Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ingeniería Mecánica Industrial

SIMULACIÓN DE CONTROL ESTADÍSTICO DE LA CALIDAD CON LENGUAJE R

Erick Suhel Marroquín Escobar
Asesorado por la Inga. Nora Leonor Elizabeth García Tobar

Guatemala, abril de 2017

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

SIMULACIÓN DE CONTROL ESTADÍSTICO DE LA CALIDAD CON LENGUAJE R

TRABAJO DE GRADUACIÓN

PRESENTADO A LA JUNTA DIRECTIVA DE LA
FACULTAD DE INGENIERÍA
POR

ERICK SUHEL MARROQUÍN ESCOBAR
ASESORADO POR LA INGA. NORA LEONOR ELIZABETH GARCÍA TOBAR

AL CONFERÍRSELE EL TÍTULO DE

INGENIERO INDUSTRIAL

GUATEMALA, ABRIL DE 2017

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

NÓMINA DE JUNTA DIRECTIVA

DECANO	Ing. Pedro Antonio Aguilar Polanco
VOCAL I	Ing. Angel Roberto Sic García
VOCAL II	Ing. Pablo Christian de León Rodríguez
VOCAL III	Ing. José Milton de León Bran
VOCAL IV	Br. Jurgen Andoni Ramírez Ramírez
VOCAL V	Br. Oscar Humberto Galicia Nuñez
SECRETARIA	Inga. Lesbia Magalí Herrera López

TRIBUNAL QUE PRACTICÓ EL EXAMEN GENERAL PRIVADO

DECANO	Ing. Murphy Olympo Paiz Recinos
EXAMINADOR	Ing. César Augusto Akú Castillo
EXAMINADORA	Inga. Priscila Yohana Sandoval Barrios
EXAMINADORA	Inga. Nora Leonor García Tobar
SECRETARIO	Ing. Hugo Humberto Rivera Pérez

HONORABLE TRIBUNAL EXAMINADOR

En cumplimiento con los preceptos que establece la ley de la Universidad de San Carlos de Guatemala, presento a su consideración mi trabajo de graduación titulado:

SIMULACIÓN DE CONTROL ESTADÍSTICO DE LA CALIDAD CON LENGUAJE R

Tema que me fuera asignado por la Dirección de la Escuela de Ingeniería Mecánica Industrial, con fecha febrero de 2014.

Erick Suhel Marroquín Escobar

Guatemala, 27 de mayo de 2016

Ingeniero
Juan Jose Peralta Dardón
Director de la Escuela
Ingeniería Mecánica Industrial
Facultad de Ingeniería
Universidad de San Carlos de Guatemala

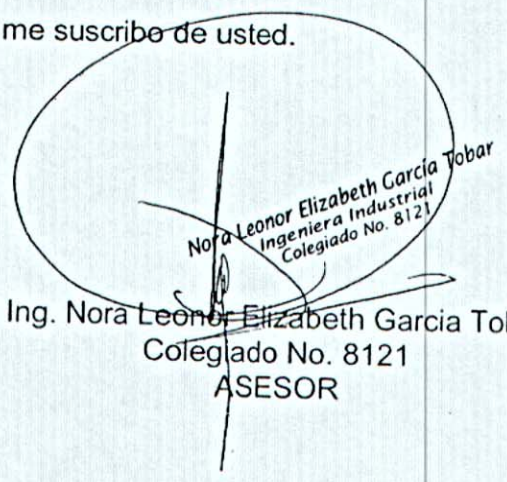
Estimado Señor Director:

Por medio de la presente informo a usted, que he asesorado y revisado el trabajo de tesis titulado SIMULACIÓN DE CONTROL ESTADISTICO DE LA CALIDAD CON LENGUAJE R, elaborado por la estudiante Erick Suhel Marroquín Escobar, con carné 1995-12063, previo obtener el título de Ingeniero Industrial

Habiendo determinado que dicho trabajo cumple con los requisitos establecidos de la Facultad de Ingeniería, y reconociendo la importancia del tema. Por todo lo anterior tanto el autor como el asesor somos responsables del contenido y conclusiones del presente trabajo de tesis y en consecuencia, por medio de la presente me permito APROBARLO, agregado que lo encuentro completamente satisfactorio.

Sin otro particular, me suscribo de usted.

Atentamente,



Ing. Nora Leonor Elizabeth Garcia Tobar
Colegiado No. 8121
ASESOR

UNIVERSIDAD DE SAN CARLOS
DE GUATEMALA

FACULTAD DE INGENIERIA

REF.REV.EMI.113.016

Como Catedrático Revisor del Trabajo de Graduación titulado **SIMULACIÓN DE CONTROL ESTADÍSTICO DE LA CALIDAD CON LENGUAJE R,** presentado por el estudiante universitario **Erick Suhel Marroquín Escobar,** apruebo el presente trabajo y recomiendo la autorización del mismo.

"ID Y ENSEÑAD A TODOS"

Inga. Miriam Patricia Rubio de Akú
Catedrático Revisor de Trabajos de Graduación
Escuela de Ingeniería Mecánica Industrial

Miriam Patricia Rubio Contreras
INGENIERA INDUSTRIAL
COL. 4074

Guatemala, agosto de 2016.

/mgp

UNIVERSIDAD DE SAN CARLOS
DE GUATEMALA

FACULTAD DE INGENIERIA

REF.DIR.EMI.042.017

El Director de la Escuela de Ingeniería Mecánica Industrial de la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala, luego de conocer el dictamen del Asesor, el Visto Bueno del Revisor y la aprobación del Área de Lingüística del trabajo de graduación titulado **SIMULACIÓN DE CONTROL ESTADÍSTICO DE LA CALIDAD CON LENGUAJE R,** presentado por el estudiante universitario **Erick Suhel Marroquín Escobar,** aprueba el presente trabajo y solicita la autorización del mismo.

"ID Y ENSEÑAD A TODOS"

Ing. José Francisco Gómez Rivera
DIRECTOR a.i.
Escuela de Ingeniería Mecánica Industrial

UNIVERSIDAD DE SAN CARLOS
DIRECCION
Escuela de Ingeniería Mecánica Industrial
FACULTAD DE INGENIERIA

Guatemala, marzo de 2017.

/mgp

Universidad de San Carlos
De Guatemala

Facultad de Ingeniería
Decanato

Ref. DTG.165-2017

El Decano de la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala, luego de conocer la aprobación por parte del Director de la Escuela de Ingeniería Mecánica Industrial, al trabajo de graduación titulado: **SIMULACIÓN DE CONTROL ESTADÍSTICO DE LA CALIDAD CON LENGUAJE R,** presentado por el estudiante universitario: **Erick Suhel Marroquín Escobar,** y después de haber culminado las revisiones previas bajo la responsabilidad de las instancias correspondientes, se autoriza la impresión del mismo.

IMPRÍMASE.

Ing. Pedro Antonio Aguilar Polanco
DECANO

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
DECANO
FACULTAD DE INGENIERIA

Guatemala, abril de 2017

/cc

ACTO QUE DEDICO A:

Mis padres	Irma Yolanda Escobar González y Mario Alberto Sandoval Nájera, por su constante amor, apoyo, comprensión y por haberme dado la vida, sabiduría y fuerza para nunca bajar los brazos.
Mis hijos	Xavier y Sofía Marroquín, por ser mi mayor triunfo.
Mis hermanos y familia	Axel Sandoval y su esposa Mariela Corado; a mi sobrina Mía Sandoval; Lisbeth Sandoval y mis sobrinos Khristal Pinto y André Hidalgo.
Mis abuelos	Cristina González, Rigoberto Rodríguez (q. e. p. d.), Zoila Nájera (q. e. p. d.), Víctor Hugo Sandoval (q. e. p. d.), y Catalina Noriega (q. e. p. d.).
Tíos y familia	Carolina, Giovanni y Marco Antonio Rodríguez (q. e. p. d.); Marta, René y Mario Escobar (q. e. p. d.), y Nery González.
Amigos y familia	Mis camaradas y hermanos de la jauría: Marvin Calderón, Wagner Molina, Romeo

Román, Byron Mazariegos, Melvin Pérez, Jorge Marroquín, German Quiroa, Edy Perensén, Luis Sirim, Cesar García (q. e. p. d.), Luis Minera (q. e. p. d.), Nery Silvestre, Marvin Toc, Jair Calderón, Joel Pérez, Manuel Álvarez y Carlos Miranda.

Mery Peña y familia

Porque siempre me diste fuerza con tu apoyo; porque me brindaste el calor y confianza de tu familia, en especial tus padres Santiago Peña y Blanca Rodríguez (q. e. p. d.).

AGRADECIMIENTOS A:

Universidad de San Carlos de Guatemala	Institución que ha siempre será mi templo del saber y formadora de mi sentir de lucha por la equidad, razón y justicia.
Facultad de Ingeniería	Por haberme aportado las bases del conocimiento técnico científico, el ímpetu del saber y la transformación de la realidad para contribuir a mi país Guatemala.
Inga. Nora García	Por darme la excelente guía y conocimientos para la elaboración de este trabajo.
Al pueblo de Guatemala	Por darme el privilegio de tener una educación superior digna. Yo soy uno de sus hijos y respondo en tu nombre…. ¡Hasta la victoria siempre!

ÍNDICE GENERAL

ÍNDICE DE ILUSTRACIONES

FIGURAS

TABLAS

LISTA DE SÍMBOLOS

Símbolo	Significado
( )	Agrupación de argumentos para cualquier función.
α	Alfa, letra griega utilizada para identificar el riesgo del productor de que se rechace un determinado lote. En general es para identificar el error tipo I.
{ }	Agrupación de las operaciones de una función, dentro de un par de llaves, pueden haber más llaves, lo que constituye un ambiente.
<-	Asignación a un objeto.
β	Beta, letra griega utilizada para identificar los parámetros de una modelo de regresión y en muestreo de aceptación es igual al riesgo del consumidor de aceptar un lote rechazable, igual al error tipo II.
ρ	Coeficiente de autocorrelación.
R^2	Coeficiente de determinación o bondad de ajuste
"A"	Cualquier símbolo alfanumérico entre comillas es

	una cadena de caracteres.
'…'	Dentro de una función significa que es factible la utilización de más atributos relacionados.
Φ	Distribución normal acumulada.
$\hat{u}$	Error calculado a partir de una muestra.
H_1	Hipótesis alternativa
H_0	Hipótesis nula
+	Indicador de nueva línea dentro de una función no cerrada.
S	Letra latina que se usa para identificar la desviación estándar de la población.
MCO	Mínimos cuadrados ordinarios.
AR(p)	Modelo autorregresivo con p rezagos.
ARMA(p,q)	Modelo autorregresivo de medias móviles.
ARIMA(p,d,q)	Modelo autorregresivo, integrado con medias móviles, con orden p, d y q.
MA(q)	Modelo de medias móviles con orden q.

#	Numeral utilizado para preceder a cualquier texto, el cual no se ejecuta en funciones ni en la línea de comando.
'…'	Observado en cualquier resultado, significa que se abrevia parte de los resultados para optimizar espacio.
Δ	Operador de primera diferencia.
$	Prefijo para indexar un objeto.
$\boldsymbol{P_a}$	Probabilidad asociada a un evento o proceso.
$\overline{\overline{\mathbf{X}}}$	Promedio de los promedios.
$\overline{\mathbf{X}}$	Promedio de los valores del objeto X.
>	*Prompt* línea de comando.
[1] 1 2 3	Resultado número uno, se presenta después de ejecutar una función o hacer una llamada a un objeto.
$\boldsymbol{X(t)}$	Serie de valores en función del tiempo.
$\boldsymbol{\sigma}$	Sigma, letra griega utilizada para identificar la desviación estándar

[f,c]	Signos para indexar subconjuntos en filas “f” y columnas “c”, si se trata de un vector, el atributo “c” no aplica.
Función (argumentos)	Sintaxis de las funciones
$\sum_{i=1}^{n} x_n$	Sumatoria de la variable desde el término i = 1, hasta n.
u	Término de error respecto a la media.
Z	Valor de la media estandarizada
X_{t-k}	Variable rezagada en k periodos

GLOSARIO

Ambiente Está definido como un objeto más en R, que representa el marco sobre el que se ejecutan ciertas condiciones.

AOQ Calidad promedio de salida o CSP, está definido por la calidad de salida después de pasar por el proceso de inspección.

AOQL Límite promedio de calidad de salida o LPCS, es el límite máximo o peor promedio de calidad que se obtiene de un programa de inspección.

AQL *Acceptance Quality Level*, es el Nivel de calidad aceptable o NCA por sus siglas en español, que está asociado con el nivel de calidad del productor y es el porcentaje de unidades que no cumplen con la calidad especificada.

Área de trabajo
Workspace Conjunto de resultados y objetos generados en una sesión de trabajo en R, se identifica en archivo como R Workspace y es almacenado en el directorio de trabajo con una extensión .Rdata.

Argumento	Son los valores, lógicos, numéricos que permiten que una función se comporte de una forma determinada.
Atributos de un objeto	Son el conjunto de características o propiedades que representan los datos asociados al objeto que constituyen en determinado momento el estado del objeto.
Carta de control	Herramienta gráfica y tabular que permite la identificación de la variabilidad del proceso, mientras que permite cuantificar la aparición o ausencia de atributos especificados.
Ciclo ***loop***	Es la sentencia que se realiza “cíclicamente” en o en forma repetida hasta que la condición asignada es cumplida. El ciclo es una pieza de código o *script*, en otras palabras por definición es un algoritmo.
Clase de un objeto ***class***	Atributo que determina de qué forma será tratado por lo que se conoce como función genérica
Control del proceso	Es el proceso por el cual, se detectan las variaciones estadísticas de un proceso productivo que puedan afectar la calidad de un producto, es en efecto, es un proceso mucho más amplio que el solo control estadístico, ya que requiere controlar el proceso mediante el análisis de las causas asignables.

Control por atributos Facilita el control de la condición de conforme o no conforme de un artículo o lote completo en contraposición a las variables que no se pueden controlar.

Control por variables Describe la variabilidad de la producción en función de magnitudes físicas continuas como, longitud, volumen, temperatura.

CRAN *Comprehensive R Archive Network* es la red en la cual están difundidos todos conocimientos desarrollados por la comunidad científica. Dichos conocimientos compartidos están depositados en paquetes y referencias bibliográficas digitales.

Curva de operación O curva característica de operación, es la relación que describe el grado de protección ofrecido por el plan de muestreo hacia la producción de lotes de diferentes calidades.

Depuración Proceso para encontrar, depurar o señalar posibles errores de programación que afecten el correcto funcionamiento de una función.

Dimensión
dim Atributo de la generalización de los vectores con un número de m de filas y un número n de columnas.

Expresión	Son una colección de una o más declaraciones a evaluar.
Función ***Function***	Objeto que permite de forma eficiente y mediante la declaración de argumentos el tratamiento de datos.
Gráfico de alto nivel	Gráficos que presentan las funcionalidades básicas.
Gráfico de bajo nivel	Gráficos que son alimentados con la adición de más funciones gráficas para revelar mayor información.
Inspección	Proceso por el cual, son medidas o examinadas las variables o los atributos respectivamente que resumen las características deseables de la calidad.
Interface gráfica para usuario *Gui*	Consola principal de R en forma gráfica.
Librería ***Library***	Conjunto de paquetes instalados y disponibles en el equipo
Límite de especificación	Es el límite de control que es establecido externamente de acuerdo a un estándar, norma o política.
Límite de control	Es el valor numérico que se establece o se calcula para limitar la zona en la cual se afirma que un proceso está bajo control.

Límite natural de control	Es el límite calculado a partir de la variación natural del proceso, en tal sentido está libre de causas asignables que causan variaciones no aleatorias, ya sea por atributos o variables.
Línea de comando ***Prompt***	Es el símbolo que indica que la línea principal de comandos está lista para recibir instrucciones, el símbolo más habitual en R es “>”
Lista ***List***	Objeto que representa una colección de componentes con distintos tipos de datos y longitud.
Longitud ***Length***	Atributo significa el tamaño de una unidad como un vector.
LPTD	Porcentaje máximo tolerable
LQL	*Limited Quality Level*, es el Nivel de Calidad Límite es conocido como el riesgo del consumidor. En principio, es el nivel de calidad que se considera como no satisfactorio, del que existe un riesgo de no rechazar un lote con un NCL mayor que cero, a este riesgo se le conoce como el riesgo del consumidor.
Marco de datos ***data.frame***	Objeto que reúne en una estructura de datos que se asemejan a los paneles de datos, donde cada columna o variable es un vector con datos del mismo tipo, no obstante las otras columnas pueden ser de otro tipo, siempre y cuando cumplan con el atributo

	de ser de la misma longitud.
Matriz ***Matrix***	Por definición es una generalización del vector con el atributo de dimensión, dicha definición aplica a las matrices de tres dimensiones o *arrays*.
Modo de un objeto ***Mode***	Tipo de almacenamiento del objeto, por ejemplo en el caso de un vector tiene modo lógico, numérico o carácter.
Muestra	Es un número “n” de elementos tomados de una población “N” donde su selección es por lo general en forma aleatoria ya que de esa forma cada elemento tiene la misma probabilidad de ser seleccionado, el número “n” depende de un cálculo estadístico.
Muestreo de aceptación	Es el muestreo que se levanta entre un lote de producción con el objetivo de aceptar o rechazarlo.
NaN	*Not a Number*, atributo de los valores que no corresponden al tipo de dato numérico contenido en un objeto.
No conformidad	Es la condición de un artículo o un lote que pasa por un proceso de inspección que identifica que no refleja los atributos deseados o la cantidad de atributos deseados.

Nombres ***names***	Atributo que da nombre a las filas, columnas o a cada valor de un vector, por lo que también aplica a las listas, factores y *data.frame.*
Núcleo base	Conjunto de paquetes instalados y disponibles que son instalados junto con la instalación de R y que funcionan por defecto. El núcleo base es parte integral del ambiente del lenguaje ya que contiene las funciones primitivas heredadas del lenguaje S.
NULL	Argumento que identifica la ausencia necesaria e inicial de un valor.
Objeto	Unidad lógica que consta de un estado en particular y un comportamiento, que bajo esa estructura se almacenan los datos.
Paquete ***package***	Conjunto de funciones y datos, que en un determinado ambiente simulan, calculan o modelan diversos fenómenos de la realidad.
Plan de muestreo de aceptación	Tiene como propósito dictaminar la aceptación o rechazo de un lote, no así, de estimar su calidad, aun cuando este lote esté dentro una serie de lotes, que son reconocidos nivel de calidad.
POO	Metodología general o paradigma para abordar problemas complejos, descomponiendo el problema en objetos.

R Commander	Paquete adicional de R, que permite en forma gráfica, reproducir las mismas funciones contenidas en el núcleo base, en tal sentido, abrevia la escritura de *script* s.
R Studio	Software externo, programado en Java, que simula las mismas características de R, en forma visual, que amplía sus funciones a la compilación de las funciones para la creación de paquetes, entre otras funciones.
R Tools	Conjunto de utilidades adicionales útiles para crear paquetes.
Serie de tiempo *ts*	Es un objeto similar al *data.frame*, con el atributo adicional que cada fila está definida por una frecuencia, inicio y final respecto al tiempo.
Source	Fuente u origen de datos, de los objetos en memoria.
TRUE/FALSE	Valores lógicos contenidos como dato o como argumentos.
Vector	Objeto como unidad atómica de datos del mismo tipo, ordenados en forma de colección.

RESUMEN

El presente texto es una guía inicial para el aprendizaje de estadística, programación orientada a objetos por medio del Lenguaje R. Dicho lenguaje es una de las herramientas utilizadas en los círculos académicos y profesionales de universidades alrededor del mundo.

La ventaja que tiene esta herramienta no es solo que no tiene costo para su uso y distribución, sino que está en constante desarrollo por los desarrolladores y los mismos usuarios. Por lo que, parte fundamental de los resultados de este trabajo es contribuir a la academia, desarrollando la guía de los aspectos que a criterio se necesitan para comprender y utilizar sin mayor dificultad el lenguaje. El punto álgido de la contribución de este trabajo resultó en la creación de dos paquetes con varias funciones que permiten el análisis y automatización en control estadístico de la calidad. Dicha contribución, está a disposición de aquel interesado que quiera practicar lo relacionado con dicha materia.

Otra contribución aportada por el texto, es la compilación de algunas funciones y paquetes, que son sin duda una herramienta útil para el cálculo de pronósticos. Lo más importante es que se abre la puerta para que la Universidad de San Carlos de Guatemala, a través de la Facultad de Ingeniería, se posicione a la vanguardia de la investigación de las herramientas computacionales.

OBJETIVOS

General

Introducir a la práctica del control estadístico de la calidad y a la estimación de series de tiempo el uso del lenguaje R que permita una mejor comprensión, y ampliar el acervo de conocimientos a temas más complejos, contribuyendo así al apoyo a las actividades pedagógicas y al fomento de la investigación científica.

Específicos

1. Sintetizar los aspectos fundamentales para iniciar al lector a una correcta aproximación de la lógica del lenguaje y paquetes de R para usos estadísticos y probabilísticos.

2. Describir y ejemplificar los detalles de la sintaxis del lenguaje para la comprensión de la estructura específica de los programas de las herramientas para casos prácticos.

3. Propiciar el intercambio de los productos, herramientas y experiencia, colocándolos al alcance de la comunidad para su uso y mejora, por medio de la explicación en manual del usuario, la generación del paquete y publicación en el repositorio *CRAN*.

4. Simular detección automática de patrones en las gráficas de control, obteniendo mayor precisión comparada con la detección visual basada

en los criterios de las reglas *Western Electric*.

5. Graficar las curvas de operación y la relación NCA y NCL en el entorno del análisis probabilístico como método para la comprensión e importancia que tiene el análisis de los niveles de calidad para el productor y para el consumidor,

6. Simular muestreos de aceptación por variables y atributos para obtener cifras correctas y oportunas para la práctica, prescindiendo del uso manual de las tablas y dedicarle más atención a los criterios que garanticen congruencia del muestreo con los niveles de calidad especificados.

7. Formular estimaciones con modelos de series de tiempo mediante la aplicación del paquete LM, como complemento a la comprensión de las aplicaciones, para facilitar los cálculos cuando los modelos estadísticos impliquen la transformación e inclusión de rezagos en el tiempo o detección de autocorrelación en la muestra de las variables consideradas como síntoma de patrones no aleatorios.

INTRODUCCIÓN

La utilización de métodos estadísticos y computarizados constituye hoy en día una herramienta no solo competitiva, sino un ventaja para superar las expectativas propias, en el sentido de anticiparse a los hechos simulando situaciones, estimando modelos matemáticos - estadísticos que describan un panorama cambiante y que no depende siempre de la toma de decisiones propias.

La generación, manipulación y almacenaje de información, seguido de una utilización productiva, no debe de ser un campo únicamente de los expertos en informática. Un ingeniero industrial puede obtener ventaja de la información y más aún, contar con cierta independencia para su manipulación, que le permite los conocimientos de estadística y matemática y por lo tanto, dado el volumen de la misma, conocer con cierta destreza el software que le facilite la tarea.

Para ello hay una cantidad grande de productos en el mercado (programas y lenguajes dedicados a la estadística), pero en este caso se necesitaba enfatizar en una herramienta de bajo costo, que sirviera a la vez de apoyo pedagógico, y que fuera una plataforma en constante desarrollo por la colaboración de la comunidad científica y profesional.

Reuniendo dichas características se seleccionó al lenguaje R, el cual es un software libre; esto quiere decir que se está en la libertad de utilizar el software para cualquier propósito, distribuir copias del programa, estudiar cómo funciona el lenguaje y adaptarlo a las necesidades, por lo que está a libre disposición el código fuente del mismo lenguaje y sus paquetes; mejor aún, hay

libertad de desarrollar mejoras y compartirlas.

En efecto, para ser usuario es absolutamente necesario contar con conocimientos de estadística y usuario estándar de computadora personal, al menos de hojas electrónicas. Para el caso de ser desarrollador, se necesita contar con un conocimiento más dotado de lenguajes de programación, bases de datos, matemática y absoluta comprensión del tema a simular.

El texto se ha desarrollado en forma lógica para que el orden de ideas vaya desde la comprensión de la plataforma base de R. Por ejemplo, entre la lectura pueden notarse algunos textos con negrilla, los cuales corresponden al simbolismo de los objetos, por ejemplo el objeto **X.prom** es un objeto que puede contener información, ya sea en forma de vector, matriz o marco de datos, entre otros; sin importar de qué clase de objeto se trate, es un objeto más que contiene información.

No obstante, en la terminología de la programación orientada a objetos, una función también es un objeto; sin embargo se optó por identificarlas con letra cursiva seguido del paréntesis, por ejemplo la función *xrs_gr*( ) que sirve para el cálculo de los límites de control de la carta X, R y S; los paréntesis tienen por objetivo recordar que necesitan para su ejecución de determinados argumentos.

Por otro lado, los ejemplos desarrollados en el *prompt* de la consola principal de R, como también las piezas de *script* de cada función en los archivos de extensión ".R", son transcritos al documento con el formato que aparece en pantalla, tanto con la simbología original como las sangrías que indican que ese párrafo está implícito en determinada parte del programa o al ambiente general.

Respecto de los resultados, en la consola principal se nota que están antecedidos del símbolo “>”, en cambio, las piezas de *script* no están antecedidas de ningún símbolo, exceptuando cuando aparece el símbolo numeral “#”, que significa en ambos casos que lo que esté a la derecha del mismo es un simple texto sin validez para efectos de cálculo. En los ejemplos que se detallan en varios temas, se utiliza esa propiedad para explicar en pocas palabras el objetivo de las siguientes líneas.

El lenguaje tiene la característica de distinguir entre mayúsculas y minúsculas para todos los objetos. Esta nota es importante ya que un carácter distinto produce un objeto distinto. También es necesario considerar que no es permitido nombrar objetos cuya cadena de texto alfanumérico comience o simplemente sea un número. Así como considerar los signos reservados para el lenguaje, como los paréntesis, las comillas, la coma, los corchetes, las llaves o el símbolo de moneda, los cuales no pueden ser utilizados para nombre de ningún objeto.

Puede notarse también, dentro de las funciones y documentación la falta de acentuación y falta utilización de caracteres que no son del tipo ASCII, como la letra “ñ”. Esto se evitó con el fin de crear conflictos del tipo advertencias dentro de las funciones creadas, no obstante los paquetes en conjunto y las funciones, puede ejecutarse sin ninguna consecuencia.

Tanto los resultados de la consola principal como las piezas de *script* son presentados con un formato distinto de los párrafos normales del texto, ya que tienen un menor tamaño de fuente, el espaciado entre líneas es menor y hay una sangría mayor a todo el párrafo. Un aspecto sumamente importante asociado a la sintaxis del lenguaje es la utilización de la coma “ , “. Dicho símbolo es utilizado como separador de los argumentos, por lo que la utilización

de la coma para símbolo decimal en los números representa un problema serio y requiere de mayor complejidad para programar. Por ello, para los usuarios principiantes, es mejor la utilización del punto como símbolo decimal.

No obstante, únicamente los cuadros de datos, son expresados por norma con la coma como símbolo decimal, para todo lo demás el punto es el símbolo decimal y la coma es empleada para separar elementos o argumentos.

El lenguaje cuenta con una plataforma base que incluye varios paquetes que por defecto son instalados al instalar por primera vez R. En ocasiones es de mucha utilidad consultar en la plataforma de ayuda y documentación de R, cómo funciona determinada función o cuál es la estructura de determinada clase de objetos.

Para ello se antecede el símbolo de interrogación a la función, nombre de paquete o de clase, por ejemplo en el caso de las matrices, se escribe ?matrix en la línea de comando y si la clase de objeto, la función o el paquete existe y tiene documentación, esta se desplegará en formato HTML, abriendo el navegador que tenga como predeterminado. En el contenido general del texto se utiliza una cantidad grande de términos provenientes del idioma inglés, se escriben dentro del texto ya sea en letra cursiva (sin anteceder a los paréntesis “( )” para diferenciarlo de las funciones) o encerrados entre comillas “ ”, dependiendo del contexto.

