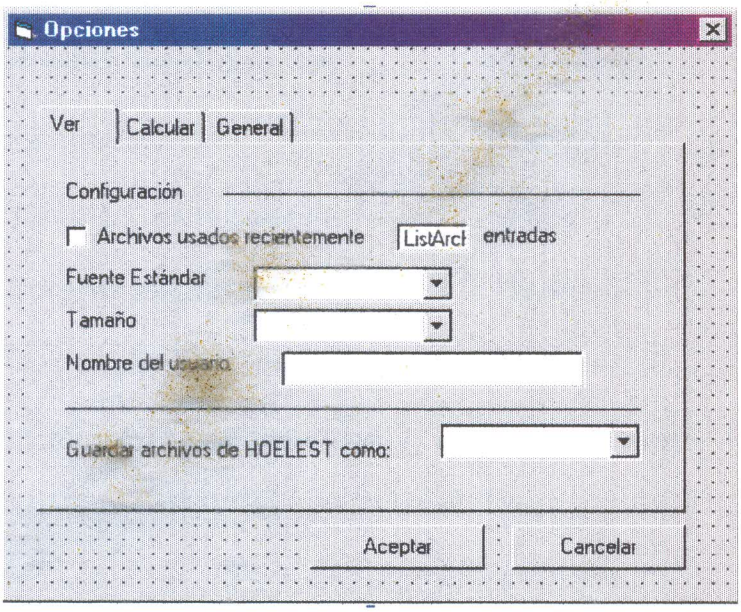


Figura 5. 22. Formulario frmOpciones. Diseño

La tabla V contiene las secciones creadas en el registro del sistema para Hoelest, las claves de cada sección y los valores que pueden aceptar.

Tabla V

Configuración de Hoelest en el Registro del Sistema

Ver	BarraFórmulas	True o False
Ver	BarraEstado	True o False
Ver	BarraDesplazamientoHorizontal	True o False
Ver	BarraDesplazamientoVertical	True o False
Ver	LineasDeDivisión	True o False
Calcular	ModoCálculo	1 (Automático), 0 (Manual
Calcular	CalcularAntesGuardar	True o False
Calcular	NumFijoDecimales	Enteros
General	NumArchivosRecientes	0<=x<=10
General	FuenteEstándar	Cadena
General	TamañoFuenteEstandar	single
General	NombreUsuario	Cadena
General	FormatoArchivoPredeterminado	1 ó 2
General	MostrarArchivosRecientes	True o False
Archivos Recientes	ArchivoReciente(Número)	Cadena

5.3.9. Prueba de Hipótesis Simple

Cuando se desea evaluar la hipótesis sobre la media poblacional, Hoelest ofrece la siguiente ventana, en donde es posible elegir varias variables, todas ellas se probarán con la media hipotética y confianza introducidos. La figura 5.23 ilustra dicho formulario.

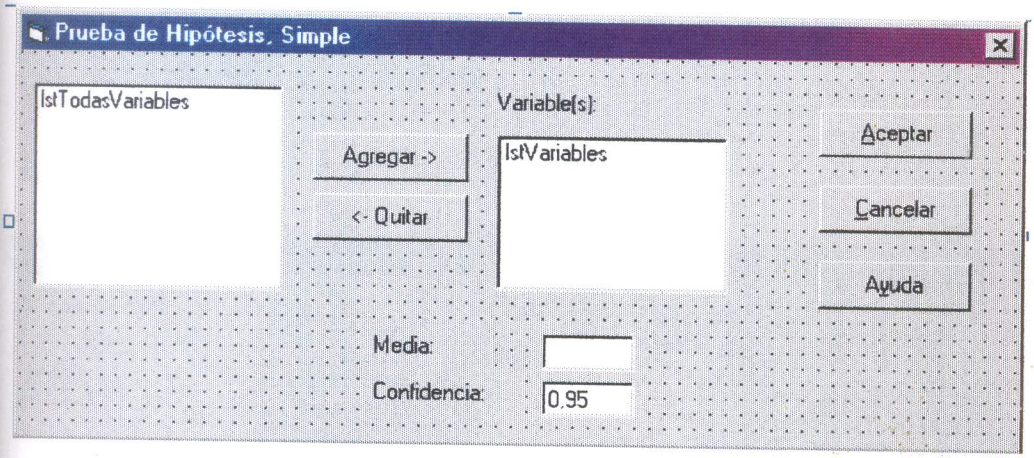


Figura 5. 23. Formulario PHSimple. Diseño

5.3.10. Prueba de Hipótesis Dos Grupos

Si en cambio, un investigador desea comparar las medias de dos poblaciones diferentes, puede accesar a Prueba de Hipótesis, Dos grupos.

Así mismo, tiene la capacidad de elegir varias variables, todas ellas se compararán entre sí con el nivel de confianza especificado. La figura 5.24 ilustra el formulario de PHDosGrupos.

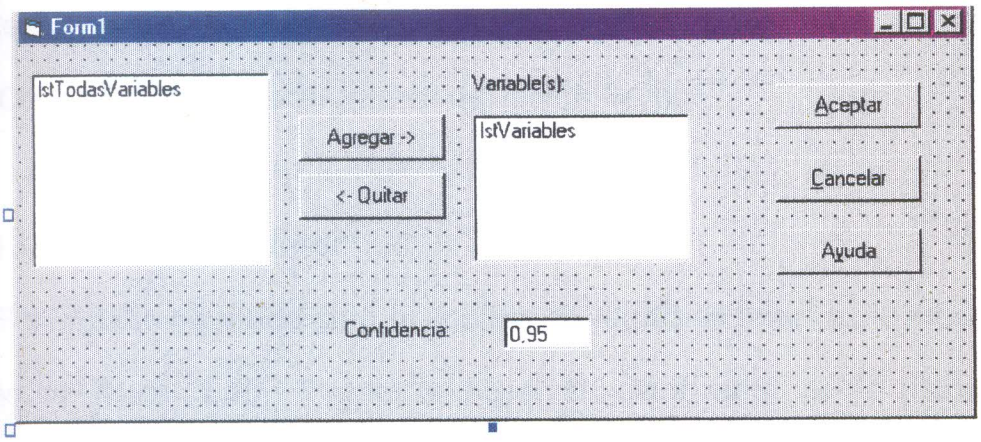


Figura 5. 24. Formulario PHDosGrupos. Diseño

5.3.11. Correlación

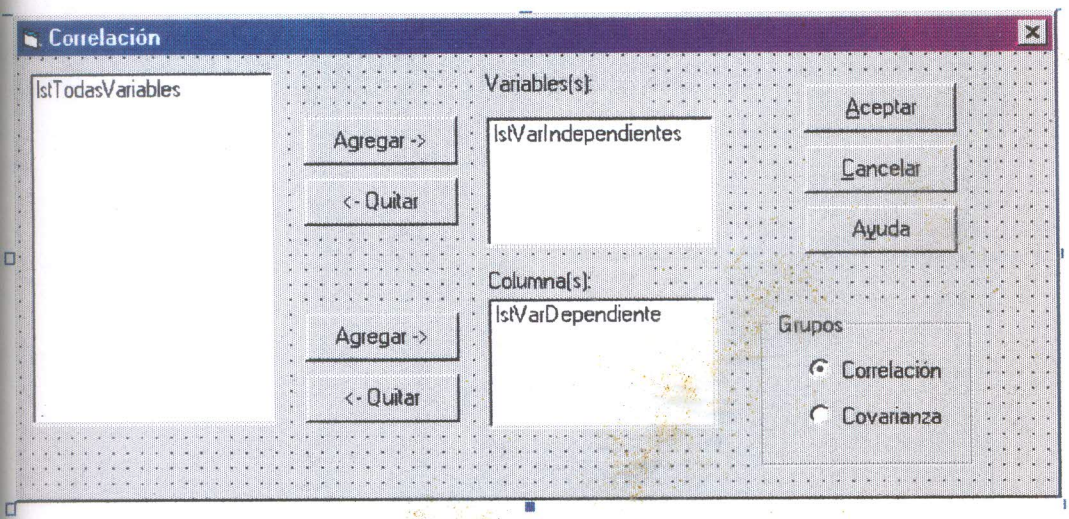


Figura 5. 25. Formulario frmCorrelación. Diseño

Podemos obtener una matriz de covarianzas o de correlaciones de un conjunto de variables. El formulario que nos facilita obtener esta información se muestra en la figura 5.25. Es posible apreciar que podemos obtener una matriz de covarianzas o correlaciones de un grupo

de variables, si es que habilitamos la opción "Covarianzas" o "Correlaciones" respectivamente del marco Grupos.

5.3.12. Regresión Lineal

El formulario diseñado para el análisis de regresión lineal es el mostrado en la figura 5.26. Podemos observar claramente como podemos especificar la variable aleatoria dependiente, en función de una o más variables no aleatorias independientes.

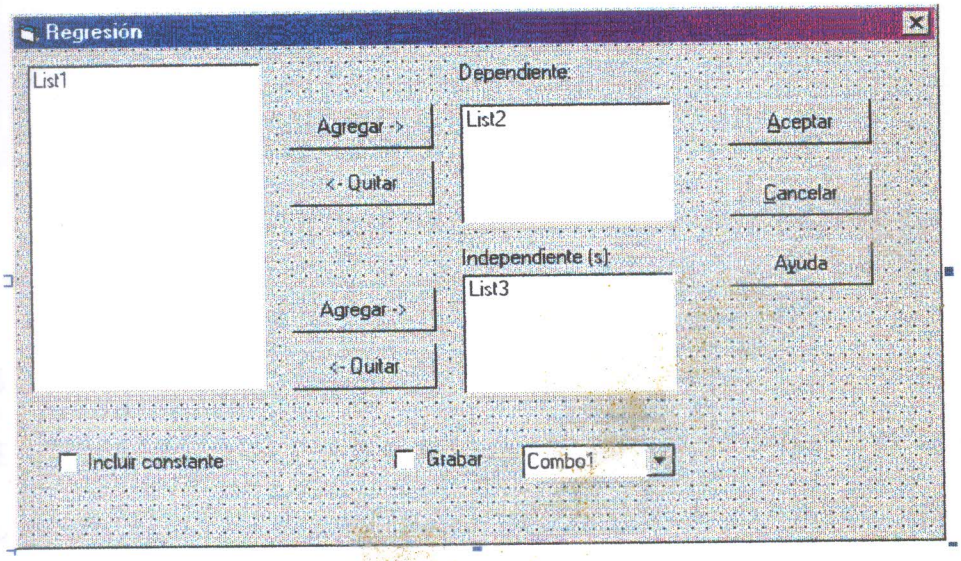


Figura 5. 26. Formulario frmRegresión. Diseño

5.3.13. Diseño de Experimentos

Los métodos de diseño experimental tiene un cometido importante en el desarrollo de procesos y en la depuración de procesos para mejorar el

rendimiento. El formulario que permite construir matrices de diseño para realizar experimentos es el de la figura 5.27. Podemos escoger de entre los tipos de diseño indicados en el cuadro combinado y habilitar opciones de impresión o diseño según el diseño escogido.

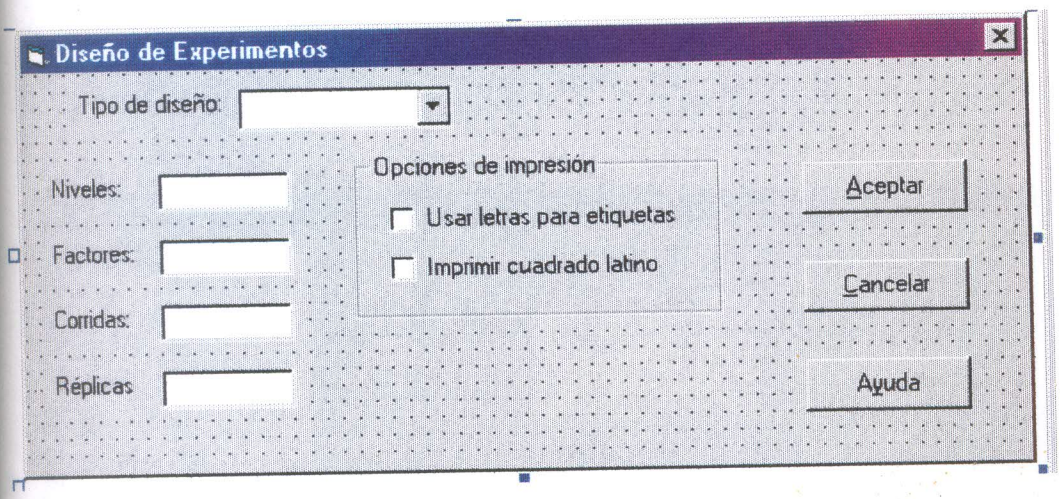


Figura 5. 27. Formulario frmDiseñoDeExperimentos. Diseño

5.3.14. Análisis de Varianza ANOVA

El procedimiento de análisis de varianza trata de analizar la variación de una respuesta y de asignar porciones (componentes) de esta variación a cada una de las variables de un conjunto de variables independientes. El formulario que presta facilidades para la realización de una análisis de varianza es el que ilustra la figura 5.28

Como es posible apreciar se puede elegir el análisis de modelos de

bloques aleatorizados y de cuadrados latinos. Para ello se considera que el primer factor en Factor(s): es el factor de interés principal y, el segundo factor describe a las columnas o bloques. De manera similar, cuando se trata de cuadrados latinos, el primer factor en la lista es el factor principal, el segundo corresponde a las columnas y el tercero a los renglones. En los demás casos no interesa el orden de aparición.

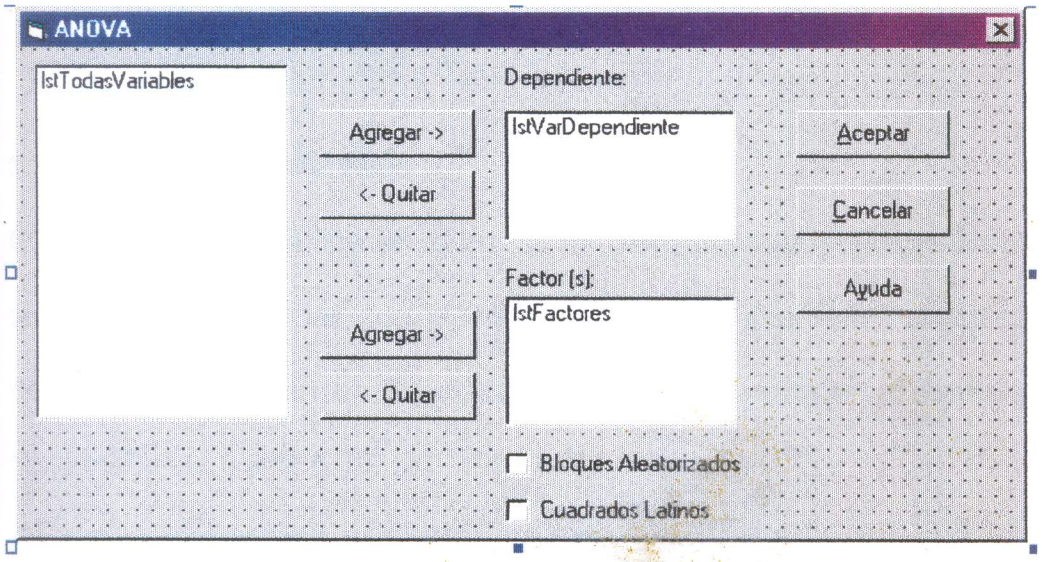


Figura 5. 28. Formulario frmANOVA. Diseño

5.3.15. Kruskal Wallis

La prueba de Kruskal Wallis constituye un procedimiento alternativo para el análisis de varianza, cuando la suposición de normalidad no se justifica. El formulario diseñado para esta prueba es el de la figura 5.29

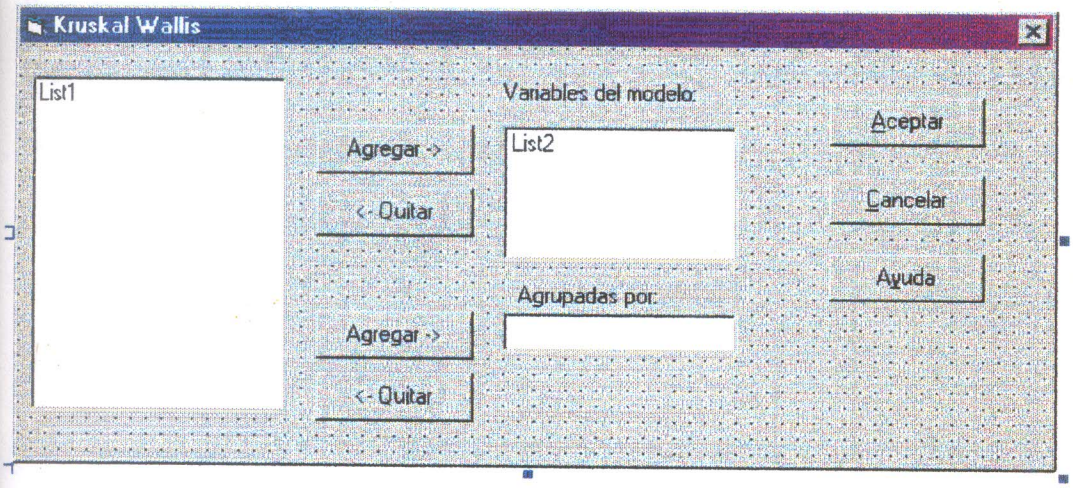


Figura 5. 29. Formulario frmKruskalWallis. Diseño

5.3.16. Prueba de Wilcoxon

Cuando nos interesamos en saber si las distribuciones poblacionales de dos variables aleatorias son idénticas, frente a la alternativa de que las distribuciones difieren en su ubicación, podemos acceder al formulario mostrado en la figura 5.30.

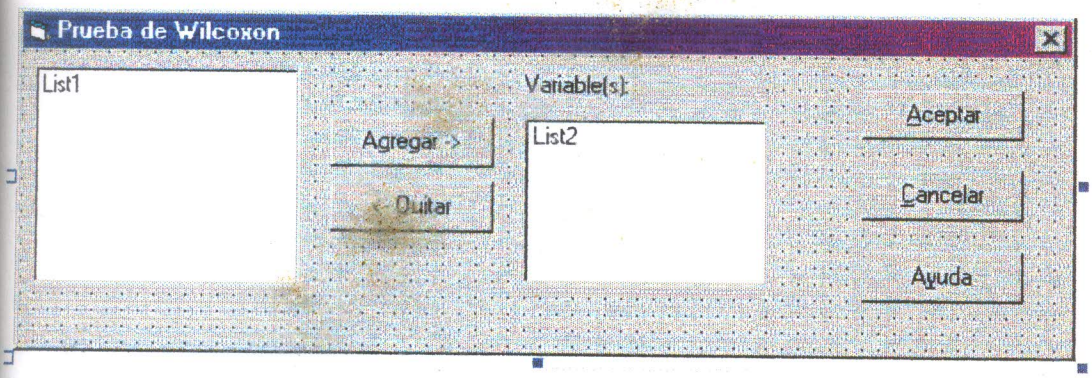


Figura 5. 30. Formulario frmPruebaDeWilcoxon. Diseño

5.4. Módulos de Código

Ciertas tareas de uso global de la aplicación se encuentran organizadas en módulos de código. Hoelest tiene 6 módulos, cada uno con un conjunto de procedimientos relacionados.

5.4.1. Módulo Búsquedas

Este módulo contiene funciones para búsqueda de cadenas dentro de una hoja electrónica, bien sea en las celdas o en las fórmulas que ésta contenga. El procedimiento BuscarCadena se encarga de llamar a los demás procedimientos según las especificaciones dadas por el usuario. Este código se ejecuta cuando el usuario hace click en el botón aceptar del formulario *Buscar*.

Los procedimientos de búsqueda son: BuscaCadenaPorColumnas, BuscaCadenaPorFilas, BuscaCadenaDentroDeFormulasPorColumnas, BuscaCadenaDentroDeFormulasPorFila, ReemplazarTodas y Reemplazar.

5.4.2. FuncionesBase

Solamente contiene dos funciones que son útiles para sumar los elementos de una matriz. Estas funciones sencillas se utilizan por el objeto FuncionesHoja, cuyas funciones más complejas consideran

además como argumentos objetos de tipo Rango. Las funciones que brinda son: *Suma* y *SumaDeCuadrados*

5.4.3. Módulo Principal

Este módulo contiene constantes, procedimientos y funciones globales. Las constantes globales y sus valores están dados en la Tabla VI:

Tabla VI

Constantes Globales de Hoelest

Constante	Valor
NúmeroMáximoColumnas	100
NúmeroMáximoFilas	10000
AltoFilaStandard	240
AnchoColumnaStandard	960

El procedimiento Main es el primer procedimiento que se ejecuta cuando se inicia Hoelest, ya sea como una aplicación de escritorio o como un componente ActiveX.

Por último los procedimientos LeerRegistroSistema y GuardarRegistroSistema devuelven y establecen valores globales de la aplicación con el Registro del Sistema.

Cuando Hoelest se inicia, lee el registro y establece algunas de sus propiedades. El objeto *origen* modifica dichas propiedades cuando se accede a la ventana Opciones desde el menú.

5.4.4. Módulo TrabajoConArchivos

Contiene procedimientos que son llamados por los objetos Hojas y Hoja, en lo referente a la apertura, guardado en disco y cierre de archivos.