El resultado de todas las consideraciones anteriores es que se obtuvieron, como producto de este trabajo, dos paquetes que fueron diseñados para efectos pedagógicos para cursos, tales como “Controles Industriales” impartido en la Escuela de Mecánica Industrial y un tanto para “Control de la Producción”. Sin embargo las aplicaciones pueden extenderse aún más.

Al respecto, es preciso mencionar que los paquetes que se diseñaron para este texto y los utilizados, cuentan con documentación; no obstante dicha información está escrita en idioma inglés, para garantizar una mayor apertura a usuarios en varias regiones del mundo.

Es necesario considerar la utilidad que resulta de auxiliarse de *R Studio*, la cual es una plataforma visual de R, que contiene no solo las mismas capacidades básicas, sino el elemento de contar con mayor organización visual para la administración de proyectos para la construcción de paquetes.

Un paquete de nombre XRSCC está diseñado con varias funciones que automatizan los procedimientos para elaborar cartas de control por variables y atributos en control estadístico de calidad, a la vez contiene análisis más probabilístico como la utilización de la curva característica de operación y la utilización de las reglas de Western Electric, para calificar si un proceso está bajo o fuera de control, enfocado no en la variabilidad sino en la asociación de ciertos grupos no aleatorios.

Por otro lado está el paquete “Planesmuestra”, que simula el cálculo de los planes de muestreo de aceptación. Para ello utiliza los planes basados en variables y en atributos. Se auxilia de las curvas de operación para estimar el riesgo de aceptación de lotes rechazables o de rechazo de lotes aceptables.

En capítulo 5 se observa el uso de la terminología de las series de tiempo y la regresión lineal, por lo que se necesita cierto grado de dominio de temas del tipo econométrico, pero en efecto constituye una compilación de ideas que se consideran útiles para efectuar cálculos con mayor criterio estadístico. Sin embargo, también se utiliza paquetes ya programados y bajados del *CRAN Comprehensive R Archive Network* (al cual también se remitieron los paquetes

XRSCC y Planesmuestra). Los paquetes utilizados fueron seleccionados para complementar algunas ideas relacionadas con el aprendizaje de la plataforma de R y otros para completar la explicación de ciertos temas los relacionados con el control estadístico de calidad y los pronósticos.

El código del programa o *script* y la documentación en inglés de cada función de ambos paquetes están detallados en los apéndices.

Es necesario mencionar que por la propia dinámica de desarrollo del lenguaje, al iniciar la articulación de este texto, la versión utilizada es 3.0.3 y al final fue la 3.3.0 y en efecto, algunos paquetes utilizados fueron actualizados a la vez para adaptarse a la plataforma. A pesar de ello, todos los descritos en texto son ejecutables.

1. ASPECTOS FUNDAMENTALES DEL LENGUAJE R

1.1. Instalación

El lenguaje R es un software libre, se está en la libertad de correr el software para cualquier propósito, distribuir copias del programa, estudiar cómo funciona el lenguaje y adaptarlo a las necesidades; en tal caso es necesario contar con el código fuente y mejor aún, de implementar mejoras y compartirlas.

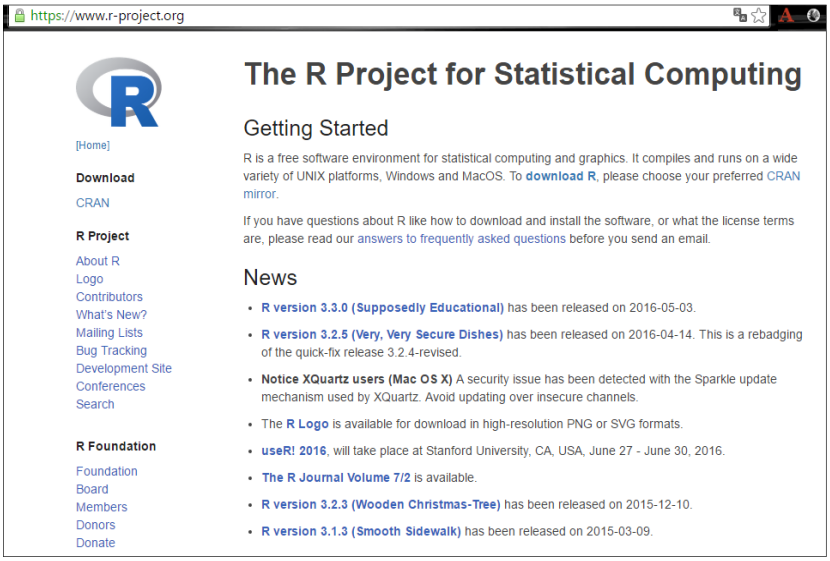
Está dividido conceptualmente en dos partes:

- El núcleo base de R, que es la parte que se baja de un *CRAN*, letras iniciales de *Comprehensive R Archive Network,* que significa "Red completa de archivos R".
- Todo los demás paquetes tales como *R Commander* y RODBC.

El paquete base de R, contenido en el núcleo es la compilación de sus funciones internas. Hay otros paquetes incluidos en núcleo base tales como: *utils, stats, datasets, graphics, grid*, entre otros.

Para la primera instalación de R es necesario contar con el último archivo de instalación o una conexión a internet, para obtenerlo; en este caso hay que ingresar la dirección http://www.r-project.org/ en la línea de comando del explorador o *browser* y aparecerá la web de R-Project (ver figura 1), en la que se puede acceder al *CRAN*. Al ocurrir esto, redirige la exploración hacia la página de *CRAN Mirrors*, que lista los servidores espejo que contienen una copia de todos los archivos del servidor original.

Figura 1. **Página web de R-Project**

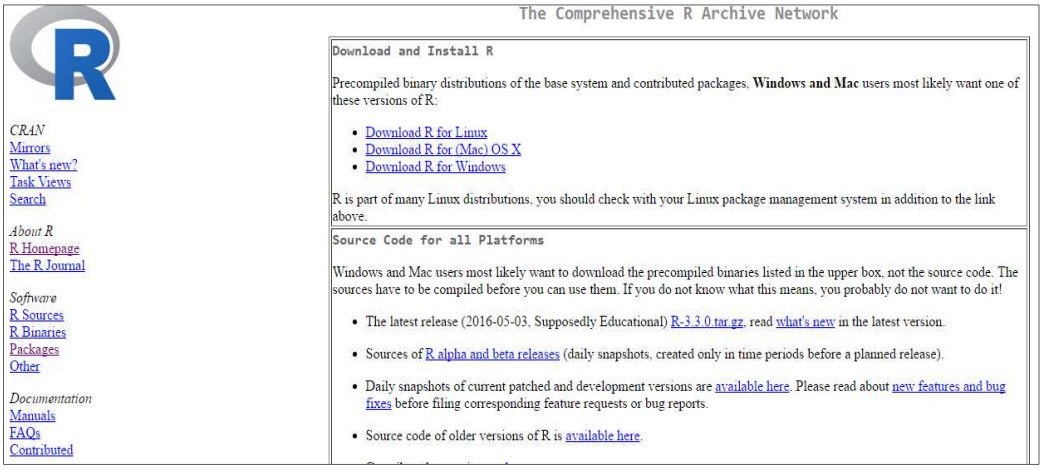


Fuente: R Project. https://www.r-project.org/.Consulta: abril de 2016.

Entre los servidores espejo están los dispuestos por universidades y centros de investigación alrededor del mundo. Para simplificar este proceso se escoge el espejo de nombre "*0-Cloud*" que abre la siguiente pantalla.

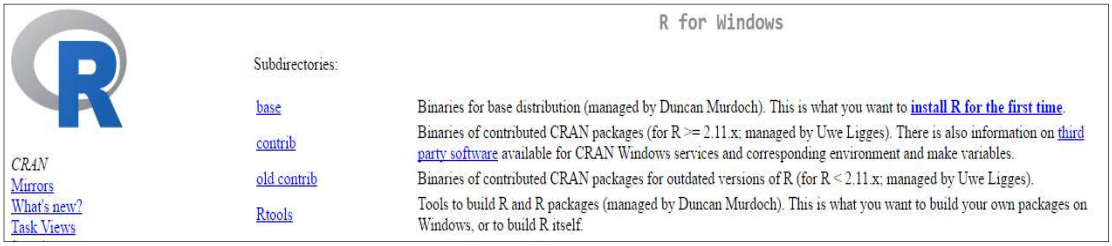
Según la figura 2, se muestran las versiones disponibles para bajar de acuerdo con el sistema operativo (Linux, OS X y Windows). En este trabajo se enfoca en la versión soportada para Microsoft Windows XP o superior. Al dar clic al vínculo "*Download R for Windows"*, accede a otra página en la que hay que continuar la opción "*base",* que consiste en el paquete base del programa del lenguaje, seguido del vínculo "*Download R for Windows"* y abrirá la siguiente página (ver figura 3).

Figura 2. **Pantalla para bajar instalador de R**



Fuente: R Project. https://cloud.r-project.org/.Consulta: abril de 2016.

Figura 3. **Lista de instaladores de R**



Fuente: R Project. https://cloud.r-project.org/. Consulta: abril de 2016.

A continuación, dar clic al vínculo "*install R for the first time*" para que muestre la siguiente página (figura 4).

Figura 4. **Vínculo para bajar R versión 3.3.0**

Fuente: R Project. https://cloud.r-project.org/. Consulta: abril de 2016.

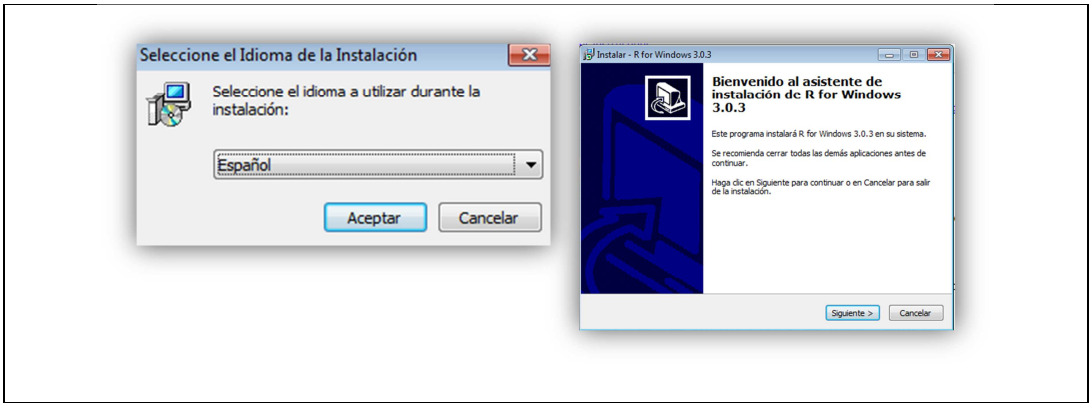
Al dar clic a "*Download R 3.3.0for Windows*", inicia el proceso de bajar el archivo de instalación, el cual, sin importar el navegador de internet o *browser* que se utilice, pedirá elegir entre "Ejecutar", "Guardar" o "Cancelar"; a continuación se da clic en "Ejecutar" (ver la figura 5).

Figura 5. **Cuadro para ejecución en *Microsoft Edge* y *Google Chrome***

Fuente: toma de pantalla *R 3.3.0,* proceso de instalación.

Una vez el instalador (R-3.3.0 –win.exe o superior) se ha descargado, se inicia el procedimiento de la instalación de R (ver la figura 6).

Figura 6. **Inicio de instalación de R**



Fuente: toma de pantalla R 3.0.3, proceso de instalación.

Al dar clic al botón “Siguiente>”, se despliega la ventana para leer la información relacionada con la licencia del programa. Al estar conforme con las condiciones de la licencia, se da clic a “Siguiente>” de nuevo. Esto abre la ventana que muestra el directorio donde se instalará el programa. Para cambiar el directorio de instalación, debe darse clic en “Examinar…”; en caso contrario, R por defecto, se instala en la ruta C:\Program Files\R\R-3.3.0.Al igual que otros tipos de software, R viene preparado para instalarse en sistemas de 32 o 64 bits.

A continuación, hay que seleccionar los componentes necesarios de acuerdo con el sistema operativo del equipo, se recomienda seleccionar la instalación del núcleo base de archivos “*Core Files*”, dar clic en “Siguiente”.

El siguiente paso de instalación pregunta si se desea configurar las opciones de configuración. Por efectos del alcance de este trabajo, se prescinde de esa configuración, por lo tanto, se da clic en “No” y “Siguiente >”.

En la ventana que a continuación abre, solicita la ubicación de la “Carpeta del menú de Inicio”, este paso no es crítico para los objetivos, así que se da clic en “Siguiente >” para dejar las opciones de instalación por defecto.

La siguiente ventana del proceso de instalación es para seleccionar las “Tareas adicionales”; se recomienda dejar las tareas habilitadas por defecto. Luego de ello se inicia el proceso de extracción de archivos; al final se mostrará una ventana en donde se avisa que el proceso se ha completado; a continuación dar clic en “Finalizar”; esto termina la instalación base de R.

1.2. Introducción a los objetos

Los objetos y la programación orientada a objetos son, respectivamente, los componentes y la estructura, para lenguajes como R, ya que el tratamiento estadístico y matemático de datos adquiere un orden de ideas distinto a otros lenguajes. Los objetos tienen características distintivas y la programación orientada a objetos está destinada a facilitar la relación dinámica de las características de los objetos.

1.2.1. Programación y los lenguajes orientados a objetos

La programación orientada a objetos es una metodología general o paradigma para abordar problemas complejos, descomponiendo el problema en objetos, Además de lenguaje, R es un software que proporciona un entorno para resolver problemas de estadística, por ello también es llamado un PSE (*Problem Solving Environment*), contrario a la programación funcional que se fundamenta en procedimientos o sucesión de funciones. Mientras que en la programación orientada a objetos se programa un concepto que contiene atributos que le proporcionan la definición. En resumen, la programación

orientada a objetos es un estilo de programación y ha tomado popularidad debido a que es fácil de programar sistemas complicados.

Fundamentado en la metodología de programación orientada a objetos, existen los lenguajes orientados a objetos, tal es el ejemplo de *Java*, *C++*, *Phyton* y por supuesto R; no obstante, en una u otra medida combinan metodologías de programación o paradigmas de la programación de forma funcional y declarativa. La lógica de la programación orientada a objetos se fundamenta en técnicas como la herencia, cohesión, abstracción, polimorfismo, entre otras.

"R en particular tiene dos sistemas de programación orientadas a objetos (POO). Uno es conocido como S3 y soporta por lo general funciones genéricas y el otro es conocido como S4 y soporta tanto clases como funciones genéricas"[1]. La entidad central de cualquier POO es el objeto y los conceptos de clase y métodos.

1.2.2. Los objetos

Los objetos como entidades, tienen varias funciones y están relacionados entre sí. "Cada objeto pertenece a una clase, de forma que las funciones pueden tener comportamientos diferentes en función de la clase a la que pertenece su objeto argumento"[2].Algunos objetos comparten algún atributo principal en común e independientemente del comportamiento; a este denominador común se le conoce como clase o *class* en inglés. Dicho de otra forma, un objeto es un caso concreto de una clase. Cada objeto posee

[1]GENTLEMAN, Robert. *R Programming for bioinformatics*. p.7.

[2] CONTRERAS, José. *Introducción a la programación estadística con R para profesores. Formación de profesores.* p. 3.

determinadas diferencias de otros objetos de la misma clase; estas diferencias son definidas como variables.

Un objeto es para la programación la representación de un concepto, el cual posee una clase y es particular debido a sus variables instancia y funciones internas o métodos instancia. Tal y como se mencionó anteriormente, la herencia constituye una característica muy importante, ya que la gran mayoría de objetos están a menudo relacionados con otros, ya que heredaron el núcleo central de la clase, más nuevas características que a su vez heredarán a nuevos objetos. Análogamente, lo mismo sucedió entre el lenguaje S y R, ya que la mayoría de las funcionalidades de S funcionan en R.

La diferencia de R y de S de otros paquetes estadísticos es que están enfocados a los objetos. Por ejemplo los resultados intermedios que se realizan en su línea de comando o consola por medio de *script s* se almacenan en objetos con nombre específico, esto permite ser observados, operados y analizados posteriormente, reduciendo así la acumulación en memoria RAM de resultados; dicho comportamiento es posible por el uso de objetos especiales como .*Generic*, .*Class*, .*Method* y .*Group* que permiten la interpretación inmediata de las instrucciones y funciones en R.

Además, como lenguaje orientado a objetos, es un lenguaje interpretado; es decir que los comandos y funciones escritos son ejecutados directamente sin necesidad de ser compilados, tal como funciona C++, Fortran o Pascal.

1.2.3. Las clases

La clase de un objeto determina de qué forma será tratado, por lo que se conoce como función genérica. Para el efecto, una clase o *class* es una función

genérica. Para un vector, las clases asociadas son *numeric, logical* y *character*. Esto quiere decir que un vector numérico solo se puede asociar libremente con otro de la clase *numeric*. No obstante es posible cambiar la clase.

1.3. Entorno y operaciones

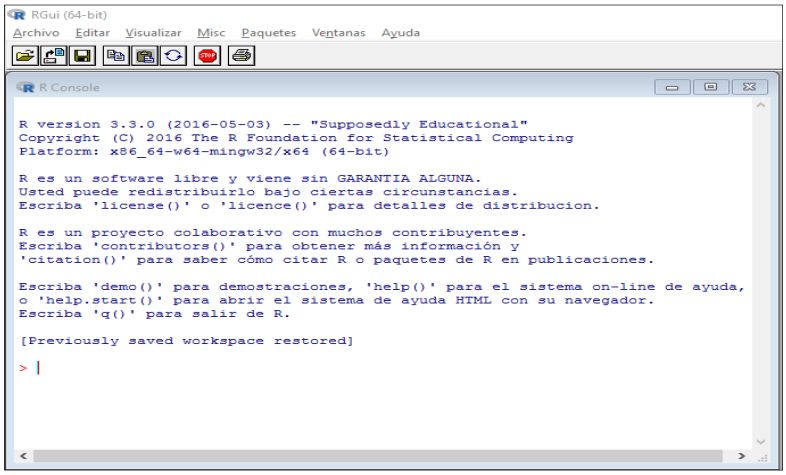
Para los usuarios principiantes, el área de trabajo de R puede aparecer bastante complicada, puesto que en este entorno o ambiente (*environment*), los comandos ingresados son interpretados directamente en la línea de comando (ver figura 7), para lo que no hay equivalente en forma gráfica por medio de ícono o selección de comando. Sin embargo, existen paquetes y software de plataforma como *R Commander* y *R Studio,* respectivamente, que sí cuentan con un entorno más gráfico; no obstante, no son totalmente gráficos.

Una ventaja del software, es que siendo un software de código abierto se puede de hecho modificar el entorno mediante la manipulación del código fuente o *Source Code*; sin embargo esta particularidad está fuera del alcance de los objetivos este texto. Por lo anterior, siguiendo la línea de y el nivel de desarrollo planteado, a continuación se presentan las principales funciones del entorno de la consola.

1.3.1. Línea principal de comando

Sobre la línea principal de comando de la interface gráfica para usuario o *Gui* en sus siglas en inglés, se observa el símbolo “>” de prompt el cual significa que la consola está lista para recibir cualquier instrucción (ver figura 7).

Figura 7. **Consola principal de R**



Fuente: toma de pantalla de RGui 3.0.03.

Posterior a cada resultado, retorna a al *prompt* indicando que ya está lista la consola para la siguiente instrucción u operación. Al ingresar cualquier función en el *prompt*, debe estar completa su sintaxis, es decir cerrado con ambos paréntesis, de lo contrario la consola pintará un signo “+” que representa el *prompt* de continuación:

```
>help(
+
```

El simple ejemplo del comando de ayuda *help*(), si no se cerró correctamente con el último paréntesis, la consola indica que hay funciones pendientes de digitar. De tratarse de un error de digitación, se procede a oprimir la tecla “Esc” e inmediatamente regresa la línea de comando al *prompt* de inicio “>”.

Los cálculos efectuados en la consola mediante alguna función, variable u

objeto se almacenan como un objeto en la memoria RAM, sin emplear archivos temporales. Al terminar la sesión de trabajo se puede digitar la función *q*() que significa lo mismo que *quit* o "salir" de otros lenguajes. Esto abre un cuadro de diálogo que consulta si se desea o no guardar el espacio de trabajo o *workspace*, en el cual si se confirma, almacena la información de la última sesión de trabajo; dicha información consiste en los objetos creados.

El *workspace* guardado se almacenará en un archivo de sesión .RData con los objetos creados en el directorio de trabajo por defecto (directorio de instalación) y el archivo .RHistory con los comandos y sentencias generadas[3].

El directorio de trabajo es por defecto en Windows la carpeta de "Mis documentos", pero este puede ser cambiado en el menú *File – Changedir*….

La función *save*(objeto, "archivo.RData"), permite guardar un objeto en memoria en archivo. Para cargar un objeto a memoria se utiliza la función *load*("archivo.RData"). Otra funcionalidad importante es que entre comandos y en los mismos *script* s, es posible escribir comentarios o textos para guiar al usuario del trabajo efectuado o de las instrucciones por venir. Para ello se escribe el texto precedido del símbolo "#", ejemplo:>#Este es un ejemplo para introducir comentarios en R.

1.3.1.1. Función *help* (ayuda)

Previamente se utilizó como ejemplo el comando *help* (), el cual posee gran importancia en el aprendizaje y guía del lenguaje. Con el simple hecho de teclear *help*() en el *prompt,* automáticamente accede al archivo de documentación principal en formato HTML, abriendo el explorador de internet

[3]MAIN, Paloma. *Análisis exploratorio de datos con R y Minitab*. p. 11.

por defecto. No obstante, al introducir dentro de los paréntesis de la función de ayuda el nombre de alguna función, opción o clase de un objeto, se abre la documentación específica:

```
>help(numeric)
># o mediante
>?numeric
```

Notar el uso del signo de interrogación “?” seguido de la función que se desea consultar, como sustituto perfecto. El anterior ejemplo debe de mostrar la documentación principal de los vectores numéricos, la cual contiene por lo general la siguiente información:

- Descripción
- Uso
- Argumentos
- Detalles
- Valores
- Referencias
- Ver también
- Ejemplo

1.3.1.2. Función *library* y librería de paquetes

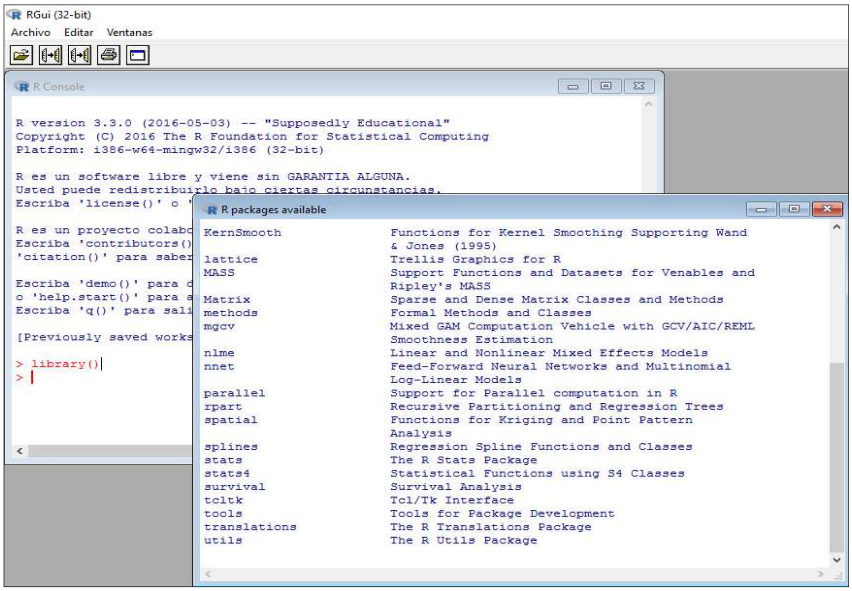
Los paquetes disponibles se encuentran localizados en el directorio C:\Program Files\R\R-3.3.0\library o en directorio donde esté ubicado R. En la instalación básica se encuentran los paquetes del núcleo principal.

Los paquetes que son bajados de cualquier *CRAN* residen en la misma ruta de almacenamiento. Para conocer la lista de los paquetes instalados y disponibles con una breve descripción, es necesario ingresar la función *library*( );el resultado se muestra en la figura 8. Al usar la función *search*( ), muestra las librerías y paquetes que se están ejecutando en la memoria RAM:

```
>search()
[1] ".GlobalEnv"        "package:stats"     "package:graphics"
[4] "package:grDevices" "package:utils"     "package:datasets"
[7] "package:methods"   "Autoloads"         "package:base"
```

Existen dos métodos para cargar a la memoria cualquiera de los paquetes previamente instalados, el primero es tecleando en el *prompt* la función library, conteniendo el nombre del paquete tal y como aparece en la lista.

Figura 8. **Librería de paquetes instalados**



Fuente: toma de pantalla de RGui 3.0.03.

```
>library(AcceptanceSampling)
>search()
 [1] ".GlobalEnv"              "package:AcceptanceSampling"
 [3] "package:stats"            "package:graphics"
 [5] "package:grDevices"        "package:utils"
 [7] "package:datasets"         "package:methods"
 [9] "Autoloads"               "package:base"
>library(Rcmdr)
Loading required package: car
Loading required package: MASS
Loading required package: nnet
Versión del Rcmdr 1.9-6
```

En el ejemplo anterior se cargó a la memoria el paquete AcceptanceSampling, luego se muestra que ya está en ejecución mediante la función *search*( ). Con el mismo método, se carga el paquete *R Commander*, con el nombre *Rcmdr.* El segundo método para cargar un paquete es ingresando al menú “Paquetes” (*Packages*), luego la opción “Cargar paquete” (*Load package*…) y seleccionar el paquete a utilizar.

La documentación de un paquete en ejecución puede ser obtenida mediante la función *library*(help=paquete). El resultado se muestra en pantalla con la descripción principal del mismo, la cual incluye el título, versión, autor, descripción, licencia, entre otros.

1.3.2. Operaciones básicas

Las operaciones básicas en R, deben entenderse como la forma particular en que las operaciones aritméticas fundamentales son calculadas y de qué forma las operaciones mismas son asignadas y almacenadas en objetos.

La complejidad de las operaciones y los datos, requiere objetos más complejos. Sobre el lenguaje en particular, la operación más común es la asignación a objetos y la operación de asignación a objetos de un subconjunto de datos es conocida como “vectorización”.

1.3.2.1. Asignación a objetos

Un paso importante para las operaciones en R, es asignar información a un objeto, ya sea por medio de la lectura de datos externos, es decir importándolos; la interpretación, derivación y generación de nueva información se da a partir de esos datos o simplemente se crean a partir de línea de comando.

Al asignar valores a un objeto, por ejemplo un vector **x,** luego se llama el mismo vector para visualizar su contenido, tal como el siguiente ejemplo:

```
> x<-c(1:3)
>x
[1] 1 2 3
```

En el anterior ejemplo, al vector **x** le son asignados los valores 1, 2 y 3, mediante la función c( ), abreviando el intervalo de los valores numéricos enteros mediante el símbolo de dos puntos “:”. Luego, con solo teclear el nombre del vector se muestra en pantalla a partir del elemento 1; por ello muestra antes del resultado el número uno entre corchetes.

Para asignar nombre a un vector o cualquier objeto es válido iniciar el nombre con cualquier letra (A-Z o a-z), ya que R discrimina el uso de mayúsculas y minúsculas.

Los siguientes caracteres válidos son otras letras, números (0-9) y puntos, por ejemplo el vector serie.A.1, que combina texto, con puntos, letras minúsculas, letras mayúsculas y números, contienen a los números del uno al diez.

```
> serie.A.1<-c(1:10); serie.A.1
[1]  1  2  3  4  5  6  7  8  9 10
```

En el siguiente ejemplo se crea un vector X**,** (equis mayúscula), el cual, aunque contenga los mismos valores, para R son objetos distintos por el nombre, pero son de la misma clase, debido a que sus valores son numéricos y de hecho el largo (*length*) es el mismo = 3.

```
> X<-c(1:3)
> x<-c(1:3)
> X
[1] 1 2 3
>x
[1] 1 2 3
> X-x
[1] 0 0 0
```

En la última operación, la cual resta **x** a **X**, el resultado concuerda con las reglas elementales de operación aritmética de vectores, ya que resta cada elemento *i* de x a cada elemento *i* de X.

1.3.2.2. Funciones aritméticas

Las operaciones básicas del tipo aritmético, de hecho, introducen el tema de objetos y funciones; estas últimas, la sintaxis de "*f(objeto, argumentos)*", en

la que la letra “*f*” simboliza la función y los paréntesis “( )”, encierran el objeto a operar y a los argumentos específicos de la operación, en un ambiente particular.

Los símbolos de las operaciones aritméticas básicas entre datos de los objetos y los mismos objetos son: +, -, *, /, ^, ( )y la función *sqrt*( ) que corresponde a la operación de raíz cuadrada.

Otro método para obtener raíces cuadradas y raíces de orden superior, en donde se puede usar el símbolo “^” operado con potencias fraccionarias, es el siguiente:

```
> 64^(1/3)
[1] 4
> 16^(1/2)
[1] 4
```

Para escribir texto sin que implique operación alguna en la consola principal, se precede la cadena de texto por el símbolo de numeral “#”; ver los siguientes ejemplos de operaciones aritméticas básicas:

```
> #Suma de constantes
> 3+4+2+0.3
[1] 9.3
> #Resta
> 4-2
[1] 2
> -6+4
[1] -2
```

```
> #Multiplicación
> 10*35
[1] 350
> (4+5)*2
[1] 18
> -(3+4)*4
[1] -28
> #División
> 8/3
[1] 2.666667
> (4*5)/10
[1] 2
```

Las operaciones elementales de aritmética son posibles en la línea de comando, no obstante, no es habitual utilizar a R como una calculadora; en tal sentido, se optimizan sus capacidades haciendo uso de las mismas funciones mediante objetos y dentro de funciones.

```
> #Potenciación
> 4^3
[1] 64
> 10^10
[1] 1e+10
```

En la operación 10^10 el resultado lo expresa por defecto mediante notación científica.

```
> #Potenciación fraccionaria o raíces
> 16^0.5
[1] 4
>sqrt(81)
```

```
[1] 9
>sqrt(5-6)
[1] NaN
Mensajes de aviso perdidos
In sqrt(5 - 6) : se han producido NaNs
```

En la operación de raíz cuadrada de la operación (5 – 6), R muestra un valor de “NaNs” porque no se especificó en números complejos, de la forma:

```
>sqrt(-1+0i)
[1] 0+1i
```

Los cálculos con números complejos se pueden efectuar con la salvedad de que la unidad imaginaria “ *i* ”, debe ir acompañada de un coeficiente diferente de cero y por otro lado, no es posible hacer la división entera y módulo:

```
> 2i+3i
[1] 0+5i
> 1+2i-(3-5i)
[1] -2+7i
```

La consola de R, puede funcionar como una calculadora, ya sea para números (escalares) u objetos numéricos, ver la tabla I.

Tabla I. **Funciones trigonométricas y de aproximación**

Función	**Sintaxis**
Seno	sin(ángulo)
Coseno	cos(ángulo)
Tangente	tan(ángulo)
Logaritmo base n	log(número, base)
Exponente base *e*	exp(exponente)
Redondear	round(número, decimales)
Cifras significativas	signif(número, cifras)

Fuente: elaboración propia.

1.3.2.3. Operadores comparativos y lógicos

Es importante tomar decisiones respecto de los datos por medio de las funciones o al programar, por ello, es necesario utilizar los operadores comparativos y los lógicos. Cuando se trata de hacer comparaciones entre conjuntos de datos, se establecen relaciones de igualdad, no igualdad y diferencia. Por ejemplo en el cálculo de la carta R, x o S, en el control estadístico de calidad, el cálculo de los límites de control tiene una secuencia iterativa, la cual debe excluir los valores mayores que el límite de control superior o LCS y menor al límite de control inferior o LCI, para luego calcular los nuevos límites de control, incluyendo el límite central.

Plantear operaciones lógicas siguiendo las leyes del cálculo proposicional y los conectivos “y” de conjunción, “o” de disyunción y la negación, también es posible y absolutamente necesario en el quehacer estadístico, tratándose de grupos y asociaciones entre ellos.

Lo anterior es particularmente útil cuando los datos se encuentran acumulados en grandes bases de datos y la definición de cada grupo viene dada por color, tamaño, lote (tratándose de la producción total o de muestras). La simbología empleada en R se detalla en la tabla II.

Con los valores lógicos, el doble símbolo “&&” y “||” implica el mismo efecto que la conjunción y disyunción, respectivamente, sin embargo valida primero el valor u objeto de la izquierda y si esta es verdadera, se detiene el análisis, ahorrando tiempo y espacio en la memoria.

Tabla II. **Simbología para los operadores comparativos y lógicos**

Comparativos		**Lógicos**	
<	Menor que	!X	Negación
>	Mayor que	X&Y	Conjunción “y”
<=	Menor o igual que	X&&Y	Igual
>=	Mayor o igual que	Y|Y	Disyunción “o”
==	Igual que	X||Y	Igual
!=	Diferente que	xor(X,Y)	“o” exclusivo

Fuente: elaboración propia.

Empleando la función *identical*( ), se comparan totalmente dos objetos[4]:

```
> x<-c(1,2,3,5,7);  y<-c(1,2,4,6,7)  # Se asigna los vectores
>length(x)>length(y) # Se compara el largo de ambos vectores
[1] FALSE
```

[4] CONTRERAS, José. *Introducción a la programación estadística con R para profesores. Formación de profesores.* p. 17.

```
>length(x)==length(y)
[1] TRUE
>x==y  # Se compara cada uno de los valores de cada vector
[1]  TRUE TRUE FALSE FALSE  TRUE
>identical(x,y)  # Se comparan totalmente ambos vectores
[1] FALSE
>x[x==y]  # se obtienen valores coincidentes
[1] 1 2 7
```

1.3.3. Creación y manipulación básica de objetos

En el inciso de las operaciones básicas se introdujo informalmente al tema de creación de objetos, fundamentalmente vectores, los cuales para efectos de aprendizaje, son los más apropiados por la facilidad de manipulación que presentan. En la asignación de valores a vectores es necesario mencionar que R opera con las llamadas estructuras de datos, que definen tanto unidades escalares como vectoriales. Entre esas estructuras, la más sencilla es un vector numérico, que en términos simples consiste en una colección ordenada de números, por ejemplo:

```
> #Creación manual de un vector y asignación
> x <- c(3.1, 4.2, 3.5, 4.3, 3.9)
```

Al vector **x** le fue asignado la colección de cinco números que son (3.1, 4.2, 3.5, 4.3, 3.9); notar que la cada número está separado por una coma “,” y la función concatenar *c*() es la que hace posible hacer la colección de valores, ya sea ordenados o sin patrón alguno. Una alternativa para asignar valores a vectores es con la función *assign*( ):

```
>assign("x", c(3.1, 4.2, 3.5, 4.3, 3.9))
```

```
>x
[1] 3.1 4.2 3.5 4.3 3.9
> #La forma más directa y conocida es:
>c(3.1, 4.2, 3.5, 4.3, 3.9)-> x; x
[1] 3.1 4.2 3.5 4.3 3.9
```

Donde el signo de asignación es “->” o “<-“, dependiendo de qué lado se encuentre el nombre del vector. Desde la aritmética, el álgebra de matrices y el análisis tensorial, son aplicables a los objetos del tipo *arrays*; sin embargo, para una aproximación simple de cálculo en vectores, ver los siguientes ejemplos:

```
> # Se muestran los números que componen el vector x
>x
[1] 3.1 4.2 3.5 4.3 3.9
> # La operación indica que se multiplica el vector x por 2
>x*2
[1] 6.2 8.4 7.0 8.6 7.8
> # Se divide el vector x entre 2
> # Una alternativa es multiplicarlo por 1/2
>x/2
[1] 1.55 2.10 1.75 2.15 1.95
> # Sumar 1 a cada elemento del vector x> x+1
[1] 4.1 5.2 4.5 5.3 4.9
```

1.3.3.1. Función *seq*()

Es común necesitar vectores que tengan una colección de valores en una secuencia definida, tal como secuencias ordinales o escalas numéricas. La secuencia más natural es la que asigna un rango de valores discretos a un vector, ejemplo:

```
> x<-c(1:10) ;x
[1]  1  2  3  4  5  6  7  8  9 10
> y<-c(20:10); y
[1] 20 19 18 17 16 15 14 13 12 11 10
```

Al vector “x” se le asignan los valores del uno al diez en orden creciente, mientras que al vector “y” se le asignan valores del 20 al 10 en orden decreciente.

Nótese que el símbolo de dos puntos “**:**” es utilizado en varios entornos de programación para señalar un rango específico. Al necesitarse una secuencia en la que un valor difiere del inmediato anterior en un factor conocido, como una progresión aritmética, se utiliza la función de secuencia *seq*( ) que usa los argumentos de dirección, rango y factor diferencia; ejemplo:

```
>seq(-10,10,by=2)->x
>x
 [1] -10  -8  -6  -4  -2   0   2   4   6   8  10
```

La secuencia indicada genera los números del -10 al 10, de dos en dos con la función “*by”*. De igual forma, se puede especificar el número de elementos, pero ahora se hace uso únicamente del inicio del intervalo, ejemplo:

```
>seq(length=11, from=-10, by=2)->y ;y
[1] -10  -8  -6  -4  -2   0   2   4   6   8  10
```

En este caso describe una secuencia de once números, comenzando desde menos diez (argumento *from*), contando de dos en dos. Por ejemplo, la proporción de productos defectuosos en una muestra sirve para construir la

curva característica de operación respecto de la probabilidad de aceptar un lote de tamaño *N*, un tamaño de muestra *n* y un número de aceptación *c*.

La proporción creciente de artículos defectuosos hace disminuir la probabilidad de aceptación del lote completo.

Para graficar esta situación es necesario contar con una secuencia de valores de la proporción máxima de artículos defectuosos; por ejemplo, hasta un 10 % desde el ideal de 0 % con un ancho de 100 valores.

```
> p<-seq(0,.1,length=100); p
[1] 0.000000000 0.001010101 0.002020202…. 0.097979798 0.098989899
0.100000000
```

1.3.3.2. Función*rep*()

Del inglés *replicate,* la función *rep*( ) es útil en la creación arreglos de datos en los que se necesita repetir series o simples colecciones de valores o cualquier tipo de arreglo, incluyendo texto:

```
> q<-rep(c(1,5,10),3); q
[1]  1  5 10  1  5 10  1  5 10
```

En el anterior ejemplo existe un vector conteniendo los números 1, 5, 10, cuya serie se repite tres veces:

```
>rep(1:3, each=2)
[1] 1 1 2 2 3 3
>rep(1:3,2)
[1] 1 2 3 1 2 3
```

```
>rep(1:3,2, length.out=5)
[1] 1 2 3 1 2
```

En este caso, una secuencia de valores del 1 al 3, se repite cada uno dos veces antes del siguiente valor mediante el argumento *each,* observar como la inclusión de este parámetro hace variar el orden de los valores.

Al omitir el argumento *each*, funciona el argumento *times*, que es un número entero no negativo, el cual indica cuantas veces se repite la lista de valores, en el ejemplo se indicó que se repitiera dos veces la secuencia de uno a tres.

Mediante el argumento *length.out* se especifica el largo del vector con un número entero y no negativo, por ejemplo cinco, cuando el largo natural del vector es seis. No obstante también se puede declarar un número mayor al largo natural del vector.

1.3.3.3. Objetos en memoria

Hasta acá se ha mostrado cómo crear objetos, para el caso de los vectores y una breve manipulación aritmética con los mismos. También es útil conocer cuales objetos están en memoria y eliminarlos si es necesario. La función para mostrar los objetos en memoria es *ls*( ):

```
>ls()
[1] "serie.A.1" "x"         "X"         "y"
```

La anterior operación muestra todos los objetos que en el momento permanecen activos en memoria RAM. Sin embargo en la práctica se puede dar el caso de que exista una cantidad grande de objetos con nombres similares o que comparten al menos un carácter en el nombre, para ello se emplea el

argumento *pattern*, abreviada dentro del paréntesis de *ls*( ) con el argumento *pat.* Sí se quisiera listar únicamente aquellos que contienen la cadena de texto “2014”, se procede de la siguiente forma:

```
>ls()
[1] "pruebas.2013"  "pruebas.2014"  "qcontrol.2014"
>ls(pat=2014)
[1] "pruebas.2014"  "qcontrol.2014"
```

De no existir ningún objeto que coincida con la búsqueda selectiva con *pattern* o la totalidad de los objetos en memoria con *ls*( ), la consola muestra el resultado como *character*(0).Para borrar un objeto de memoria se utiliza la función *rm*( ), digitando entre los paréntesis el nombre del objeto u objetos (separados por una coma “ , ”) que se desean eliminar permanentemente.

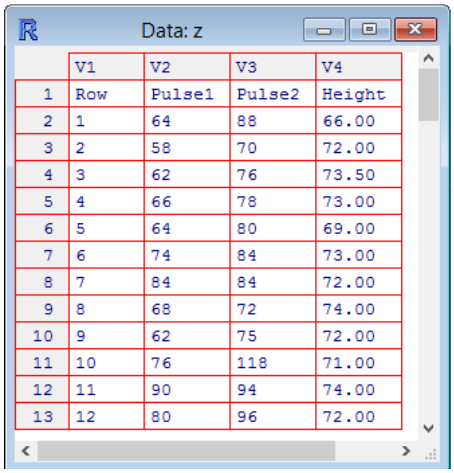
```
>rm(pruebas.2013, pruebas.2014, qcontrol.2014)
>ls()
character(0)
```

Otra forma es, tomar al conjunto de objetos en memoria como una lista:

```
>rm(list=ls())
>ls()
```

En ocasiones es necesario ver los datos contenidos en los objetos como en los marcos de datos, matrices, listas o vectores. Para ello se utiliza la función *view*( ), que permite ver el conjunto de datos en forma de hoja electrónica. Ver figura 9.

Figura 9. **Función *view* para vista tipo hoja electrónica**



	V1	V2	V3	V4
1	Row	Pulse1	Pulse2	Height
2	1	64	88	66.00
3	2	58	70	72.00
4	3	62	76	73.50
5	4	66	78	73.00
6	5	64	80	69.00
7	6	74	84	73.00
8	7	84	84	72.00
9	8	68	72	74.00
10	9	62	75	72.00
11	10	76	118	71.00
12	11	90	94	74.00
13	12	80	96	72.00

Fuente: toma de pantalla de función view( ).

Seguido de visualizar los datos, es frecuente que se necesite editar los datos en la misma forma de hoja electrónica. Con la función *fix*( ) se accede a dicha forma de visualización pero con las facilidades de edición de cortar o pegar por ejemplo. Con la función *edit*( ) se puede editar un conjunto de datos en forma de archivo de texto.

1.4. Manejo de datos

El intercambio de información y datos entre R y otros lenguajes u otras fuentes de información digital, es un tema que interviene directamente en la calidad del tratamiento y análisis estadístico, considerando en primera instancia que R no es software para almacenamiento de información, en tal caso, se aporta una aproximación de cómo acceder o interactuar con los formatos de datos más comunes.

1.4.1. Formatos y fuentes de datos soportados

El formato más fácil para acceder a datos e importarlos a un objeto es un archivo de texto de la forma “archivo.txt” y esto es a menudo aceptable para problemas de pequeña y mediana dificultad. Sin embargo, no son pocos los investigadores y profesionales en la estadística que están familiarizados con formatos más comerciales como hojas electrónicas de Microsoft Excel®, archivos de datos en XML, archivos de SPSS®, es decir archivos fácilmente portables en una memoria *USB*; no obstante también son accesibles los datos de bases de datos más robustas como SQL. Para los sistemas de bases de datos como SQL y/o bases de datos relacionales, suelen utilizarse los llamados “sistemas de administración de bases de datos” traducido de inglés *Data base management systems* (DBMSs), como MySQL, PostgreSQL, Microsoft Access®.

El formato base en R para la manipulación de datos es el archivo de texto. Para todos los demás formatos se necesita de paquetes específicos o apoyarse con los DBMSs a través de un *query* para exportarlo a un archivo de texto o una hoja electrónica de Excel.

Acceder a SQL es posible mediante los paquetes RODBC y RMySQL, el cual proporciona un servicio de cliente servidor, lo que implica que es posible acceder a datos desde en otro equipo conectado a una red LAN o conexiones remotas mediante conexiones URL como “http://”, “ftp://” y file://. El paquete RODBC, permite leer bases de datos DBF u Oracle y escribir otras mediante los controladores dBase ODBC de Microsoft. Así cómo es posible acceder a la información de los archivos de base de datos como dBase (.DBF), también lo es con las tablas tipo .DIF, .FWF y.CSV, hojas electrónicas tipo .ODS, archivos de Stata y SAS, incluso para usuarios más avanzados existe la opción de utilizar los archivos binarios.

Ahora es importante mencionar, que en R se trabaja con un directorio de trabajo (*working directory*), de no declararse otra ruta, la información debe de estar en esa carpeta para ser leída o escrita. Este parámetro puede cambiarse accediendo al menú Archivo (*File*) y la opción “Cambiar directorio” (*Changedir*…) donde se busca y se selecciona la carpeta deseada para directorio de trabajo. Una forma manual de hacer lo anterior es mediante la función *setwd*(DIR), donde *DIR* es la ruta de la carpeta que se desea establecer el directorio de trabajo:

```
>setwd(“C:/Rdata”)
```

Notar que al ingresar la ruta del directorio de trabajo, es necesario encerrarlo entre comillas (“ ”); de igual manera la consola presenta entre comillas la ruta al consultar y cuál es el directorio de trabajo actual.

Para conocer dicho directorio puede aplicarse la función *getwd*( ).

```
>getwd()
>[1] "c:/Rdata"
```

A menos que exista el directorio, no imprimirá ningún resultado adicional, de lo contrario pintará *NULL* cuando el directorio de trabajo no esté disponible. Por defecto, al abrir R, tiene como directorio de trabajo la carpeta de documentos del usuario en la versión para Windows. Para listar los archivos contenidos en el directorio de trabajo se utiliza la función *list.files*( ), la cual genera un vector de caracteres con los nombres entre comillas de los archivos contenidos en ese directorio, no obstante, la función cuenta con más resultados, dependiendo de los argumentos empleados; para ello consultar la ayuda de la función con *help*(list.files).

>list.files()
[1] "mudiametromm.csv" "mudiametromm.txt" “mudiametromm.xls”

1.4.2. Importar datos

La acción de importar datos tiene como propósito tomar la información deseada y trasladarla hacia objetos como vectores, tablas, matrices, factores y *arrays*.

Partiendo de la fuente más habitual de datos tabulados, una hoja electrónica de MS Excel, sin utilizar ningún paquete para leer directamente los datos, es necesario guardar el archivo formato *.xls, es decir hojas electrónicas previas a la versión 2007. Una vez guardado el archivo como delimitado por tabulaciones (*tab-delimited*) .TXT o la forma delimitado por comas (*comma-separated form*) .CSV, se emplea la función *read.delim*( ) y *read.csv*( ), respectivamente. Se puede utilizar previamente la opción de “guardar como” en el menú de archivo en MS Excel 2007 o superior para guardarlo como .XLS, ya que los formatos .XLSX provocan que las propiedades de los archivos .CSV no se trasladen adecuadamente.

Para guardar un archivo de MS Excel como un archivo .TXT, se recomienda que al tener la información en el archivo, contenerla en una sola hoja de cálculo con los datos; luego guardarlo como “Texto MS-DOS”.

El archivo .TXT resultante está apto ya para leerse. En el siguiente ejemplo, inicialmente se tiene un archivo de MS Excel con el nombre mudiametromm.xls (nótese la extensión del archivo) con una sola hoja de cálculo con nombre “datos” (el nombre de la hoja no es crítico para el proceso); el archivo contiene 20 muestras tomadas de diámetros internos de piezas

mecánicas; el archivo se abre y se guarda como un archivo “Texto MS-DOS” con el nombre **mudiametromm**.

1.4.2.1. Función *read.delim*()

Al ver el archivo en el directorio de trabajo tiene el nombre “mudiametromm.txt”. Ahora se procede en la consola a leer el archivo y asignar los datos a un objeto de nombre **tabla1**. Por conveniencia, en los resultados, se escribe el signo de dos puntos “:” para simbolizar que se omite intencionalmente datos sin perder la lógica del orden.

```
> tabla1<-read.delim("mudiametromm.txt");  tabla1
   MUESTRA DIAMETRO_MM
1       1     45.678
2       2     45.493
3       3     45.680
:       :      :
20     20      45.555
```

La anterior operación es de las más fundamentales en el análisis de datos en R. Como se puede apreciar, la función *read.delim*( ) lee el contenido del archivo .TXT y por defecto reconoce que la primera fila contiene el encabezado con los nombres de las variables, luego de leer el archivo y asignados los valores al objeto **tabla1** el cual también por defecto tiene la clase *data.frame* o hoja de datos.

1.4.2.2. Función *read.csv*( )

La importación de datos del formato .CSV es por medio de la función *read.csv*( ). El primer paso es guardar el archivo como CSV (MS-DOS), sin más

hojas de cálculo que la de nombre “datos”.

Empleando el mismo libro de MS Excel el resultado de **tabla2** tendrá los mismos datos de **tabla1**:

```
> tabla2<-read.csv("mudiametromm.csv"); tabla2
   MUESTRA DIAMETRO_MM
1        1      45.678
2        2      45.493
3        3      45.680
:        :      :
20      20       45.555
```

En este paso de importar datos, es muy importante observar que el anterior ejemplo es la forma más sencilla entre los numerosos casos posibles tan solo para los formatos .CSV y .TXT, ya que por defecto las funciones *read.delim*( ) y *read.csv*( ) asumen que la primer fila del contenido del archivo origen es un encabezado con los nombres de las variables o campos, sin embargo esto no es siempre es cierto.

Por otro lado, se mencionó que por defecto la clase del objeto asignado es una hoja de datos; este parámetro se puede especificar para asignar los datos a un objeto como un vector.

Los argumentos para la función *read.delim*( ) y *read.csv*( ) se presentan a continuación:

```
read.delim(file, header = TRUE, sep = "\t", quote = "\"",
dec = ".", fill = TRUE, comment.char = "", ...)
```

Tabla III. **Argumentos de la función *read.delim*( )**

Argumento	Descripción
file	Es el nombre del archivo entre comillas (“ ”). De no contener ninguna ruta o *path* el archivo deberá estar contenido en el directorio de trabajo. De igual forma, la información puede estar ubicada en una ubicación *URL* o *Universal Resource Locator.*
header	Es un valor lógico (TRUE o FALSE), que indica si la primera línea contiene los nombres de los campos o variables, el cual se puede omitir cuando es falso.
sep	Es el carácter de separación entre campos o variables, por defecto el carácter un espacio tabulado denotado con “\t” para el caso de ser un archivo.TXT. Si se tratara un .CSV el separador por defecto es una coma “,”.
quote	Son los caracteres para citar variables tipo carácter
dec	Es el carácter usado para punto decimal, por lo general es un punto “.”
fill	Agrega espacios en blanco cuando existen líneas sin valor de alguna variable.
comment.char	Es un carácter que define comentarios dentro del archivo. Las líneas que comiencen con este carácter son ignoradas.

Fuente: elaboración propia, con ayuda de R <help(base)>.

Se ha mencionado que por defecto, los objetos creados con las funciones *read* son designados como hoja de datos (*data.frame*), a menos que se especifique otro tipo de clase, ejemplo:

```
> tabla1m<-as.matrix(read.delim("mudiametromm.txt"))
>class(tabla1m)
```

```
[1] "matrix"
>dim(tabla1m)
[1] 20  2
>
```

Utilizando el mismo origen de datos, se creó una matriz (*matrix*) de dimensión 20 por 2 (20 filas por 2 columnas) con el nombre de las variables como nombre de las columnas. Además se puede acceder o llamar una fila y columna en particular de una hoja de datos o una matriz mediante la notación:

- objeto[i , j] Imprime en pantalla un valor en particular del objeto, la intersección de la columna *i* con la columna *j.*
- objeto[i ,] Imprime en pantalla todos los valores de la fila *i.*
- objeto[, j] Imprime en pantalla todos los valores de la columna *j.*

Ejemplo:

```
>tabla1[,2]
 [1] 45.678 45.493 45.680 45.623 45.544 45.597 45.670 45.652
45.579 45.673
[11] 45.613 45.569 45.628 45.664 45.613 45.624 45.641 45.626
45.596 45.555
>tabla1[4,]

  MUESTRA DIAMETRO_MM
4       4      45.623
>tabla1[4,2]
[1] 45.623
```

Las matrices son el caso en el que la dimensión del objeto (vector) es

$m \times n$, es decir de dos dimensiones; un objeto donde los datos descritos se proyectan a más de dos dimensiones es un *array*. Por ejemplo, al tratarse de tres dimensiones cada elemento tiene el índice (*i*, *j*, *k*) y su tratamiento tiene el mismo fundamento que una matriz.

1.4.2.3. Función *read.table*( )

Mediante la función *read.table*( ) se puede obtener el mismo resultado, sí y solo sí se declaran los separadores y encabezados.

```
> tabla3<-read.table("mudiametromm.csv",sep=",",header=TRUE)
> tabla3
   MUESTRA DIAMETRO_MM
1       1     45.678
2       2     45.493
3       3     45.680
4       4     45.623
5       5     45.544
:       :      :
20      20     45.555
>class(tabla3)
[1] "data.frame"
> tabla4<-read.table("mudiametromm.txt",sep="\t", header=TRUE)
> tabla4

   MUESTRA DIÁMETRO_MM
1       1     45.678
2       2     45.493
3       3     45.680
:       :       :
20      20     45.555
```

```
>class(tabla4)
[1] "data.frame"
```

Se puede observar que la diferencia estructural en los ejemplos de la función *read.table*( ) es el argumento de separador.

Para los formatos de separación por tabulaciones los datos están separados por un espacio tabular y para los delimitados por comas la separación es una sola coma.

Los argumentos para *read.table*( ) y *read.delim*( ), respectivamente, son:

```
read.table(file, header = FALSE, sep = "", quote = "\"",dec = ".", row.names,
col.names,na.strings = "NA, blank.line.skip=TRUE, nrows=-1)
```

Tabla IV. **Argumentos para *read.table*( )**

Argumento	Descripción
what	Especifica el tipo o tipos de los datos, el tipo defecto es el numérico.
row.names	Es un vector con el nombre de las filas. Puede ser un vector columna conteniendo los nombres reales de las filas. Puede ser el número de columna o el nombre de la columna de la tabla que contiene los nombres de las filas.
col.names	Es un vector con nombres opcionales para las variables, dado que sucede con frecuencia que el archivo de texto no contiene los nombres de las columnas. Al omitirse este paso R nombra a cada columna con una “V” seguido del número de columna.

Continuación de la tabla IV

Argumento	Descripción
na.strings	Vector que contiene las cadenas de texto que identifican los valores no disponibles "*NA*".
blank.lines.skip	Un valor lógico (TRUE o FALSE), que hace ignorar las líneas en blanco.
nrows	Número entero y no negativo de filas a ser leídas; los valores no válidos y negativos son ignorados.