Los procedimientos son: ActualizarMenuArchivo, GuardarSTD, GuardarTXT, GrabarArchivoComo, ExtraerSoloNombreArchivo, AbrirSTD, AbrirTXT, ExtraerSoloRutaAcceso, EstaEnListaArchivosRecientes.

5.4.5. EvalúaFórmulas

Las fórmulas introducidas en tiempo de ejecución por un usuario, son evalúadas con el código de este módulo. La función Evalúa, utiliza la función Convertir para calcular el resultado de una expresión. Convertir transforma la expresión introducida por el usuario en notación postfix, de esta manera facilita la evaluación de la misma.

5.4.6. Funciones Internas

Contiene funciones para edición de celdas y fórmulas (ProcEdicionCopiar, ProcEdicionCortar, ProcEdicionPegar), procedimientos que interactúan

con el registro del sistema (ObtenerArchivosRecientes, EscribirArchivosRecientes) y funciones para verificar la existencia de objetos o la validez de expresiones (Quedaalgunahoja, ExisteFormula, NombreVálido)

Los resultados del objeto EstadísticaInferencial pasan al procedimiento DibujarResultadoEnHoja, que se encarga de mostrar los resultados en pantalla.

5.5. Constantes de Hoelest

Hoelest proporciona un conjunto de constantes con nombre a los programadores de aplicaciones cliente. Entre los beneficios de las constantes con nombre está la legibilidad del código de desarrollo. La Tabla VII resume las constantes que proporciona este componente:

Tabla VII

Constantes con Nombre de Hoelest

MHLTFormatoArchivo	mHLTtexto = 1 mHLTHoelest = 2
mHLTModoCalculo	mHLTManual = 0 mHLTAutomatico = 1
mHLTEstadoVentana	mHLTNormal = 0 mHLTMinimizado = 1 mHLTMaximizado = 2
mHLTOrdenarOrden	mHLTOrdenAscendente mHLTOrdenDescendente
mHLTOrden	mHLTAbajoLuegoDerecha mHLTDerechaLuegoAbajo
mHLTTamañoPapel	MHLTPapelCarta MHLTPapelOficio mHLTPapelUsuario
MHLTPagOrientación	MHLTPagOrientaciónVertical mHLTPagOrientaciónHorizontal
MHLTAlineamiento	mHLTAlinIzquierda mHLTAlinCentro mHLTAlinDerecha

CAPITULO 6

6. EN TIEMPO DE EJECUCION

El tema fundamental de la Tesis es la **Estadística Inferencial**. Este capítulo contiene una breve descripción de las técnicas estadísticas que están dentro del alcance de HOELEST, además de, demostrar mediante ejemplos sencillos, cómo pueden utilizarse en la aplicación. Básicamente, tratamos de cubrir dos aspectos:

1. El aspecto estadístico. Cada técnica estadística que revisemos será abordada con el mismo esquema: qué hace, por qué lo hace y cómo interpretar los resultados. Con ello se pretende que el lector tenga una idea más clara de la utilización de las mismas; capacitándose así, para utilizar correctamente las técnicas que se acoplen al diseño y a las características de sus datos cuando lo requiera.

2. El aspecto informático, por otro lado, demuestra cómo aplicar cada técnica estadística y cómo se encuentra organizada la salida (resultados de la prueba, presentada en pantalla o impresa) para su posterior interpretación. Además se detallan nociones básicas relacionadas con trabajo de archivos, manipulación de datos y definición de variables para iniciar un estudio estadístico.

6.1. Nociones Básicas de Hoelest

Como indicamos anteriormente, Hoelest es una hoja electrónica que puede ser utilizada como una aplicación de escritorio o como un servidor ActiveX. Este capítulo demuestra su utilización si se ejecuta como una aplicación de escritorio, como por ejemplo desde el Explorador de Windows.

En esta sección indicaremos como acceder a una opción de menú de Hoelest, por medio de una cadena en letra cursiva, que contiene los nombres de los menús y submenús separados por coma. Por ejemplo, para efectuar un Análisis de Regresión Lineal, se accede al menú *Estadísticas* por medio del ratón o del teclado, al desplegarse el menú se escoge la opción o submenú *Regresión* y posteriormente la opción *Lineal*. La explicación anterior se resume con la siguiente cadena: *Estadísticas,Regresión,Lineal*.

6.1.1. Cómo ejecutar Hoelest

Se inicia una instancia de Hoelest para su utilización por un usuario común por medio de dos alternativas: Desde el Explorador de Windows, dando un click sobre el archivo ejecutable Hoelest.exe o digitando "RutaDeAcceso"\Hoelest en el cuadro del diálogo Ejecutar, de la opción Ejecutar del menú Inicio.

La figura 6.1 ilustra la la aplicación Hoelest. Bajo la pantalla inicial se pueden apreciar dos hojas electrónicas organizadas en mosaicos; en la hoja de la izquierda se definieron variables de estudio y en la otra se presentan resultados con un formato personalizado.

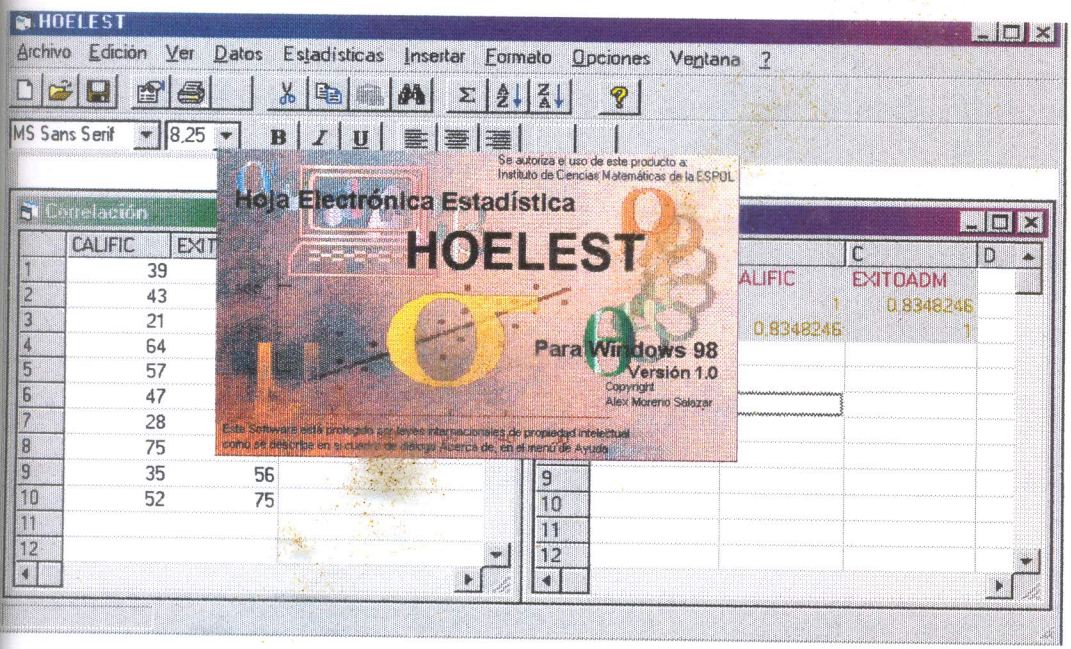


Figura 6. 1. Aplicación Hoelest

6.1.2. Cómo ejecutar Comandos

Existen dos formas de ejecutar comandos:

1. Por medio de los *Menús*. Se puede acceder a los menús con ayuda de la tecla ALT, considerando dos posibilidades: 1. Presionar la tecla ALT seguida de la letra subrayada del comando deseado y 2. Presionar la tecla ALT y desplazarse con ayuda de las teclas direccionales. Para salir de los menús se vuelve a presionar la tecla ALT.

   Entre las opciones que brinda el menú principal de HOELEST están: 1.Menú Archivo (ver sección 6.1.3); 2.Menú Edición, con opciones para buscar, reemplazar datos y trabajo con el portapapeles del sistema; 3.Menú Ver, con opciones para controlar la visualización de barras de herramientas, de comandos y de estado; 4.Menú Datos, (ver sección 6.1.4); 5.Menú Estadísticas, con opciones para realizar pruebas estadísticas inferenciales; 6.Menú Insertar, con opciones para insertar celdas, filas y columnas; 7.Menú Formato, con opciones para establecer fuentes, colores de fondo,etc.; 8.Menú Opciones, permite registrar preferencias del usuario en el Registro del Sistema; 9. Menú Ventana, para manipular las hojas electrónicas y 10.Menú Ayuda (?), que brinda al usuario la posibilidad de consultar sobre las técnicas estadísticas disponibles.y el manejo de la aplicación.

2. Por medio de las *Barras de Herramientas*. Para ejecutar un comando de la barra de herramientas, basta con situar el puntero del ratón sobre el icono del comando deseado y dar un click. Inmediatamente se ejecutará la orden señalada. HOELEST cuenta con dos barras de herramientas: 1. *Formato*, con opciones útiles para dar formato al texto en pantalla e impreso y 2. *Estándar*, con los comandos que se utilizan con mayor frecuencia, como grabar o abrir documentos.

Se cuenta además con la *Barra de Fórmulas*. En ella sólo se pueden escribir fórmulas simples para realizar cálculos sencillos. Lás únicas funciones incorporadas en la aplicación que pueden utilizarse en esta barra son: SUMA y SUMACUADRADOS (ver sección 5.1.9.2).

6.1.3. Trabajo con archivos

Por medio del menú Archivo, se acceden a las opciones de crear, abrir, guardar, cerrar e imprimir documentos. Además se encuentra la opción de configurar página que le permite personalizar la impresión.

6.1.4. Definición de variables de estudio

Para definir variables de estudio en una hoja electrónica, se accede a una ventana que permite crear, editar y eliminar variables por medio del menú Datos,Variables. La figura 6.2 ilustra la ventana Variables de Hoelest.

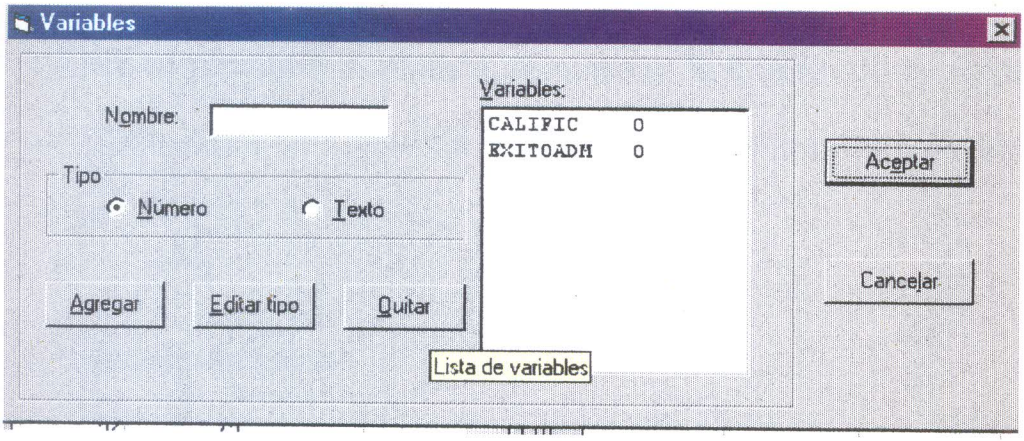


Figura 6. 2. Ventana para la Definición de Variables de Estudio

En la figura 6.2 se puede apreciar que existen dos variables en una hoja electrónica. Cuando se accede a esta ventana se lee información de la hoja electrónica activa; si ésta contiene variables definidas se listarán en *Variables*. Para crear variables se especifica el nombre y tipo, luego se presiona agregar. Para editar nombre o tipo de una variable, se escoge la variable deseada en *Variables* y se presiona EditarTipo. Para eliminar una variable, se escoge primero de la lista Variables y se presiona Quitar.

Los encabezados de las columnas de la hoja electrónica se modifican y toman los nombres de las variables según su orden de creación. Para los cálculos detallados en las secciones siguientes, se supone que existen variables definidas en la hoja electrónica de trabajo, de esta manera, cuando se accede a la ventana del cálculo deseado, Hoelest lee la información de la hoja activa y establece las variables con las que puede

trabajar. Por ejemplo, para calcular las covarianzas de un grupo de variables se accede a la ventana Correlación, la misma que en su parte izquierda muestra un listado de las variables disponibles para realizar el cálculo.

6.2. Correlación Lineal

6.2.1. Introducción

Con frecuencia la intersección de dos o más eventos tiene importancia fundamental para el investigador. De mayor importancia para los estadísticos son las intersecciones que ocurren al efectuar un muestreo. Es necesario entonces, obtener la probabilidad de la muestra observada, es decir, la probabilidad de la intersección de un conjunto de eventos numéricos. En un análisis de covariación, el coeficiente de correlación lineal simple de Pearson tiene un peso fundamental en las técnicas estadísticas.

6.2.2. La covarianza de dos variables aleatorias

La dependencia entre dos variables aleatorias Y_1 y Y_2 existe si una variable, digamos Y_1, crece o decrece cuando Y_2 cambia. Las figuras 6.3 a) y b) representan los puntos que corresponden a dos muestras aleatorias de n=10 unidades experimentales sacadas de una población. Si todos los puntos caen sobre un línea recta, como en la figura 6.1 a), Y_1 y

Y_2 son obviamente dependientes. Al contrario, la figura 6.1 b) indicaría poca o ninguna dependencia entre Y_1 y Y_2.

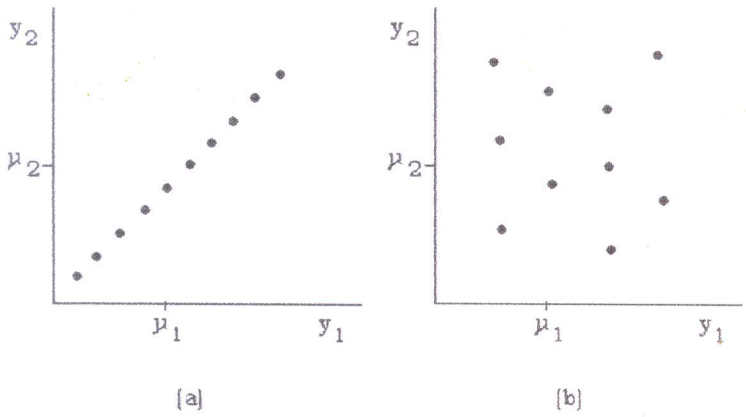


Figura 6. 3. Observaciones dependientes e independientes para Y_1 y Y_2

Supongamos que conocemos los valores esperados μ_1 y μ_2 de Y_1 y Y_2. Podríamos entonces determinar las desviaciones $(Y_1-\mu_1)$ y $(Y_2-\mu_2)$. Si en la figura 6.3 a) escogiéramos todos y cada uno de los puntos y determináramos las desviaciones, veríamos que ambas tendrían el mismo signo y por tanto su producto $(y_1-\mu_1)(y_2-\mu_2)$ será positivo, por lo que el promedio del producto de las desviaciones será grande y positivo.

Si la relación lineal en la figura 6.3 a) tuviera pendiente negativa, todos los pares correspondientes de las desviaciones habrían sido de signo opuesto, y el valor de medio de $(y_1-\mu_1)(y_2-\mu_2)$ habría sido grande negativo.

La situación anterior no se presenta en la figura 6.3 b), en donde hay poca dependencia entre Y_1 y Y_2. Las desviaciones correspondientes, $(y_1-\mu_1)$ y $(y_2-\mu_2)$ tendrían el mismo signo algebraico para algunos puntos y signos opuestos para otros. Así el producto $(y_1-\mu_1)(y_2-\mu_2)$ será positivo para algunos puntos, negativo para otros, y tendrá un promedio cercano a cero. El valor medio de $(y_1-\mu_1)(y_2-\mu_2)$ nos dará un medida de la dependencia lineal de Y_1 y Y_2. Esta cantidad se denomina covarianza de Y_1 y Y_2, cuya expresión es:

$$Cov(Y1,Y2) = \frac{\sum_{i=1}^{N}(y_{1i}-\mu_1)(y_{2i}-\mu_2)}{N} = \frac{\sum_{i=1}^{N} y_{1i}y_{2i}}{N} - \mu_1\mu_2$$

A mayor valor absoluto de la covarianza de Y_1 y Y_2 corresponde una mayor dependencia lineal entre Y_1 y Y_2. Valores positivos indican que Y_1 crece cuando Y_2 crece; valores negativos indican que Y_1 decrece cuando Y_2 crece. Un valor cero de la covarianza indicará que no hay dependencia lineal entre Y_1 y Y_2.

Desafortunadamente, es difícil utilizar la covarianza como una medida absoluta de le dependencia porque su valor depende de la escala de medición y por consiguiente es difícil determinar si una covarianza en particular es grande a simple vista. Se puede eliminar este problema

estandarizando su valor, utilizando el coeficiente simple de correlación lineal.

6.2.3. El Coeficiente de Correlación Lineal Simple de Pearson

Utilizar puntuaciones estandarizadas en lugar de diferenciales, resulta equivalente a dividir la covarianza entre las desviaciones típicas de las variables implicadas. El coeficiente de correlación lineal se simboliza con la letra minúscula **r** cuando se calcula con los datos muestrales, y con la letra griefa rho (ρ) cuando se refiere a los valores de la población. Karl Pearson creó el índice el pasado siglo, por lo que lleva su nombre y se define como:

$$\rho = \frac{Cov(Y_1, Y_2)}{\sigma_1 \sigma_2}$$

La fórmula es útil sólo para contextos de dos variables. Para diferenciar la utilización de esta medida en situaciones de dos variables con respecto a aplicaciones con múltiples variables, se utiliza el calificativo de simple.

El término covarianza se refiere, literalmente, a variación conjunta. Con la covarianza no nos preguntamos si una variable varía en función de la otra, sino que se observa una determinada magnitud de variación conjunta (digamos que ambas variables se mueven a la vez:cuando una

se mueve, se mueven las dos). El coeficiente **r** basado en la covarianza, también recoje variación conjunta, pero con un control mayor de los aspectos que no se refieren directamente al grado de relación entre las variables. Por este motivo, suele decirse que **r** mide, más que relación, correlación.

Por último, una relación lineal, regida por una línea recta, una variable puede ser puesta en función de la otra según la ecuación de una recta. Así pues, nos referimos a r como el coeficiente de correlación lineal simple de Pearson.

Puede demostrarse que el coeficiente de correlación ρ satisface la desigualdad $-1 <= \rho <= 1$. Así, -1 ó 1 implica una correlación perfecta, con todos los puntos sobre una línea recta. El valor $\rho=0$ implica una covarianza igual a cero y ninguna correlación.

El signo del coeficiente de correlación depende del signo de la covarianza. Así, un coeficiente de correlación positivo indica que Y2 crece cuando Y1 crece. Un coeficiente de correlación negativa implica que Y2 decrece cuando Y1 crece.

6.2.4. Correlación lineal simple de Pearson en Hoelest

Por medio del menú *Estadísticas, Correlación* se accede a una pantalla que permite especificar las variables que se desean estudiar. La figura 6.4 ilustra la ventana *Correlación* de Hoelest.

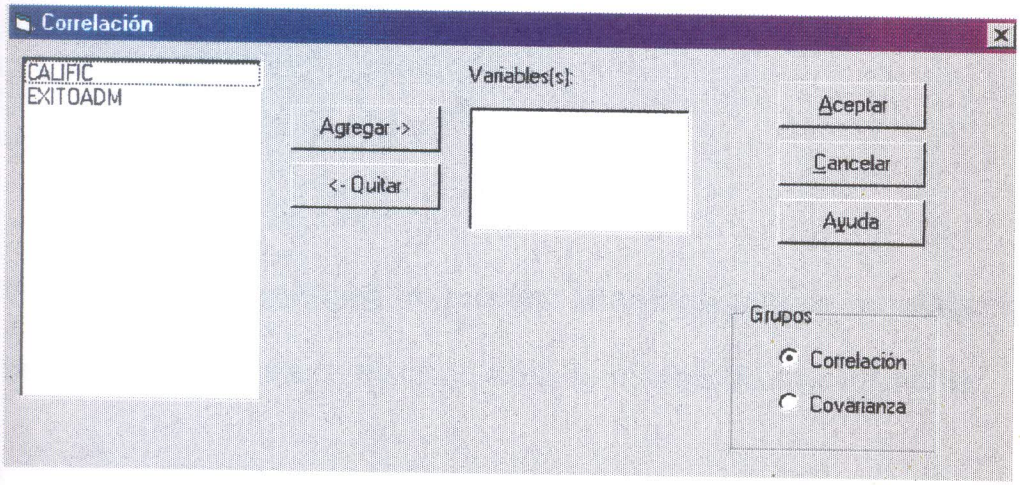


Figura 6. 4. Ventana para el cálculo de Correlación Lineal

En la parte izquierda de la ventana *Correlación* se listan todas las variables definidas en la hoja de cálculo; por medio del botón Agregar, se añaden a una segunda lista que contiene a las variables con las que se desea trabajar.

Cuando se presiona el botón Aceptar, Hoelest realiza el cálculo especificado en el recuadro *Grupos*, que pueden ser Correlación o Covarianzas. El valor por defecto es Correlación.

Cuando se haya realizado el cálculo, Hoelest presenta los resultados (correlaciones o covarianzas) en un matriz, cuyas filas y columnas están etiquetadas con los nombres de las variables. La intersección de una fila y una columna, por decir, la primera fila con la segunda columna, contiene la correlación o covarianza entre las variables que representan la fila y la columna.

Ejemplo:

Consideremos el problema de predecir la calificación que un aspirante a gerente recibirá después de cinco años de trabajar en una compañía.

La clasificación en la escala del éxito ***y*** es el promedio de las evaluaciones subjetivas de cinco gerentes, en la escala de 0 (mal) a 100 (muy bien).

La calificación (de 0 a 100) del aspirante ***x*** es la obtenida en una prueba combinada de logros y de personalidad, realizada al final del curso de entrenamiento administrativo. La compañía seleccionó aleatoriamente a diez gerentes que trabajan en ella hace cinco años. Se muestra en la Tabla VIII su clasificación en la escala de éxito y sus calificaciones como aspirantes.

Tabla VIII

Datos para el ejemplo de Correlación lineal

Gerente	Calificación como aspirante (x)	Clasificación de éxito administrativo (y)
1	39	65
2	43	78
3	21	52
4	64	82
5	57	92
6	47	89
7	28	73
8	75	98
9	34	56
10	52	75

Se definen las variables CALIF y EXITOADM en Hoelest, que representan las variables Calificación como aspirante y Clasificación de éxito. Accedemos a la ventana Correlación de Hoelest, por medio de *Estadísticas,Correlación* y agregamos ambas variables a la lista de variables de estudio. Presionamos el botón aceptar y se presentará la siguiente matriz en una hoja electrónica.

	CALIFIC	EXITOADM
CALIFIC	1	0,8348246
EXITOADM	0,8348246	1

Entonces, vemos que el coeficiente de correlación lineal simple de Pearson entre CALIF y EXITOADM tiene un valor de 0,84. Existe entonces, una relación lineal positiva entre ambas variables, en otras palabras, a medida que un aspirante obtiene una mayor calificación en la prueba final del curso, mayor será su clasificación en la escala de éxito administrativo.

6.3. Pruebas de Hipótesis

6.3.1. Introducción

Cuando hablamos de pruebas de hipótesis sobre una variable, el interés no es la descripción de una variable, sino la decisión desde el punto de vista probabilístico sobre una hipótesis que se establece con respecto a la distribución de la variable o sobre otras características de su comportamiento. En muchos aspectos el procedimiento formal para la prueba de hipótesis es similar al método científico. El científico observa la naturaleza, establece una teoría y después prueba su teoría respecto de la observación. En este contexto el científico propone una teoría relativa a los valores específicos de uno o más parámetros poblacionales. Luego

obtiene una muestra de la población y compara la observación con la teoría. Si las observaciones se contraponen a la teoría, el científico rechaza la hipótesis. En caso contrario concluye que la teoría es válida o bien que la muestra no detectó la diferencia entre los valores reales y los valores de la hipótesis respecto de los parámetros poblacionales.

Contraste o comparación de medias. En una variable medida, se obtiene una media aritmética con un valor concreto. La pregunta de investigación es si puede considerarse que la muestra proviene de una población caracterizada por un valor determinado y conocido para la media aritmética. La hipótesis nula establece que la diferencia que se observa entre ambos valores es debida al azar, y por tanto no significativa, por lo que la muestra sí proviene de esa población.

Contraste o comparación de proporciones. En la variable que interesa se observa una proporción concreta de valores de una categoría con respecto al N total de la muestra. Mediante esta prueba de hipótesis se establece la probabilidad de que la proporción observada sea realmente (en la población de origen) una proporción teórica o esperada. Si el grado de significación es menor que el nivel de significación, se rechazará la hipótesis de nulidad, estableciendo que la muestra proviene de otra

población, o bien que la diferencia entre lo observado y lo esperado es significativa y no despreciable.

Prueba de comparación con un modelo teórico de distribución. Las frecuencias observadas de la muestra pueden seguir más o menos un patrón tal que se pueda mantener una hipótesis de nulidad en los siguientes términos: no existe diferencia significativa entre la distribución de datos observada y la que cabría esperar si tales datos siguieran un modelo de distribución teórico conocido y concreto (normal, Poisson o uniforme). Si el grado de significación es menor que el nivel de significación, se rechazará la hipótesis de nulidad, estableciendo que la variable en estudio no sigue como distribución el modelo especificado.

6.3.2. La Hipótesis Estadística

El momento de la decisión acerca de si existe o no relación, tiene lugar cuando se analizan las mediciones realizadas. El análisis de mediciones usual es el estadístico.

Si hemos partido de que no existe relación, podemos llamar a este enunciado hipótesis de partida. En el momento del análisis estadístico, a esta negación se la denomina hipótesis nula y se simboliza con H_o . Es una hipótesis porque se trata de un enunciado que se establece antes de

comprobar si es cierto o no. Y se denomina nula, porque se redacta de forma que niega la relación.

El análisis estadístico va a consistir básicamente en recrear la situación en la que la hipótesis nula es cierta y calcular la probabilidad de que, en ese contexto se observen los resultados de la investigación.

La utilidad de esta asignación es inmediata. Si se trata de una probabilidad muy pequeña, puedo considerar que H_o es cierta, pues es poco probable que me equivoque al realizar este acto de rechazo.

El fundamento de este proceder es considerar que no es posible realizar una afirmación tajante y duradera. Puedo encontrarme en un salón y anotar el color de pelo de todo aquel que entra. Tras 256 observaciones, he notado 246 de pelo negro. Podría concluir que en ese salón sólo entran personas de pelo negro. No obstante, en la observación 257 aparece un rubio. Mi afirmación, pues, era errónea. El problema reside en que nunca se podrá realizar una afirmación libre de error, por muchas observaciones que se realicen, salvo que se lleven a cabo todas la observaciones posibles. La solución está en considerar el error en vez de obviarlo. Así, en el ejemplo, a pesar de encontrar una persona rubia en la observación 257, podré afirmar que sólo entran personas de pelo negro,

suponiendo que voy a equivocarme al mantener esta afirmación una vez cada 257 ocasiones y considerando esta probabilidad de error (1/257) muy pequeña. Simbolizaremos con *p* la probabilidad de que ocurra la observación K en la situación de que H_o es cierta.

La hipótesis alternativa es la que se adopta cuando se rechaza la hipótesis nula y, en realidad es la hipótesis que se desea investigar. Si se logra demostrar que la hipótesis nula es falsa entonces se adopta la hipótesis alternativa como verdadera.

6.3.3. Riesgo Límite

¿Cómo considerar que *p* es lo suficientemente pequeña como para considerar que H_o no es verdadera?. La solución es comparar *p* con una probabilidad límite, a partir de la cual *p* dejará de ser considerada pequeña o suficientemente pequeña. Este término de comparación, que llamaremos riesgo límite, se denota por la letra griega alfa (α) Realizando la comparación entre *p* y α, sólo caben tres posibles resultados:

a) Si $p < \alpha$. Se concluye que existe relación, ya que negamos Ho= "no existe relación"
b) Si $p > \alpha$. No consideramos falsa a Ho. No se ha encontrado evidencia sufiente de relación entre variables.

c) Si $p = \alpha$. Es muy difícil que ocurra esta coincidencia. Hacemos la misma conclusión que el literal b.

6.3.4. Elementos de una Prueba Estadística

Toda prueba de hipótesis consta de 4 elementos: La hipótesis nula (H_o), la hipótesis alternativa (H_a), la estadística de prueba y la región de rechazo. Hemos hablado ya de los dos primeros. La decisión de rechazar la hipótesis nula y de aceptar la hipótesis alternativa, se basa en la información contenida en una muestra de *n* mediciones. Los valores muestrales se utilizan para calcular un solo número que corresponde a un punto de una recta. Tal número sirve para tomar decisiones y se denomina estadístico de prueba.

Se divide todo el conjunto de valores que puede tomar una estadística de prueba en dos regiones: uno que corresponde a la región de rechazo y otro que corresponde a la región de aceptación. Si el valor de la estadística de prueba cae dentro de la región de rechazo, se descarta la hipótesis nula a favor de la alternativa.

6.3.5. Errores en la prueba de hipótesis

Hay dos tipos de error en las pruebas de hipótesis: El error tipo I y el error tipo II. En un error Tipo I la hipótesis nula es rechazada cuando realmente

es verdadera. La probabilidad de este error es precisamente α, el nivel de significancia de la prueba. En un error tipo II, la hipótesis nula no es rechazada cuando ésta en realidad es falsa. La probabilidad del error tipo II se denota por β. Las dos posibilidades para la hipótesis nula –verdadera o falsa- junto con las dos desviaciones que el experimentador puede tener, se muestran en Tabla IX.

Tabla IX

Tabla de decisiones

	Hipótesis nula	
Decisión	**Verdadera**	**Falsa**
Rechazar Ho	Error Tipo I	Decisión correcta
Aceptar Ho	Decisión correcta	Error Tipo II

6.3.6. Prueba de hipótesis para una media poblacional

Supóngase que $X_1, \ldots, X_n$ representa una muestra aleatoria de tamaño n de una distribución normal con una media desconocida μ y varianza desconocida σ^2. En esta prueba, el parámetro que se desea probar es la media poblacional μ. Se utiliza un estimador de μ que es la media muestral . $\bar{x}$ y la desviación estándar $\sigma_{\hat{\theta}}$ de la distribución muestral de $\bar{x}$ es $\sigma / \sqrt{n}$ para construir el estadístico de prueba T, que tendrá una distribución t de Student con (n-1) grados de libertad.

En Hoelest se realiza la siguiente prueba de hipótesis:

Hipótesis nula: Ho: $\mu = \mu_o$

Hipótesis alternativa: Ha: $\mu \neq \mu_o$

Estadística de prueba:

$$t = \frac{\ddot{x} - \mu_o}{\sigma_{\ddot{x}}} = \frac{\ddot{x} - \mu_o}{\sigma / \sqrt{n}}$$

Región de rechazo: Prueba de dos extremidades:

$$t > t_{\alpha/2} \text{ o bien } t < -t_{\alpha/2}$$

Los valores de t y $t_{\alpha/2}$ se basan en (n-1) grados de libertad.

6.3.6.1. Prueba de Hipótesis Simple en Hoelest

Por medio del menú *Estadísticas,Pruebas de Hipótesis,SImple* se accede a una pantalla que permite especificar las variables que se desean estudiar. La figura 6.5 ilustra la ventana *Prueba de Hipótesis SImple* de Hoelest. En la parte izquierda de la ventana *Prueba de Hipótesis Simple* se listan todas las variables definidas en la hoja de cálculo; por medio del botón Agregar, se añaden a una segunda lista que contiene a las variables con las que se trabajará.

Cuando se presiona el botón Aceptar, Hoelest hace una prueba de hipótesis con respecto a la media especificada en el cuado *Media*, y con el nivel de confianza especificada en *Confidencia*. Se hacen los cálculos

para cada variable especificada en Variables(s).

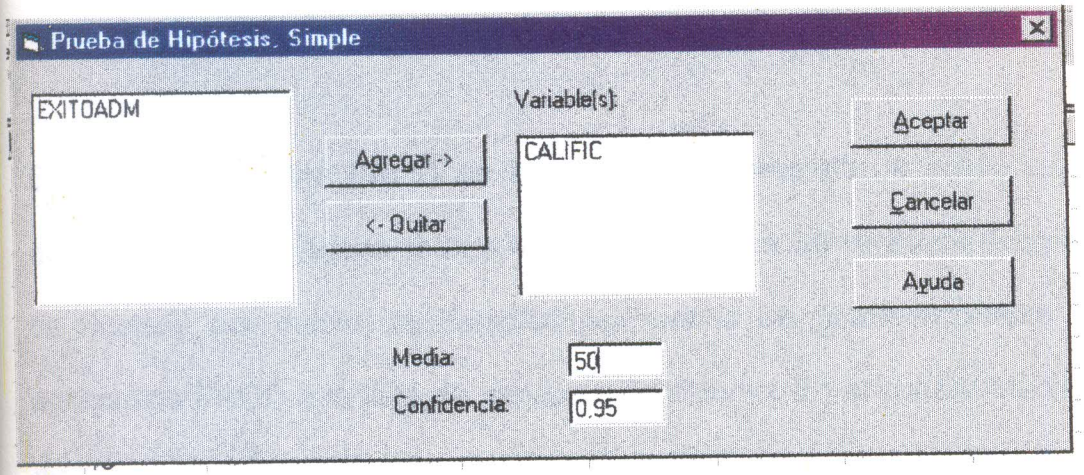


Figura 6. 5. Ventana para efectuar una Prueba de Hipótesis Simple

Hoelest presenta los resultados de la prueba para cada variable individualmente. La salida contiene el número de casos de la variable estudiada, un intervalo de confianza al nivel especificado, el valor del estadístico de prueba calculado para la media especificada y el nivel de significancia asociado con dicho estadístico.

Ejemplo:

La producción diaria en una planta industrial química registrada durante n=50 días, tiene una media muestral y una desviación estándar de 871 toneladas y 21 toneladas, respectivamente. Se desea probar la hipótesis

de que el promedio de la producción diaria del producto químico es μ = 880 toneladas por día, contra la alternativa de que μ es mayor o menor que 880 toneladas

Se define la variable PROD en Hoelest, que representa la producción diaria del químico. Accedemos a la ventana Prueba de Hipótesis Simple de Hoelest, por medio de *Estadísticas,Pruebas de Hipótesis,Simple* y agregamos PROD a la lista de variables de estudio. En el cuadro Media, digitamos la hipótesis nula, es decir, 880. Se utilizará α=0,05, es decir, trabajaremos con una confianza del (1-α)100%= 95%. Presionamos el botón aceptar y se presentará la siguiente salida en una hoja electrónica.

PH simple para CALIFIC		N casos 10		Ho: Media= 50	
Media:	871	0,95 IC :	865,18	a	876,82
Desv.Est:	21	Valor t Student:	-3,03		
		Prob:	0,954		

Para α = 0.05, la región de rechazo es z > 1.96 o z < 1.96. Como el valor calculado de z cae en la región de rechazo, se descarta la hipótesis de que μ=880 toneladas. La probabilidad de rechazar H_o, suponiendo que sea certa es solamente α=0,05; por lo que se está razonablemente seguro de que la respuesta es cierta. El intervalo de confianza nos notifica que,

en un muestreo repetitivo, (1-α)100% de los intervalos contendrán a μ. Al observar que μ=880 no cae en el intervalo, nos inclinaríamos a rechazar μ = 880 como un valor probable y se concluiría que el promedio de la producción diaria es menor.

6.3.7. Prueba de hipótesis para comparar dos medias poblacionales

Supóngase que se eligen dos muestras independientes de dos distribuciones normales, X_{11}, X_{12}, ..., X_{1n1} de la primera y X_{21}, X_{22}, ...,X_{2n2} de la segunda, en donde la media y la varianza de la i-ésima población son μ_i y σ^2, i=1,2. Además, supóngase que X_i y S_i,, i=1,2 son las medias y las varianzas muestrales correspondientes (Note que ambas distribuciones tienen varianzas iguales). Cuando se satisfacen estas condiciones, se tiene que

$$S^2 = \frac{(n1-1)S_1^2 + (n2-1)S_2^2}{ni + n2 - 2}$$

es un estimador ponderado para σ^2, entonces el estadístico de prueba T:

$$t = \frac{\ddot{x}1 - \ddot{x}2 - D_o}{S\sqrt{\frac{1}{n1} + \frac{1}{n2}}}$$

Tiene una distribución t de Student con (n_1+n_2 - 2) grados de libertad.

En Hoelest se realiza la siguiente prueba de hipótesis:

Hipótesis nula: $H_o: \mu_1 - \mu_2 = D_o$

Hipótesis alternativa: $H_a: \mu_1 - \mu_2 \neq D_o$

Estadística de prueba:

$$t = \frac{\ddot{x}1 - \ddot{x}2 - D_o}{S\sqrt{\frac{1}{n1} + \frac{1}{n2}}}$$

Región de rechazo: Prueba de dos extremidades:

$t > t\alpha/2$ o bien $t < -t\,\alpha/2$

Los valores de t y $t\alpha/2$ se basan en (n1+ n2 - 2) grados de libertad.

Hoelest siempre considera que D_o tiene el valor de cero. Por lo que la prueba descrita arriba, se convierte en una prueba de igualdad de medias de las dos poblaciones

6.3.7.1. Prueba de hipótesis para comparar dos medias en Hoelest

Por medio del menú *Estadísticas,Pruebas de Hipótesis,Dos grupos* se accede a una pantalla que permite especificar las variables que se desean estudiar. La figura 6.6 ilustra la ventana *Prueba de Hipótesis Dos grupos* de Hoelest. En la parte izquierda de la ventana *Prueba de Hipótesis Dos Grupos* se listan todas las variables definidas en la hoja de cálculo; por medio del botón Agregar, se añaden a una segunda lista que contiene a las variables con las que se trabajará. Cuando se presiona el

botón Aceptar, Hoelest realiza una prueba de hipótesis con el nivel de confianza especificado en *Confidencia* para cada combinación posible de dos (por parejas) de las variables incluídas en el análisis.

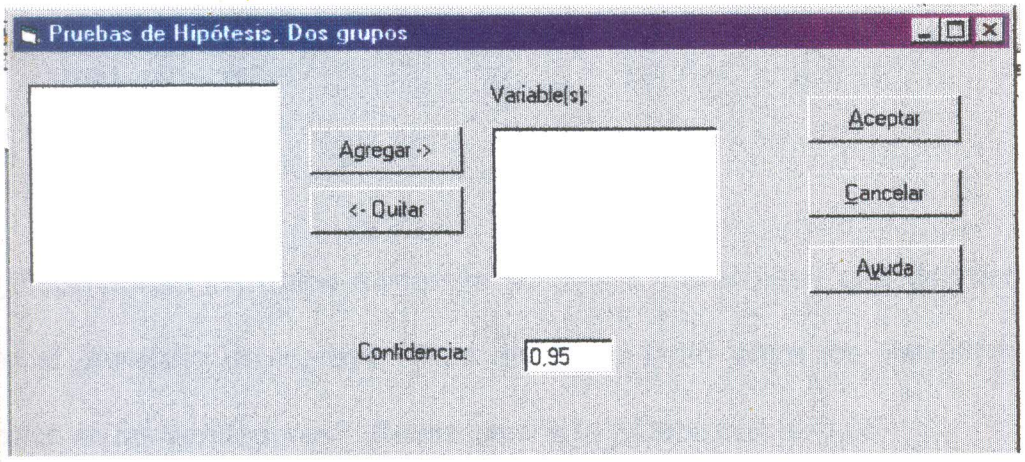


Figura 6. 6. Ventana para una Prueba de Hipótesis de Dos Medias

Hoelest presenta los resultados de la prueba para cada pareja de variables por separado. La salida contiene la diferencia de medias, diferencia de desviaciones, un intervalo de confianza al nivel especificado, el valor del estadístico de prueba calculado para la media especificada y el nivel de significancia asociado con dicho estadístico.

Ejemplo:

Una compañía desea comparar las expectativas salariales anuales de su personal de ventas femenino y masculino, según un nuevo plan de

compensaciones ventas-más-comisión. Se pidió a n_1=40 vendedoras, y a n_2=40 vendedores, muestreados al azar, predijeran sus ingresos anuales bajo el nuevo plan. Las medias muestrales y las desviaciones muestrales eran

$$\ddot{x}_1 = 31083 \quad \ddot{x}_2 = 29745$$
$$s_1 = 2312 \quad s_2 = 2569$$

¿Proporcionan los datos evidencia suficiente que indique una diferencia en el promedio de ingreso anual esperado tanto entre los vendedores como en las vendedoras?. Realizaremos la prueba con α=0.05

Se definen las variables EXPHOMB y EXPMUJ en Hoelest, que representan las variables expectativas de ingresos de los vendedores y vendedoras respectivamente. Accedemos a la ventana *Prueba de Hipótesis, Dos grupos* de Hoelest, por medio de Estadísticas,PruebasDeHipótesis,DosGrupos y agregamos ambas variables a la lista de variables de estudio.

Se utilizará α=0.05, es decir, trabajaremos con una confianza del (1-α)100%= 95%. Presionamos el botón aceptar y se presentará la siguiente salida en una hoja electrónica.

PH por pares para EXPMUJ vs. EXPHOMB N casos: 40

Media: EXPMUJ 31083

Media: EXPHOMB 29745

Diferencia de medias 1338 0,95 IC: 16,00949 a 43,79052

Desviación Dif 546,468 Valor t Student: 2,45

Prob: 0,9858

AL utilizar la prueba de dos colas con α=0,05, rechazaremos H_o si z > 1.96 o bien z < -1.96. Ya que el valor observado de z, z= 2.45 excede a 1.96, la estadística de prueba cae en la región de rechazo. Se descarta H_o. (igualdad de medias) y concluimos que existe una diferencia en el promedio de las expectativas salariales anuales entre los vendedores y las vendedoras. Hay mucha confianza en haber tomado la decisión correcta, puesto que la probabilidad de rechazar H_o, cuando es verdadera es solamente α=0,05.