Fuente: elaboración propia con ayuda de R <help(base)>.

Una forma opcional de nombrar las columnas es asignar posteriormente a la lectura del archivo de datos un vector de caracteres con igual número de cadenas de texto y en el mismo orden que las columnas del archivo de texto leído por medio de la función *names*(), ejemplo:

```
> tabla1<-read.table("mudiametromm2.txt",header=FALSE, sep="\t")
> tabla1
   V1     V2
1   1 45.678
2   2 45.493
3   3 45.680
:     :   :
19 19 45.596
20 20 45.555
>names(tabla1)
[1] "V1" "V2"
>names(tabla1)<-c("MUESTRA","DIAMETRO_MM")
>names(tabla1)
[1] "MUESTRA"     "DIAMETRO_MM"
```

Además, se puede acceder a datos ubicados en alguna dirección *URL*, por ejemplo:

```
> x<-read.table("http://www.mat.ucm.es/~palomam/aed/datos/datos1.dat")
>x
   V1
1   49
2  -67
3    8
:    :   :
14 -48
15  29
> mean(x$V1)
[1] 20.93333
```

1.4.2.4. Función *scan*( )

Con la función *scan*( ) es factible leer una fuente de datos, con los mismos argumentos de los anteriores, con el agregado de que se puede especificar el tipo de variables con el argumento *what* = *list* ( ). Dentro del paréntesis se especifica el tipo de datos, ya sea texto “” o numérico por ejemplo = “0”, el que es el tipo de datos por defecto a menos que se argumente lo contrario; asimismo, puede emplearse el formato numérico *double*(0).

Esto convierte a la estructura de datos resultante como una lista, de la cual puede a su vez extraerse a vectores cada tipo de datos contenido en la lista; ejemplo:

```
>inp<- scan("input.dat", list("",0,0))
>label<- inp[[1]]; x <- inp[[2]]; y <- inp[[3]]
```

```
>#  Se puede crear la lista dummy para nombrar los vectores componentes
>inp<- scan("input.dat", list(id="", x=0, y=0))
>#  Y asignar cada vector a uno externo a la lista
>label<- inp$id; x <- inp$x; y <- inp$y
```

1.4.2.5. Importar datos desde una hoja electrónica

La hoja electrónica es un formato muy común y versátil para manipular datos en forma tabular, por lo que es absolutamente necesario para un gran segmento de usuarios exportar datos hacia un objeto en R.

Ya se detalló el método manual para guardar los datos de una hoja electrónica a un archivo de texto o un formato de separación por comas; no obstante, también es posible leer la información directamente desde el archivo de hoja electrónica, el formato más habitual es de una hoja electrónica de MS Excel y existen sendas opciones específicas para leer formatos previos a la versión 2007 y posteriores a este.

Al contar con acceso a MS Excel en el mismo equipo donde se encuentra instalado R, una entre varias opciones, es guardar la información en formato de archivo de texto o en delimitación por comas. De no contar con MS Excel, para acceder a la hoja electrónica, es necesario tener instalado en la librería de R, el paquete RODBC mediante la función o *dbcConnectExcel*( ), la cual está disponible para los usuario de R de 32 bits para el formato .XLS, la cual cuenta con la capacidad de acceder a hojas, columnas y filas específicas.

Otro paquete que se recomienda para la lectura de datos en hojas electrónicas de MS Excel es XLConnect.

Siguiendo con el ejemplo del archivo “mudiametromm.xls”, los pasos a

seguir son los siguientes:

- Cargar el paquete RODBC con library(RODBC)

- Leer el archivo de MS Excel y crear el canal de conexión mediante la función *odbdConnectExcel*( ), el objeto “diámetro” ahora el canal queda en estado “abierto”.

```
>diametro<-odbcConnectExcel("mudiametromm.xls"); diametro
RODBC Connection 3
Details:
case=nochange
DBQ=C:\Rdata\mudiametromm.xls
DefaultDir=C:\Rdata
Driver={Microsoft Excel Driver (*.xls)}
DriverId=790
MaxBufferSize=2048
PageTimeout=5
```

- Listar las tablas disponibles del canal de conexión con la función *sqlTables*( ); las tablas son las hojas contenidas en el archivo.

```
>sqlTables(diametro)
TABLE_CAT
1 C:\Rdata\mudiametromm
TABLE_SCHEMTABLE_NAMETABLE_TYPE REMARKS
1       <NA>mudiametromm$ SYSTEM TABLE    <NA>
```

- Listar el contenido dentro de una tabla empleando la función *sqlFetch*()y asignar a un objeto tipo *Data Frame:*

```
>sqlFetch(diametro, "mudiametromm", rownames="")->diametro1; diametro1
   MUESTRA DIAMETRO_MM
1       1     45.678
2       2     45.493
3       3     45.680
:     :   :
19 19 45.596
20 20 45.555
```

En el ejemplo se asignó la información de la tabla leída a un objeto tipo "*data frame*" de nombre **diametro1** mediante la función *sqlFetch*( ); los argumentos son:

Tabla V. **Argumentos de función *sqlFetch*( )**

Argumento	Descripción
channel	Es el nombre de la conexión creada.
sqtable	Es el nombre de la tabla a leer.
colnames	Es un valor lógico (TRUE o FALSE) que indica si la primera fila tiene el nombre de las columnas. Por omisión el valor es TRUE.
rownames	Es un valor lógico o carácter con el nombre de las filas. Si es lógico, adopta la primera columna de la tabla como los nombres de las filas. Si es carácter, se especifica el nombre de la columna dentro de la tabla para el nombre de las filas.

Fuente: elaboración propia, con ayuda de R <help(base)>.

Al emplear la anterior función *sqlFetch*( ), se procede a almacenar los

datos de la consulta en un objeto de clase *data frame*.

```
>sqlFetch(diametro, "mudiametromm", rownames="MUESTRA")->dm
```

En el último paso se especificó que la columna que contiene el nombre de las filas es la que lleva el nombre de “MUESTRA”; en caso contrario, les asigna un número entero correlativo a cada una.

- Crear un objeto con la consulta con los valores deseados mediante una definición SQL, la cual consiste en un texto siempre y cuando cumpla con la estructura fundamental SQL de “SELECT – FROM – WHERE”, donde *SELECT* es el conjunto de campos o valores calculados a mostrar en la consulta; *FROM* es el conjunto de tablas (en este caso hojas) que contienen los campos (columnas) a mostrar y *WHERE* define los criterios de selección y llaves principales. Para ello se recomienda crear un objeto de clase “character” con escritura continua, es decir, sin saltos de línea

```
>sqldm<-"SELECT DIAMETRO_MM FROM[mudiametromm$]
WHEREDIAMETRO_MM> 45.6 AND DIAMETRO_MM< 45.65"
    >class(sqldm)
    [1] "character"
```

La definición del vector “sqldm” tiene un dato de tipo carácter (cadena de texto), el cual describe la selección de los datos del campo DIAMETRO_MM de la tabla **mudiametromm**, donde el campo DIAMETRO_MM tenga los valores mayores a 45,6 y menor a 45,65.

- Utilizar la definición de la consulta “sqldm” creada en el objeto anterior con la función *sqlQuery*( ):

```
    >sqlQuery(diametro, sqldm)
```

```
  DIAMETRO_MM
1      45.623
2      45.613
3      45.628
4      45.613
5      45.624
6      45.641
7      45.626
```

La sintaxis empleada en la anterior función es: sqlQuery([canal],[consulta sql]).

En el ejemplo se utilizó exclusivamente la columna DIAMETRO_MM, no obstante, en ocasiones es necesario dejar la referencia de cuál es el número de muestra original, por ello se deja a continuación un ejemplo completo de todo el proceso de consulta desde la conexión hasta la selección de datos:

```
>library(RODBC)
>diametro<-odbcConnectExcel("mudiametromm.xls")
>diametro
RODBC Connection 1
Details:
case=nochange
DBQ=C:\Rdata\mudiametromm.xls
DefaultDir=C:\Rdata
Driver={Microsoft Excel Driver (*.xls)}
DriverId=790
MaxBufferSize=2048
PageTimeout=5
>sqlTables(diametro)
TABLE_CAT
```

```
1 C:\Rdata\mudiametromm
TABLE_SCHEMTABLE_NAMETABLE_TYPE REMARKS
1       <NA>mudiametromm$ SYSTEM TABLE    <NA>
>sqlFetch(diametro,"mudiametromm")
  MUESTRA DIAMETRO_MM
1       1      45.678
2       2      45.493
3       3      45.680
:       :       :
19      19      45.596
20      20      45.555
>sqldm<-"SELECT  *  FROM  [mudiametromm$]WHERE  DIAMETRO_MM>  45.6
AND DIAMETRO_MM< 45.65"
>sqlQuery(diametro, sqldm)
  MUESTRA DIAMETRO_MM
1       4      45.623
2      11      45.613
3      13      45.628
:       :       :
7      18      45.626
```

Como se puede apreciar, se colocó un asterisco “ * “ para abreviar todos los campos de la tabla “mudiametromm”; esta a su vez debe de ir encerrada entre corchetes con la terminación del signo “ $ ”. Los resultados son aquellos registros que cumplen con los criterios.

Por último, para cerrar la conexión abierta se utiliza la función *odbcClose*() del paquete, con un único argumento que es el nombre del canal a cerrar. De no cerrarse el canal, será imposible actualizarlo desde el programa donde se crea. Si hubiera más de un canal de conexiones abiertas se puede utilizar la función *odbcCloseAll*( ).

```
>odbcClose(diametro)
>odbcCloseAll()
```

Un ejemplo con un formato .XLSX, en el que se tiene una muestra 30 grupos de 5 muestras de los pesos en gramos de azúcar blanca en presentación de sacos de kilogramo. Por razones de estandarización o legales, es necesario garantizar el peso que la etiqueta del saco indica. Para ello se cuenta con un archivo de MS Excel, del cual se calculan en consultas separadas los valores promedios para los valores mayores a 1000 y los menores a 1000 gramos.

```
>odbcConnectExcel2007("mupesogr.xlsx")->peso
>sqlpesom<-"SELECT * FROM [pesogramos$] WHERE
(peso1+peso2+peso3+peso4+peso5)/5 < 1000"
>sqlQuery(peso, sqlpesom)

MUESTRA peso1 peso2 peso3 peso4 peso5
1       2   986  1003   999   983   993
2       3  1011   987  1002   990  1002
3       4  1008  1006   998   996   990
:       :     :       :      :       :      :
22     27  1002  1010   991   996   994
23     28   988  1007  1002   981   985
24     30   983  1007   984  1006   986
>sqlpesom<-"SELECT * FROM [pesogramos$] WHERE
(peso1+peso2+peso3+peso4+peso5)/5 > 1000"
>sqlQuery(peso, sqlpesom)

MUESTRA peso1 peso2 peso3 peso4 peso5
1       1   987  1001  1011  1000  1008
2       7   995  1010  1010  1008   990
```

```
:     :    :     :     :     :     :
5    21  999   992  1003  1009  1004
6    29 1007   993   993  1004  1011
```

En el anterior ejemplo, solo las filas 1, 7, 16, 17, 21 y 29 cumplen con el criterio de selección, es decir, un promedio de fila mayor a 1000 gramos por unidad, las demás del bloque anterior no superan dicho promedio.

1.4.2.6. Importar datos con copy/paste (*clipboard*)

Una forma muy útil, inmediata y muy familiar a la mayoría de usuarios es usar la función copy / paste compatible con la mayoría de sistemas operativos. Asumiendo que se trabaja con datos residentes en una hoja electrónica de MS Excel, se procede a marcar el rango deseado incluyendo los encabezados o nombres de las variables, luego dar clic a copiar o en forma abreviada *Ctrl* + C.

Es necesario advertir, que el pegado o *paste* no es posible con *Ctrl* + V ni existe en R*Gui* un icono de “pegar”, en lugar de ello, se utiliza la función *read.delim*( ) para leer lo que está almacenado en memoria del ordenador, ya que R tiene la capacidad de utilizar esa información como un objeto con el argumento *clipboard*.

```
>copypaste<-read.delim('clipboard'); copypaste
      D1     D2     D3     D4     D5
1  45.678 45.597 45.514 45.514 45.512
2  45.493 45.557 45.592 45.620 45.580
3  45.680 45.539 45.549 45.525 45.596
:      :      :      :      :      :
19 45.596 45.604 45.609 45.574 45.622
20 45.555 45.549 45.601 45.530 45.604
```

1.4.2.7. Importar datos desde SPSS

Los paquetes *foreign* o *Hmisc* permiten acceder a los archivos en formato ".SAV" compatibles con todas las plataformas de SPSS. Para ejemplificar el procedimiento para importar la información se utiliza un archivo "ENEI_II_2013.SAV"[5]

```
>library(foreign)
>personas<-read.spss("ENEI_II_2013_PERSONAS.sav",to.data.frame= TRUE)
>dim(personas)
[1] 17981   249
>write.table(personas, "personas.txt", sep="\t",
+ na="NA", dec=",", row.names= TRUE, col.names = TRUE)
```

Se puede importar gran cantidad de información, en este caso es la base de datos de la Encuesta Nacional de Empleo e Ingresos 2013, la cual "tiene como principal objetivo obtener información estadística sobre variables de empleo, desempleo, subempleo y actividad e inactividad económica de la población". Además, se puede apreciar el método que se empleó para exportar los datos contenidos en el objeto "personas" hacia un archivo "personas.txt", el cual se logró mediante el uso de la función *write.table*( ) la cual abre el tema para exportar información.

1.4.3. Exportar datos

Mediante la función *write.table*( ), es posible crear archivos de texto (formato .TXT) a partir de los datos contenidos en objetos de R, por ejemplo un *data frame*, una lista, una matriz o simplemente un vector.Los argumentos necesarios de la función figuran en la siguiente tabla.

[5] Fuente: Instituto Nacional de Estadística –INE–.

Tabla VI. **Argumentos de función *write.table*( )**

Argumento	Descripción
x	Es el nombre del objeto cuyo contenido se va a exportar.
file	Es el nombre del archivo entre comillas (“ ”). De no contener ninguna ruta o *path* el archivo deberá estar contenido en el directorio de trabajo.
sep	Es el carácter de separación entre campos o variables, por defecto el carácter un espacio tabulado denotado con “\t” para el caso de ser un archivo.TXT. Si se tratara un .CSV el separador por defecto es una coma “,”.
na	Es la cadena de texto que indica cómo se visualizarán los valores perdidos.
dec	Es el carácter usado para punto decimal, por lo general es un punto “.”.
row.names	Valor lógico que indica si las filas tienen nombre o bien pueden ser un vector; por omisión es verdadero (*true*) e identifica mediante un vector de caracteres.
col.names	Al igual que *row.names* es un valor lógico o un vector indicando el nombre de las columnas.

Fuente: SAEZ, Antonio. *Métodos estadísticos con R y R Commander*. p. 39.

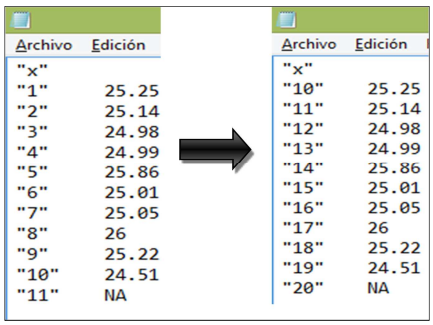
Tomando un ejemplo de un simple vector **x**, el cual contiene unos pesos en libras de un producto y se desea exportar los datos a un archivo de texto para su posterior manipulación y análisis, por defecto, las filas son numeradas, sin embargo, mediante el argumento row.names, se puede nombrar a las filas, ya sea un vector de números o de caracteres; en este caso se observa la numeración de 10 a 20 del lado derecho de la figura 10.

```
> x<-c(25.25,25.14,24.98,24.99,25.86,
+ 25.01,25.05,26,25.22,24.51,NA); x
[1] 25.25 25.14 24.98 24.99 25.86 25.01 25.05 26.00 25.22 24.51    NA
>write.table(x, file="pesoslibras.txt", sep="\t")
>write.table(x, file="pesoslibras.txt", sep="\t", dec=".",
+ row.names=c(10:20), na="NA")
```

Para crear un archivo con formato del formato .CSV es necesario variar la extensión de .TXT a .CSV y el argumento del separador "*sep*" de tabulador por una coma " , ", mientras los demás argumentos funcionan por igual.

```
>write.table(x, file="pesolibras.csv", sep=",", dec=".", col.names="pesoslb
```

Figura 10. **Archivo de texto con write.table**



Fuente: toma de pantalla de archivo de texto.

En resumen, se pueden manipular datos desde afuera y hacia afuera de R. En tal sentido al terminar este punto es posible volver a leer el archivo .TXT o el .CSV mediante las funciones *read.delim*( ) y *read.csv*( ) respectivamente o genéricamente con la función *read.table*( ).

En ocasiones es necesario escribir no solo los datos, sino los resultados de una función o cualquier expresión evaluada. Para ello se utiliza la función *sink*( ), la cual tiene como argumento una cadena de texto entre comillas con el nombre del archivo de texto o de extensión TXT, donde se escriben todas las salidas del *script*. El archivo de texto, queda en el directorio de trabajo, para anular la instrucción de guardar en el archivo descrito, se escribe "*sink*( )", es decir, sin argumento.

1.4.4. Entorno gráfico R Commander (paquete Rcmdr)

R Commander es un paquete de plataforma independiente de entorno gráfico, para ejecutar comandos y funciones estadísticas de forma abreviada, para lo cual funciona con menús y submenús.

Como no forma parte del núcleo base de R, *R Commander* necesita descargarse e instalarse desde un *CRAN.* Al buscarlo, el paquete tiene el nombre de Rcmdr, la búsqueda también mostrará más paquetes con nombres similares, los cuales son complementos optativos al paquete base de *R Commander*. Una vez instalado el paquete o cualquier otro, ya es posible cargarlo, para lo cual hay dos vías, la primera es mediante los menús propios de R, con la siguiente secuencia:

- Menú: paquetes
  - Opción: cargar paquete….

- Seleccionar con un clic y aceptar el paquete deseado de la lista de opciones, como la muestra la ilustración de la derecha.

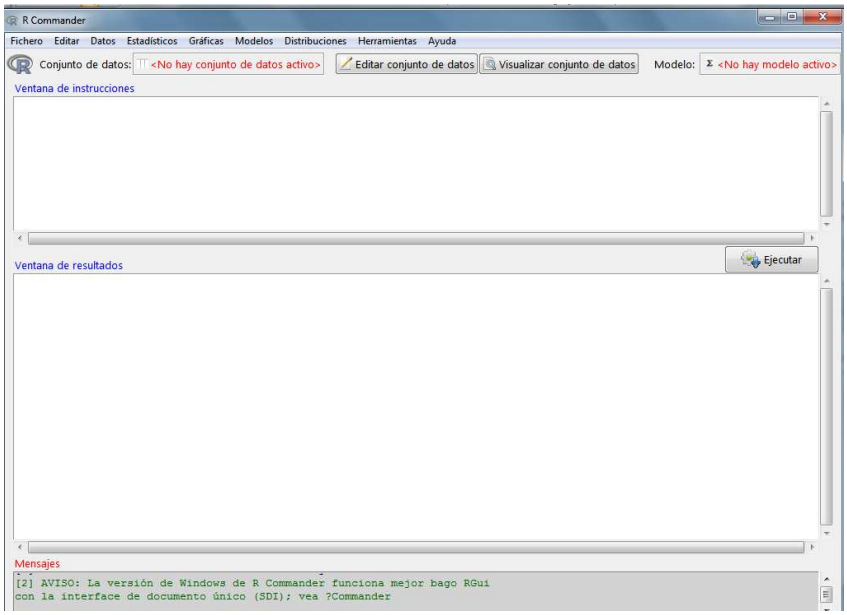
La segunda vía y más sencilla forma de cargar este paquete y otros es mediante la función *library*(paquete) escrita en el *prompt.*

```
>library(Rcmdr)
Loading required package: splines
Loading required package: car
VersióndelRcmdr 2.2-4
```

Como se aprecia en la figura 11, la pantalla principal de *R Commander* tiene una estructura consistente en una barra de menús, una barra de datos, una ventana de instrucciones (*R script*) y una de resultado o salida. La ventana de *R script* muestra la instrucción que el lenguaje traduce del entorno gráfico a lenguaje formal de R, en la que se pueden escribir funciones y *script*s para ejecutar, lo que se puede hacer al dar clic en el botón “Ejecutar” que se encuentra al pie de la ventana de *R script* ; más abajo se observa un cuadro para mensajes.

Se sugiere al usuario principiante, apoyarse en el uso de *R Commander* para el aprendizaje de R, ya que cada opción en los menús desplegables equivale a uno o un conjunto de instrucciones propias de R, en otras palabras, *R Commander* es un interface gráfica del lenguaje de R. A continuación se presentan las diferentes opciones de *R Commander* disponibles en la versión 2.2-4.

Figura 11. **Pantalla principal de *R Commander***



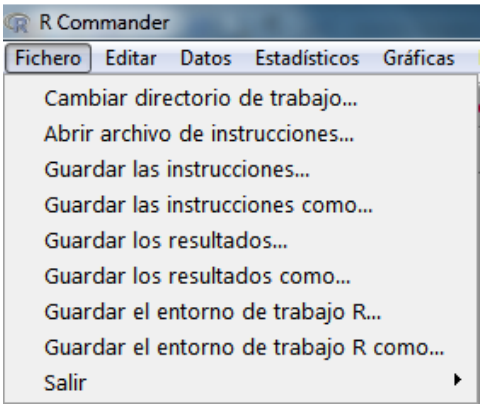
Fuente: toma de pantalla de *R Commander* 2.2-4.

1.4.4.1. Menú “Fichero”

Mediante este menú es factible gestionar y usar datos depositados en ficheros externos al entorno de programación y de los paquetes, así como guardar resultados y el área de trabajo.

En el menú “Fichero” (figura 12), también se puede encontrar la opción de cambiar la ruta de trabajo.

Figura 12. **R Commander, menú “Fichero”**

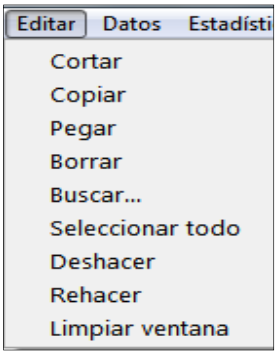


Fuente: toma de pantalla *R Commander* 2.2-4.

1.4.4.2. Menú “Editar”

Aporta las funciones estándares de operación de datos e información, como cortar, copiar, pegar borrar, seleccionar, entre otras funciones aplicables a la “Ventana de instrucciones” (ver la figura 13).

Figura 13. **R Commander, menú "Editar"**



Fuente: toma de pantalla de *R Commander* 2.2-4.

1.4.4.3. Menú “Datos”

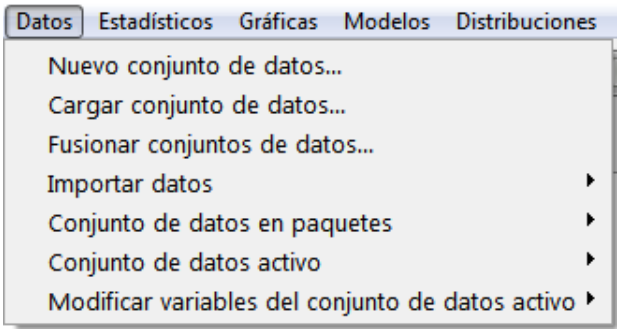
En la figura 14 se observa que es la instancia donde en forma general se cargan a la memoria los datos a ser analizados. Acá las opciones hacen lo mismo, como si se tratase de la consola principal de R, por ejemplo al importar datos desde el archivo mudiamtro.csv, la ventana R *Script* muestra la siguiente salida.

```
Datos<- read.table("C:/path/mudiametromm.csv", header=TRUE, sep=",",
na.strings="NA", dec=".", strip.white=TRUE)
fix(Datos)
```

Y en la ventana de Salida se muestra lo siguiente:

```
>Datos<- read.table("C:/path/mudiametromm.csv", header=TRUE, sep=",",
+   na.strings="NA", dec=".", strip.white=TRUE)
>fix(Datos)
```

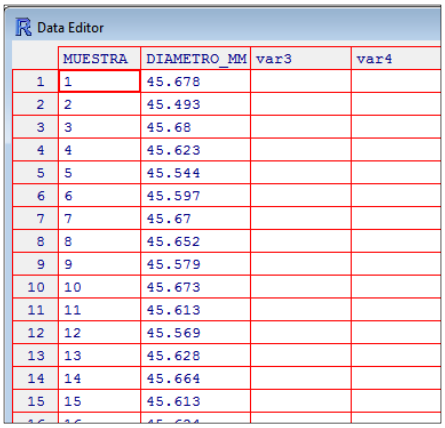
Figura 14. ***R Commander*, menú "Datos"**



Fuente: toma de pantalla de *R Commander* 2.2-4.

Por último, se ordena editar los datos del conjunto de datos, por lo que abre la misma ventana que se abriría en R para editar los datos en forma tabular (ver figura 15).

Figura 15. ***R Commander*, ventana para edición de datos**



	MUESTRA	DIAMETRO_MM	var3	var4
1	1	45.678		
2	2	45.493		
3	3	45.68		
4	4	45.623		
5	5	45.544		
6	6	45.597		
7	7	45.67		
8	8	45.652		
9	9	45.579		
10	10	45.673		
11	11	45.613		
12	12	45.569		
13	13	45.628		
14	14	45.664		
15	15	45.613		

Fuente: toma de pantalla de *R Commander* 2.2-4.

La ventana de R *script* s mostrará de forma similar a lo mostrado en con el paquete RODBC cuando se tratase de importar datos provenientes de hojas electrónicas, por ejemplo:

- Datos
  - Importar datos
    - Desde conjunto de datos MS Excel, MS Access, dBase…

Digitar el nombre de objeto a crear (diámetro) con el conjunto de datos; en este caso archivo de MS Excel (mudiametromm.xls).

```
diametro<- sqlQuery(channel = 1, select * from [mudiametromm$])
```

La estructura para importar datos es la misma a la visto anteriormente, con un poco más de simplicidad en los pasos, ya que no se necesita de conocer las funciones y el lenguaje de R, sin embargo no aporta la posibilidad de personalizar el *query* para refinar el conjunto de datos a utilizar. Para ello es preciso revelar que esta opción no fuese posible sin que se tuviera de respaldo el paquete RODBC, ya que al llamar a esta opción se abre dicho paquete; de no tenerlo en la librería de paquetes no fuera posible importar datos desde bases de datos incluyendo libros de MS Excel.

Otra opción muy útil, es crear el conjunto de datos desde cero mediante la secuencia:

- Datos
  - Nuevo conjunto de datos

Luego digitar el nombre a utilizar en el nuevo conjunto de datos. Al abrirse automáticamente la ventana para edición de datos, se puede con dar clic al título, asignar nombre personalizado a cada columna, campo o variable y cambiar el tipo de dato, por defecto es numérico y se puede cambiar a carácter. El *script* para efectuar esa operación es:

```
Datos <- edit(as.data.frame(NULL))
fix(Datos)
```

Como se puede apreciar, el objeto por defecto creado es del tipo *data frame*.