6.4. Regresión Lineal

6.4.1. Introducción

El punto máximo de la formalización de una relación entre variables es la construcción de una fórmula matemática, una ecuación algebraica donde unas variables se expresan en relación a otras. Se pueden utilizar muchos tipos diferentes de funciones matemáticas para representar el modelo de

una respuesta que sea función de una o más variables independientes. Es posible clasificar estos modelos en dos categorías, los modelos determinísticos y los modelos probabilísticos. Un modelo determinístico no permite error alguno en la predicción, como en la siguiente ecuación:

$$Y=\beta_0 +\beta_1 x$$

donde Y siempre toma una valor exacto de acuerdo al valor de X. En cambio, un modelo probabilístico contempla además un componente aleatorio, que representa el error en la predicción. Por ejemplo, podemos representar las respuestas de la figura 6.7 mediante el modelo

$$E(Y)=\beta_0+\beta_1 x$$

que equivale a $Y=\beta_0+\beta_1 x+\varepsilon$

en donde $E(Y)=\beta_0+\beta_1 x$ es el componente determinístico de Y ,y ε es el componente aleatorio con una distribución de probabilidad específica con media cero. Al considerar el componente aleatorio, el modelo probabilístico constituye una mejor representación de la realidad que el modelo determinístico.

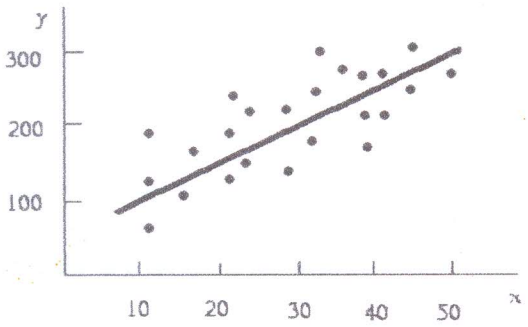


Figura 6. 7. Representación gráfica de los datos.

En física relacionamos volumen, presión y temperatura a través de una ecuación algebraica que siempre se cumple. Por ejemplo la ley de Newton que expresa la relación entre la fuerza que actúa sobre un cuerpo en movimiento, su masa y su aceleración:

F=ma

En psicología, por ejemplo, se relaciona el rendimiento en los estudios con las variables horas de estudio, ambiente familiar e inteligencia. Pero en este caso, la adecuación de la fórmula es restringida y se expresa en términos de seguridad probabilística. Mediante regresión lineal, un psicólogo puede poner la variable satisfacción laboral en función de, por ejemplo, las variables remuneración económica real, percepción sobre la

remuneración económica, antigüedad en la empresa, estatus en la empresa y nivel de actividad

6.4.2. La idea de Regresión Lineal

6.4.2.1. Modelos Estadísticos Lineales

Si Y es la variable de respuesta y X una variable independiente, parece razonable utilizar el modelo $E(Y)=\beta_0+\beta_1 x$, para valores desconocidos de los parámetros β_0 y β_1. Como podrá apreciar en este modelo E(Y) es una función lineal de x (para β_0 y β_1 dados) y también es una función lineal de β_0 y β_1 *[ya que* $E(Y)=c\beta_0+d\beta_1$ *con* $c=1$ *y* $d=x$*]*. En el modelo $E(Y)=\beta_0+\beta_1 x^2$, E(Y) no es una función lineal de x, pero sí es una función lineal de β_0 y β_1 [*con* $c=1$ *y* $d=x^2$]. Cuando nos referimos a un modelo estadístico lineal para Y, se deriva que E(Y) es una función lineal de los parámetros desconocidos β_0 y β_1 y no necesariamente una función lineal de x. Por lo tanto $Y=\beta_0+\beta_1(e^x)+\varepsilon$ es un modelo lineal (suponiendo que e^x es una constante conocida).

Un modelo de regresión lineal simple es aquel que expresa a E(Y) como función de β_0 y β_1 únicamente. Si hay más de una variable independiente de interés, digamos $x_1, x_2, \ldots, x_k$, y si el modelo de E(Y) es

$$E(Y) = \beta_0+\beta_1 x + \ldots. + \beta_k$$

Entonces se dice que el modelo lineal es de regresión múltiple. En este contexto, las variables. Y y X_i, reciben denominaciones acordes con la función que cumplen en la recta de regresión:

Y: variable dependiente (depende de los X_i), variable predicha (por los X_i) o variable criterio (referente para la construcción de la ecuación).

X_i: variable independiente (no depende de Y), variable predictora (de los valores de Y), regresor (parte activa de la ecuación de regresión).

6.4.2.2. Procedimiento de ajuste

Dado que una recta queda perfectamente definida a partir de dos puntos, teniendo dos puntos en un plano debemos ser capaces de encontrar la ecuación de la recta que los une. Por ejemplo, con los datos:

X	Y
3	7
6	9

Se construyen las ecuaciones y=5 + 2x/3 o bien x=1.5y – 7.5 Cuando contamos con más de dos puntos (figura 6.8), resulta evidente que no existe una línea recta que los una; es entonces cuando tenemos lo que se denomina una nube de puntos o diagrama de dispersión. Lo factible es utilizar una línea recta en lugar de la nube de puntos.

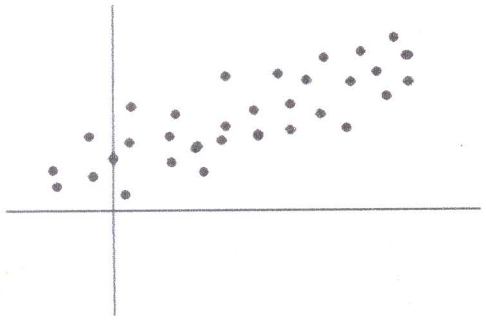


Figura 6. 8. Nube de puntos o diagrama de dispersión

Como se observa, en la representación de la figura 6.8 existe una tendencia lineal. Conforme aumenta el valor de la variable de abscisas, aumenta (con cierta dispersión) el valor de la variable de ordenadas. Si tal gráfica representara algún aspecto de la conducta que está en estudio, ésta resultaría mucho más abordable, simplificando la diversidad de la nube de puntos por una línea recta (Figura 6.9)

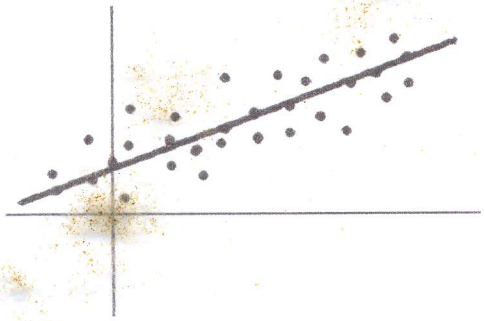


Figura 6. 9. Representación de una nube de puntos a través de una línea recta.

El procedimiento matemático por el que se representa una nube de puntos mediante una línea recta se denomina procedimiento de ajuste. Existen varios procedimientos de ajuste, por ello al momento de elegir, nos interesa aquel que genera el menor error al sustituir la nube por la recta.

¿Cómo promediar los errores producidos por el ajuste? Si se suman los errores y se divide por el número de puntos, no tendremos una idea clara del error. Ocurre que unos errores serán por exceso (positivos) y otros por defecto (negativos), con lo que unos y otros se equilibrarán y podremos obtener un ajuste pésimo con una suma de errores igual a cero. Una solución consiste en calcular la media de los valores absolutos de los errores. No obstante, en estadística se prefiere controlar el signo mediante la operación aritmética de elevar al cuadrado, con lo que todos los signos se convierten en positivos y se realzan las diferencias. Así pues, el criterio rey para escoger una recta que represente la nube de puntos es proceder seleccionando la que genere menos error cuadrático.

6.4.2.3. El Método de los Mínimos Cuadrados

Supóngase que se desea ajustar el modelo

$$E(Y)=\beta_0+\beta_1 x$$

Al conjunto de puntos ordenados de la Figura 6.10. En otras palabras, se postula que $Y=\beta_0+\beta_1 x+\varepsilon$ en donde ε tiene una distribución con $E(\varepsilon)=0$. Si $\hat{\beta}_0$ y $\hat{\beta}_1$ son estimadores de los parámetros β_0 y β, entonces $\hat{Y}=\hat{\beta}_0+\hat{\beta}_1 x$ es obviamente un estimador de E(Y).

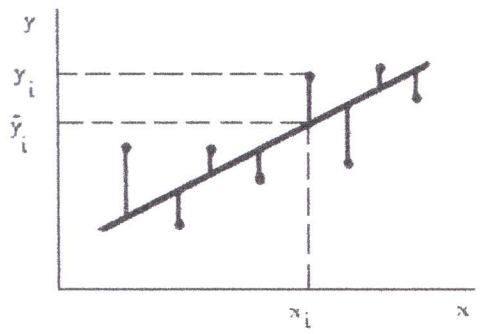


Figura 6. 10. Ajuste de una línea recta a través de un conjunto de puntos

Como indicamos en la sección anterior, se pretende minimizar la suma de los cuadrados de las desviaciones verticales de la recta ajustada (figura 6.10). Por lo tanto si $\hat{Y}=\hat{\beta}_0+\hat{\beta}_1 x$ es el valor que se predice del i-ésimo valor de *y* (cuando $x=x_i$), entonces la desviación del valor observado de *y* a partir de la recta (llamada error o residual) es $y_i-\hat{y}_i$ y la suma de los cuadrados de las desviaciones que deben minimizarse es:

$$SCE=\sum\varepsilon^2=\sum_{i=1}^{n}(y_i-\hat{y}_i)^2$$

Ocurre que la SCE se encuentra en función de dos variables: origen (β_0) y pendiente (β_1). Para poder calcular qué expresiones de estas dos variables hacen mínimo a la SCE, se tendrán que despejar dos ecuaciones: la derivada parcial de la SCE con respecto al origen (β_0) igualada a cero, y la derivada parcial de la SCE con respecto a la pendiente (β_1) igualada a cero. Así obtenemos dos ecuaciones lineales:

$$\frac{\partial SCE}{\partial \hat{\beta}_o} = -2\left(\sum_{i=1}^{n} y_i - n\hat{\beta}_o - \hat{\beta}_1 \sum_{i=1}^{n} x_i\right) = 0$$

$$\frac{\partial SCE}{\partial \hat{\beta}_1} = -2\left(\sum_{i=1}^{n} x_i y_i - \hat{\beta}_o \sum_{i=1}^{n} x_i - \hat{\beta}_1 \sum_{i=1}^{n} x_i^{\,2}\right) = 0$$

Las ecuaciones de arriba se denominan *ecuaciones de los mínimos cuadrados* para estimar los parámetros de una recta.

6.4.2.4. ¿Por qué Regresión?

Imaginemos que realizamos una serie de mediciones en un conjunto de personas. Como una consecuencia, relacionamos los valores obtenidos en dos variables y se genera una nube de puntos; es muy probable que los valores más alejados de la nube correspondan a mediciones con una influencia sensible al azar. Si se realiza otra nueva medición en aquellos sujetos que se encuentran en los extremos de la nube de puntos, es fácil que ocurra lo siguiente: los valores son ahora menos extremos; es decir,

más cercanos a la media del conjunto. Este efecto fue bautizado como regresión hacia la media.

En una ecuación de regresión, la recta representa los valores medios de la variable de ordenadas para cada valor observado de la variable de abscisas. Si se produce una regresión hacia la media, ocurrirá que en una siguiente medición los puntos extremos de la nube de puntos se encontrarán más cercanos a la recta llamada, por tanto, de regresión. Por supuesto, se observará que otros puntos, originalmente más cercanos a la recta de regresión, se alejarán en la segunda medición, de forma que, en términos globales, no hay porqué esperar diferencias sensibles en la apariencia global de la nube de puntos entre dos mediciones sucesivas.

6.4.2.5. Ecuación de Regresión Simple vs. Múltiple

En la práctica, suelen utilizarse varios regresores. Se trata de la llamada regresión lineal múltiple. No se habla ya de una recta de regresión, sino de un plano de regresión (caso de dos regresores) o un hiperplano de regresión (con más de dos regresores).

La representación gráfica, pues, comienza a hacerse inviable, con lo que únicamente queda el recurso algebraico para comprender los conceptos.

6.4.3. La Proporción de Variación Explicada

6.4.3.1. La Perspectiva de la Variación

Cuando en un estudio contamos con una sola variable podríamos realizar un análisis de los datos de ella. Sin embargo, si pretendemos explicar las razones por las cuales varían sus valores nos encontraríamos limitados a encontrar soluciones porque contamos con poca información; es necesaria entonces, la intervención de más variables de las que creamos puede obtenerse cierta comprensión de la variación de la variable de interés. Pongamos que la variable cuya explicación nos preocupa es Y. Si representamos su variación con un círculo, obtenemos la figura 6.11:

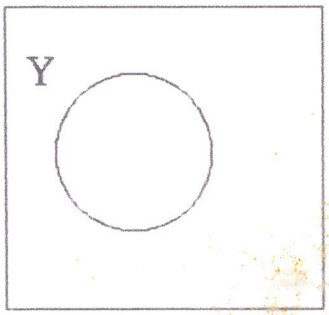


Figura 6. 11. Variación observada de la variable Y

Ahora, existe otra variable X que suponemos influye sobre el comportamiento de Y. Al representar las dos variables (figura 6.12), podremos observar que la inclusión de X resta variación no explicada en la variable Y. Dicho de otro modo: la variable Y que antes no era

explicada por nada, resulta ahora más comprensible debido a la influencia de la variable X.

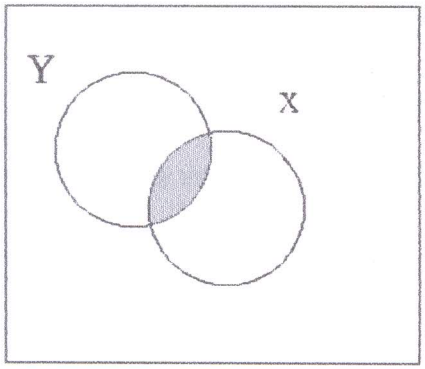


Figura 6. 12. Variación de la variable Y que es explicada por la variable X

Digamos que Y es el rendimiento académico y, que X representa al coeficiente intelectual de los alumnos. Tratamos de averiguar si la inteligencia de los alumnos es un factor que incide en el rendimiento académico.

Según la figura 6.12 parte del área de variación del rendimiento ha sido consumida por la inteligencia, de forma que lo que queda aún por explicar es menos.

6.4.3.2. El Coeficiente de Determinación

Como sabemos la varianza es una medida de la dispersión de los datos con respecto a la media. Su expresión original es:

$$S^2 = \frac{\sum_{i=1}^{n}(x_i - \ddot{x})^2}{n}$$

El numerador de esta expresión, ya que surge de elevar al cuadrado las diferencias con respecto a la media, se conoce como suma de cuadrados. Y dado que se refiere a toda la variación de la variable, se denomina mas propiamente suma de cuadrados total (SCtot).

El referente en la suma de cuadrados total es la media. Si la expresión se calcula con respecto a las puntuaciones predichas, tenemos una medida de variación no explicada, puesto que el $y_i - \hat{y}_i$ es precisamente el error, residuo, o parte del caso i, no explicado por el regresor.

Por esta razón se la podría conocer como suma de cuadrados no explicada o suma de cuadrados residual (SCcres). Por otro lado, la diferencia entre la predicción y la media es precisamente de lo que se ocupa el modelo. Si no existiera nada más que la variable dependiente, la única predicción de valores admisible sería la media. La inclusión de un regresor aumenta este poder predictivo con valores que se separan de la media según la potencia del regresor como variable explicativa. Así pues, a la suma de los cuadrados de las distancias entre las predicciones y la media, se la conoce como suma de cuadrados explicada (SCexp)

Por último, la suma de cuadrados total en el modelo de regresión lineal con ajuste por mínimos cuadrados coincide con la suma de los cuadrados explicada más residual; es decir:

SCtot= SCexp + SCres

$$\sum\left(Y_i - \ddot{Y}\right)^2 = \sum\left(\hat{Y}_i - \ddot{Y}\right)^2 + \sum\left(Y_i - \hat{Y}_i\right)^2$$

Dividiendo ambos términos de la igualdad entre la suma de cuadrados total:

$$\frac{SCTot}{SCTot} = \frac{SC\exp}{SCTot} + \frac{SCres}{SCTot}$$

El primer término, por coincidir numerador y denominador, tiene como valor 1. El segundo se lee como proporción de variación explicada. El tercero será proporción de variación residual. La proporción de variación explicada, también recibe el nombre de coeficiente de determinación, simbolizado por R^2. La proporción de variación residual también recibe el nombre de coeficiente de alienación o de no determinación. Así pues:

Coeficiente de determinación = R^2

Coeficiente de alienación = $1- R^2$

La representación gráfica que corresponde a estos coeficientes será la que consta en la figura 6.13.

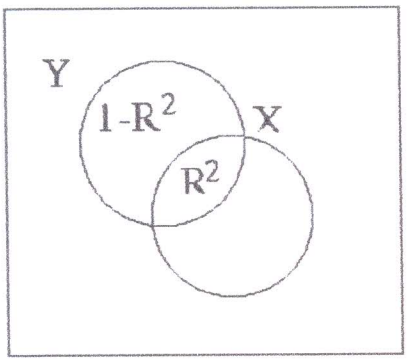


Figura 6. 13. Significado gráfico de los coeficientes de determinación y alienación.

El coeficiente de determinación tiene un papel fundamental en el contexto de la regresión lineal. Registra la proporción de variación explicada por los regresores, lo que nos permitirá conocer un poco más cada uno de los regresores a medida que el número de éstos se va incrementando en el mismo proceso de explicación.

Podemos con estos conocimientos entrever que un objetivo primordial en la regresión es aumentar en lo posible R^2, sin complicar excesivamente el modelo; es decir, sin incluir demasiados regresores.

Con frecuencia, se denomina también a R^2 *bondad de ajuste*. A medida que su valor sea mayor, estaremos más satisfechos del ajuste realizado, ya que la variación explicada es también mayor, por lo que el desajuste o error provocado por la sustitución de los valores observados por los predichos es menor.

Por último, el coeficiente de determinación coincide con el cuadrado del coeficiente de correlación lineal de Pearson. De ahí el símbolo de R^2. En regresión lineal simple, R^2 representa tanto la proporción de Y explicada por X como la proporción de X explicada por Y.

En regresión múltiple, esta conclusión no es igualmente válida. Dado que la proporción se calcula con respecto a una de las variables (la dependiente), sólo cabe interpretar R^2 como proporción de variación del criterio explicada por el conjunto de los regresores.

6.4.4. Regresión Lineal en Hoelest

Por medio del menú *Estadísticas, Regresión,Lineal* se accede a una pantalla que permite especificar la variable dependiente y las variables independientes del modelo a analizar. La figura 6.14 ilustra la ventana *Regresión* de Hoelest.

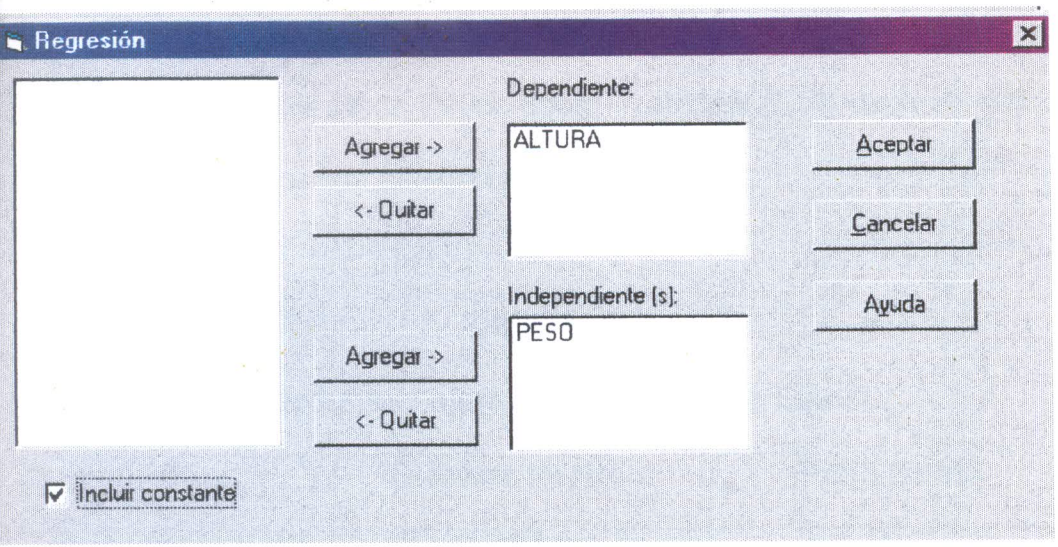


Figura 6. 14. Ventana para el cálculo de Correlación Lineal

En la parte izquierda de la ventana *Regresión* se listan todas las variables definidas en la hoja de cálculo; por medio del botón Agregar (superior), se añaden a una segunda lista la variable dependiente y con el botón Agregar (inferior) las independientes. Si se desea incluir una constante en el modelo de regresión, debe constar una flecha en la casilla de verificación *Incluir constante*.

Cuando se presiona el botón Aceptar, Hoelest realiza el análisis del modelo y presenta la salida en la que constan: el coeficiente de determinación, el error estándar, la tabla de regresión y una tabla de análisis de varianza.

Ejemplo:

Se anotaron las estaturas y los pesos de de 8 personas en un sala. La Tabla X resume los datos observados. Trataremos de explicar cuanto influye la estatura de una persona en su peso.

Tabla X

Datos para el ejemplo de Regresión Lineal

Individuo	Estatura (metros)	Peso (kilos)
1	183	96
2	162	52
3	175	75
4	173	65
5	168	70
6	169	69
7	176	74
8	162	49

Se definen las variables ESTATURA y PESO en Hoelest. Accedemos a la ventana Regresión Lineal de Hoelest, por medio de Estadísticas,Regresión,Lineal y agregamos la variable ESTATURA al grupo de Variables independientes, luego escogemos la variable PESO como dependiente. Incluiremos una constante en el modelo, para lo cual revisamos si *Incluir Constante* está chequeada, si no lo está, la

chequeamos. Presionamos el botón Aceptar y obtendremos la siguiente salida:

Análisis de Regresión Lineal

Var Dep: ESTATURA

N: 8

Múltiple R:	0,942340972445947
R Cuadrado:	0,888006508350372
Error Estándar:	2,60658221303939

La primera linea indica el nombre de la variable dependiente, en este caso ALTURA; abajo se contabiliza el número de observaciones con las que se realizó el cálculo. El coeficiente de correlación lineal múltiple (Múltiple R) vale 0,9423. Dado que sólo existe un regresor se trata del coeficiente de correlación lineal simple de Pearson. El valor nos indica que la relación lineal entre ambas variables es muy alta y positiva. Al elevar este coeficiente al cuadrado, obteniendo el coeficiente de determinación R^2 (R Cuadrado) con el valor de 0,888, la variable PESO explica el 89% de la variación observada en la ESTATURA. Por último, se indica el error estándar. Es la raíz cuadrada de la varianza residual. La varianza residual es un valor promedio de la varianza no explicada; más concretamente, es la suma de cuadrados no explicada dividida por el número de casos

menos el número de variables implicadas (regresores más el criterio). Cuanto mayor sea el error estándar, mayor será la distancia entre el modelo y lo observado. Así pues, lo ideal sería un error estándar de cero, que sólo ocurrirá cuando $R^2=1$; es decir, cuando toda la variación sea explicada.

Análisis de Varianza

Fuente	Suma Cuadrados	GL	Medias Cuadrados	F	Prob
Regresión	323,2344	1	323,234	47,5745	0,0005
Error	40,7656	6	6,794		

Lo que sigue es una tabla de análisis de varianza típico. Podemos ver dos fuentes de variación: explicada por la ecuación (Regresión) y no explicada (Error). Se calculan las sumas de cuadrados para ambas, generando la columna *Suma Cuadrados* . Al dividir la suma de cuadrados por los grados de libertad (GL), obtenemos las medias cuadráticas (Medias Cuadrados) o medidas de varianza, que expresan cómo varían los datos (según la fuente de variación) por término medio. Nótese que, en efecto, la raíz cuadrada de la varianza residual es el error estándar ($\sqrt{6,835}=2,614$). La varianza explicada es mayor que la residual; un procedimiento consiste en dividir la explicada para la no explicada

obteniendo el índice F, que en este caso vale 47,25. (la varianza explicada es casi 50 veces superior a la no explicada). La hipótesis nula en este caso establece que F es mayor que 1 por casualidad. Vemos que la probabilidad de que sea así es insignificante (0,0005), por lo que podemos rechazar tal hipótesis y afirmar que el modelo explica más de lo que deja sin explicar, o decimos que el modelo es válido.

Variables en la Ecuación

Efecto	Coeficiente	Std Error	T	P 2c
Constante	139,0501	4,7246	29,4305	0,0005
PESO	0,4647	0,0674	6,8947	0,0001

La tabla *Variables en la Ecuación* contiene los resultados que más interesan al investigador. La columna Coeficiente se refiere a los coeficientes de la ecuación en puntuaciones directas, lo que nos permite construir la expresión:

Estatura = 139,0501+ 0,4647 * Peso

La columna Std Error contiene los errores estándar para cada coeficiente. Cuanto mayor sea el error estándar, mayor variación de posibles valores cabe esperar de ese coeficiente de cara a obtener los verdaderos índices

en la población. Como los errores son relativos al tamaño del coeficiente se calcula un índice relativo T. La T es un estadístico que muestra lo diferente que es el coeficiente de cero. Si es estadísticamente diferente, el coeficiente sirve para algo (el regresor que lo posee explica suficiente del criterio). Si no resulta estadísticamente diferente de cero, es que el coeficiente no es nulo por casualidad. Se observa que ambos coeficientes son muy significativos (0,0005 para el peso y 0,0001 para la constante).

6.5. Diseño de Experimentos

6.5.1. Introducción.

El diseño de experimentos es un tema muy amplio referente a los métodos de muestreo para reducir la variación o perturbación en un experimento, a fin de amplificar la señal de la naturaleza y adquirir así una cantidad especificada de información a un costo mínimo.

Los objetos sobre los cuales se hacen mediciones se denominan unidades experimentales. Si un experimentador somete a un conjunto de n=10 ratas a un estímulo y mide la respuesta de cada una, una rata en particular es una unidad experimental, el grupo de n=10 mediciones es una muestra.

Consideremos que deseamos estudiar el efecto de diferentes "tipos de conexión de redes computacionales" y "tipos de procesadores de computadores personales " en la rapidez de un sistema de ventanillas de una institución bancaria. Cada combinación de tipos de redes y tipos de computadores representa un tratamiento diferente.

Estas variables son controlables por el experimentador, y se denominan variables independientes. Como se estudia la rapidez de comunicación, y ésta es una función de las variables independientes, se le denomina variable dependiente.

Las variables experimentales independientes se denominan factores. Dichos factores pueden ser cuantitativos o cualitativos. En el primer caso, el factor puede tomar valores correspondientes a los puntos de una recta real. Los factores que no son cuantitativos se denominan cualitativos.

Un ejemplo común es el estudio del efecto de la temperatura de horneado sobre la densidad de un tipo específico de pastel. Si se elaboran 3 pasteles a una temperatura de 350, 400 y 450 grados, se tendrán diversos resultados al medir la densidad de cada uno. Las tres temperaturas, representan tres niveles del factor cuantitativo "temperatura

de horneado". Se tiene entonces que al grado de intensidad de un factor se le denomina nivel.

En el ejemplo sobre la rapidez del sistema de ventanillas de una institución bancaria, el factor cualitativo "tipos de conexión de redes computacionales" podría tener dos niveles a y b. Del mismo modo el factor cualitativo "tipos de procesadores de computadores personales" podría tener 4 niveles: m,n,o,p. Lo que se hace con las unidades experimentales y que las vuelve diferentes de una población a otra, se llama tratamiento. Un tratamiento es una combinación específica de niveles de un factor o de factores.

El experimento puede involucrar un solo factor, como la temperatura en el ejemplo del pastel. O podría estar compuesto de combinaciones de niveles de dos o más factores, como en el experimento de rapidez de un sistema de ventanillas, donde cada combinación representaría un tratamiento.

El diseño experimental implica un problema final. Después de seleccionar las combinaciones de los factores (tratamientos) a utilizar en el experimento, debe decidirse como asignar los tratamientos a las unidades experimentales. ¿Habría que asignar aleatoriamente los tratamientos a

las unidades experimentales o debería de utilizarse un modelo semialeatorio? Por ejemplo, ¿habría que asignar aleatoriamente los neumáticos de dos fabricantes A y B a todas las ruedas de un automóvil, o asignar un neumático de cada tipo A y B a las ruedas traseras de cada auto?.

6.5.2. ¿Qué es el Diseño Experimental?

Un experimento diseñado es una prueba o serie de pruebas en las cuales se produce un cambio deliberado en las variables de entrada (independientes) de un proceso o sistema, de manera que sea posible observar e identificar las causas de los cambios en la respuesta de salida. Entre los objetivos del experimento están:

a) Determinar cuales variables o factores tienen mayor influencia en la respuesta o variables de interés.

b) Determinar la mejor combinación de los factores que influyen en la respuesta, de modo que ésta tenga casi siempre un valor cercano al valor nominal deseado.

c) Determinar la mejor combinación de los factores que influyen en la respuesta, de modo que su variablidad sea pequeña.

d) Determinar la mejor combinación de los factores que influyen en la respuesta, de modo que se minimicen los efectos de variables incontrolables.

6.5.3. Procedimiento del Diseño de un Experimento

Primero se seleccionan los factores que deben incluirse en el experimento y se especifica el (los) parámetros poblacionales de interés. Segundo, se decide cúanta información conviene utilizar acerca de el (los) parámetros de interés. Como tercer paso se seleccionan los tratamientos que deben utilizarse en el experimento y se decide el número de unidades experimentales que deben asignarse a cada uno. Por último, se decide cómo deben aplicarse los tratamientos a las unidades experimentales.

6.5.4. Diseño de Experimentos en Hoelest

Por medio del menú *Estadísticas,Diseño de Experimentos...* se accede a una pantalla que permite escoger el tipo de diseño deseado. La figura 6.15 ilustra la ventana *Diseño de Experimentos* de Hoelest. La ventana contiene un cuadro de lista *Tipo de Diseño* de donde se puede escoger tres tipos: Factorial, Bloques y Latinos. Constan también los cuadros de texto para especificar el número de niveles, el número de factores y el número de réplicas.

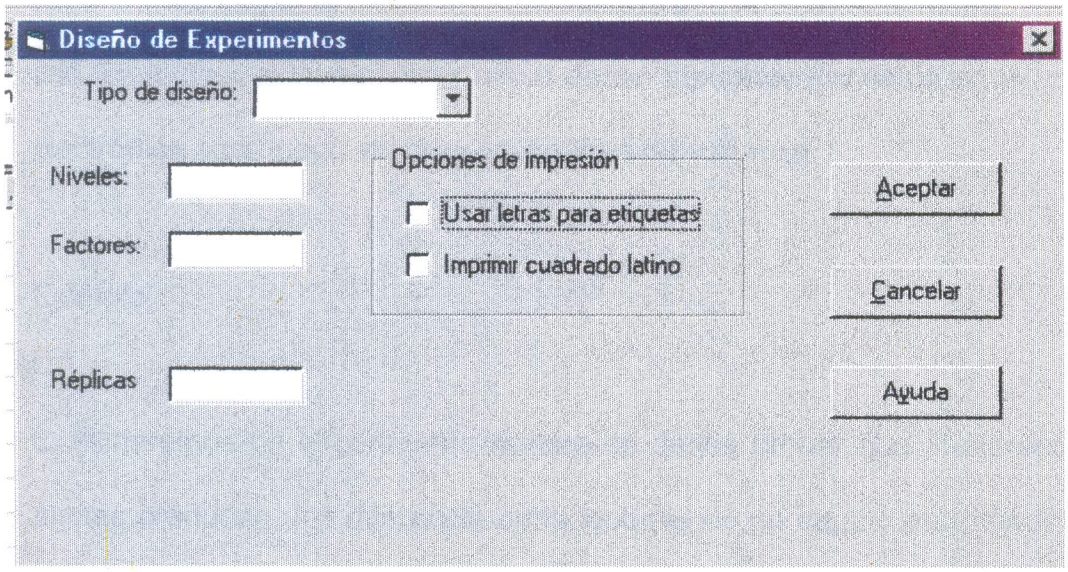


Figura 6. 15. Ventana para generar Matrices de Diseño.

En el recuadro *Opciones de impresión*, se puede escoger *Usar letras para etiquetas* e *Imprimir cuadrado latino*, éste último sólo disponible cuando Tipo de diseño es Cuadrado Latino. El procedimiento asigna un número a cada tratamiento de cada factor, por ejemplo, si hay dos factores y el número de niveles es 2 entonces los elementos de la matriz de diseño serán números entre 1 y 2. Usar letras para etiquetas modifica este comportamiento, ya que en vez de asignar un número a cada nivel, le asigna un letra, en este caso a y b por haber dos niveles.

Cuando se presiona el botón Aceptar, Hoelest genera una matriz de diseño del modelo especificado. Note que no se realiza ningún análisis. La matriz de diseño resultado sirve para que el experimentador realice su

experimento según el orden de tratamientos que le indique la matriz. Otro uso, es indicar al investigador cómo deben inputarse los datos en la hoja electrónica para luego efectuar un análisis de varianza.

Ejemplo:

Consideremos un experimento en que se desee probar que 3 diferentes puntas producen una diferencia en la lecturas de un equipo para medir la dureza. La máquina funciona presionando la punta sobre una probeta de metal y determinando la dureza de la probeta a partir de la profundidad de la marca que se produce.

El experimentador ha decidido obtener 3 observaciones para cada punta. Sólo existe un factor –tipo de punta-, y el diseño de un factor completamente aleatorizado consiste en asignar aleatoriamente cada uno de los 3 x 3 = 9 ensayos a una unidad experimental, o sea, una probeta de metal, y tomar las lecturas de la dureza correspondientes.

Un diseño por bloques aleatorizado, ayudaría a reducir el error experimental. La Tabla XI contiene los datos de la dureza observada en la escala C de Rockwell menos 40.

Tabla XI

Datos para el experimento de prueba de dureza

	Probeta (Bloque)		
Tipo Punta	1	2	3
1	9.3	9.4	9.6
2	9.4	9.3	9.8
3	9.2	9.6	9.5

Accedemos entonces, a la ventana *Diseño de Experimentos* de Hoelest, por medio de Estadísticas,DiseñoDeExperimentos. Escogemos tipo de diseño: bloques y digitamos el número 3 en Niveles y en Réplicas digitamos 3, puesto que se tomarán 3 observaciones por nivel. Note que Factores toma el valor de 1. Presionamos el botón aceptar y se generará la Matriz de Diseño. La primera parte de la salida presenta información del diseño escogido:

Matriz de Diseño

9 corridas

2 Factores

cada uno con tres niveles

La segunda parte de la salida es la matriz de diseño. En la primera columna se listan el número de corridas y las demás columnas son los factores. Las filas de los factores son todas las combinaciones posibles

entre los niveles de todos los factores. En este caso, la matriz de diseño es:

Corridas	Factor 1	Factor 2
1	1	1
2	1	2
3	1	3
4	2	1
5	2	2
6	2	3
7	3	1
8	3	2
9	3	3

Note que en este caso, estamos generando la matriz para conocer como deben de imputarse los datos en una hoja electrónica para efectuar un análisis de varianza.

Si no contáramos con los datos observados, podríamos haber hecho el mismo procedimiento, y contar con la matriz para efectuar el muestreo del experimento y obtener las mediciones.

6.6. Análisis de Varianza

6.6.1. Introducción

La mayoría de los experimentos consisten en el estudio de los efectos de una o más variables independientes sobre una respuesta Y. Estas variables independientes, que pueden controlarse en el experimento se denominan factores. El objetivo del análisis de varianza es identificar variables independientes importantes en un estudio y determinar cómo interactúan y afectan a la respuesta.

6.6.2. Procedimiento de Análisis de Varianza

El procedimiento de análisis de varianza trata de analizar la variación de una respuesta, y de asignar porciones (componentes) de esta variación a cada una de las variables de un conjunto de variables independientes.

El razonamiento se basa en que las variables de respuestas se modifican por la variación de algún conjunto de variables independientes desconocidas. Es muy poco probable que se incluyan todas las variables que afectan a la respuesta en un experimento, por lo que es posible observar una variación aleatoria en la respuesta.

La variabilidad de un conjunto de n mediciones es proporcional a la suma de los cuadrados de las desviaciones, cantidad que se utiliza para

calcular la varianza de la muestra. El análisis de varianza divide la suma de los cuadrados de las desviaciones llamada "suma total de los cuadrados de las desviaciones" (SCTotal), en partes, cada una de las cúales se atribuye a una de las variables independientes en el experimento, más un residuo que se asocia al error aleatorio. Esto puede observarse en el diagrama de la figura 6.16 para 3 variables independientes.

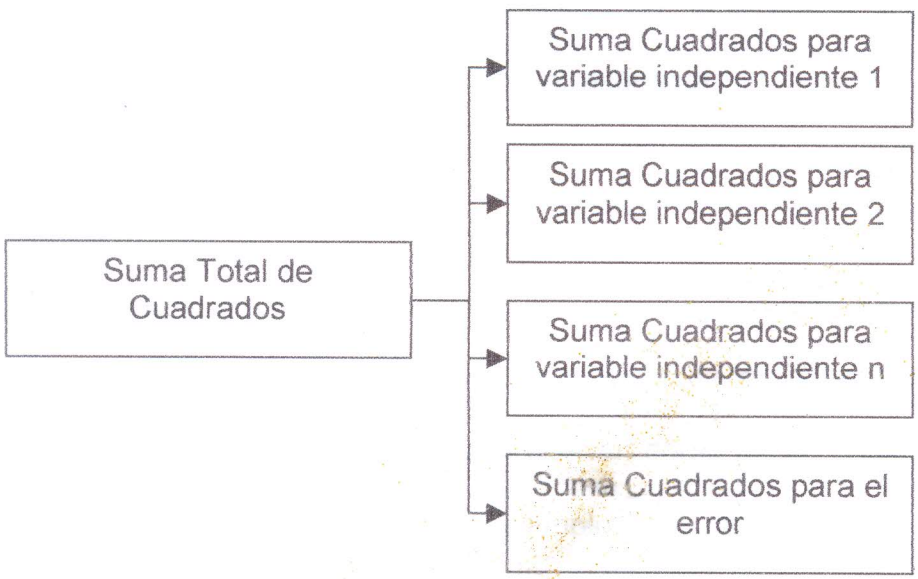


Figura 6.16. Porción de la Suma Total de Cuadrados de las desviaciones

La porción de la suma total de los cuadrados de las desviaciones asignadas al error se designa como SCE. De acuerdo con la hipótesis de que la respuesta no está relacionada con las variables independientes, se puede demostrar que cada uno de los componenentes de SCTotal,

dividida entre una constante apropiada, proporciona un estimador independiente e insesgado para la varianza del error experimental, σ^2. Cuando una variable está muy relacionada con la respuesta, su porción (llamada "suma de los cuadrados" para esa variable) estará exagerada. Se puede detectar esta condición comparando la estimación de σ^2 de una variable independiente en particular, con la estimación obtenida a partir de SCE aplicando una prueba F. Si la estimación para la variable independiente es significativamente mayor, la prueba F rechazará la hipótesis de que la variable independiente no tiene efecto y generará evidencia que indique una relación con la respuesta.

6.6.3. Experimentos con un solo factor

Supóngase que se desea comparar ***a*** tratamientos o niveles de un factor único. La respuesta que se observa en cada uno de los *a* tratamientos es una variable aleatoria. Los datos aparecerían como en la Tabla XII. Una entrada de la citada tabla, por ejemplo Y_{ij}, representa la j-ésima observación del tratamiento i. En general, habrá *n* observaciones del tratramiento i. Es útil describir las observaciones mediante el modelo estadístico lineal

$$Y_{ij} = \mu + \tau_i + \varepsilon_{ij} \begin{cases} i = 1,2,...,a \\ j = 1,2,....n \end{cases}$$

En donde Y_{ij} es la (ij)-ésima observación, μ es un parámetro común a todos los tratamientos denominado media global, τ_i es un parmámetro único para el i-ésimo tratamiento llamado efecto del tratamiento i-ésimo, y ε_{ij} es la componente aleatoria del error. El objetivo es probar hipótesis apropiadas respecto a los efectos del tratamiento y hacer una estimación de ellos. Para probar la hipótesis se supone que los errores del modelo son variables aleatorias independientes con distribución normal, con media cero y varianza σ^2. Se supone que la σ^2 es constante para todos los niveles del factor.

Tabla XII

Datos típicos para un experimento unifactorial

Tratamiento		Observaciones		
1	Y11	Y12	...	Y1n
2	Y21	Y22	...	Y2n
.	-..	..	...	...
a	Ya1	Ya2	...	Yan

Este modelo se denomina análisis de varianza clasificación en un sentido porque sólo se investiga un factor. Además se requiere que el experimento se realice en orden aleatorio, de manera que el medio ambiente en el que se usan los tratamientos (unidades experimentales) sea lo más uniforme posible. Por tanto, este diseño experimental es un diseño completamente aleatorizado.

Si los *a* tratamientos fueron seleccionados específicamente por el experimentador, las conclusiones sólo se aplican a los niveles estudiados. En este caso el modelo se denomina modelo de efectos fijos. Si en cambio, los a tratamientos constituyen una muestra aleatoria de una población mayor de tratamientos, el modelo recibe el nombre de modelo de efectos aleatorios o de componentes de varianza. Esta sección analizará solamente el modelo de efectos fijos.

6.6.3.1. Análisis Estadístico

La hipótesis nula que se investiga es la igualdad de medias de los tratamientos.

$H_o : \mu_1 = \mu_2 = ... = \mu_a$

Denotemos el número de tratamientos del factor con la letra *a*, y el número de observaciones con la letra N. Este modelo divide la suma de cuadrados de las desviaciones (SCTotal) en dos componentes. El primero que se denomina Suma de cuadrados de tratamientos (SCT) sirve para calcular una medida de la variación entre las medias muestrales de los tratamientos. El segundo componente, llamado Suma de Cuadrados del error (SCE), se usa para medir la variación dentro de las muestras. Entonces:

SCTotal = SCT + SCE

Si se divide SCT para (a–1), se obtiene la Media de Cuadrados de Tratamientos (MCT). De la misma forma, si SCE se divide para (N-a) se obtiene la Media de Cuadrados del Error (MCE). La MCT sigue una distribución Ji Cuadrado con (a-1) grados de libertad y MCE también sigue una distribución Ji Cuadrado con (N-a) grados de libertad. Con ayuda del Teorema de Cochran se construye un estadístico F_o para probar la hipótesis de la igualdad de medias de los tratamientos (11). El estadístico F_o es:

$$Fo = \frac{SCT/(a-1)}{SCE/(N-a)} = \frac{MCT}{MCE}$$

y sigue una distribución F de Fisher con (a-1) y (N-a) grados de libertad en el numerador y en el denominador respectivamente.