Otra opción útil para practicar con datos es utilizar los datos adjuntos con los paquetes que están en el núcleo base o en otros paquetes disponibles,

siguiendo el orden de:

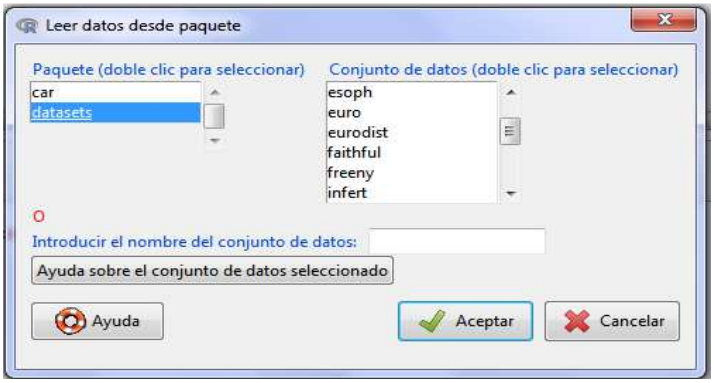
- Datos
  - Conjunto de datos en paquetes
    - Leer conjunto de datos desde paquete adjunto…

El anterior procedimiento abre la ventana según la figura 16, en la cual hay dos paquetes que tienen a su vez datos adjuntos que pueden ser útiles para la comprensión de la utilización de los mismos y de otras opciones internas de *R Commander*.

Si a partir de los datos existentes se quisiera calcular una nueva variable, se procede siguiendo la siguiente secuencia:

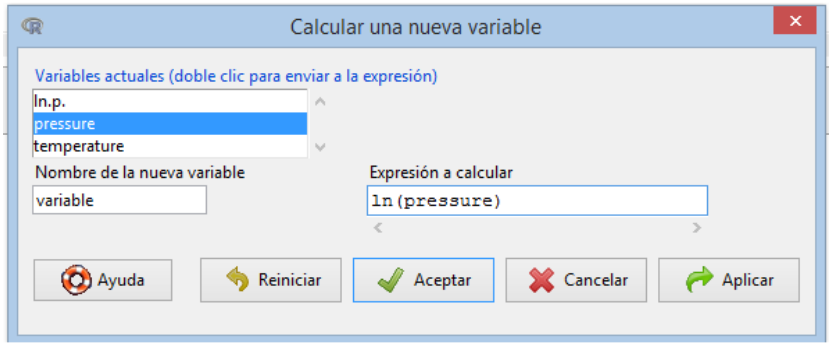
- Datos
  - Modificar variables del conjunto de datos activo
    - Calcular una nueva variable

Figura 16**.** ***R Commander*, lectura de datos desde paquete**



Fuente: toma de pantalla de *R Commander* 2.2-4.

Figura 17. ***R Commander*, cálculo de una nueva variable**



Fuente: toma de pantalla *R Commander* 2.2-4.

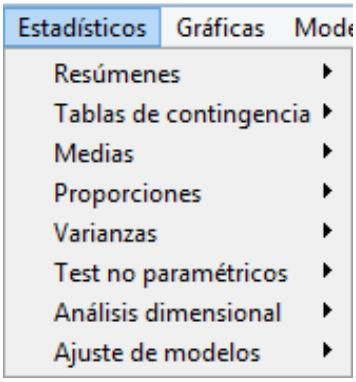
Estas trasformaciones son optativas, ya que en modelos econométricos o simplemente el modelo a evaluar pueden especificarse directamente sobre una variable y sus transformaciones implícitas. No obstante, lo correcto es especificar modelos en función de la teoría.

En los pronósticos industriales, este tipo de manipulaciones tiene gran utilidad, ya que no se necesita hacer más cálculos posteriores, en cambio, se pueden calcular las regresiones utilizando las funciones logarítmica, exponencial o recíproca en su forma lineal. En el capítulo 5 se detalla el procedimiento para estos cálculos.

1.4.4.4. Menú “Estadísticos”

Resaltan de este menú las opciones que permiten hacer análisis estadístico descriptivo e inferencial (ver figura 18).

Figura 18. ***R Commander*, menú "Estadísticos"**



Fuente: toma de pantalla *R Commander* 2.2-4.

Los siguientes resultados fueron adaptados manualmente para observarse en este formato; haciendo uso de los datos de diámetros importados en el menú de datos, el resultado de salida, con la opción “Resúmenes numéricos…” es:

```
>numSummary(Datos[,"DIAMETRO_MM"], statistics=c("mean", "sd", "IQR",
"quantiles", "cv", "skewness", "kurtosis"), quantiles=c(0,.25,.5,.75,1), type="2")

mean     sd              IQR        cv           skewness      kurtosis
0%       25%             50%             75%          100%          n
 45.6159                 0.04977729 0.06325 0.001091227 -0.7571017 0.3912548
45.493   45.59175        45.6235         45.655       45.68         20
```

La anterior función es utilidad cuando se desea hacer estadísticos resumidos por grupos y luego la variables por separado. La opción “Conjunto de datos activo”, equivale a la función *summary*(variable) en el *prompt* de R. Los resultados son exactamente los mismos, ejemplo:

```
>summary(Datos)
```

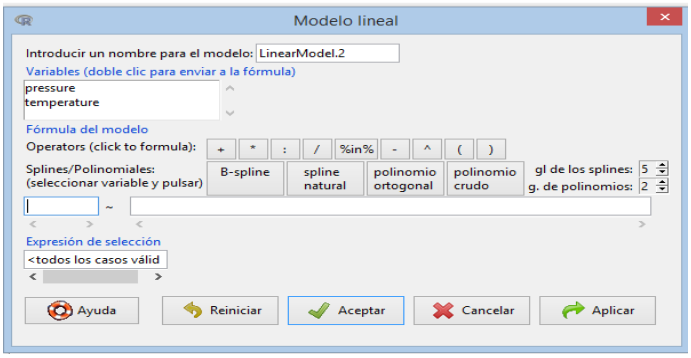
```
MUESTRA    DIAMETRO_MM
Min.: 1.00         Min.:45.49
1st Qu.: 5.75      1st Qu.:45.59
Median :10.50   Median :45.62
 Mean   :10.50   Mean   :45.62
 3rd Qu.:15.25             3rd Qu.:45.66
Max.   :20.00      Max.   :45.68
```

Como se mencionó, se puede utilizar el *script* de entrada para refinar los datos de salida, por ejemplo:

```
>summary(Datos$DIAMETRO_MM)
   Min.   1st Qu.         Median         Mean   3rd Qu. Max.
  45.49    45.59          45.62          45.62   45.66    45.68
```

En el submenú de “Ajuste de modelos” se calculan modelos de regresión predefinidos como el modelo regresión lineal simple con dos variables y multivariable, así también, permite especificar puntualmente el modelo, (ver figura 19).

Figura 19. ***R Commander*, ajuste de modelo**



Fuente: toma de pantalla de *R Commander* 2.2-4.

1.4.4.5. Menú “Distribuciones”

Este menú contiene las distribuciones estadísticas acumuladas, la probabilidad de densidad, cuartiles y las respectivas gráficas de distribuciones estandarizadas, que en efecto, son útiles e ideales como sustitutos para el análisis estadístico, ya que se pueden utilizar como referencia de las tablas y gráficas estadísticas y desde luego emplear dicha información en simulaciones.

Más adelante, en el inciso 1.8, se trabaja con mayor profundidad las distribuciones de probabilidad en R, que servirán de fundamento estadístico en las curvas de operación y en los muestreos de aceptación; no obstante, en este contexto es necesario plantear un ejemplo con una variable x, normalmente distribuida con media μ y varianza σ^2, cuya notación es $x \sim N(\mu, \sigma^2)$, implica además que la curva de la distribución normal es simétrica respecto de la media y unimodal conocida como la curva de campana[6].

1.4.4.6. Menú “Herramientas”

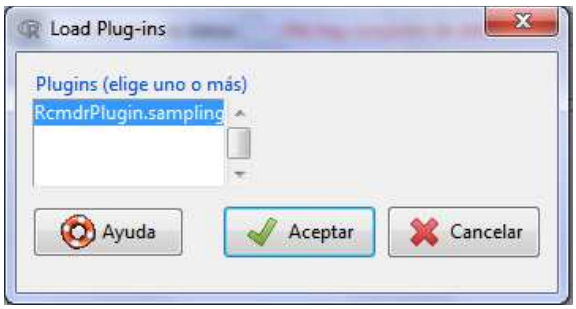
El menú de herramientas contiene los accesos para cargar paquetes a R y cargar los *plugin* para *R Commander*, en tal caso hay que bajarlos previamente del *CRAN*, para que aparezcan en la lista.

Al cargar el *plugin* con la opción que muestra la figura 20, se agrega a la barra de menús principal un nuevo menú con el nombre de este. En este caso el *plugin Rcmdr Plugin.sampling*, será de gran utilidad más adelante. La opción para cargar paquetes, replica la misma operación manual de *library*(paquete). En cuanto a “Opciones…”, es el acceso a personalizar la forma en que se

[6] MONTOGOMERY, Douglas. *Introduction to statistical quality control*. p. 61.

visualiza y como funciona *R Commander,* en relación con el manejo de memoria.

Figura 20. ***R Commander*, cargar plugins**



Fuente: toma de pantalla de *R Commander* 2.2-4.

1.4.5. Entorno gráfico *R Studio*

El entorno de *R Studio*, es una plataforma más amigable para interactuar con datos y el lenguaje R. A su vez, también cuenta con la característica de ser un software de código abierto. La plataforma principal del mismo está programada con el lenguaje C ++. La última versión de *R Studio* está disponible en http://www.rstudio.com; la empleada en este documento es la 0.99.902 para sistema operativo basado en Windows ©. *R Studio* funciona de forma independiente, es decir, que es otro programa, pero vinculado a núcleo base de R, por lo que se pueden utilizar los paquetes instalados en la librería.

Al observar la figura 21, el área “a” que señala la barra de menú, contiene los accesos básicos de edición (menú *Edit*); los referidos a los archivos de código, de sesión, área de trabajo (menús *File* y *Session*). Asimismo contiene las herramientas en el menú *Plots* como apoyo para la elaboración de gráficas. Las herramientas del menú *Debug* se utilizan para depurar *scripts*. El área “b”, es la consola de comandos donde se ingresan las instrucciones y muestran los

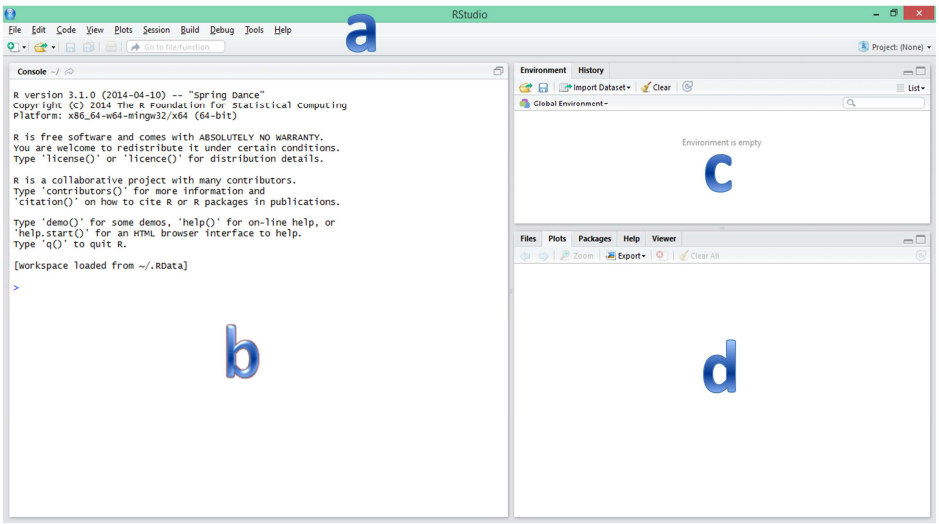
resultados y en la parte superior, las fuentes o *Source*y los datos en forma tabular.

En el área “c” se muestran los objetos en memoria (ambiente) y el historial de las instrucciones procesadas. Al dar clic a uno de los objetos en memoria (datos, valores y funciones), equivale a la función *View*( ).

Si al abrir R Studio aparecen objetos en memoria, quiere decir que se reabrió con la anterior área de trabajo y sesión guardada.

En el área “d”, se encuentra la información en etiquetas de los archivos contenidos en el directorio de trabajo *File*, los resultados de la gráficas instruidas *Plots*, los paquetes instalados en la librería de R y la opción para instalar otros, etiqueta *Packages*; por último la ayuda *Help* que muestra el índice general de referencia de R.

Figura 21. ***R Studio*, pantalla principal**



Fuente: toma de pantalla *R Studio* 0.99.902.

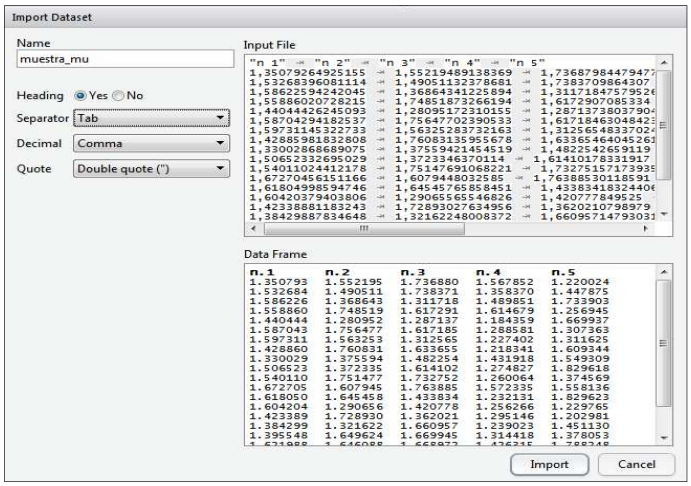
1.4.5.1. Importar datos

Para importar datos debe aplicarse la siguiente secuencia:

- Menú *Tools.*
- Opción *ImportDataset.*
- Seleccionar *Heading* (*Yes*), cuando el archivo de texto tiene encabezado.
- Seleccionar el separador de cada campo, en el ejemplo es separado por comas, formato CSV.
- Seleccionar el par de comillas para leer columnas como texto, por defecto es (“ “).

En el procedimiento anterior se visualiza mejor observando la figura 22 que aplica tanto para el formato .CSV, como .TXT, por ello el *script* de salida se muestra a continuación:

Figura 22. ***R Studio*, importar datos**



Fuente: toma de pantalla *R Studio* 0.99.902.

Como se puede apreciar, al igual que *R Commander*, con *R Studio*, importar datos por el método visual resume los principales argumentos de cada función.

1.5. Operaciones fundamentales

Este inciso trata sobre las operaciones básicas a enunciar y evaluar en la programación de los *script* s de las propuestas en los capítulos posteriores. En la sección 1.3.2.2 se inició el tema con las operaciones aritméticas fundamentales. En el capítulo 2 se detallan aspectos más complejos relacionados con los objetos del tipo expresión o *expression*, más apropiadas para la evaluación interna y secuencial dentro de un *script*, entiéndase, temas enfocados al lenguaje R propiamente dicho.

1.5.1. Manipulaciones aritméticas y estadísticas

No es posible referirse a operaciones estadísticas básicas sin hablar de operaciones aritméticas como la suma, resta, producto, división, potenciación, y agrupación; aunque es obvio que R y paquetes para el mismo cuenta con sus propias funciones estadísticas para hacer tratamientos más complejos. En tal sentido, este inciso tiene por objetivo mostrar las funciones estadísticas básicas y el equivalente a lo que posteriormente se ampliará con la revisión de paquetes específicos en estadística.

1.5.1.1. Medidas de tendencia central y de dispersión

Para el efecto, se emplea una fuente propia de datos, consistentes en 20 muestras de tamaño cinco, de diámetros internos (como variable a controlar) de

piezas de aluminio que pasan por un proceso de torno, tal y como se verán ejercicios para las cartas de control estadístico de calidad. Se emplea el paquete RODBC para conectarse al archivo de hoja electrónica con nombre mudiametromm2.xls.

```
>cdiam<-odbcConnectExcel("mudiametromm2.xls")
>sqlTables(cdiam)
TABLE_CATTABLE_SCHEMTABLE_NAMETABLE_TYPE REMARKS
1 C:\\Rdata\\ mudiametromm2 <NA>mudiametromm$ SYSTEM TABLE
<NA>
>sqlFetch(cdiam, "mudiametromm", rownames="MUESTRA")->diametro2
> diametro2
 D1    D2    D3    D4    D5
1  45.678 45.597 45.514 45.514 45.512
2  45.493 45.557 45.592 45.620 45.580
:     :     :     :     :     :
19 45.596 45.604 45.609 45.574 45.622
20 45.555 45.549 45.601 45.530 45.604
```

Una vez los datos cargados en la memoria con el objeto **diametro2** tipo *data.frame* (hoja de datos) se puede proceder ya a hacer manipulación de los datos y los cálculos que interesan en este inciso. Algo fundamental es calcular sumatorias en forma horizontal de cada muestra o en forma vertical de cada columna D_i del ejemplo con la función *sum*( ) y utilizando las técnicas de índices aplicable a matrices, vectores y marcos de datos.

```
> #Sumar horizontalmente la muestra 10
>sum(diametro2[10,])
[1] 227.927
> #Sumar la muestra 15 a la 20
>sum(diametro2[15:20,])
```

```
[1] 1367.594
> #Sumar los valores del diámetro 1 "D1"
>sum(diametro2$D1)
[1] 912.318
> #Equivalente a:
>>sum(diametro2[,1])
[1] 912.318
> #Sumar los valores del diámetro 1 "D1" de la muestra 1 a la 5
>sum(diametro2$D1[1:5])
[1] 228.018
```

Es muy probable que se desee hacer la sumatoria horizontal de cada fila de diámetro D_i para calcular la sumatoria como un nuevo campo o variable "sumDi", con la función *with*( ) cuyos argumentos son "datos" y expresión; el símbolo "$" entre el objeto y el nombre del nuevo campo de campo (tabla$campo), ejemplo:

```
> diametro2$sumDi <- with(diametro2, D1+ D2+ D3+ D4+ D5)
> diametro2
 D1     D2     D3     D4     D5  sumDi
1  45.678 45.597 45.514 45.514 45.512 227.815
2  45.493 45.557 45.592 45.620 45.580 227.842
:      :      :      :      :      :      :
19 45.596 45.604 45.609 45.574 45.622 228.005
20 45.555 45.549 45.601 45.530 45.604 227.839
```

De la misma forma con que se determinó el campo de la suma, en control estadístico de calidad es necesario calcular el promedio de cada grupo de muestras (ver anexo 1). Creando una variable nueva de nombre **xprom** $(\bar{X})$ con la siguiente instrucción:

```
> diametro2$xprom<-with(diametro2,sumDi/5)
> diametro2
       D1      D2      D3      D4      D5     sumDixprom
1  45.678 45.597 45.514 45.514 45.512 227.815 45.5630
2  45.493 45.557 45.592 45.620 45.580 227.842 45.5684
3  45.680 45.539 45.549 45.525 45.596 227.889 45.5778
:      :       :       :       :       :       :       :
19 45.596 45.604 45.609 45.574 45.622 228.005 45.6010
20 45.555 45.549 45.601 45.530 45.604 227.839 45.5678
```

Por último, se tiene el promedio de los promedios (ver en anexo 1 el formulario completo) **Xprom**$(\bar{\bar{X}})$ más simple de calcular.

```
>mean(diametro2$xprom)
[1] 45.58186
```

Posteriormente se debe calcular el rango (*R*) por muestra y el rango general $(\bar{R})$**,** pero se desarrollará posteriormente, ya que previamente se necesita introducir a temas como las declaraciones y los ciclos. Antes de ello, aprovechando la información del objeto **diametro2,** los datos se convierten a un vector **diametro3,** de largo 100 y clase *numeric o* numérico, conteniendo en orden los registros de cada diámetro D_i.

Observar el empleo de la función *as.vector*( ), que fuerza un objeto como una *data frame* o una matriz y/o cualquiera de sus variables o columnas a fusionarse en un vector como nuevo objeto de nombre **diametro2** del tipo numérico.

```
> diametro3<-as.vector(c(diametro2$D1,diametro2$D2,
+ diametro2$D3, diametro2$D4, diametro2$D5),
```

```
+ mode = "numeric")
> diametro3
  [1] 45.678 45.493 45.680 45.623 45.544...
[22] 45.557 45.539 45.545 45.566 45.599…
[43] 45.549 45.599 45.579 45.544 45.535…
[64] 45.543 45.590 45.535 45.600 45.558…
[85] 45.621 45.596 45.534 45.584 45.558…
```

Esto permite tener información para continuar con el tema de estadístico de las medidas de tendencia central. El análisis inicia con la función *summary*(), la cual dependerá de la clase de objeto el resultado. En este caso, siendo un vector numérico, da como resultado el resumen de los mínimos, máximos, primer cuartil, mediana (segundo cuartil), media y tercer cuartil.

```
>summary(diametro3)
Min. 1st Qu.Median    Mean 3rd Qu.    Max.
45.49   45.55   45.58   45.58   45.61   45.68
```

Es información que indica en forma sencilla las medidas de tendencia central, sin embargo, se puede obtener información más amplia, haciendo uso del *script* de salida que se vio en la sección en el capítulo 1, al emplear *R Commander*.

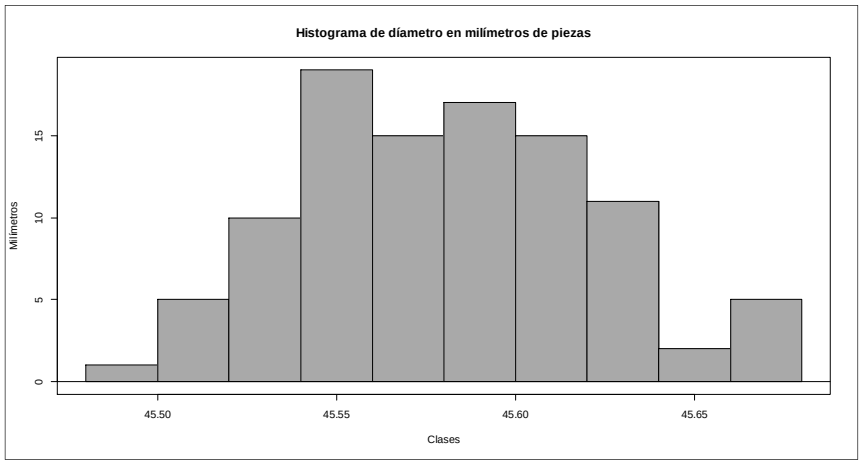
```
>numSummary(diametro3[,"D"], statistics=c("mean", "sd", "IQR", "quantiles", "cv",
"skewness", "kurtosis"), quantiles=c(0,.25,.5,.75,1), type="2")
meansd IQR  cv  skewness   kurtosis
0%    25%    50%    75%  100%   n
45.58186 0.04064605 0.06 0.0008917155 0.2510087 -0.3486197
45.493 45.549 45.582 45.609 45.68 100
```

Tomando una lectura de los resultado numéricos tal y como se colocaron con los argumentos de la función *numSummary*( ), se observa ahora mayor

información respecto de la distribución de los datos del vector **diametro3** con **D** como única variable.

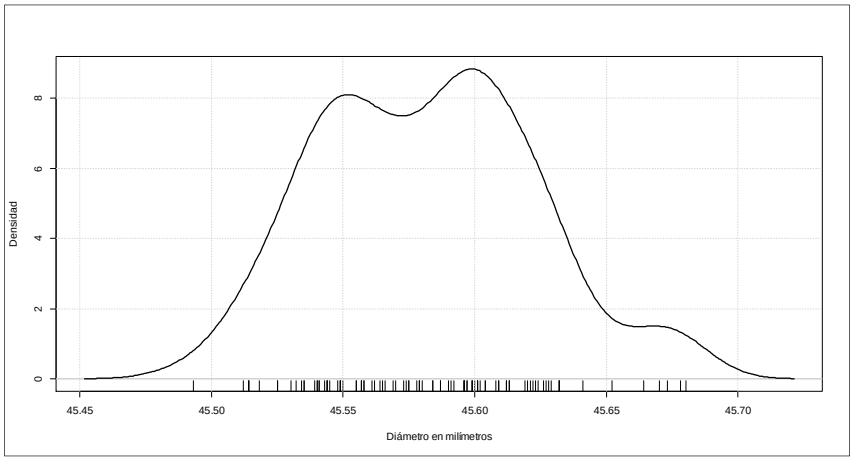
Con una desviación estándar “sd” de 0,04064605; el recorrido intercuartílico “IQR” de 0,06 que quiere decir que entre el cuartil uno y el tres, hay un rango de 0,06 milímetros, un coeficiente de variación “cv” de 0,0008917155, un coeficiente de sesgo o de asimetría positivo de 0,2510087 y una curtosis de -0,348697, es decir una curva platicúrtica. En la figura 23se pueden apreciar los estadísticos indicados, además al suavizar la curva como en la figura 24, tratándose de datos continuos y a medida que se incrementa la muestra, la distribución de probabilidad tiende a asemejarse a una distribución normal.Por último, en la figura 24 se observa una distribución bimodal, representada en los valores cercanos a 45,55 mm y 45,60 mm.

Figura 23. **Histograma de díámetros**



Fuente: elaboración propia, empleando la función *hist*( ).

Figura 24. **Curva con ajuste gaussiano**



Fuente: elaboración propia, empleando *R Commander*.

1.6. Gráficos y paquetes de gráficos

En el apartado anterior se utilizaron a manera de apoyo visual dos gráficas que representan la distribución de los datos del diámetro interno de piezas expresado en milímetros: histograma y la función de densidad, respectivamente. Para ello se recurrió al uso de las gráficas que *R Commander* permite elaborar. No obstante, hay diversos paquetes que proporcionan una funcionalidad más avanzada.

Previo a mostrar las opciones para gráficos de *R Commander* y otros paquetes, es necesario revisar brevemente las principales funciones contenidas en el paquete *Graphics*[7] del núcleo base. En dicho paquete base están definidos tres grupos fundamentales para plotear gráficas[8]:

[7] R, help(graphics).

[8] VENABLES, William. *An introduction to R*. p. 68.

- Alto nivel: permite hacer gráficos con todos los parámetros básicos y los optativos de configurar los ejes, etiquetas, leyendas, títulos, intervalos, colores, entre otros; una vez elaborada una gráfica, la siguiente es escrita sobre la anterior, es decir que es una totalmente nueva y hay que declarar nuevos argumentos incluyendo el objeto que contiene la información.

- Bajo nivel: consiste fundamentalmente en emplear más argumentos que permitan mostrar más información gráfica, texto y estilo. Lo anterior implica alternar los argumentos del alto y bajo nivel.

- Interactivos: permiten agregar y extraer datos de un gráfico con un dispositivo para señalar como un *mouse* o un *pad*.

1.6.1. Función de gráficos *curve*()

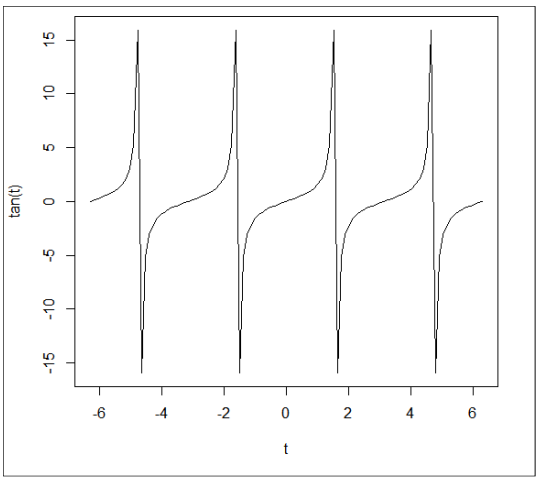
La función *curve*( ) dibuja una gráfica, con "expresión" y el intervalo [a, b] como principales argumentos. Por expresión, se entiende a una expresión matemática cualquiera y el intervalo limita la amplitud de la gráfica; en el siguiente ejemplo se dibujó la expresión trigonométrica $f(t) = \tan t$ en el intervalo$(-2\pi,\ 2\pi)$; ver los argumentos de la función seguido de la figura 25.

curve(expr, from = NULL, to = NULL, xname = "x")

- *expr:* Nombre de la función o la expresión misma como función de x.
- *xname*: el nombre del eje x.

```
>curve(tan, -2*pi, 2*pi, xname="t")
```

Figura 25. **Gráfica con función *curve***



Fuente: elaboración propia, empleando función *curve*( ).