La MCE es un estimador insesgado de la varianza del error experimental σ^2. Si la hipótesis nula es verdadera, MCT resulta ser un estimador insesgado de σ^2. Sin embargo, si la hipótesis nula es falsa, el valor esperado de MCT es mayor que σ^2. Por tanto, el valor esperado del numerador en la estadística de prueba es mayor que el valor esperado del

denominador si la hipótesis alterna es verdadera y, en consecuencia debe rechazarse H_o si el valor de tal estadística es demasiado grande. Esto implica una región crítica unilateral superior. En otras palabras, se rechaza H_o si:

$F_o > F_{\alpha,a-1,N-a}$

El procedimiento de prueba se resume en la Tabla XIII

Tabla XIII

Tabla de análisis de varianza de un modelo unifactorial

Fuente de Variación	Suma Cuadrados	GL	Medias Cuadrados	Fo
Entre tratamientos	SCT	a-1	SCT / (a-1)	MCT / MCE
Error	SCE	N-a	SCE / (N-a)	
Total	SCTotal	N-1		

6.6.3.2. ANOVA de un solo factor en Hoelest

Por medio del menú Estadísticas,ANOVA se accede a la ventana ANOVA de Hoelest (figura 6.17). En dicha ventana se puede definir un modelo lineal, relacionando una variable dependiente con varias independientes.

Los botones Agregar y Quitar superiores e inferiores, ayudan al usuario a crear el modelo deseado.

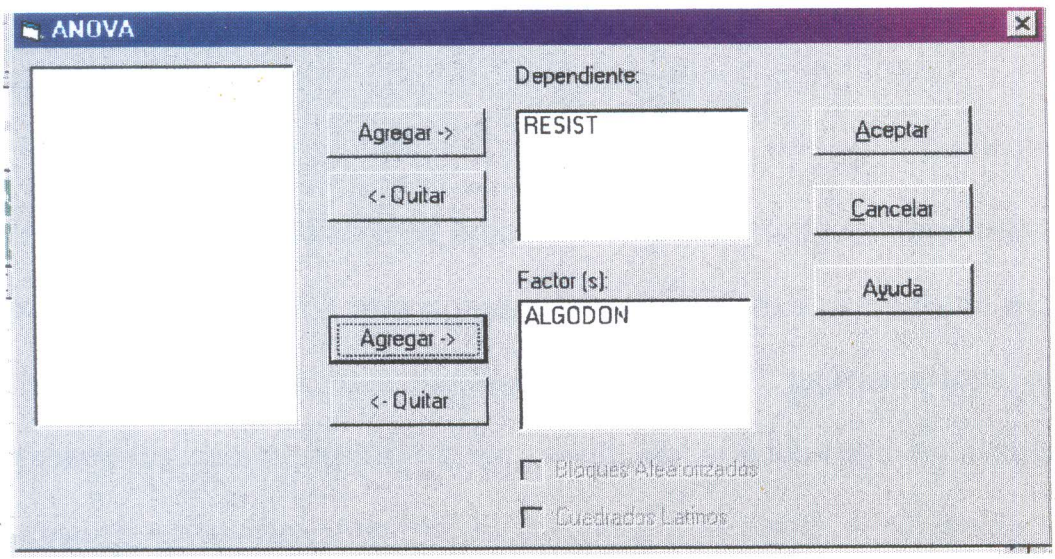


Figura 6. 17. Ventana ANOVA de Hoelest

Cuando en la lista *Factores,* que son las variables independientes, hay dos elementos, se habilita la casilla de verficación *Bloques aleatorizados*, para permitir al usuario realizar un análisis de este modelo, considerando que el primer factor es en realidad el factor de interés y el segundo factor representa a los bloques.

Cuando en la lista *Factores,* hay tres variables o elementos, se habilita la casilla de verficación *Cuadrados Latinos*, para permitir al usuario realizar un análisis de este modelo, considerando que el primer factor es en

realidad el factor de interés, el segundo factor representa a los renglones y el tercer factor representa a las columnas. Se accede a esta pantalla para realizar cualquier análisis de varianza descrito en este capítulo.

Ejemplo:

Un ingeniero de desarrollo de productos está interesado en maximizar la resistencia a la tensión de una nueva fibra sintética que se empleará en la manufactura de tela para camisas de hombre. El ingeniero sabe por experiencia que la resistencia es influida por el porcentaje de algodón presente en la fibra. Además, el sospecha que elevar el contenido de algodón incrementará la resistencia, al menos inicialmente. También sabe que el contenido de algodón debe variar aproximadamene entre 10 y 40% para que la tela resultante tenga otras características de calidad que se desean.

El ingeniero decide probar muestras a cinco niveles de porcentaje de algodón: 15, 20, 25, 30 y 35 %. Así mismo, decide ensayar cinco muestras a cada nivel de contenido de algodón. En este experimento hay a=5 niveles y n=5 repeticiones, por lo que deben hacerse 25 observaciones. La Tabla XIV tabula los valores observados a cada nivel.

Tabla XIV

Datos del experimento de Resistencia a la tensión

	Resistencia a la tensión observada				
% algodón	1	2	3	4	5
15	7	7	15	11	9
20	12	17	12	18	18
25	14	18	18	19	19
30	19	25	22	19	23
35	7	10	11	15	11

Se definen dos variables: RESIST y ALGODÓN que representan los datos observados de la resistencia y los porcentajes de algodón (relacionados con las observaciones). Para saber cómo se deben ingresar dichos datos en la hoja electrónica se puede generar una matriz de diseño, de acuerdo a este modelo. Accedemos a la Ventana ANOVA por medio del menú Estadísticas,ANOVA. Agregamos la variable RESIST al cuadro Variable Dependiente y la variable ALGODÓN a la lista de Factores (variables independientes). Hoelest presenta la siguiente salida:

Factor: ALGODON

Niveles(5): 15, 20, 25, 30, 35,

Var Dep: OBSERV

R: 0,86424700386959

R Cuadrada: 0,746922883697564

Análisis de Varianza

Fuente	Suma Cuadrados	GL	Medias Cuadrados	F	Prob
RESIST	475,76	4	118,94	14,7568	0,0000
Error	161,2	20	8,06		
Total	636,96	24			

La primera linea indica el nombre del factor (ALGODÓN). La segunda línea detalla los niveles de dicho factor. Luego se informa cual es la variable dependiente junto con los coeficientes de correlación y de determinación. Así pues, la variable o factor ALGODÓN explica el 74% de la variación observada de RESIST.

Lo que sigue es una tabla de análisis de varianza típico. Podemos ver dos fuentes de variación: explicada por RESIST y no explicada (Error). Se calculan las sumas de cuadrados para ambas, generando la columna *Suma Cuadrados* . Al dividir la suma de cuadrados por los grados de libertad (GL), obtenemos las medias cuadráticas (Medias Cuadrados) o medidas de varianza, que expresan cómo varían los datos (según la fuente de variación) por término medio. La varianza explicada o la media de cuadrados entre tratamientos (118,94) es mucho mayor que la residual

(8,06). Esto indica que es improbable que las medias de los tratamientos sean iguales.

Calculando la razón F = 118,94 / 8,06 = 14.76 y comparándola con el estadístico F α,4,20=4.43, la hipótesis nula debe rechazarse y concluir que las medias de tratamientos difieren; es decir, el porcentaje de algodón en la fibra afecta significativamente sus resistencia media. EL valor en Prob es el nivel de significancia alcanzado por la prueba.

6.6.4. Bloques Aleatorizados

En muchos problemas de investigación es necesario diseñar experimentos en los que pueda controlarse sistemáticamente la variabilidad producida por diversas fuentes extrañas. Se busca que el error experimental sea lo más pequeño posible, en otras palabras, se busca sustraer del error experimental la variabilidad producida por los bloques.

Si se utiliza este diseño, los bloques forman una unidad experimental más homogénea con la cual compara los tratamientos, lo que mejora efectivamente la precisión en las comparaciones al eliminar la variabilidad entre los bloques. Este diseño es quizás el diseño experimental más ampliamente utilizado. Los datos aparecerían tabulados como en la Tabla

XII, teniendo en cuenta que cada columna de la tabla representa ahora un bloque.

Supóngase que en general se tienen *a* tratamientos (que deben ser comparados) y *b* bloques. Se realiza una observación por tratamiento en cada bloque, y el orden en que los tratamientos son medidos en cada bloque se determina aleatoriamente. El modelo estadístico para este diseño es:

$$Y_{ij} = \mu + \tau_i + \beta_j + \varepsilon_{ij} \begin{cases} i = 1,2,\ldots,a \\ j = 1,2,\ldots b \end{cases}$$

En donde Y_{ij} es la (ij)-ésima observación, μ es una media general, τ_i es el efecto del i-ésimo tratamiento, β_j es el efecto del j-ésimo bloque y ε_{ij} es la componente aleatoria del error, distribuida normalmente con media cero y varianza σ^2.

Se supone que la σ^2 es constante para todos los niveles del factor. Inicialmente se considera que tanto los tratamientos como los bloques son factores fijos. Más aún, los efectos de los tratamientos y de bloques se consideran como desviaciones de la media general. Este modelo se denomina modelo aleatorizado por bloques completos.

6.6.4.1. Análisis Estadístico

La hipótesis nula que se investiga es la igualdad de medias de los tratamientos.

$H_o : \mu_1 = \mu_2 = ... = \mu_a$

Versus la hipótesis alternativa que indica que existe al menos una media que difiere de otra

$H_a: \mu_i \neq \mu_j$

Denotemos el número de tratamientos del factor con la letra *a*, el número de bloques con la letra *b* y el número de observaciones con la letra N. Este modelo divide la suma de cuadrados de las desviaciones (SCTotal) en tres componentes:

El primero que se denomina Suma de cuadrados de tratamientos (SCT) sirve para calcular una medida de la variación entre las medias muestrales de los tratamientos. El segundo componente, llamado Suma de cuadrados de bloques (SCB) sirve para calcular una medida de la variación entre las medias de los bloques. El tercer componente, llamado Suma de Cuadrados del error (SCE), se usa para medir la variación de las

diferencias entre las observaciones de los tratamientos dentro de los bloques (que mide el error experimental). Entonces:

SCTotal = SCT + SCB + SCE

Si se divide SCT para (a–1), se obtiene la Media de Cuadrados de Tratamientos (MCT). De la misma forma, si SCB se divide para (b-1) se tiene la Media de Cuadrados de Bloques (MCB). Por último, si SCE se divide para (a-1)(b-1) se obtiene la Media de Cuadrados del Error (MCE). La MCT, MCB y MCE siguen una distribución Ji Cuadrado con (a-1), (b-1) y (a-1)(b-1) grados de libertad respectivamente. Para probar la igualdad en las medias de tratamiento, hay que usar la estadística:

$$Fo = \frac{SCT/(a-1)}{SCE/(a-1)(b-1)} = \frac{MCT}{MCE}$$

que sigue una distribución F de Fisher con (a-1) y (a-1)(b-1) grados de libertad en el numerador y en el denominador respectivamente. Para probar la igualdad en las medias de bloque, hay que usar la estadística

$$Fo = \frac{SCB/(b-1)}{SCE/(a-1)(b-1)} = \frac{MCB}{MCE}$$

que sigue una distribución F de Fisher con (b-1) y (a-1)(b-1) grados de libertad en el numerador y en el denominador respectivamente.

La MCE es un estimador insesgado de la varianza del error experimental σ^2. Si la hipótesis nula es verdadera, MCT resulta ser un estimador insesgado de σ^2. Sin embargo, si la hipótesis nula es falsa, el valor esperado de MCT es mayor que σ^2. Por tanto, el valor esperado del numerador en la estadística de prueba es mayor que el valor esperado del denominador si la hipótesis alterna es verdadera y, en consecuencia debe rechazarse H_o si el valor de tal estadística es demasiado grande. Esto implica una región crítica unilateral superior. En otras palabras, se rechaza H_o si:

$$F_o > F_{\alpha, a-1, (a-1)(b-1)}$$

De forma análoga, para probar la igualdad en las medias de bloques se utiliza el estadístico F_o calculado para los bloques. Se rechaza la hipótesis de igual de medias de bloques si:

$$F_o > F_{\alpha, b-1, (a-1)(b-1)}$$

El procedimiento de prueba se resume en la Tabla XV.

Tabla XV

Tabla de Análisis de Varianza para un diseño aleatorizado por bloques completos.

Fuente de Variación	Suma Cuadrados	GL	Medias Cuadrados	Fo
Tratamientos	SCT	a-1	SCT / (a-1)	MCT / MCE
Bloques	SCB	b-1	SCB / (b-1)	MCB / MCE
Error	SCE	(a-1)(b-1)	SCE / (a-1)(b-1)	
Total	SCTotal	N-1		

6.6.4.2. ANOVA de un Diseño de Bloques Aleatorizado en Hoelest

Para efectuar el ANOVA se accede a la pantalla ANOVA como se explicó en la sección 6.6.3.2.

Ejemplo:

Supongamos que se desea determinar si cuatro diferentes puntas producen una diferencia en las lecturas de un equipo para medir la dureza. La máquina funciona presionando la punta sobre una probeta de metal y determinando la dureza de la probeta a partir de la profundidad de la marca que se produce. El experimentador ha decidido obtener cuatro observaciones para cada punta.

Sólo existe un factor –tipo de punta-, y el diseño de un factor completamente aleatorizado consiste en asignar aleatoriamente cada uno de los 4 x 4 = 16 ensayos a una unidad experimental, o sea, una probeta de metal, y tomar las lecturas de la dureza correspondientes. Por lo tanto, se requerirían 16 probetas de metal para realizar este experimento, una para cada ensayo. La Tabla XVI tabula los valores de la dureza medidos en la escada C de Rockell menos 40.

Tabla XVI

Datos del experimento de Prueba de Dureza

	Tipo de punta			
Probeta	**1**	**2**	**3**	**4**
1	9.3	9.4	9.6	10
2	9.4	9.3	9.8	9.9
3	9.2	9.4	9.5	9.7
4	9.7	9.6	10	10.2

Se definen tres variables: DUREZA, PUNTA y PROBETA que representan los datos observados de la dureza, los tipos de punta y las probetas. Para saber cómo se deben ingresar dichos datos en la hoja electrónica se puede generar una matriz de diseño, de acuerdo a este modelo. Accedemos a la Ventana ANOVA por medio del menú Estadísticas,ANOVA. Agregamos la variable DUREZA al cuadro Variable Dependiente y las variables PUNTA y PROBETA (en ese orden) a la lista de Factores. Al presionar el botón Aceptar se realiza el análisis y Hoelest

presenta la siguiente salida (la interpretación de la primera parte es análoga a la realizada en experimentos de un solo factor).

Factor: PUNTA

Niveles(4): 1, 2, 3, 4,

Var Dep: DUREZA

R: 0,968495996958186

RCuadrada: 0,937984496124031

Análisis de Varianza

Fuente	Suma Cuadrados	GL	Medias Cuadrados	F	Prob
PUNTA	38,5	3	12,83333	14,4375	0,0008
Bloques	82,5	3	27,5	30,9375	0,0000
Error	8	9	0,8888889		
Total	129	15			

El coeficiente de determinación es alto, podemos decir que el modelo explica el 94% de la variación observada de DUREZA. En la tabla de análisis de varianza, podemos ver tres fuentes de variación: explicada por PUNTA, por BLOQUES y no explicada (Error). Se calculan las sumas de cuadrados para todas, generando la columna *Suma Cuadrados*. Al dividir la suma de cuadrados por los grados de libertad (GL), obtenemos las

medias cuadráticas (Medias Cuadrados) o medidas de varianza, que expresan cómo varían los datos (según la fuente de variación) por término medio. La varianza explicada o la media de cuadrados entre tratamientos (12.83) es casi 15 veces mayor que la residual (0,89). Si α=0,05, el valor crítico de F es $F_{0.05,3,9}$ = 3.86. A partir de que 14.44 > 3.86, se concluye que el tipo de punta afecta la lectura del nivel medio de dureza. También puede haber diferencia significativa entre las probetas (bloques) porque la media de cuadrados de bloques es relativamente grande comparada con la del error, inclusive es mayor que de los tratamientos (30,93).

6.6.5. Cuadrados Latinos

El diseño de cuadrados latinos se utiliza para eliminar dos fuentes de variabilidad problemáticas; en otras palabras, permite analizar sistemáticamente por bloques en dos direcciones. En este diseño, los bloques y las columnas, representan en realidad dos restricciones a la aleatorización.

En general, un cuadrado latino para *p* factores, o un cuadrado latino *p* x *p*, es un cuadrado que contiene *p* renglones y *p* columnas. Cada una de las p^2 celdas contiene una de las *p* letras que corresponde a un tratamiento, y cada letra aparece una sola vez en cada renglón y columna. A continuación un ejemplo de un cuadrado latino 4 x 4:

A	B	C	D
B	C	D	A
C	D	A	B
D	A	B	C

El modelo estadístico para este diseño es:

$$Y_{ijk} = \mu + \alpha_i + \tau_j + \beta_k + \varepsilon_{ijk} \begin{cases} i = 1,2,...,p \\ j = 1,2,..,p \\ k = 1,2,..,p \end{cases}$$

En donde Y_{ijk} es la observación correspondiente al i-ésimo renglón, la k-ésima columna y el j-ésimo tratamiento, μ es una media general, α_i es el i-ésimo efecto de renglón, τ_j es el j-ésimo efecto de tratamiento, β_k es el k-ésimo efecto de columna y ε_{ijk} es el error aleatorio.

6.6.5.1. Análisis Estadístico

La hipótesis nula que se investiga es la igualdad de medias de los tratamientos.

$H_o : \mu_1 = \mu_2 = ... = \mu_a$

Versus la hipótesis alternativa que indica que existe al menos una media

que difiere de otra

H_a: $\mu_i \neq \mu_j$

Denotemos el número de tratamientos del factor con la letra *p*, que a su vez es el número de renglones y columnas. El número de observaciones $N = p^2$.

Este modelo divide la suma de cuadrados de las desviaciones (SCTotal) en cuatro componentes. El primero que se denomina Suma de cuadrados de tratamientos (SCT) sirve para calcular una medida de la variación entre las medias muestrales de los tratamientos. El segundo componente, llamado Suma de cuadrados de renglones (SCR) sirve para calcular una medida de la variación entre las medias de renglones. El tercer componente, llamado Suma de cuadrados de bloques (SCB) sirve para calcular una medida de la variación entre las medias de los bloques. El cuarto componente, llamado Suma de cuadrados del error (SCE), mide el error experimental. Entonces:

SCTotal = SCT + SCR + SCB + SCE

Si se divide SCT, SCR y SCB para (p – 1), se obtiene la Media de

Cuadrados de Tratamientos (MCT), la Media de Cuadrados de Renglones (MCR) y la Media de Cuadrados de Bloques (MCB). Por último, si SCE se divide para (p-2)(p-1) se obtiene la Media de Cuadrados del Error (MCE). La MCT, MCR, MCB siguen una distribución Ji Cuadrado con (p-1) grados de libertad. La MCE sigue una distribución Ji cuadrado con (p-2)(p-1) g.l. Para probar la igualdad en las medias de tratamiento, hay que usar la estadística

$$Fo = \frac{SCT/(p-1)}{SCE/(p-2)(p-1)} = \frac{MCT}{MCE}$$

que sigue una distribución F de Fisher con (p-1) y (p-2)(p-1) grados de libertad en el numerador y en el denominador respectivamente.

Para probar la igualdad en las medias de renglones y de bloque, basta con reemplazar el numerador por la Suma Cuadrática correspondiente. La MCE es un estimador insesgado de la varianza del error experimental σ^2. Si la hipótesis nula es verdadera, MCT resulta ser un estimador insesgado de σ^2. Sin embargo, si la hipótesis nula es falsa, el valor esperado de MCT es mayor que σ^2. Por tanto, el valor esperado del numerador en la estadística de prueba es mayor que el valor esperado del denominador si la hipótesis alterna es verdadera y, en consecuencia debe rechazarse H_o

si el valor de tal estadística es demasiado grande. Esto implica una región crítica unilateral superior. En otras palabras, se rechaza H_o si:

$$F_o > F_{\alpha, p-1, (p-2)(p-1)}$$

De forma análoga, para probar la igualdad en las medias de renglones y de bloques se utiliza el estadístico F_o calculado para cada uno. Se rechaza la hipótesis de igualdad de medias si en ambos casos:

$$F_o > F_{\alpha, p-1, (p-2)(p-1)}$$

El procedimiento de prueba se resume en Tabla XVII.

Tabla XVII

Tabla de Análisis de Varianza para un diseño de Cuadrado Latino.

Fuente de Variación	Suma Cuadrados	GL	Medias Cuadrados	Fo
Tratamientos	SCT	p-1	SCT / (p-1)	MCT / MCE
Renglones	SCR	p-1	SCR / (p-1)	MCR / MCE
Bloques	SCB	p-1	SCB / (p-1)	MCB / MCE
Error	SCE	(p-2)(p-1)	SCE / (p-2)(p-1)	
Total	SCTotal	p^2 -1		

6.6.5.2. ANOVA de un Diseño de Cuadrados Latinos en Hoelest

Para efectuar el ANOVA se accede a la pantalla ANOVA como se explicó en la sección 6.6.3.2.

Ejemplo:

Supongamos que un experimentador está estudiando el efecto de cinco fórmulas diferentes de la mezcla de dinamita sobre la fuerza explosiva observada. Cada fórmula se prepara usando un lote de materia prima, lo suficientemente grande para que sólo se hagan cinco mezclas. Más aún, las mezclas las preparan cinco operadores, pudiendo existir una diferencia sustancial en la habilidad y experiencia de ellos.

Al parecer hay dos efectos extraños que se deben cancelar en el diseño: lotes de materia prima y operadores.

El diseño apropiado consiste en probar cada fórmula exactamente un vez, utilizando cada lote de materia prima, y en que cada fórmula sea preparada exactamente una vez por cada uno de los cinco operadores.

El diseño que resulta aparece en la tabla XVIII.

Tabla XVIII

Datos del experimento de la fórmula de dinamita

	Operadores				
Lotes	**1**	**2**	**3**	**4**	**5**
1	A=24	B=20	C=19	D=24	E=24
2	B=17	C=24	D=30	E=27	A=36
3	C=18	D=38	E=26	A=27	B=21
4	D=26	E=31	A=26	B=23	C=22
5	E=22	A=30	B=20	C=29	D=31

Se definen cuatro variables: FUERZA, FORMULA, LOTE y OPER que representan los datos observados de la fuerza explosiva, los tipos de fórmula, los lotes de materia prima (renglones) y los operadores (columnas). Para saber cómo se deben ingresar dichos datos en la hoja electrónica se puede generar una matriz de diseño, de acuerdo a este modelo.

Accedemos a la Ventana ANOVA por medio del menú Estadísticas,ANOVA. Agregamos la variable FUERZA al cuadro *Variable Dependiente* y las variables FORMULA, LOTE y OPER (en ese orden) a la lista de Factores. Al presionar el botón Aceptar se realiza el análisis y Hoelest presenta la siguiente salida (la interpretación de la primera parte es análoga a la realizada en experimentos de un sólo factor). El coeficiente de determinación R^2 indica que el modelo explica el 81% de la variación observada en FUERZA (fuerza explosiva).

Factor: FORMULA

Niveles(5): 1, 2, 3, 4, 5,

Var Dep: FUERZA

R: 0,900361531593817

R Cuadrada: 0,810650887573965

Análisis de Varianza

Fuente	Suma Cuadrados	GL	Medias Cuadrados	F	Prob
PUNTA	330	4	82,5	7,73437	0,0025
Renglones	68	4	17	1,59375	0,2390
Bloques	150	4	37,5	3,51562	0,0403
Error	128	12	10,66667		
Total	676	24			

En la tabla de análisis de varianza, podemos ver cuatro fuentes de variación: explicada por FORMULA, RENGLONES (LOTES), COLUMNAS (OPERADORES) y no explicada (Error). Se calculan las sumas de cuadrados para todas, generando la columna *Suma Cuadrados* . Al dividir la suma de cuadrados por los grados de libertad (GL), obtenemos las medias cuadráticas (Medias Cuadrados) o medidas de varianza, que expresan cómo varían los datos (según la fuente de variación) por término medio. La varianza explicada o la media de cuadrados entre tratamientos

(82,5) es mayor que la residual (10,67). Si α=0,01, el valor crítico de F es $F_{0.01, 4, 12}$ = 5,41. A partir de que 7.73 > 5.41, se concluye que existe una diferencia significativa en la fuerza explosiva media debido a las cinco fórmulas diferentes. También hay una indicación de diferencia entres los operadores, por lo tanto, controlar ese factor fue una buena precaución. No existe una evidencia fuerte de que haya una diferencia entre los lotes de materia prima, y al parecer, en este experimento hubo una inquietud innecesaria en cuanto a esa fuente de variabilidad. Sin embargo, controlar localmente los lotes de materia prima siempre resulta adecuado.

6.6.6. Experimentos con dos o más factores

Muchos experimentos se llevan a cabo para estudiar los efectos producidos por dos o más factores. Los diseños factoriales son los más eficientes para este tipo de experimentos.

Por diseño factorial se entiende aquel en el que investigan todas las posibles combinaciones de los niveles de los factores en cada ensayo completo o réplica del experimento. Por ejemplo, si existen *a* niveles del factor A y *b* niveles del factor B, entonces cada réplica del experimento contiene todas las *ab* combinaciones de los tratamientos. Se dice que los factores están cruzados cuando éstos se arreglan en un diseño factorial.

El efecto de un factor se define como el cambio en la respuesta producida por un cambio en el nivel del factor. Se lo conoce como efecto principal porque se refiere a los factores de interés primordial del experimento. También existe la interacción entre los factores. Al comparar las medias de tratamientos para un experimento factorial, necesitamos normalmente más de una observación por tratamiento. Por ejemplo, si obtenemos dos observaciones para cada uno de los tratamientos de un experimento factorial completo, decimos que tenemos dos réplicas. El análisis de varianza discutido en esta sección, requiere que un diseño factorial de dos o más factores tenga por lo menos 2 réplicas.

Los tipos más sencillos de diseños factoriales implican sólo dos factores o conjuntos de tratamientos. Hay *a* niveles del factor A y *b* niveles del factor B, dispuestos en un diseño factorial; esto es, cada repetición o réplica del experimento contiene todas las combinaciones de tratamiento *ab*. En general, hay n repeticiones.

Los datos observados se verán como en la Tabla XIX. En el caso general, sea Y_{ijk} la respuesta observada cuando el factor A se encuentra en el i-ésimo nivel (i=1,2, .. n). El orden en el cual se toman las *abn* observaciones es aleatorio, de modo que éste es un diseño completamente aleatorizado. El modelo estadístico para este diseño es:

$$Y_{ijk} = \mu + \tau_i + \beta_j + (\tau\beta)_{ij} + \varepsilon_{ijk} \begin{cases} i = 1,2,...,a \\ j = 1,2,..,b \\ k = 1,2,..,n \end{cases}$$

En donde μ es el efecto medio general, τ_i es el efecto del i-ésimo nivel del factor renglón A, β_j es el efecto del j-ésimo nivel del factor columna B, $(\tau\beta)_{ij}$ es el efecto de la interacción entre τ_i y β_j, y ε_{ijk} es el componente del error aleatorio. Hay un total de abn observaciones porque se realizan n réplicas.

Tabla XIX

Disposición general para un Diseño Bifactorial

		Factor B			
		1	2	...	b
Factor A	1	$y_{111}, y_{112},\ldots$ $\ldots\ldots y_{11n}$	$Y_{121}, y_{122},$ $\ldots\ldots y_{12n}$	.....	$Y_{1b1}, y_{1b2},$ $\ldots\ldots y_{1bn}$
	2	$Y_{211}, y_{212},$ $\ldots\ldots y_{21n}$	$Y_{221}, y_{222},$ $\ldots\ldots y_{22n}$	.....	$Y_{2b1}, y_{2b2},$ $\ldots\ldots y_{2bn}$
	...	.....	.....	.....	.....
	a	$y_{a11}, y_{a12},$ $\ldots\ldots y_{a1n}$	$Y_{a21}, y_{a22},$ $\ldots\ldots y_{a2n}$	.....	$Y_{ab1}, y_{ab2},$ $\ldots\ldots ,y_{abn}$

6.6.6.1. Análisis Estadístico

En un diseño factorial de dos factores, tanto los factores (o tratamientos) de renglón como de columna tienen la misma importancia. Específicamente el interés consiste en probar hipótesis acerca de la

igualdad de los efectos de tratamiento de renglón, es decir:

$H_o : \tau_1 = \tau_2 = ... = \tau_a = 0$

H_a: al menos una $\tau_i \neq 0$

Y de la igualdad de los efectos de tratamiento de columna

H_o: $\beta_1 = \beta_2 = \ldots = \beta_b = 0$

H_a: al menos una $\beta_i \neq 0$

También es interesante determinar si los tratamientos de renglón y columna interaccionan, es decir:

H_o: $(\tau\ \beta)_{ij} = 0$

H_a: al menos una $(\tau\ \beta)_{ij} \neq 0$

Denotemos el número de niveles del factor A con la letra *a*, el número de niveles del factor B con la letra *b*, el número de réplicas con la letra n, y el número de observaciones total N = abn.

Este modelo divide la suma de cuadrados de las desviaciones (SCTotal) en cuatro componentes. El primero que se denomina Suma de cuadrados del factor A (SCA) sirve para calcular una medida de la variación entre las medias de los niveles del factor A. El segundo componente, llamado Suma de cuadrados del factor B (SCB) sirve para calcular una medida de la variación entre las medias de niveles del factor B. El tercer componente, llamado Suma de cuadrados de la interacción AB (SCAB) sirve para calcular una medida de la interacción de los factores. El cuarto componente, llamado Suma de Cuadrados del error (SCE), mide la variación de las observaciones dentro de un tratamiento. Entonces:

SCTotal = SCAT + SCB + SCAB + SCE

Si se divide SCA para el número de niveles que tiene el factor A menos 1 (a-1) se obtiene la Media de Cuadrados del Factor A (MCA). Si se divide SCB para el número de niveles que el factor B tiene menos 1 (b-1) se obtiene la Media de Cuadrados del Factor B (MCB). Si se divide SCAB para el número de niveles de cada factor menos (a-1)(b-1) se obtiene la Media de Cuadrados de la interacción de los factores A y B (MCAB). Por último, si SCE se divide para la multiplicación del número de niveles de los factores involucrado por el número de réplicas menos 1, es decir, la expresión ab(n-1), se obtiene la Media de Cuadrados del Error (MCE).

La MCA, MCB, MCAB y MCE siguen una distribución Ji Cuadrado con (a-1), (b-1), (a-1)(b-1) y ab(n-1) grados de libertad respectivamente. Para probar la igualdad en las medias del factor A, hay que usar la estadística:

$$Fo = \frac{SCA/(a-1)}{SCE/ab(n-1)} = \frac{MCA}{MCE}$$

que sigue una distribución F de Fisher con (a-1) y ab(n-1) g.l. Para probar la igualdad en las medias del factor B, hay que usar la estadística:

$$Fo = \frac{SCB/(b-1)}{SCE/ab(n-1)} = \frac{MCB}{MCE}$$

que sigue una distribución F de Fisher con (b-1) y ab(n-1) g.l.

Para probar los tratamientos de renglón y columna interaccionan, hay que usar la estadística:

$$Fo = \frac{SCAB/(a-1)(b-1)}{SCE/ab(n-1)} = \frac{MCAB}{MCE}$$

que sigue una distribución F de Fisher con (a-1)(b-1) y ab(n-1) g.l.

Para las tres hipótesis a probar se tiene que si el valor esperado del numerador en la estadística de prueba es mayor que el valor esperado del denominador, la hipótesis alterna es verdadera y, en consecuencia debe rechazarse H_o si el valor de tal estadística es demasiado grande. Esto implica una región crítica unilateral superior. En otras palabras, se rechaza H_o si:

$F_o > F_{\alpha, a-1, ab(n-1)}$ Para la hipótesis sobre las medias del factor A

$F_o > F_{\alpha, b-1, ab(n-1)}$ Para la hipótesis sobre las medias del factor B y,

$F_o > F_{\alpha, (a-1)(b-1), ab(n-1)}$ Para la hipótesis sobre la interacción de los factores.

El análisis de varianza puede extenderse a más factores, digamos k. Entonces aparecerían k fuentes de variación principales (por los k factores) y más fuentes de variación por cada combinación posible de los factores. Por ejemplo si hay 3 factores A, B y C, tendríamos 3 fuentes de variación principales, 3 fuentes de variación por la interacción entre dos factores AB, AC y BC, una variación por la interacción de los 3 factores ABC y la varianza residual. En total analizaríamos 3 +3 +1 +1 = 8 fuentes

de variación en este caso. El procedimiento de prueba se resume en la Tabla XX.

Tabla XX

Tabla de Análisis de Varianza para un Diseño Bifactorial.

Fuente de Variación	Suma Cuadrados	GL	Medias Cuadrados	Fo
Tratamientos A	SCA	a-1	SCA / (a-1)	MCA / MCE
Tratamientos B	SCB	b-1	SCB / (b-1)	MCB / MCE
Interacción	SCAB	(a-1)(b-1)	SCAB /(a-1)(b-1)	MCAB /MCE
Error	SCE	ab(n-1)	SCE / ab(n-1)	
Total	SCTotal	abn -1		

6.6.6.2. ANOVA de un Diseño Multifactorial en Hoelest

Para efectuar el ANOVA se accede a la pantalla ANOVA como se explicó en la sección 6.6.3.2.

Ejemplo:

Consideremos el caso en el que un ingeniero diseña una batería para su uso en un dispositivo que será sometido a ciertas variaciones extremas de

temperatura. El único parámetro de diseño que él puede seleccionar en este punto es el material de la cubierta de la batería, y tiene tres alternativas.

Cuando el dispositivo se manufactura y se envia al campo, el ingeniero no tiene control sobre los extremos de temperatura a que será expuesto el dispositivo, y sabe por experiencia, que es probable que la temperatura influya en la duración efectiva de la batería. Sin embargo, es posible controlar la temperatura en los laboratorios de desarrollo.

El ingeniero decide probar los tres materiales de la cubierta a tres niveles de temperatura (15º, 70º y 125º F) consistentes en el entorno de uso final de producto. Se prueban cuatro baterías a cada combinación de material de la cubierta y temperatura, y las 36 pruebas se ejecutan al azar. En la tabla XXI se presentan el experimento y los datos resultantes de duración observada de las baterías.

El ingeniero desea contestar preguntas como: ¿Qué efecto tienen el tipo de material y la temperatura sobre la duración de la batería? Y ¿Existe una elección del material que dé por resultado una duración uniformemente larga sin importar la temperatura?

Tabla XXI

Datos del experimento de diseño de una batería

	Temperatura					
Tipo material	**15**		**70**		**125**	
1	130	155	34	40	20	70
	74	180	80	75	82	58
2	150	188	126	122	25	70
	159	126	106	115	58	45
3	138	110	174	120	96	104
	168	160	150	139	82	60

Se definen tres variables: DURACION, TEMPERAT y MATERIAL que representan los datos observados de la duración de las baterías, la temperatura y el tipo de material de la cubierta. Para saber cómo se deben ingresar dichos datos en la hoja electrónica se puede generar una matriz de diseño, de acuerdo a este modelo.

Accedemos a la Ventana ANOVA por medio del menú Estadísticas,ANOVA. Agregamos la variable DURACION al cuadro *Variable Dependiente* y las variables TEMPERAT y MATERIAL a la lista de Factores (no importa el orden). Al presionar el botón Aceptar se realiza el análisis y Hoelest presenta la siguiente salida (la interpretación de la primera parte es análoga a la realizada en experimentos de un sólo factor). El coeficiente de determinación R^2 indica que el modelo explica el 76% de la variación observada en DURACION.

Factor: TEMPERAT

Niveles(): 15, 70, 125,

Factor: MATERIAL

Niveles(): 1, 2, 3

R : 0,87476271972521

R Cuadrada: 0,765209815821046

Análisis de Varianza

Fuente	Suma Cuadrados	GL	Medias Cuadrados	F	Prob
Temperatura	39118,72	2	19559,36	28,9677	0,0000
Material	10683,72	2	5341,861	7,91137	0,0019
Interacción	9613,777	4	2403,444	3,55953	0,0186
Error	18230,75	27	675,2128		
Total	77646,97	35			

En la tabla de análisis de varianza, podemos ver cuatro fuentes de variación: explicada por TEMPERATURA, MATERIAL, INTERACCION y no explicada (Error). Se calculan las sumas de cuadrados para todas, generando la columna *Suma Cuadrados*. Al dividir la suma de cuadrados por los grados de libertad (GL), obtenemos las medias cuadráticas (Medias Cuadrados) o medidas de varianza, que expresan cómo varían los datos (según la fuente de variación) por término medio.

Se concluye que existe una interacción significativa entre el tipo de material y la temperatura porque $F_{0.05,\ 4,\ 27} = 2{,}73$. Además también son significativos los efectos principales del tipo de material y de la temperatura, porque $F_{0.05,\ 2,\ 27} = 3.35$.

6.7. Estadística No Paramétrica

6.7.1. Introducción

Algunos experimentos producen mediciones de respuesta que son difíciles de cuantificar. Los métodos estadísticos no paramétricos son útiles para analizar este tipo de datos.

Los procedimientos estadísticos no paramétricos se aplican no sólo a observaciones que son difíciles de cuantificar. También son particularmente útiles para hacer inferencias en situaciones en donde existe gran incertidumbre acerca de las suposiciones requeridas por la metodología estándar. Por ejemplo, la prueba t para comparar dos medias de la sección 5.3.7 se basa en el supuesto de que ambas poblaciones tienen distribuciones normales con varianzas iguales.

El experimentador nunca sabrá si estos supuestos se cumplen en una situación práctica, pero a menudo estará razonablemente seguro de que las desviaciones de las suposiciones serán lo bastante pequeñas como

para que las propiedades del procedimiento estadístico no sean afectadas. Esta dificultad se soluciona utilizando una prueba estadística no paramétrica, y así se evita confiar en un conjunto muy inseguro de suposiciones.

El término "estadística no paramétrica" no tiene una definición estándar que acepten todos los estadísticos. Sin embargo, la mayoría está de acuerdo en que los métodos estadísticos no paramétricos funcionan bien en suposiciones muy generales acerca de las características de cualquier distribución de probabilidad o parámetros involucrados en un problema inferencial. Las investigaciones han demostrado que las pruebas estadísticas no paramétricas son casi tan capaces de detectar diferencias entre poblaciones como los métodos paramétricos analizados en las secciones anteriores.

6.7.2. Prueba de Wilcoxon de Rangos con Signo para un Experimento Aparejado

Supóngase que tenemos n observaciones aparejadas de la forma (X_i, Y_i) y que $D_i=X_i-Y_i$. Estamos interesados en probar la hipótesis de que las X y las Y tienen la misma distribución, frente a la alternativa de que las distribuciones difieren en ubicación.

Se esperaría (en promedio) que la mitad de las diferencias entre las X y las Y sean negativas y la otra mitad sean positivas. Es decir, el número esperado de los pares con diferencias negativas sería n/2, donde n es el número de pares. Además, las diferencias positivas y las diferencias negativas con el mismo valor absoluto, deberían ocurrir con la misma probabilidad.

Si se ordenan las diferencias según su valor absoluto y se les asigna un rango de menor a mayor, el valor esperado de las sumas de los rangos para las diferencias negativas y positivas sería igual. Diferencias considerables entre las sumas de los rangos asignados a las diferencias positivas y negativas proporcionan evidencia para indicar un desfasamiento en la ubicación de las distribuciones.

Para realizar la prueba de Wilcoxon, se calcula las diferencias (D_i) para cada uno de los n pares. Se eliminan las diferencias que son iguales a cero, y se reduce conforme a ello el número de pares n. Luego se ordenan los valores absolutos de las diferencias asignándoles el rango 1 al más pequeño, el rango 2 al siguiente, etc. Si dos o más valores absolutos de las diferencias empatan para un mismo rango, entonces se asigna a cada miembro del conjunto empatado el promedio de los rangos que habrían asignado a estas diferencias.

Luego se calcula la suma de los rangos para las diferencias negativas (T^-) y también para las diferencias positivas (T^+). Para una prueba de dos colas utilizamos la menor de estas dos cantidades, T, como un estadístico de prueba, para probar la hipótesis nula de que los histogramas de frecuencias relativas de las poblaciones son iguales. A menor valor de T, mayor el peso de la evidencia a favor del rechazo de la hipótesis nula. Por lo tanto, rechazaremos la hipótesis nula si T es menor o igual a cierto valor, digamos T_o.

El valor T tiene una distribución aproximadamente normal cuando n es grande (25 o más). Esto permite construir una prueba Z para muestras grandes, en donde T = menor(T^+, T^-)

$$E(T) = \frac{n(n+1)}{4} \quad \text{y} \quad V(T) = \frac{n(n+1)(2n+1)}{24}$$

Entonces, el estadístico Z,

$$Z = \frac{T - E(T)}{\sqrt{V(T)}}$$

se puede utilizar como un estadístico de prueba.

En Hoelest se realiza la siguiente prueba de rangos:

Hipótesis nula: H_o: Las distribuciones de frecuencias relativas poblacionales para las X y las Y son idénticas.

Hipótesis alternativa: H_a: Las dos distribuciones de frecuencias relativas poblacionales difieren en ubicación (pruebas de dos colas).

Estadística de prueba:

$$Z = \frac{T - E(T)}{\sqrt{V(T)}}$$

Región de rechazo:

Se rechaza Ho si Z >= $Z_{\alpha/2}$ o bien para Z <= $-Z_{\alpha/2}$

6.7.2.1. La prueba de Wilcoxon en Hoelest

Por medio del menú *Estadísticas,EstadísticaNoParamétrica,Wilcoxon* se accede a la pantalla que permite especificar las variables que se desean estudiar. La figura 6.18 ilustra la ventana *Prueba de Wilcoxon* de Hoelest.