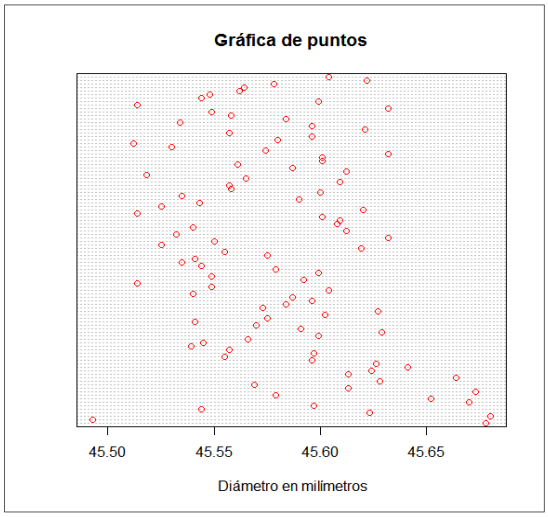
1.6.2. Función de gráficos *dotchart*( )

El gráfico de puntos o *dotchart* son un sustituto ideal para la gráfica de barras en la presentación de la distribución de datos. Siguiendo con el ejemplo de los diámetros internos obtenidos en un proceso de torno en piezas de aluminio, se observa que a medida que los datos de los diámetros se acercan a la media, se yuxtaponen uno sobre otro los puntos rojos, según la figura 26.

```
> dotchart(diametro3, main="Gráfica de Puntos",
+ color = "red", xlab="Díametro en milímetros")
```

Observar los argumentos empleados previamente al gráfico de la figura 26, con *main* para identificar el título principal del gráfico; *xlab* para detallar el título en el eje X, y *color* para configurar los puntos del gráfico. Es necesario mencionar que estos parámetros son aplicables para otros tipos de gráficos.

Figura 26. **Gráfico de puntos, función *dotchart*( )**



Fuente: elaboración propia, empleando función *dotchart*( ).

1.6.3. Función de gráficos *plot*( )

Esta función relaciona pares de variables tipo X-Y, para auxiliar al análisis y detección de posible comportamiento determinístico. Por ejemplo, para este tipo de gráfico se anticipará el uso de los objetos tipo *ts* o *time* series por su nombre en inglés para visualizar la evolución en el tiempo de la varianza en cada mes desde septiembre 2011 a septiembre 2014 de diferentes muestras de los diámetros internos de las piezas fabricadas en procesos de torno. Las varianzas de una serie de 36 meses son llevados a un objeto **temp** tipo *data.frame*, que captura la información desde una hoja de MS Excel, luego se traslada la información de la variable **V1**al objeto **var_d** de tipo *ts*.

```
> temp<-read.delim('clipboard', header=FALSE)
> class(temp)
[1] "data.frame"
```

```
> dim(temp)
[1] 36  1
> var_d<-ts(c(temp$V1), start=c(2011,9), frequency=12)
> class(var_d)
[1] "ts"
```

Las propiedades del objeto *ts* se discutirán más adelante; se deja de momento la muestra del despliegue de la información del objeto **var_d** (ver figura 27). Ahora se procede a plotear los valores de las varianzas (Y) respecto del tiempo (X), mediante la función gráfica *plot*( ). Notar la variabilidad en relación con el tiempo en los primeros dos años, mientras que en el último año se estabiliza la variación; en términos estadísticos se dice que hay presencia de heteroscedasticidad de los datos.

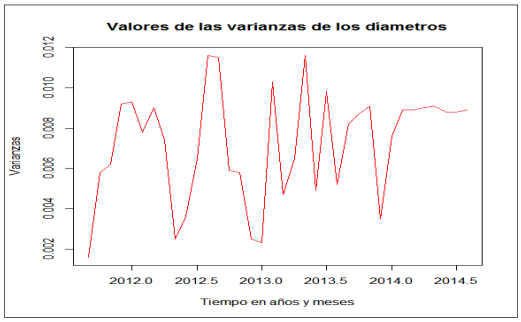
Figura 27. **Serie de tiempo**

```
> var_d
        Jan    Feb    Mar    Apr    May    Jun    Jul    Aug    Sep    Oct
2011                                                             0.0016 0.0058
2012 0.0093 0.0078 0.0090 0.0074 0.0025 0.0036 0.0065 0.0116 0.0115 0.0059
2013 0.0023 0.0103 0.0047 0.0065 0.0116 0.0049 0.0098 0.0052 0.0082 0.0087
2014 0.0076 0.0089 0.0089 0.0090 0.0091 0.0088 0.0088 0.0089
        Nov    Dec
2011 0.0062 0.0092
2012 0.0058 0.0025
2013 0.0091 0.0035
2014
```

Fuente: toma de pantalla de RGui.

En función de las causas asignables para el aseguramiento de calidad, la tendencia a “no igual” varianza respecto del tiempo supone la existencia de una “curva de aprendizaje”, ya sea por el proceso de aprendizaje atribuido a operarios nuevos, introducción de métodos de producción nuevos o variación de los mismos, pero dada alta variabilidad en un periodo prolongado del periodo de análisis, (ver figura 28), hace suponer un proceso “fuera de control”, sin haber existido ninguna acción correctiva.

Figura 28. **Gráfica con función *plot*( )**



Fuente: elaboración propia, empleando función *plot*( ).

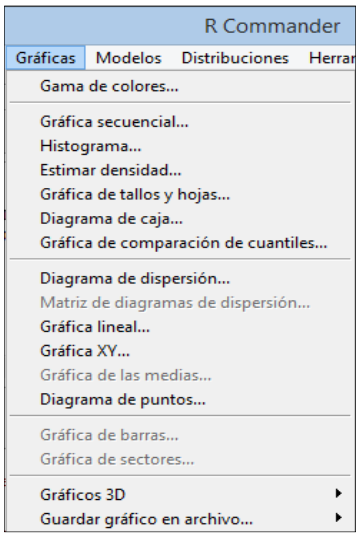
```
> plot(var_d, main="Valores de las varianzas de los diámetros",
+ xlab="Tiempo en años y meses", ylab="Varianzas", col="red")
```

Otros argumentos contenidos en el paquete *Graphics* serán ejemplificados en los siguientes capítulos.

1.6.4. R Commander menú “Gráficas”

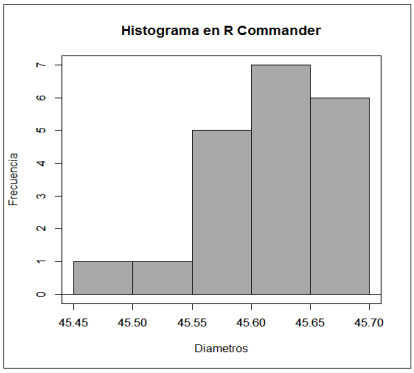
Las funciones del menú “Gráficas” (figura 29) con métodos útiles para identificar patrones, distribuciones y correlación entre variables previo a los procedimientos estadísticos y/o pruebas paramétricas. En *R Commander* es posible generar un histograma o un diagrama de densidad con una función acumulada de Gauss.

Figura 29. ***R Commander*, menú "Gráficas"**



Fuente: toma de pantalla de *R Commander* 2.2-4.

Figura 30. ***R Commander*, histograma**



Fuente: elaboración propia, empleando *R Commander* 2.2-4.

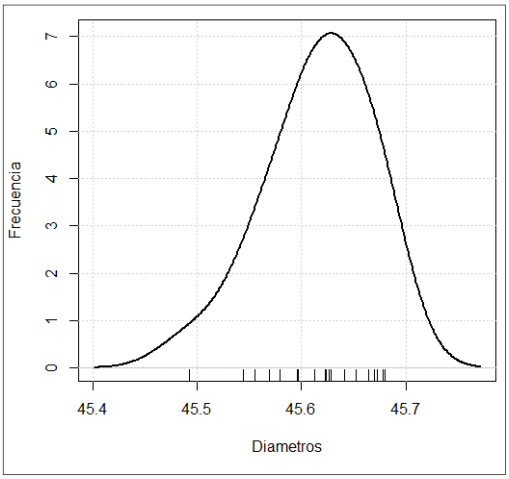
La figura 30, es el resultado de la opción de “Histograma” sobre los datos de “diámetros”, seguido del *script* equivalente en R, el *script* relacionado es el siguiente:

>with(Datos, Hist(DIAMETRO_MM, scale="frequency", breaks="Sturges", col="darkgray", xlab="Diametros", ylab="Frecuencia", main="Histograma en *R Commander*"))

La función de densidad (figura 31) es un nivel más complejo de un histograma, ya que es la forma normalizada de este último. En ambos se puede notar cierta asimetría hacia la izquierda (*skewness* = -0,757).

>densityPlot(~ DIAMETRO_MM, data=Datos, bw="SJ", adjust=1, kernel="gaussian",
+ xlab="Diametros", ylab="Frecuencia")

Figura 31. ***R Commander*, diagrama de densidad**



Fuente: elaboración propia, empleando *R Commander* 2.2-4.

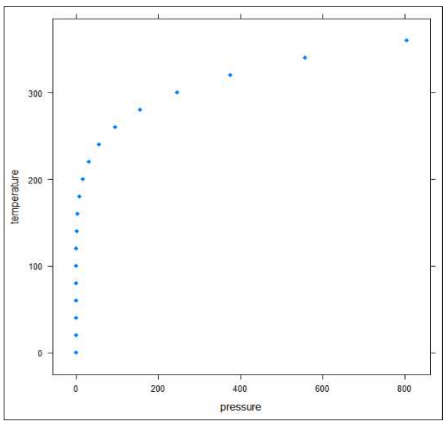
El *script* de entrada para la función de densidad equivalente, muestra los argumentos necesarios, uno importante es la declaración de la función *kernel* para normalizar la distribución, la cual, se refiere al núcleo; en este caso es el micro núcleo por el cual se calcula la función de normalización. Lo que produce

un objeto de clase *tskernel* con un coeficiente “coef” y una dimensión “m”.

En el caso de las relaciones causa y efecto, variable explicativa – variable explicada, se procede mediante la opción; en este ejemplo se importaron los datos adjuntos en el conjunto de datos vistos en el menú de Datos. Estos son presión y temperatura, ploteados en una “Gráfica XY”, el resultado gráfico en la figura 32.

```
>data(pressure, package="datasets")
>library(lattice, pos=4)
>xyplot(temperature ~ pressure, type="p", pch=16, auto.key=list(border=TRUE),
+   par.settings=simpleTheme(pch=16), scales=list(x=list(relation='same'),
+   y=list(relation='same')), data=pressure)
```

Figura 32. **R Commander, gráfica presión vrs temperatura**



Fuente: elaboración propia, empleando *R Commander* 2.2-4, con datos de paquete *Datasets*.

R Commander se auxilia del paquete *lattice*, el cual está dentro del núcleo base, el cual está diseñado para hacer análisis multivariable.

1.7. Estadística descriptiva

En este inciso se analizan los resultados de salida, en análisis netamente descriptivo de datos, utilizando algunos paquetes no contenidos en el núcleo base de R, por ejemplo se cita a: *descr, DescTools.*

1.7.1. Paquete *descr*

El paquete *descr*, versión 1.0.4, cuyo nombre es *Descriptive statistics* es utilizado para describir variables categóricas ponderadas y funciones.

A continuación se presenta el resultado que se puede obtener con el vector **diametro** ya utilizado en anteriores párrafos:

```
> #Cargar el Paquete desde el prompt
> library(descr)
> #describir las medidas de tendencia central
> descr(diametro)
D1
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
45.49   45.55   45.58   45.58   45.61   45.68
```

La función *descr*( ), es similar a la función *summary*( ) contenida en el núcleo central.

Empleando el vector **diametro3,** se pueden ajustar los datos a una estructura del tipo factor, ya que en este ejemplo se desea construir una tabla de frecuencias a partir del vector mencionado y luego que el paquete muestre un histograma con la tabla de frecuencias, pera ello se emplea una combinación entre las funciones *cut*( ) del núcleo central y el paquete *descr*, ver figura 33.

```
> #Crear el factor  diamf mediante la función cut( )
> diamf<-factor(cut(diametro3,breaks=min(diametro3)+0.023*(0:7)))
> diamf
  [1] <NA><NA><NA>          (45.61,45.63] (45.54,45.56]
  [6] (45.59,45.61] <NA>          (45.63,45.65] (45.56,45.59] <NA>
 [11] (45.61,45.63] (45.56,45.59] (45.61,45.63] <NA>          (45.61,45.63]
 [16] (45.61,45.63] (45.63,45.65] (45.61,45.63] (45.59,45.61] (45.54,45.56]
 ……..
 [96] (45.54,45.56] (45.56,45.59] (45.56,45.59] (45.61,45.63] (45.59,45.61]
7 Levels: (45.49,45.52] (45.52,45.54] (45.54,45.56] ... (45.63,45.65]
```

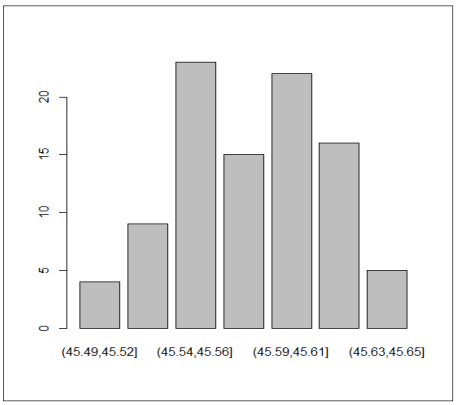
La misma función que da como resultado lo datos de la figura 33, presenta el polígono de frecuencias, en tal sentido, se le puede dedicar más tiempo al análisis gráfico, donde se puede observar de forma clara, un distribución bimodal o de dos intervalos con las frecuencias más altas, no estando contiguos (figura 34).

Figura 33. **Tabla de frecuencias, función *freq*( )**

```
> freq(diamf)
diamf
               Frequency Percent Valid Percent
(45.49,45.52]          4       4         4.255
(45.52,45.54]          9       9         9.574
(45.54,45.56]         23      23        24.468
(45.56,45.59]         15      15        15.957
(45.59,45.61]         22      22        23.404
(45.61,45.63]         16      16        17.021
(45.63,45.65]          5       5         5.319
NA's                   6       6
Total                100     100       100.000
```

Fuente: toma de pantalla de RGui, empleando función *freq*( ).

Figura 34. **Polígono de frecuencias, función *freq*( )**



Fuente: elaboración propia, empleando función *freq*( ).

Otra forma de presentar las clases es poniéndoles etiqueta, por ejemplo se sugieren los puntos medios de clase:

```
> diamf<-factor(cut(diametro3,breaks=min(diametro3)+0.023*(0:7)),
+ labels=c(min(diametro3)+0.023/2*(0:6)))
```

1.7.2. Paquete DescTools

El paquete *DescTools*, es más completo que *descr*, tanto así, que no es necesario trabajar en conversiones previas de tipo de objetos; en su lugar, está la función*Freq*( ).

```
> library(DescTools)
> Freq(diametro3)
```

Figura 35. **Tabla de frecuencias, función *Freq*( )**

```
> library(DescTools)
> Freq(diametro3)
           level freq perc cumfreq cumperc
1   [45.48,45.5]    1 0.01       1    0.01
2    (45.5,45.52]    5 0.05       6    0.06
3  (45.52,45.54]   10 0.10      16    0.16
4  (45.54,45.56]   19 0.19      35    0.35
5  (45.56,45.58]   15 0.15      50    0.50
6   (45.58,45.6]   17 0.17      67    0.67
7   (45.6,45.62]   15 0.15      82    0.82
8  (45.62,45.64]   11 0.11      93    0.93
9  (45.64,45.66]    2 0.02      95    0.95
10 (45.66,45.68]    5 0.05     100    1.00
```

Fuente: toma de pantalla, empleando paquete DescTools.

El paquete *DescTools*, es un conjunto diverso de las funciones básicas estadísticas y matemáticas. Para algunos casos prácticos, es necesario obtener de forma separada algunos de los parámetros estadísticos como la media, la moda o la desviación estándar, en lugar de obtener en un solo resultado las medidas de tendencia central más habituales (ver tabla VII).

La complejidad de los gráficos del paquete ayuda a una mejor presentación para el usuario común de R. Por ejemplo, observar la figura 36, la cual es generada por medio de la función *PlotFdist*( ); en ella se observa que combina el histograma con la función de densidad ajustada con la función Gaussiana; abajo, el diagrama de caja que indica el sesgo hacia la izquierda que presentan los datos; más abajo figura la función acumulada, donde se observa que aproximadamente el 75 % de los casos observados de los diámetros internos están por debajo del 45,61 milímetros.

Tabla VII. **Paquete DescTools, funciones estadísticas**

Función	Parámetro	Resultado
Gmean(x)	Media geométrica	45,58184
Gsd(x)	Desviación estándar geométrica	1,000892
MeanAD(x)	Desviación absoluta media	0,03358
MeanSD(x)	Error estándar de la media	0,004064605
Skew(x)	Sesgo	0,2435286
Kurt(x)	Apuntalamiento	-0,4427593
Mode(x)	Moda	45,596

Fuente: elaboración propia.

1.8. Distribuciones de probabilidad

Una distribución de probabilidad, es "un modelo matemático que relaciona el valor de una variable con la probabilidad de ocurrencia de ese valor en una población"[9]. En tal sentido la variable en mención es aleatoria porque toma distintos valores por mecanismos estrictamente aleatorios. Anteriormente, se pudo observar que mediante *R Commander* es posible generar de forma demostrativa las gráficas correspondientes a cada tipo de distribución. A continuación se presentan las diferentes distribuciones, tanto continuas como discretas, con ejemplos desarrollados desde el entorno de R y los paquetes relacionados con los dos tipos de distribuciones que son:

- Distribuciones de probabilidad discreta
- Distribuciones de probabilidad continua

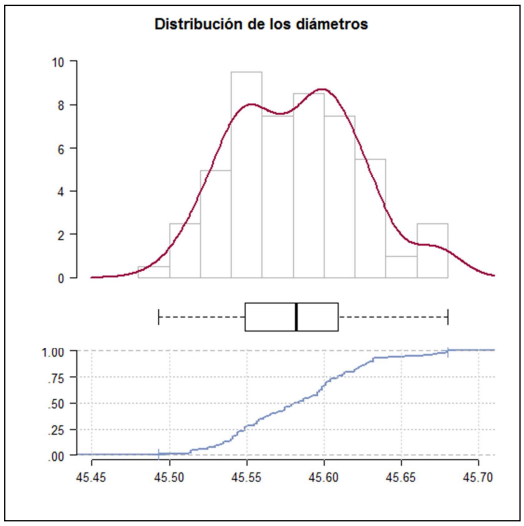
[9] MONTOGOMERY, Douglas. *Introduction to statistical quality control*. p. 51.

En el ejemplo que se ha venido trabajando, la escala de medida es de carácter continuo, es decir un diámetro medido en milímetros; en otras palabras, una distribución continua es un modelo que se fundamenta en valores que son representados por escalas continuas, tal como las dimensiones de longitud, volumen, masa (peso) o temperatura, entre otros.

1.8.1. Distribuciones discretas de probabilidad

Son definidas como tal cuando los valores pueden tomar solo ciertos valores enteros. Por ejemplo, en control estadístico de la calidad se obtienen número entero de artículos con disconformidad o un número entero de disconformidades por artículo.

Figura 36. **Paquete DescTools, función *PlotFdist*( )**



Fuente: elaboración propia, empleando función *PlotFdist*( ).

1.8.1.1. Distribución hipergeométrica

Esta distribución relaciona algún número *D* de artículos o *ítems* con algún interés en particular (o grupo de interés), de una población finita de *N* ítems. De esa población es tomada una muestra de n ítems sin reemplazo, de esa muestra, existe un número *x* que cae por definición en el grupo de interés. En tal sentido, *x* es la variable aleatoria hipergeométrica, con una distribución de probabilidad descrita en el siguiente modelo:

$$p(x) = \frac{\binom{D}{x}\binom{N-D}{n-x}}{(N-n)}$$

Donde *x*, puede tomar los valores $x = 0,1,2,3 \dots \min(n, D)$

Y la notación $\binom{D}{x}$ por ejemplo, es la simbología de una combinación, donde:

$$\binom{D}{x} = \frac{D!}{x(D-x)!}$$

En el interés de este texto se puede argumentar que *D* es el número de ítems no conformes de una población *N*, de la que se toma una muestra n sin reemplazo, en la que x representa el número de los ítems no conformes que se obtienen de la muestra. En tal sentido, se trata de la probabilidad de que esta configuración suceda.

Por ejemplo, se tiene un lote de 200 prendas de vestir, del cual 15 de ellas son declaradas como “no conformes”, por lo que se extrae una muestra de 10 unidades sin remplazo, y se desea conocer cuál es la probabilidad de que de esa muestra se obtengan dos o menos prendas “no conformes”. De tal forma

que se tienen los siguientes datos: N = 200, D = 15, n = 10, x ≤ 2.En tal sentido, hay que calcular la probabilidad acumulada.

$$P(x \leq 2) = P\{x = 2\} + P\{x = 1\} + P\{x = 0\}$$

Con la función *dhyper*( ) contenida en el núcleo base, se calcula la probabilidad para vector $(n - x)$yx = 2, 1, 0.

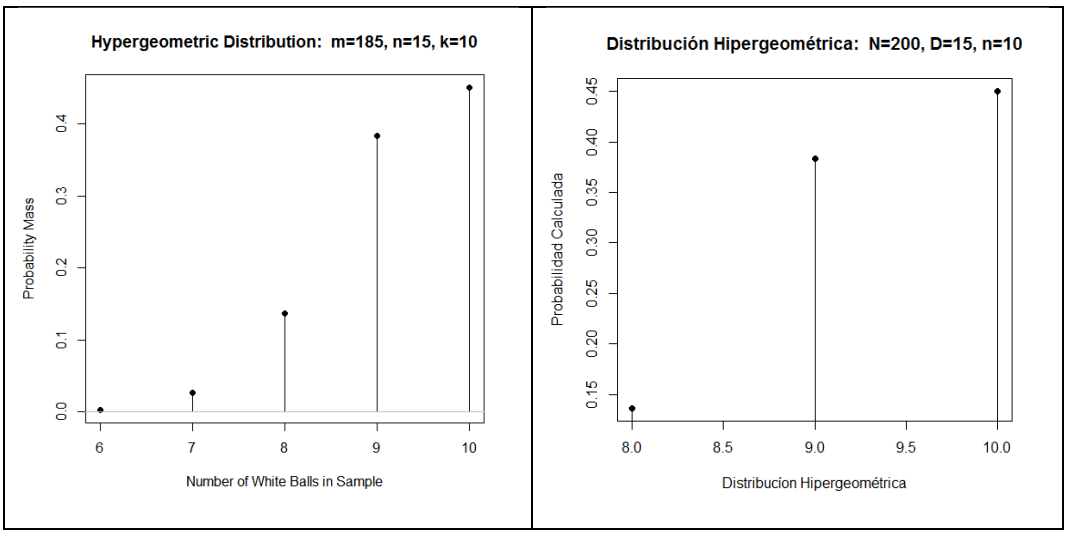
```
> dhyper(c(8,9,10), 185, 15, 10)->P
> P
[1] 0.1365115 0.3835322 0.4500111
> sum(P)
[1] 0.9700548
```

Otra forma de expresar la misma probabilidad acumulada es utilizar la función de suma acumulada *cumsum*( ), la que da el vector de la suma acumulada desde $n - x$ hasta $x = 0$.

```
> cumsum(dhyper(c(8,9,10), m=185, n=15, k=10))
[1] 0.1365115 0.5200437 0.9700548
```

La probabilidad acumulada es del 97%, de que al menos dos prendas de la muestra sean “no conformes”. Notar los argumentos empleados en la función, donde $185 = N - D$ y el vector con los valores 8, 9 y 10, equivale a las variables de la expresión $n - x$ hasta $x = 0$, entonces es 10 – 2, 10 – 1 y 10 – 0.

Figura 37. **Distribución hipergeométrica**



Fuente: elaboración propia, empleando función *plotDistr*( ).

La gráfica asociada a la distribución hipergeométrica se puede obtener directamente desde *R Commander*. Notar que los puntos descritos entre la relación $(x, P(x))$, son discretos en el sentido estricto de la palabra y no son mutuamente excluyentes, debido a que no hay reemplazo al extraer la muestra.

En el *script* de salida de *R Commander* se han hecho algunos cambios al *script* de entrada inicial, sobre todo en los argumentos de los títulos y los valores a plotear; el cambio es posible, luego de plotear una primera gráfica y con el *script* de entrada hacer los cambios deseados, escribiendo sobre los atributos por defecto, marcar el *script* y luego dar clic al botón “Ejecutar” los cuales quedan como sigue:

```
.x <- 8:10
```

```
plotDistr(.x, dhyper(.x, m=185, n=15, k=10), xlab="Distribución Hipergeométrica",
ylab="Probabilidad Calculada",
main="Distribución Hipergeométrica:  N=200, D=15, n=10", discrete=TRUE)
remove(.x)
```

De tal forma que, la función para graficar las distribuciones es *plotDistr*( ), la que en este caso plotea todos los pares ordenados$(.x, P(x))$, donde $P(x)$ es la probabilidad hipergeométrica de que se tenga $n - x$ prendas de vestir "no conformes" de la muestra de$n = 10$, la probabilidad acumulada es la sumatoria $\sum_{i=n-x}^{n} P(x)$, de $x = 0,1,2$.

Otros parámetros importantes a la distribución hipergeométrica son:

- La media $\mu = \frac{nD}{N}$
- La varianza $\sigma^2 = \frac{nD}{N}\left(1 - \frac{D}{N}\right)\left(\frac{N-n}{N-1}\right)$

1.8.1.2. Distribución binomial

Considerar una serie de ensayos del tipo *Bernoulli* donde cada uno es independiente del siguiente, donde hay una probabilidad$p$ constante de "éxito" en cada ensayo, por lo que hay un número x de "éxitos" en n ensayos de *Bernoulli* que tienen una distribución del tipo binomial. El modelo matemático que describe dicha distribución es:

$$p(x) = \binom{n}{x} p^x (1-p)^{n-x}$$

Donde $n \geq 0$, $0 \leq p \leq 1$ y $x = 0,1, \dots, n$. Con media $\mu = np$ y varianza $\sigma^2 = np(1-p)$.

La distribución binomial es adecuada en el control de calidad en muestreos de ítems con poblaciones bastante grandes; en este sentido,$p$representa la proporción de los artículos o ítems defectuosos o “no conformes” encontrados en una muestra aleatoria de tamaño n, en la que x es el número de los “no conformes”.Por ejemplo, se tiene una muestra de tamaño 5 y la proporción de los artículos defectuosos se ha calculado en 8%.

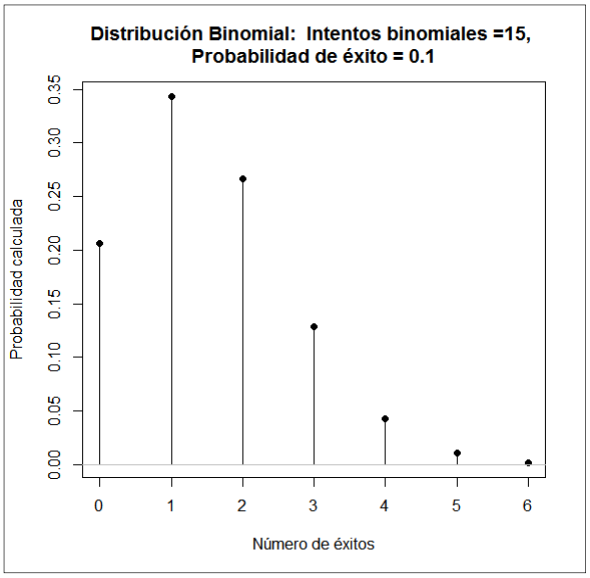
```
> dbinom(0:15, size=15, prob=0.1)->.b
> round(.b, digits=4)
[1] 0.2059 0.3432 0.2669 0.1285 0.0428 0.0105 0.0019 0.0003 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
[16] 0.0000
```

El *script* de la figura 38 para la distribución binomial es:

```
.x <- 0:15
plotDistr(.x, dbinom(.x, size=15, prob=0.1), xlab="Número de éxitos",
ylab="Probabilidad calculada",
  main="Distribución Binomial:  Intentos binomiales =15,
Probabilidad de éxito = 0.1", discrete=TRUE)
remove(.x)
```

Es posible aproximar gráficamente la distribución binomial a la normal y graficar la relación por medio de la función interna *lines*( ) para gráficos de bajo nivel. Para el siguiente ejemplo se amplía el número de intentos para un mejor ajuste de la aproximación (ver la figura 39). Los resultados desde el *script* de salida son posibles al readecuar el *script* de entrada en *R Commander*.

Figura 38**.** **Distribución binomial**



Fuente: elaboración propia, empleando función *plotDistr*( ).

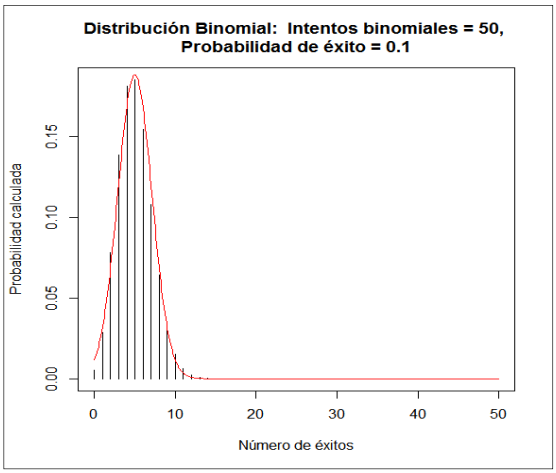
```
bx <- 0:50
by<- dbinom(bx, size=50, prob=0.1)
plot(bx,by,type= "h",
, xlab="Número de éxitos", ylab="Probabilidad calculada",
  main="Distribución Binomial:  Intentos binomiales = 50,
Probabilidad de éxito = 0.1")
nx <- seq(0,50,length=500)
ny<-dnorm(nx,50*0.1,
sqrt(50*0.1*(1-0.1)))
lines(nx,ny,col="red")
```

Los resultados presentados son dados por la función *dbinom*( ) que muestra la probabilidad simple para cada ensayo; no obstante, es posible mostrar los resultados de la probabilidad acumulada para cada ensayo

mediante la función *pbinom*( ), ejemplo:

> pbinom(c(0:15), size=15, prob=0.1, lower.tail=TRUE)
[1] 0.2058911 0.5490430 0.8159389 0.9444444 0.9872795 0.9977503 0.9996894
0.9999664 0.9999972 0.9999998 ……. [11] 1.0000000

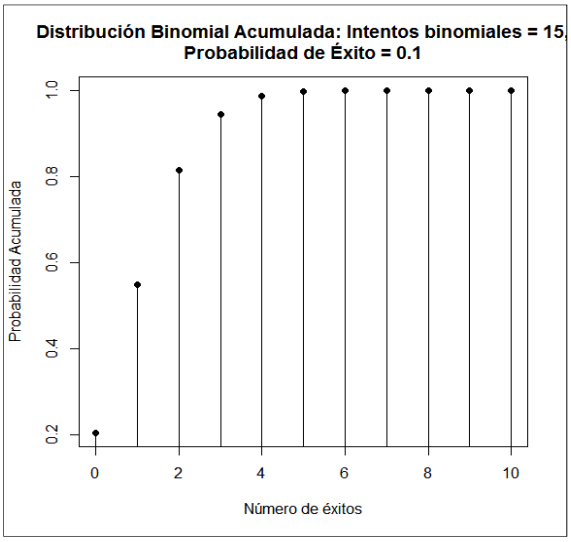
Figura 39. **Aproximación distribución binomial a distribución normal**



Fuente: elaboración propia, empleando función *plot*( ).

Al plotear los resultados de la probabilidad acumulada hasta el ensayo número 10, se tiene la figura 40, en la que lógicamente la probabilidad acumulada tiene un máximo de 1,0 a partir del ensayo 10; por ello los puntos son asintóticos a la probabilidad del 100 %. Si se necesitara conocer la gráfica de la distribución binomial a distintas probabilidades $p(x)$ de éxito en iguales números de ensayos, observar la figura 41 donde se tiene que p = (0,05 0,1, 0,25 y 0,5) y n = 15.

Figura 40. **Distribución binomial acumulada**

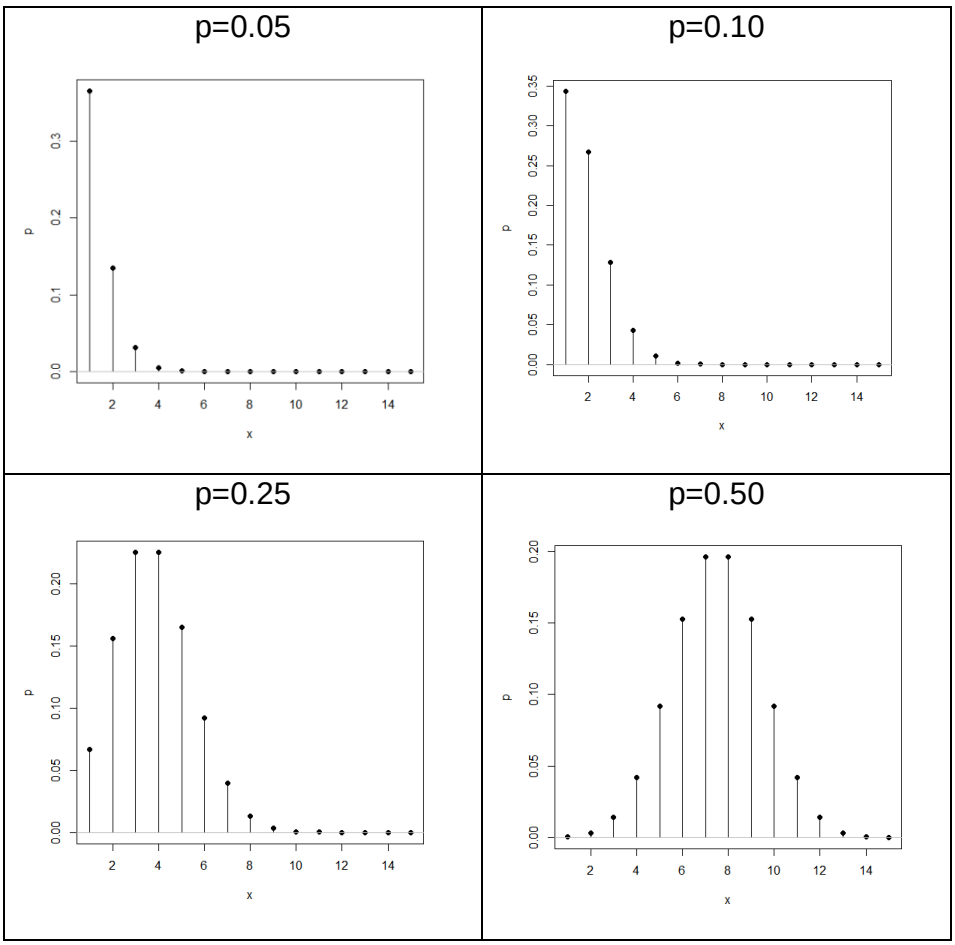


Fuente: elaboración propia, empleando función *plotDistr*( ).

A medida que la proporción p se acerca a 0.5 y x se acerca a n, la distribución binomial se acerca a una distribución normal. Otra forma de presentar el mismo arreglo de varias gráficas a la vez es utilizando la función *layout* o los parámetros de la función *par*( )[10].

[10] CORREA, Juan Carlos.*Gráficos estadísticos con R*. p. 250.

Figura 41. **Distribución binomial a distintas probabilidades de éxito**



Fuente: elaboración propia, empleando función *plotDistr*( ).

Los *scripts* utilizados para plotear cada gráfica por separado son:

```
> .x<-1:15
> plotDistr(.x,dbinom(.x,size=15,p=0.05), discrete=TRUE)
> plotDistr(.x,dbinom(.x,size=15,p=0.10), discrete=TRUE)
> plotDistr(.x,dbinom(.x,size=15,p=0.25), discrete=TRUE)
> plotDistr(.x,dbinom(.x,size=15,p=0.50), discrete=TRUE)
```

Si se deseara sobreponer las gráficas, es necesario primero hacer dos matrices conteniendo los valores discretos de cada ensayo por un lado y por otro, la matriz de las probabilidades de cada ensayo y cada columna representa los valores calculados a distintas probabilidades de éxito (ver la figura 42).

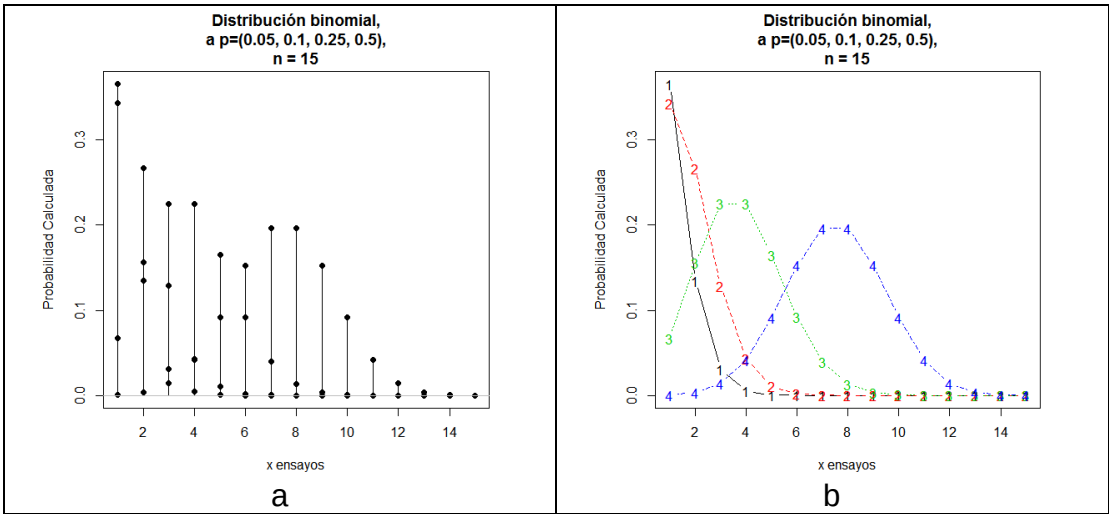
```
> x<-matrix(rep(1:15,4), nrow=15); x
      [,1]  [,2]     [,3]     [,4]
 [1,]    1  1        1        1
 [2,]    2  2        2        2
    :       : :          :        :
 [14,]   14          14       14       14
 [15,]   15   15   15       15
> px<-matrix(c(dbinom(x[,1], size=15, prob=0.05),
+ dbinom(x[,1], size = 15, prob = 0.1),
+ dbinom(x[,1], size = 15, prob=0.25),
+ dbinom(x[,1], size = 15, prob=0.5)),nrow=15)
> dimnames(px)<-list(1:15,c("p=0.05", "p=0.1","p=0.25","p=0.5"))
> round(px,4)
   p=0.05  p=0.1 p=0.25  p=0.5
1  0.3658 0.3432 0.0668 0.0005
2  0.1348 0.2669 0.1559 0.0032
: :          :          :          :
14 0.0000 0.0000 0.0000 0.0005
15 0.0000 0.0000 0.0000 0.0000
```

La información de la figura 42-a no resulta visualmente útil por la utilización de líneas y puntos que se utilizan comúnmente en las gráficas de las distribuciones de probabilidad discretas; por ello, en la figura 42-b se utilizó la función *matplot*( ) para sobreponer, las gráficas de cada probabilidad en una línea tipo densidad; el *script* que se utilizó es el siguiente:

```
> matplot(x,px,main = "Distribución binomial,
+ a p=(0.05, 0.1, 0.25, 0.5),
+ n = 15",xlab = "x ensayos",
+ ylab = "Probabilidad Calculada",
+ type = "b")
```

Utilizando las opciones de gráficas de bajo nivel, se aproximan las anteriores distribuciones binomiales a las distribuciones normales, incrementando el número de ensayos a 50. Por último, se sobreponen ambos conjuntos de gráficos a distintas probabilidades de éxito. Ver el siguiente código de entrada en la figura 43.

Figura 42. **Distribución binomial, con matriz de datos**

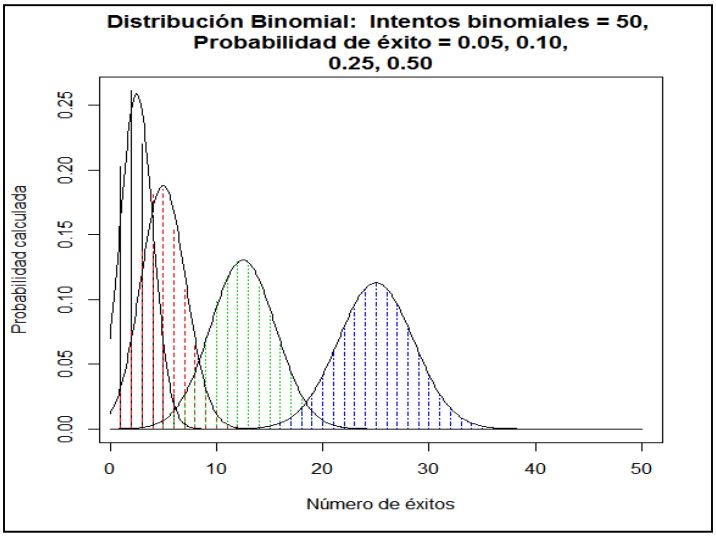


Fuente: elaboración propia, empleando función *matplot*( ).

```
bx<-c(1:50)
yb0.05<-dbinom(bx,size=50,prob=0.05)
yb0.10<-dbinom(bx,size=50,prob=0.1)
yb0.25<-dbinom(bx,size=50,prob=0.25)
```

```
yb0.50<-dbinom(bx,size=50,prob=0.5)
matplot(cbind(bx,bx,bx,bx), cbind(yb0.05,yb0.10,yb0.25,yb0.50),type= "h",
xlab="Número de éxitos", ylab="Probabilidad calculada",
  main="Distribución Binomial:  Intentos binomiales = 50,
Probabilidad de éxito = 0.05, 0.10,
0.25, 0.50")
xn<-seq(0,50,length=500)
yn0.05<-dnorm(nx,50*0.05,sqrt(50*0.05*(1-0.05)))
yn0.10<-dnorm(nx,50*0.1,sqrt(50*0.1*(1-0.1)))
yn0.25<-dnorm(nx,50*0.25,sqrt(50*0.25*(1-0.25)))
yn0.50<-dnorm(nx,50*0.5,sqrt(50*0.5*(1-0.5)))
lines(xn,yn0.05)
lines(xn,yn0.10)
lines(xn,yn0.25)
lines(xn,yn0.50)
```

Figura 43. **Transición de distribución binomial a normal**



Fuente: elaboración propia, empleando función *matplot*( ).

1.8.1.3. Distribución de *Poisson*

La distribución de *Poisson* es la aproximación matemática más adecuada para estudiar cualquier fenómeno aleatorio que ocurre por unidad de área, por unidad de volumen, por unidad de tiempo, entre otros.

El parámetro "lambda" λ, es a su vez la media y la varianza de la distribución de *Poisson* y significa el número entero y aleatorio de ocurrencia de un fenómeno por unidad; el modelo matemático que describe la probabilidad $p(x)$,de que ocurra x (inconformidades o defectos en control de calidad) veces determinado fenómeno con una distribución de *Poisson* con parámetro λ es:

$$p(x) = \frac{e^{-\lambda}\lambda^{x}}{x!} \text{y} \quad x = 0,1,2 \ldots.$$

Un ejemplo de dicha distribución, puede verse como la probabilidad de que en un taller de mecánica se reciben en promedio 4 quejas por hora, de clientes insatisfechos. Se desea conocer cuál es la probabilidad de que se reciban al menos 3 quejas por hora. El parámetro $\lambda = 4$ y el número de quejas $x = 3$; por lo tanto, la probabilidad combinada, es la sumatoria entre de $x = 0,1,3$.

$$p(x \leq 3) = \sum_{x=3}^{3} \frac{e^{-4}4^{x}}{x!}$$

Calculado por medio de R, se utiliza la función *dpois*( ); se tiene que:

```
> cumsum(dpois(0:3, lambda=4))
[1] 0.01831564 0.09157819 0.23810331 0.43347012
```

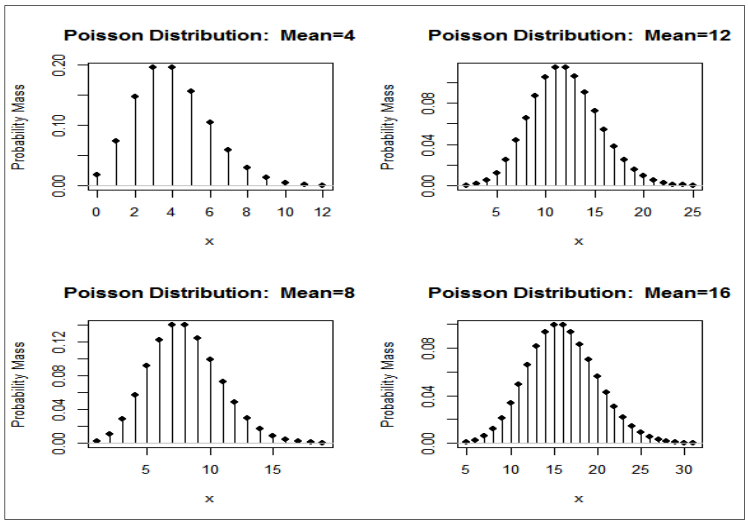
Dado que es una suma acumulada, la probabilidad de que se reciban al

menos 3 quejas por hora de los clientes es del 43,3%. Sin embargo pueden ser más de cuatro; la gráfica asociada a esta distribución dependerá del parámetro *lambda*; se ha tomado $\lambda = 4,8,12,16$, para demostrar que a medida que el parámetro es mayor, la gráfica presenta cierta apariencia simétrica, pero con una cola más larga hacia la derecha (ver figura 44). Mientras que el *script* usado para presentar dicha gráfica es:

```
>mat<-matrix(1:4,2,2)
> # crear la disposición con la matriz creada
> layout(mat)
>layout.show(4)
>.x <- 0:12
> plotDistr(.x, dpois(.x, lambda=4), xlab="x", ylab="Probability Mass",
main="Poisson Distribution:  Mean=4",
+   discrete=TRUE)
> remove(.x)
> .x <- 1:19
> plotDistr(.x, dpois(.x, lambda=8), xlab="x",
ylab="Probability Mass", main="Poisson Distribution:  Mean=8",
+   discrete=TRUE)
> remove(.x)
> .x <- 2:25
> plotDistr(.x, dpois(.x, lambda=12), xlab="x",
ylab="Probability Mass", main="Poisson Distribution:  Mean=12",
+    discrete=TRUE)
> remove(.x)
> .x <- 5:31
> plotDistr(.x, dpois(.x, lambda=16), xlab="x",
```

```
ylab="Probability Mass", main="Poisson Distribution:  Mean=16",
+    discrete=TRUE)
> remove(.x)
```

Figura 44. **Distribución de *Poisson*, a distintos valores de *lambda***



Fuente: elaboración propia, empleando función *plotDistr*( ).

1.8.2. Distribuciones continuas de probabilidad

Son las distribuciones que se adaptan matemáticamente a los valores que toman ciertas variables, es decir, que se ajustan a la ocurrencia de un valor sobre una escala o magnitud.

1.8.2.1. Distribución normal

La distribución normal representa la probabilidad asociada $p(x)$ a una variable aleatoria x con media μ y varianza σ^2; en resumen se dice que la variable aleatoria y continua x, está normalmente distribuida de acuerdo con la

siguiente notación: $x \sim N(\mu, \sigma^2)$.La forma gráfica de la distribución normal es simétrica respecto de la media, la única moda, la mediana y la curva dibujan claramente una campana. La función matemática que define la distribución normal es:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2} \quad -\infty < x < \infty$$

Es sumamente importante hacer mención del área debajo de la curva normal de probabilidades a $\mu \pm 1\sigma$ corresponde al 68,26 % ~ 68%, a $\mu \pm 2\sigma$ es el 95,46 % ~ 95 % y a $\mu \pm 3\sigma$ es el 99,73% ~ 99,7 % de los casos de la población; esto se puede visualizar graficando la curva normal de probabilidades.

Para ello se utilizan más parámetros gráficos y funciones de los ya vistos, entre ellos: *dnorm*( ) que define la distribución normal; *expression*( ) que define un objeto tipo expresión; *eval*( ) que evalúa una expresión; *segments*( ) dibuja una o un conjunto de líneas; *remove*( ) elimina un objeto; *text*( ) inserta texto o fórmulas dentro de un gráfico; tanto este último como *segments*( ), son funciones para gráficos de bajo nivel; para cada función se puede acceder en la ayuda de R, con "?función" o *help*(función).Se observa en la figura 45 el aspecto simétrico y los límites de los múltiplos de desviaciones estándar señalados por las líneas verticales rojas. Estas opciones o parámetros gráficos explorados son sumamente útiles para el ejemplo propuesto:

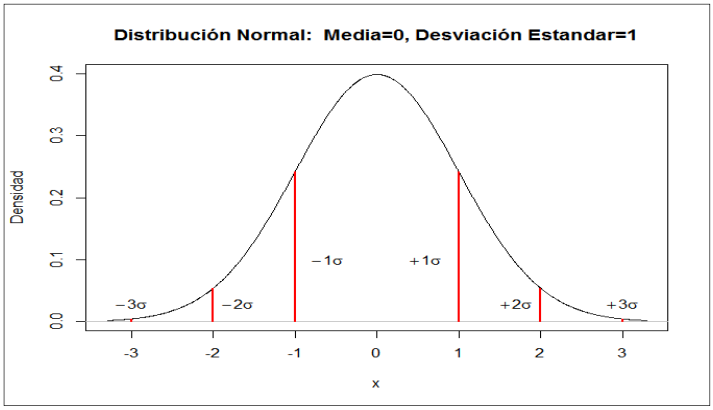
```
.x <- seq(-3.291, 3.291, length.out=1000)
plotDistr(.x, dnorm(.x, mean=0, sd=1),
cdf=FALSE, xlab="x", ylab="Densidad",
main=paste("Distribución Normal:  Media=0, Desviación Estándar=1"))
remove(.x)
```

```
x0<-c(-3,-2,-1,1,2,3)
y0<-rep(0,6)
sd<-1
fx0<-expression((1/(sqrt(2*pi)))*exp(-0.5*((x0/sd)^2)))
segments(x0, y0, x1=x0,y1=eval(fx0), col=2,lwd=2)
text(c(-3,-1.7,-0.6,0.6,1.7,3), c(0.03,0.03,0.1,0.1,0.03,0.03),
c(expression(-3*sigma),expression(-2*sigma),
expression(-1*sigma),expression(+1*sigma),
expression(+2*sigma),expression(+3*sigma))
```

Por otro lado, hay una función para calcular la probabilidad a partir de la variable Z estandarizada o calcular la variable estandarizada a partir del percentil o probabilidad:

$$z = \frac{x - \mu}{\sigma}$$

Figura 45. **Distribución normal**



Fuente: elaboración propia, empleando función *plotDistr*( ).

En tal sentido, se calcula $P(Z \leq z)$ por medio de la función *pnorm*( ) para

obtener el área bajo l, a curva de Z, la función de probabilidad acumulada de esa variable[11]; matemáticamente se obtiene por medio del modelo:

$$P(Z \leq z) = \int_{-\infty}^{z} f(y, 0{,}1)dy \qquad \text{Que se denomina} \Phi(z)$$

Donde se ya ha definido indirectamente a $f(y, 0{,}1)$ como la función de densidad. En sustitución de la tabla de la distribución normal, se emplea la función *pnorm*( ), por ejemplo: se tiene una variable estandarizada $Z = 1{,}15$, se pide calcular la probabilidad acumulada hasta Z.

```
> pnorm(c(1.15), mean=0, sd=1, lower.tail=FALSE)
[1] 0.1250719
```

Pueden notarse que los parámetros por defecto: los numéricos como la media igual a cero y la desviación estándar igual a uno. Dicho de otra forma, el resultado es equivalente a $P(Z \geq 1{,}15)$, lo que define el argumento lógico *lower.tail*. El *script* correcto debe de ser:

```
> pnorm(c(1.15), mean=0, sd=1, lower.tail=TRUE)
[1] 0.8749281
```

En efecto, el resultado correcto se pudo calcular determinando la diferencia de 1 o el complemento como: (1 – 0,1250719 = 0,8749281).La forma inversa del anterior ejemplo es considerar el percentil para determinar la variable Z.

Considerando el último resultado se utiliza ahora la función *qnorm*( ), notar el uso de los mismos parámetros:

[11]DEVORE, Jay L. *Probabilidad y estadística para ingeniería y ciencias*. p. 162.

> qnorm(c(0.8749281), mean=0, sd=1, lower.tail=TRUE)
[1] 1.15

1.8.2.2. Otras distribuciones continuas

A continuación, se presenta un resumen del equivalente de R para algunas distribuciones continuas, describiendo la función para calcular la probabilidad y los principales argumentos.

Tabla VIII. **Otras distribuciones de probabilidad continua**

Distribución	Modelo matemático	Función	Argumentos
t de Student	$f(t) = \frac{\Gamma\left(\frac{(\nu+1)/2}{2}\right)}{\sqrt{\upsilon\pi}\Gamma(\nu+2)}(1+x^2/\nu)^{-(\nu+1)/2}$	dt	Densidad (x,df,ncp)
		pt	Distribución de la función (q,df,ncp, lower.tail=TRUE)
		qt	En cuantiles (q,df,ncp, lower.tail=TRUE)
		rt	Generación aleatoria (n,df,ncp)
	$\nu > 0$ Grados de libertad real $\nu = n - 1$ = df		lower.tail es el parámetro lógico sobre la cola inferior.

Continuación de la tabla VIII.

Distribución	Modelo matemático	Función	Argumentos
F	$f(x)=\frac{\Gamma\left(\frac{n_1+n_2}{2}\right)}{\Gamma\left(\frac{n_1}{2}\right)\Gamma\left(\frac{n_2}{2}\right)}\left(\frac{n_1}{n_2}\right)^{\left(\frac{n_1}{2}\right)} x^{\left(\frac{n_1}{2}-1\right)}\left(1+\left(\frac{n_1}{n_2}\right)x\right)^{-\frac{(n_1+n_2)}{2}}$	df pf qf rf	Densidad (x,df1,df2,ncp) Distribución de la función (q,df1,df2,ncp, lower.tail=TRUE) En cuantiles (q, df1,df2,ncp, lower.tail=TRUE) Generación aleatoria (n, df1,df2,ncp), lower.tail es el argumento lógico sobre la cola inferior Con df1 grados de libertad en el numerador y df2 grados de libertad en el denominador. Para x>0

Continuación de la tabla VIII.

Distribución	Modelo matemático	Función	Argumentos
Chi Cuadrado	$f(x) = \frac{1}{2^{n/2}\Gamma(n/2)} x^{n/2-1} e^{-x/2}$	dchisq	Densidad (x,df,ncp)
		pchisq	Distribución de la función (q,df,ncp, lower.tail=TRUE)
		qchisq	En cuantiles (q,df,ncp, lower.tail=TRUE)
		rchisq	Generación aleatoria (n,df,ncp) Lower.tail es el parámetro lógico sobre la cola inferior
	Con $n \geq 0$ grados de libertad Para $x > 0$		

Fuente: elaboración propia.