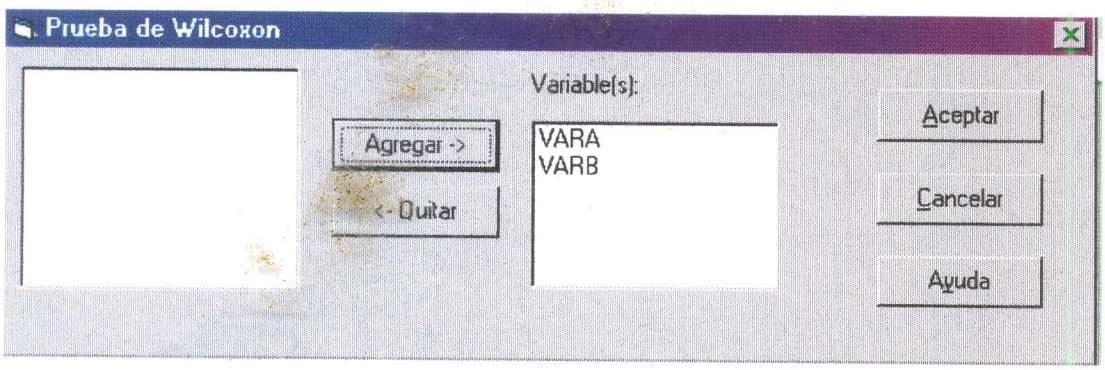


Figura 6. 18. Ventana para la prueba de Wilcoxon

En la parte izquierda de la ventana *Prueba de WIlcoxon* se listan todas las variables definidas en la hoja de cálculo; por medio del botón Agregar, se añaden a una segunda lista que contiene a las variables con las que se desea trabajar. Cuando se presiona el botón Aceptar, Hoelest calcula una prueba para cada pareja posible de las variables seleccionadas. Ejemplo:

Pruebe la hipótesis nula de que no hay diferencia entre las distribuciones poblacionales de la densidad de los pasteles para un experimento de diferencias aparejadas. Se utilizan seis pares de pasteles, uno preparado con la mezcla A y el otro con la mezcla B. Los datos observados, las diferencias y los rangos asignados se muestran en la Tabla XXII:

Tabla XXII

Datos para el ejemplo de la Prueba de Wilcoxon

A	B	A-B	Rango
0.135	0.129	0.006	3
0.102	0.120	-0.018	5
0.108	0.112	-.0004	15
0.141	0.152	-0.011	4
0.131	0.135	-0.004	1.5
0.144	0.163	-0.019	6

Se definen las variables VARA y VARB en Hoelest, que representan las variables de las mezclas A y B respectivamente. Accedemos a la ventana Prueba de Wilcoxon de Hoelest y agregamos ambas variables a la lista de variables de estudio. Presionamos el botón Aceptar y se presentarán dos matrices en una hoja electrónica. La primera matriz contiene los estadísticos de la prueba para cada pareja de variables, en este caso el valor de z=1,57.

	VARA	VARB
VARA	0	
VARB	1,5724	0

La segunda matriz contiene las probabilidades normales asociadas al estadístico, es decir el nivel de significancia de la prueba para cada pareja de variables: En este caso el valor es 0,1158

	VARA	VARB
VARA	0	
VARB	0,1158	0

Si consideramos α=0.10, el valor crítico de Z es 1.64. Luego, el valor del

estadístico de prueba calculado es 1.5724 que es menor que 1.64, por lo que no se rechaza la hipótesis nula, es decir, no hay evidencia suficiente de la diferencia entre las distribuciones poblacionales de A y B.

6.7.3. Prueba H de Kruskal-Wallis para un Diseño Completamente Aleatorizado

La prueba H de Kruskal-Wallis es una técnica no paramétrica para probar si *k* poblaciones difieren en ubicación. Supóngamos que tenemos muestras aleatorias independientes de *k* poblaciones que solamente difieren en ubicación. No es necesario suponer que estas poblaciones tengan distribuciones normales. Los tamaños muestrales de cada población no necesariamente son iguales, entonces se tiene que n_i=1,...,k es el tamaño de la muestra seleccionada de la i-ésima población.

Para realizar esta prueba se combinan todas las $n_1 + n_2 + ... + n_k = n$ observaciones, se ordenan y se les asigna el rango 1 a la menor observación, el rango 2 a la segunda y así hasta el final. Si dos o más observaciones están empatadas para un mismo rango, se asigna a cada miembro del grupo empatado el promedio de los rangos que se hubieran asignado a estas observaciones. Cuando se manejan rangos, la suma de rangos de cualquier distribución siempre suministra el mismo valor. Si hay

N valores la suma de los rangos de los N valores es la suma de los primeros N números, cuyo valor es (N * (N+1)) / 2.

Cuando se han ordenado los N casos de la muestra, según sus valores en la variable dependiente, y se han calculado la suma de rangos R_i para cada grupo o población *i*, la suma de los k resultados R_i es la suma total de rangos; es decir, siempre poseerá el valor (N * (N+1)) / 2. Pero esta suma global puede encontrarse dispersa por los *k* grupos de forma muy diferente. Si no existe ninguna diferencia entre los grupos, todas las sumas R_i tendrán el mismo valor. Cuanto mayores sean las diferencias entre los grupos, mayor será también la dispersión entre las sumas R_i, y por tanto, mayor la media de las sumas al cuadrado de rangos por grupo. (R_i^2 / n_i).

La prueba H de Kruskal-Wallis se basa en la media de las sumas al cuadrado de los rangos por grupo, pero la modifica para controlar el efecto del número de datos y el número de grupos o poblaciones, de forma que sigue una distribución Ji cuadrado con k-1 grados de libertad. La expresión concreta, denominada H de KrusKal-Wallis, es:

$$H = \frac{12}{N(N+1)} \sum_{i=1}^{k} \frac{R_i^2}{n_i} - 3(N+1)$$

Hoelest se encarga de utilizar esta aproximación cuando los tamaños n_i de las poblaciones son "grandes". Se dice que los n_i son grandes cuando son mayores o iguales a 5.

En Hoelest se realiza la siguiente prueba H de Kruskal-Wallis:

Hipótesis Nula: H_o: Las k distribuciones poblacionales son idénticas.

Hipótesis alterna: H_a. Al menos dos de las distribuciones poblacionales difieren en ubicación.

Estadístico de prueba:

$$H = \frac{12}{N(N+1)} \sum_{i=1}^{k} \frac{R_i^2}{n_i} - 3(N+1)$$, en donde

N_i = número de mediciones en la muestra de la población i.

R_i = suma de los rangos para la muestra i, en donde el rango de cada medición se calcula según su tamaño relativo en el conjunto global de $n= n_1+ n_2 + ... + n_k$ observaciones obtenidas al combinar los datos para las k muestras.

Región de rechazo:

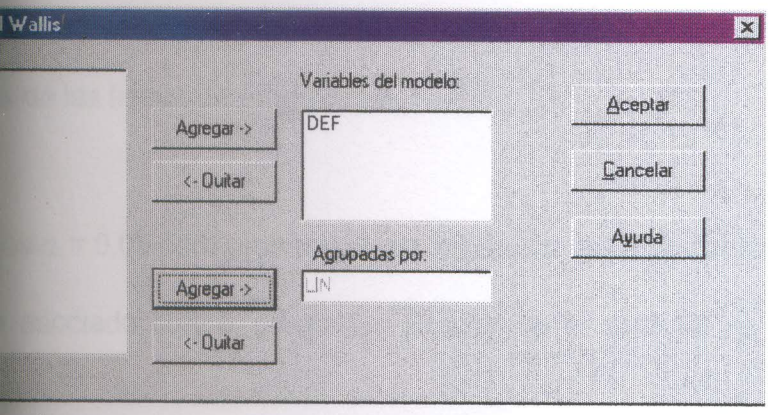
Se rechaza H_o si $H > \chi_\alpha^2$ con (k-1) grados de libertad.

Supuestos: 1. Las k muestras se extraen al azar e independientemente.

2. Hay cinco o más mediciones en cada muestra.

a Prueba de H de Kruskal-Wallis en Hoelest

del menú *Estadísticas,EstadísticaNoParamétrica,Kruskal-Wallis*
e a una pantalla que permite especificar las variables que se
tudiar.



ura 6. 19. Ventana para la prueba de Kruskal-Wallis

6.19 ilustra la ventana *Prueba de KrusKal-Wallis* de Hoelest. En
quierda de la ventana *Prueba de Kruskal-Wallis* se listan todas
les definidas en la hoja de cálculo. Con ayuda de los botones
Quitar superiores e inferiores, se determina el modelo que se
zar.

ero de control de calidad seleccionó muestras independientes de
ción de tres líneas de montaje en una fábrica de componentes

electrónicos. Para cada línea, se examinó el resultado de diez horas de producción seleccionadas al azar, para encontrar productos defectuosos.

Verificaremos si los datos que constan en la Tabla XXIII proporcionan suficiente evidencia de que las distribuciones de probabilidad del número de defectuosos por hora de producción difieren en ubicación para, por lo menos, dos de las líneas de producción.

Utilizaremos $\alpha = 0,05$ para probar nuestra hipótesis, además obtendremos el valor p asociado con la prueba, o lo que es lo mismo, el nivel de significación.

Tabla XXIII

Datos para el ejemplo de la Prueba de Kruskal-Wallis

	Linea 1	Linea 2	Linea 3
Defectos	6	34	13
	38	28	35
	3	42	19
	17	13	4
	11	40	29
	30	31	0
	15	9	7
	16	32	33
	25	39	18
	5	27	24

Se definen las variables DEF y LIN en Hoelest, que representan las variables Defectos y Línea de producción respectivamente. Accedemos a la ventana Prueba de Kruskal-Wallis de Hoelest y escogemos a LIN como la variable que agrupa, dado que este es el factor que afecta a la variable DEF, que la colocaremos en el grupo Variables del modelo. Al presionar el botón Aceptar se realiza la prueba y Hoelest presenta la siguiente salida:

Factor: LIN

Niveles(): 1, 2, 3,

Var Dep: DEF

Estadístico H: 6,09741935483871

Prob JiCuadrada: 4,74091220144367E-02con 2GL

Grupo	Ocurrencias	Suma Rangos
1	10	120
2	10	210,5
3	10	134,5

La primera línea indica el factor en el modelo, en este caso LIN que se escogió como variable agrupadora. LIN representa las líneas de producción que son tres, de allí el número de niveles. Luego se identifica la variable dependiente DEF. La prueba de Kruskal-Wallis se basa en el

estadístico H, cuyo valor en este ejercicio es 6,09 con una significancia de 0.004.

Dado que todas los n_i son mayores o iguales a 5, podemos utilizar la aproximación para la distribución de H según H_o y rechazar la hipótesis nula de ubicaciones iguales si el estadístico H es mayor que el valor crítico de la Ji cuadrada con (k-1)=2 grados de libertad $\chi_{2, 0.05} = 5.99147$. Por lo tanto, rechazamos la hipótesis nula a un nivel de $\alpha = 0,05$ y concluimos que al menos una de las tres líneas tiende a producir un mayor número de defectuosos que las otras.

En la matriz se tabulan los niveles del factor, el número de ocurrencias en cada uno de ellos y la suma de rangos del mismo.

CONCLUSIONES Y RECOMENDACIONES

1. Cuando comenzó el proyecto Hoelest, se eligió la edición Profesional de Visual Basic 5.0 como el lenguaje de desarrollo; debido a que éste ofrecía un conjunto completo de herramientas para el desarrollo de aplicaciones. Actualmente, ya está disponible en el mercado la versión 6.0 de Visual Basic. De aquí surge la primera recomendación: generar una nueva versión de Hoelest con la nueva versión de Visual Basic, debido a las ventajas que ofrece, como por ejemplo: mayor rapidez del código de ejecución.

2. La Hoja Electrónica Estadística es un sistema conformado por dos componentes de código ejecutables: EstadísticaInferencialDLL y Hoelest.exe. El primero relacionado estrechamente con la parte estadística de la aplicación; mediante su modelo de objetos proporciona todas las funciones de cálculo estadístico. El segundo, Hoelest.exe, relacionado con la parte informática, proporciona la interfaz visual de una hoja electrónica común. Por esta razón, se pueden optimizar ambos independientemente, es decir, crear nuevas versiones de cada uno de ellos especializadas en el ámbito correspondiente.

3. El componente EstadísticaInferencialDLL es un servidor en proceso, que proporciona todas las funciones estadísticas contempladas en el alcance de la elaboración de la Hoja Electrónica Estadística. La única manera de acceder a su funcionalidad es por medio de una aplicación cliente. El componente Hoelest.exe mantiene una referencia a este componente, por lo que es un cliente del mismo.

4. EL componente Hoelest.exe es un servidor fuera de proceso, que permite interactuar con el usuario común. Puede ser ejecutado como una aplicación de escritorio (como Microsoft Word) o como un componente servidor. Los objetos que proporciona permiten manipularlo mediante código, así como Microsoft Excel puede ser manipulado por medio de macros.

5. Cada componente de la Hoja Electrónica Estadística (HOELEST) tiene su propio modelo de objetos. Esta característica, propia de los componentes ActiveX, permiten que puedan ser utilizados como servidores en cualquier plataforma y por cualquier programa que permita Automatización. Sin embargo, Hoelest se desarrolló exclusivamente para su uso en el sistema operativo Windows. Se recomienda para futuras versiones de Hoelest, expandir su nivel de operación a otras plataformas, como Macintosh por ejemplo.

6. Quedan fuera del alcance de Hoelest varias técnicas estadísticas como por ejemplo: análisis de correlación canónica, análisis de series de tiempo, herramientas para el control de calidad, etc. Se recomienda para futuras versiones de Hoelest, que se aumente la capacidad de análisis de la hoja electrónica estadística, generando una segunda versión del componente EstadísticaInferencialDLL con las nuevas funciones, y también generando una segunda versión de Hoelest.exe con la inclusión de pantallas y ayuda que permitan interactuar con el usuario para que utilice sus nuevas características.

7. Hoelest.exe y EstadísticaInferencialDLL proporcionan a programadores de clientes ActiveX, algunas constantes incorporadas que resultan de gran utilidad con diversas instrucciones. Esto les facilitará la escritura del código, además de colaborar con la legibilidad del mismo.

8. En tiempo de ejecución, es decir cuando un usuario está utilizando Hoelest, se generan errores. Hoelest intercepta estos errores gracias a su código de tratamiento de errores, por lo que se convierte en una aplicación más robusta. Además permite a una aplicación externa interceptar los errores que se producen en él, enviando la siguiente información: Un número entero que indica el último error que ocurrió. La descripción y el origen del error (el nombre de la aplicación del objeto que generó el error, en este caso Hoelest.exe).

9. El componente Hoelest.exe tiene una interfaz de múltiples documentos, que le permite trabajar con varios archivos abiertos a la vez. Esta es una característica que no suele presentarse en las aplicaciones existentes de cálculo estadístico. Sería interesante modificar la interfaz visual para que se convierta en una aplicación más amigable, siendo una opción atractiva implementar una interfaz que tenga la apariencia del Explorador de Windows, es decir con un panel en su lado izquierdo que muestre los archivos abiertos mediante un árbol.

10. Los objetos están encapsulados; es decir, contienen su propio código y sus propios datos, lo cual facilita su mantenimiento más que los métodos de escritura de código tradicionales.

11. Las descripciones de las clases de EstadísticaInferencialDLL y Hoelest.exe se almacenan en bibliotecas de tipos y se pueden ver con examinadores de objetos.

12. El tipo de enlace determina la velocidad con la que se tiene acceso a los métodos de un objeto mediante la variable de objeto. El componente Hoelest.exe se comunica internamente entre sus objetos por medio de enlaces en tiempo de diseño, lo que mejora su rendimiento sustancialmente.

13. Las interfaces múltiples son una característica del Modelo de Objetos Componentes (COM); permiten que tanto EstadísticaInferencialDLL como Hoelest.exe evolucionen con el tiempo, agregando nueva funcionalidad sin afectar al código existente.

14. Los componentes ActiveX EstadísticaInferencialDLL y Hoelest.exe permiten a los programadores combinar fragmentos de código reutilizables para crear aplicaciones y servicios. De esta manera, un programador puede convertir a Microsoft Excel en un cliente de EstadisticaInferencialDLL y utilizar sus métodos. También se puede convertir Hoelest.exe en un cliente, digamos de un servidor de funciones financieras, para utilizar sus métodos y presentar los resultados como lo hace con EstadísticaInferencialDLL.

15. Un programador de aplicaciones cliente no puede tener acceso al funcionamiento interno de los objetos de EstadísticaInferencialDLL o Hoelest.exe. Los programadores sólo necesitan saber cómo pueden utilizar la funcionalidad ofrecida por los objetos, por ello se proporciona información al respecto en la biblioteca de tipos del componente, además de un archivo de ayuda creado para cada componente que ilustra con mayor detalle su funcionalidad.

16. Debido a la compatibilidad binaria y desarrollo multiplataforma, las aplicaciones cliente y los servidores compatibles con COM son

ependientes del lenguaje de desarrollo y del sistema operativo, en as palabras, un programador de Visual C++ puede utilizar la cionalidad de análisis de datos que proporciona tadísticaInferencialDLL, que es un componente desarrollado en Visual sic.

tualización más sencilla debido al control de versiones. Nunca se rde funcionalidad cuando se actualizan los componentes; siempre se jora o se amplía. Esto significa que es posible crear una nueva versión EstadísticaInferencialDLL y aquellas aplicaciones cliente, como elest.exe, que utilizaban las versión anterior no generarán problemas.

comunicación entre el cliente Hoelest.exe y el servidor tadísticaInferencialDLL proporciona el intercambio de información más ciente. Como el cliente y el componente servidor comparten el mismo pacio de direcciones, las llamadas entre ellos se hacen directamente.

ando se compila un componente, el compilador crea identificadores icos globales (GUID) para cada clase e interfaz pública del mponente. Esto asegura su unicidad universal.

el proyecto de la Hoja Electrónica Estadística no se crearon controles tiveX. Sería interesante diseñar, por ejemplo, un control *ntrolDeCalidad* que se encargue exclusivamente de la presentación de

os de control de calidad.

aces claramente definidas. A la funcionalidad implementada en un onente de código ActiveX se tiene acceso mediante propiedades, los y eventos. Estas interfaces se documentan a sí mismas en o a que el componente de código las expone automáticamente. Los ios de Automatización pueden examinar la información descriptiva da interfaz mediante el Examinador de objetos u otras herramientas.

odelo de objetos de los componentes EstadísticaInferencialDLL y est.exe define una relación entre los objetos de cada componente de o. Al definir las relaciones entre los objetos utilizados en el ama, un modelo de objetos organiza los objetos de forma que sea fácil programar.

ecomienda que esta aplicación (su estructura interna) se ponga a sición de los alumnos de la Escuela Superior Politécnica del Litoral siguen carreras afines a la computación, para que sea utilizada como rial de referencia en cursos de Programación Orientada a Objetos y logía de componentes ActiveX. Para los estudiantes de Ingeniería stadística Informática, reviste principal interés, debido al componente dístico que forma parte del sistema, pudiendo ser de utilidad en os de Estadística Computacional.

BIBLIOGRAFIA

ALFRED, HOPCROFT JOHN, ULLMAN JEFFREY. Estructuras de
s y Algoritmos. Editorial Addison-Wesley Iberoamérica. Estados
os 1988.

RDEN RICHARD, FAIRES DOUGLAS. Análisis Numérico. Grupo
torial Iberoamérica. México 1985

OYANES AGUILAR LUIS. Microsoft C/C++ 7. Manual de Bolsillo.
ditorial McGraw-Hill e Interamericana de España. 1994.

OYANES AGUILAR LUIS. Programación Orientada a objetos,. Editorial
McGraw-Hill. 1996

KENNEDY WILLIAM, GENTLE JAMES. Statistical Computing. Editorial
Marcel Dekker. Estados Unidos de América 1980

6. MANZANO VICENTE. Inferencia Estadística. Aplicaciones con SPSS/PC+. Editorial RA-MA. Madrid 1995

7. MENDEHALL WILLIAM. Estadística para administradores. Grupo Editorial Iberoamérica. México 1995.

8. MENDEHALL WILLIAM, WACKERLY DENNIS, SCHEAFFER RICHARD. Estadística Matemática con aplicaciones, Segunda Edición. Grupo Editorial Iberoamérica. México 1994.

9. MICROSOFT CORPORATION. Libros en Pantalla de Visual Basic, versión 1.0. Año 1996

10. MICROSOFT CORPORATION. Mastering Visual Basic 5, versión 1.0. Año 1997

11. MONTGOMERY DOUGLAS. Diseño y Análisis de Experimentos, Tercera Edición. Grupo Editorial Iberoamérica

12. PACHNER JAROSLAV. Handbook of Numerical Analysis Applications. Editorial McGraw-Hill. Estados Unidos de América 1984

13. SAND PAUL. Advanced Pascal Programming Techniques,. Editorial Osborne/McGraw-Hill 1984.

14. TENENBAUM AARON, LANGSAM YEDIDYAH, AUNGENSTEIN MOSHE. Data Structures using C. Editorial Prentice-Hall. Estados Unidos de América 1990.

15. WEISS MARK ALLEN. Estructuras de Datos y Algoritmos. Editorial Addison-Wesley Iberoamérica. Estados Unidos de América 1995