Las anteriores funciones y las revisadas anteriormente con mayor detalle, están contenidas en el paquete *stats* del conjunto del núcleo base; notar que los parámetros son similares en el caso de *x*, *q* que pueden ser definidos por vectores, lo que ha permitido plotear funciones de densidad, tanto de las distribuciones continuas como de las discretas.

1.9. Regresión lineal

La metodología de regresión lineal determina la existencia de relación lineal entre variables aleatorias, unas dependientes y otras independientes. Con el auxilio de modelos matemáticos, la regresión lineal también explica cómo una variable puede variar respecto del tiempo; a esta relación con el tiempo se le conoce como “serie de tiempo”; no obstante, es importante partir del modelo clásico de regresión lineal, ya que de allí se facilita comprender y aplicar los instrumentos diseñados para R, debido a que se asume que la cantidad de datos, registros y variables es en efecto grande.

En este inciso, el tema de regresión lineal se aborda de forma introductoria, ya que en temas posteriores se emplea metodología más compleja, seleccionada para cubrir dos temas: pronósticos y detección de patrones no aleatorios.

1.9.1. Modelo de clásico de regresión lineal

El modelo clásico de regresión lineal es especificado siguiendo la función muestral primero, seguido de la función estocástica. La especificación del modelo clásico de regresión lineal tiene en términos matemáticos una relación funcional muy estrecha con el modelo aditivo para las series de tiempo a revisar posteriormente.

$\widehat{Y_i} = \widehat{\beta_0} + \widehat{\beta_1} X_i$	Modelo regresión lineal
$Y_i = \widehat{\beta_0} + \widehat{\beta_1} X_i + u_i$	Función estocástica
$Y_i = \widehat{Y_i} + u_i$	Especificación de la variable aleatoria
$u_i = Y_i - \widehat{Y_i}$	Especificación del error

Donde Y_i es la variable dependiente observada y X_i es la variable indepediente; el símbolo bajo el gorro "$\widehat{\ }$", significa que representa la función calculada; el error u_i, se calcula como resultado de la diferencia de función observada y la calculada; esto conduce al método de los mínimos cuadrados ordinarios, por sus siglas MCO; el cálculo de los parámetros se consigue por medio de minimizar la suma del cuadrado de:

$$\sum_{i=1}^{n} u_i^2 = \left(Y_i - \widehat{Y_i}\right)^2$$

$$\sum_{i=1}^{n} u_i^2 = \left(Y_i - \widehat{\beta_0} - \widehat{\beta_1} X_i\right)^2$$

Mediante derivadas parciales de los parámetros y la igualación a cero de las mismas, se soluciona por medio de un poco de álgebra; el resultado es el parámetro que equivale a la pendiente de recta, como un estimador insesgado del verdadero valor de la población.

$$\widehat{\beta_1} = \frac{\sum_{i=1}^{n} x_i y_i}{\sum_i^n x_i^2}$$

Notar las letras minúsculas de las variables en las sumatorias, que indican que están expresadas en términos de error del valor observado respecto de la media. Por medio del mismo parámetro, se obtiene el estimador del intercepto $\widehat{\beta_0}$ mediante la función $\widehat{\beta_0} = \bar{Y} - \widehat{\beta_0}\bar{X}$. Para calcular los parámetros de modelos en los que existen dos o más variables independientes o explicativas, es necesario el uso del álgebra lineal.

1.9.2. Regresión lineal en R

La utilidad de los modelos de regresión lineal y series de tiempo, apoyada por las funciones y paquetes de R a revisar, es fundamental en los estudios de pronósticos y tendencias; esta última tiene importancia en las series de tiempo para diferenciar una serie estacionaria de una que no lo es. En la planificación industrial los modelos de regresión lineal son útiles en la estimación de pronósticos de demanda y de producción.

En las cartas de control, $\bar{X}$, $\bar{R}$, $\bar{S}$, se observan en los modelos que se ajustan a la relación de cambios en la tendencia con causas asignables.

Es preciso repasar la sintaxis de la función base de R para calcular los coeficientes, la cual sirve para generar objetos nuevos con la información estadística que permita utilizarlos para las trasformaciones de los datos y de los modelos mismos y efectuar pruebas paramétricas y no paramétricas.

Los tipos de objetos centrales en el análisis de regresión son: los *data.frame* como fuente de información, los vectores como fuente de criterio en los parámetros del modelo, la función *formula*( ), la cual define el modelo matemático que se ajustará a los datos y como resultado se obtiene un objeto del tipo *list* para consolidar los coeficientes y estadísticos resultantes del modelo.

La función *lm*( ), contenida en el paquete base *stats* es la empleada para ajustar modelos lineales, regresión lineal y análisis de varianza y covarianza. Los modelos en esta función son especificados simbólicamente, por medio del argumento *formula*; la sintaxis de la función es la siguiente:

lm(formula, data, subset, weights, na.action,
method = "qr", model = TRUE, x = FALSE y = FALSE, qr = TRUE,
singular.ok = TRUE, contrasts = NULL, offset, ...)

El argumento *formula* se especifica en una sintaxis particular de R, para ajustarse al modelo matemático. La estructura de un argumento *formula* es semejante a la ecuación lineal $\mathbf{Y} = \boldsymbol{\beta}\mathbf{X}$, pero con la nomenclatura $y \sim modelo$, donde el modelo es la relación funcional entre los vectores o matrices que contienen a las variables independientes.

Del modelo, la función *lm*( ) estimará la serie de parámetros. Citado textualmente la tabla IX, contiene las referencias de la forma correcta de especificar un modelo.

Tabla IX. **Fórmulas de R para regresión lineal**

Modelo	Descripción
a + b	Efectos de a y b.
X	Si X es una matriz, especifica un efecto aditivo para cada una de las columnas; por ejemplo X[,1]+X[,2]+…+X[,ncol(X)]; algunas de las columnas se pueden seleccionar con índices numéricos (por ej., X[,2:4].
a:b	Efecto interactivo entre a y b.
a*b	Efecto aditivo e interactivo entre a y b (idéntico a*a* + *b* + *a:b*).
poly(a,n)	Polinomios de a hasta grado *n.*
^n	Incluye todas las interacciones hasta el nivel *n*, por ej., (a+b+c)^2 es idéntico a *a* + *b* + *c* + *a:b* + *a:c* + *b:c*
b%in%a	Los efectos de b están anidados en a (idéntico a *a* + *a:b* o *a/b*)

Continuación de la tabla IX.

Modelo	Descripción
a-b	Remueve el efecto de b, por ejemplo (a + b + c)^2 – a:b es idéntico a *a + b + c + a:c + b:c*
-1	y ~ x -1 regresión a través del origen (igual para y ~ x + 0 o 0 + y ~ x
1	y ~ 1 ajusta un modelo, sin efectos (solo el intercepto)
offset(…)	Agrega un efecto al modelo sin estimar los parámetros (e.g., offset(3*x).

Fuente: PARADIS, Emmanuel.*R para principiantes*. p. 49.

En el caso de los modelos especificados por medio de la fórmula $\mathrm{y} \sim \mathrm{x1} + \mathrm{x2}$, equivale al modelo $y = \beta_1 x1 + \beta_2 x2 + \alpha$. Lo recién expuesto, es de suma importancia en la estimación correcta de modelos. El argumento *data* es el conjunto de datos para ajustar a través del modelo, mientras que los demás argumentos son hasta cierto punto de carácter optativo y útiles para refinar los cálculos en el momento y posteriores. Para ejemplificar el uso de la función *lm*( ) se utiliza un conjunto de datos *cars* contenido en el paquete *datasets*(ver detalle en anexo 3). El enunciado es el siguiente:

"La distancia recorrida por un vehículo para llegar al reposo está directamente relacionada con la velocidad del mismo antes de la desaceleración (frenado)". Considerando que los datos fueron tomados en la década de 1920, se pide:

- Dibujar un diagrama de puntos. ¿Es creíble el modelo representado por la línea recta de regresión?
- Encontrar los estimadores por mínimos cuadrados de la pendiente y el intercepto en una regresión lineal simple.
- Estimar la distancia a recorrer por un vehículo antes de llegar al reposo, cuando este viaja a 20 millas por hora.
- Plotear los datos calculados y comentar a qué se asemeja.

Ver los siguientes resultados iniciando en la primera columna hasta el final y concluyendo en la segunda columna. Considerar los parámetros obtenidos, con mayor interés en la pendiente, la cual está expresada en pies por millas por hora. El resultado se traduce en la ecuación de regresión:

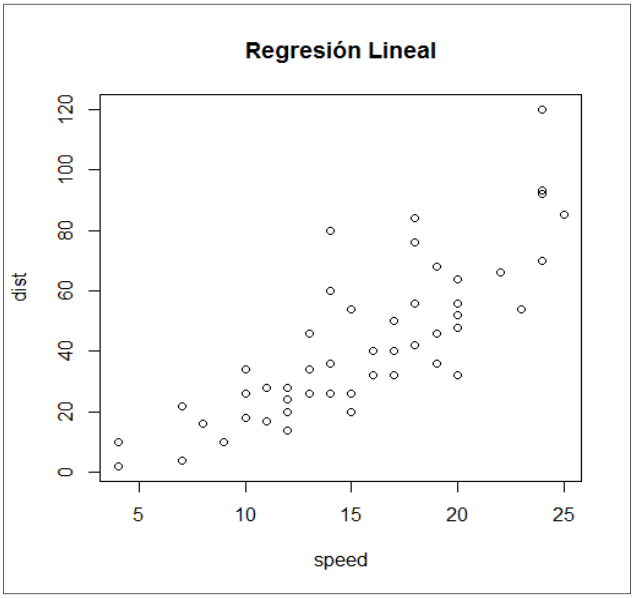
$$Distancia_{pies} = -17{,}579 + 3{,}932V_{mph}.$$

Técnicamente hablando, al considerar el intercepto -17,579 pies, no es algo que sucede en la realidad, por lo que para efectos prácticos se necesita que el modelo de regresión pase por el origen, ya que se supone que no ha recorrido ninguna distancia a partir del momento en que el vehículo empieza a frenar, por lo tanto se omite el intercepto en el siguiente modelo para obtener de hecho un mejor ajuste o R^2 que en el anterior modelo.

El diagrama de puntos se extrae de los datos originales mediante la función *plot*( ).

```
> library(datasets)
> plot(cars)
```

Figura 46. **Diagrama de puntos con función *plot*( )**



Fuente: elaboración propia, empleando función *plot*( ).

Los nombres de las variables se obtienen de la siguiente forma:

```
> names(cars)
[1] "speed" "dist"
```

Con los nombres de las variables se formula el modelo de la función de regresión con un objeto de clase formula y luego el cálculo de los resultados del modelo viene dado por la función *lm*( ) y los resultados ampliados por la función *summary*( ).

```
>distancia<-formula(cars$dist~cars$speed)
> lm(distancia)

Call:
```

```
lm(formula = distancia)
Coefficients:
(Intercept)   cars$speed
   -17.579        3.932

> summary(lm(distancia))
Call:
lm(formula = distancia)

Residuals:
   Min      1Q  Median      3Q     Max
-29.069  -9.525  -2.272   9.215  43.201

Coefficients:
             Estimate        Std. Error       t value            Pr(>|t|)
(Intercept)       -17.5791        6.7584           -2.601           0.0123 *
cars$speed         3.9324         0.4155           9.464            1.49e-12 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
Residual standard error: 15.38 on 48 degrees of freedom
Multiple R-squared:  0.6511,    Adjusted R-squared:  0.6438
F-statistic: 89.57 on 1 and 48 DF,  p-value: 1.49e-12
```

Los resultados muestran una adaptación del modelo a los datos, no obstante, hay que observar el diagrama de puntos, el cual sugiere una línea que imaginaria que pasa por el origen, es decir la media, lo que apoya a la teoría, sobre la distancia recorrida a una velocidad inicial de cero, debe de ser igual a cero, por eso el modelo se especifica de nuevo de la siguiente forma:

```
> distancia2<-formula(cars$dist~cars$speed-1)
> summary(lm(distancia2))
```

```
Call:
lm(formula = distancia2)

Residuals:
   Min     1Q  Median     3Q    Max
-26.183 -12.637  -5.455   4.590  50.181

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
cars$speed   2.9091     0.1414   20.58   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 16.26 on 49 degrees of freedom
Multiple R-squared:  0.8963,    Adjusted R-squared:  0.8942
F-statistic: 423.5 on 1 and 49 DF,  p-value: < 2.2e-16
```

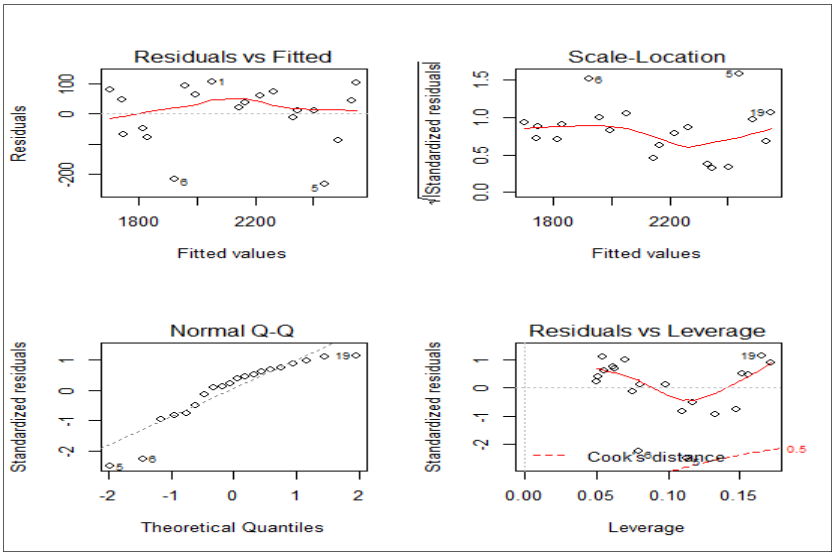
Considerando el anterior modelo y considerando que un vehículo viaja a 20 millas por hora, la distancia que toma el vehículo se calcula de la siguiente forma:

```
> predict(lm(cars$dist~cars$speed-1))
1                 2                 ……   19              20
11.63653          11.63653          37.81872       40.72785
```

A una velocidad de 20 millas por hora, la distancia que recorrería el vehículo es de 40,7 pies. Ver el uso de la función *predict*( ) sobre el modelo seleccionado. Asimismo, en ocasiones será de utilidad tener una referencia visual de los residuales calculados a partir del modelo, para ello se extrae en forma automática la información del objeto de la regresión:

```
> opar<-par()
> par(mfcol = c(2,2))
> plot(lm(distancia2))
```

Figura 47. **Regresión lineal, gráfica de residuales**



Fuente: elaboración propia, empleando función *plot*( ).

2. DESCRIPCIÓN FORMAL LENGUAJE R

2.1. Objetos en R

El “objeto” dentro la terminología de lenguajes de programación orientados a objetos, es una unidad lógica que consta de un estado en particular y un comportamiento, que bajo esa estructura se almacenan los datos. El objeto interactúa con la colección de *script* s o rutinas y subrutinas. En otras palabras un objeto representa una entidad observable que tiene identidad, comportamiento y estado.

La identidad como su nombre lo indica, es la diferenciación de un objeto de otro; el comportamiento está relacionado con su funcionalidad y especifica las operaciones que puede realizar y participar. El estado de un objeto es un conjunto de atributos o combinación de atributos en un instante dado. Por todas estas razones se dice que el objeto es una entidad dinámica. Para efectos del desarrollo posterior de este texto, de hecho debe de considerarse que casi todo en R es un objeto, entre ellos se tiene:

Tabla X. **Objetos del lenguaje y de datos**

Objetos de lenguaje	
call	Llamada
expression	Colección de declaraciones
names	Nombres

Continuación de la tabla X.

function	Las funciones constan de lista de argumentos, código y entorno (*environment*)
NULL	Sin objeto
Objetos de datos	
vector	Colección ordenada de datos del mismo tipo.
matrix* y *array	Es la generalización multidimensional del vector, es decir del mismo tipo.
data.frame	Una composición de varios vectores de diferente tipo.
factor	Reúne datos cualitativos para un vector.
list	Arreglo de datos de distinto tipo.

Fuente: elaboración propia.

Los objetos están definidos por la estructura en sus datos. La estructura fundamental es el vector o un escalar en su defecto, definido por el largo o el número de filas (mayor que uno e igual a uno, respectivamente).

2.1.1. Atributos de los objetos

Los atributos son el conjunto de características o propiedades que representan los datos asociados al objeto; estos, como se mencionó, constituyen en un determinado momento el estado del objeto.

2.1.1.1. Modo

El principal atributo de un objeto es el modo, el cual se refiere al tipo de almacenamiento del objeto; los tipos principales de objetos son:

Tabla XI. **Atributos de objetos**

NULL	name	pairlist	closure
environment	promise	call	function
character	logical	numeric	raw
complex	character	“. . . “	any
expression	List	bytecode	externalptr
weakref	S4		

Fuente: elaboración propia.

En el caso de los modos *numeric*, *complex*, *character*, *logical*, *raw* en las estructuras de datos, son mutuamente excluyentes. En tal sentido el modo, constituye entre los atributos de cualquier objeto en R una característica principal, tanto de lenguaje como de datos.

Por ejemplo, todos los vectores, como estructuras básicas de datos en R, tienen un modo, el cual se puede conocer mediante la función *mode*( ). Asimismo, se puede conocer el tipo de dato por medio de la función *typeof*( ).

Tabla XII. **Tipo de datos más mencionados en R**

NULL	Symbol	Pairlist	closure
environment	Promise	Language	special
builtin	Char	logical	integer
double	complex	character	“. . . “
any	expression	list	bytecode
externalptr	weakref	raw	S4

Fuente: elaboración propia.

Para los vectores como unidades fundamentales y objetos de datos, los tipos asociados son *integer, double, complex, character, logical*.

```
> x2<-c(1,2,TRUE)
> typeof(x2)
[1] "integer"
```

2.1.1.2. Clase

En el primer capítulo se introdujo brevemente a la clase, sin mencionar que consiste en un atributo de todo objeto en R. Se mencionó también, que los tipos principales de clase son *numeric, character, logical, complex*, cuando de objetos de datos se trate.

```
> class(x2)<-"double"
> class(x2)
[1] "numeric"
```

Por ejemplo, también se puede cambiar la clase de un objeto, aunque siga siendo del mismo tipo, el vector **x2**, ha sido creado como *integer*, mediante la función *class*( ) se cambia a *double*.

```
> x3<-c("a","b")
> class(x3)
[1] "character"
> class(x3)<-"integer"
Mensajes de aviso perdidos
In class(x3) <- "integer" : NAs introducidos por coerción
```

Pero si se quisiera cambiar un vector de tipo *character* a numérico, no es posible hacerlo por definición, no así en caso contrario:

```
> x6<-(1:5); x6
 [1] 1 2 3 4 5
> y<-as.character(x6); y6
[1] "1" "2" "3" "4" "5"
> z6<-as.integer(y6); z6
[1] 1 2 3 4 5
#Implica que los vectores x6 y z6 son iguales.
```

Otra función que asigna un tipo de clase distinta a un objeto mediante otro objeto es *as.character*( ), complementando la idea que un vector numérico puede ser del tipo *character*. Tanto la clase (*class*) como la longitud (*length*) son los atributos intrínsecos de un objeto.

```
> class(class)
[1] "function"
> class(length)
[1] "function"
```

2.1.1.3. Longitud

La longitud (*length*) es el número de datos contenidos en el vector. Por definición un escalar es de longitud “1”. No obstante pueden existir vectores con longitud “cero”, debido a que el tipo de datos almacenados es *NULL*, por lo tanto no se puede modificar.

```
> x4=2; x4
[1] 2
```

```
> length(x4)
[1] 1
> x5<-c(NULL)
> x5
NULL
> length(x5)
[1] 0
```

Por medio de la función *length*( ) se puede modificar la longitud de un objeto, sin embargo, si ya tiene datos y no se especifican más, la nueva longitud implicará que los datos posteriores serán “No disponibles” o *Not Avaliable* “*NA*”.

```
> length(x2)<-4
> x2
[1]  1  2  1 NA
```

2.1.1.4. Dimensión

El atributo de dimensión (*dim*) en los objetos de datos como *data.frame, matrix, list* o *array*, identifica el número de filas (*row*) y columnas (*col*) con que cuenta. La dimensión, es vista como la generalización de más de una dimensión de la longitud de los vectores.

En el caso de *matrix* o matriz, se refiere a uno o varios vectores con una característica adicional definida por otro vector *dim* de longitud 2 que especifica su dimensión *m* x *n* (filas x columnas):

```
> m2<-matrix(1:8,4)
> m2
[,1]        [,2]
```

```
[1,]    1   5
[2,]    2   6
[3,]    3   7
[4,]    4   8
> dim(m2)
[1] 4 2
```

2.1.1.5. Dimnames

A menudo es necesario asignar nombres a cada una de las filas y las columnas de una matriz, por ejemplo, el atributo en la función *matrix*( ) o la función *dimnames*( ), se emplea para asignar los nombres por medio de una lista de nombres en el sentido estricto de la palabra, ya que tratándose de dicha lista, ambos vectores pueden ser de distinto tipo, conteniendo un primer vector para el nombre de las filas y un segundo para el nombre de las columnas:

```
> m2<-matrix(1:8,4); m2
     [,1] [,2]
[1,]    1    5
[2,]    2    6
[3,]    3    7
[4,]    4    8
> dimnames(m2)<-list(c(),c("a","b")); m2
     a b
[1,] 1 5
[2,] 2 6
[3,] 3 7
[4,] 4 8
> attributes(m2)
$dim
```

```
[1] 4 2
$dimnames
$dimnames[[1]]
NULL
$dimnames[[2]]
[1] "a" "b"
> class(m2)
[1] "matrix"
```

2.1.1.6. Atributos rownames y colnames

Respectivamente, estos atributos se refieren a los nombre de las filas y columnas en las estructuras como los *data.frame*. En el caso de una matriz se puede emplear indistintamente que el efecto sea asignar el atributo *dimnames.*

```
> rownames(m2)<-c("c","d","e","f")
> attributes(m2)
$dim
[1] 4 2
$dimnames
$dimnames[[1]]
[1] "c" "d" "e" "f"
$dimnames[[2]]
[1] "a" "b"
```

2.1.1.7. Atributos de series de tiempo

Las series de tiempo o *"ts*" como objetos de datos en R poseen atributos particulares, uno de esos atributos es el "*tsp*" que contiene la fecha inicial en unidades de tiempo, la fecha final de la serie y la frecuencia consistente en el número de observaciones por unidad de tiempo. A continuación se presentan

los objetos más utilizados, haciendo referencia a los atributos acá mencionados.

2.1.2. Vectores

Un vector es una colección ordenada de objetos; en tal sentido, un vector puede ser también una colección de vectores, siempre y cuando todos los objetos sean del mismo tipo. Los vectores son un tipo de objeto de datos en R, así como lo son también las funciones, las matrices, factores, hoja de datos (*data.frame*), series de tiempo y las listas.

Tabla XIII. **Tipos de datos en objetos**

Objeto	**Tipos de datos**	**Varios tipos**
vector	numeric, character, complex, logic	No
factor	numeric, character	No
array	numeric, character, complex, logic	No
matrix	numeric, character, complex, logic	No
data.frame	numeric, character, complex, logic	Sí
ts	numeric, character, complex, logic	Sí
list	numeric, character, complex, logic, function, expression	Sí

Fuente: elaboración propia.

Como se puede apreciar, las listas o *list,* los marcos de datos *data.frame* y las series de tiempo *ts*, pueden contener distintos tipos de datos; sin embargo, únicamente en el caso de las listas, los datos de la misma columna pueden ser de distinto tipo.

En los siguientes ejemplos se hace referencia a los atributos ya citados y junto con algunas posibles situaciones que surgen a partir de la manipulación de ellos, se exploran otras funciones del lenguaje que serán útiles en el desarrollo de los capítulos posteriores. En la creación y asignación de vectores, es necesario, modificar, sumar, comparar, eliminar y fusionar: ver los siguientes ejemplos:

Tabla XIV. **Creación y asignación de vectores**

Script	**Explicación**
> c(4,7,10)->y	Se crea el vector **y**
> c(5,0,-2)->z	Se crea el vector **z**
> 3*y+2->v > v [1] 14 23 32	Se crea un vector **v** a partir de una operación del vector **y** con los escalares 3 y 2.
> z^2-10->w > w [1] 15 -10 -6	Se crea un vector **w** a partir de una operación del vector **z** con los escalares 2 y 10.
r<-c(y,z,0,1) > r [1] 4 7 10 5 0 -2 0 1	Se asigna al vector **r** los elementos que contiene **y**, **z** y los números 0 y 1 en ese orden. La longitud *length* o número de elementos del nuevo vector **r** es afectada en forma aditiva por la longitud de cada elemento.
> c(6,3,0)->m	Se crea el vector **m**
> c(-3,3,1,5)->n	Se crea el vector **n**

Continuación de la tabla XIV.

Script	***Explicación***
> 2*m+n->p Mensajes de aviso perdidos In 2 * m + n : La longitud de objeto mayor no es múltiplo de la longitud de uno menor	Se crea un vector **p** a partir de una operación con los vectores **m** y **n y** muestra el aviso de diferencia en el tamaño o largo de los vectores.
> p [1] 9 9 1 17	El resultado muestra ahora un vector **p** de longitud = 4, cuyo último elemento (17) es el resultado de R, regresa a calcular la operación con el primer elemento de **m** con el último de **n**.

Fuente: elaboración propia.

Para encontrar la sumatoria de los elementos numéricos de un vector se utiliza la función *sum*(), la cual equivale a la notación simple de suma con la notación$\sum_{i=1}^{n} x_i$, que significa la suma de todos los elementos incluidos en el vector.

Se ha hablado de la longitud como atributo de un vector; al no conocerlo este se puede determinar mediante la función *length*(), tomando los datos de los vectores **m, n** y **p**:

```
>length(m)
[1] 3
>length(n)
[1] 4
>length(p)
[1] 4
```

La longitud de un vector se puede modificar asignando valor al atributo *length* por medio de la función *length*( ), un entero mayor igual o mayor que cero, citando el ejemplo del vector **n**, el que es de longitud 4, se puede cohesionar a longitud 3 o hacer que el vector **m** tenga una longitud 4; sin embargo crea un valor NA, ya que en efecto, no hay ningún valor disponible en el índice m[4].

```
> 3->length(n); n
 [1] -3  3  1
> 4->length(m); m
[1]  6  3  0 NA
```

La función *max*( ) y *min*( ),representan el valor máximo y mínimo, respectivamente, y permiten ser adecuarlas a nuevas funciones relacionadas con análisis estadístico.

```
>max(p)
[1] 17
>min(p)
[1] 1
```

En complemento a lo anterior, las funciones *max*() y *min*() integran implícitamente un vector de longitud2, que se obtiene directamente con la función *range*() o rango.

```
>range(p)
[1]  1 17
```

Se puede calcular el rango de forma indirecta, creando un nuevo vector con los valores máximo y mínimo en forma ascendente.

```
> s<-c(min(p),max(p))
>s
[1]  1 17
```

Como se puede apreciar, algunas las funciones internas de R, son atajos para efectuar operaciones que se pueden declarar con otras funciones aritméticas; para un vector *v* cualquiera se resumen las siguientes funciones:

Tabla XV. **Funciones y formas alternativas**

Descripción	**Función *f()***	**Operación alternativa**
Rango de un vector	*range*()	v<-c(*min*(), *max*())
Media aritmética	*mean*()	*sum*()/*length*()
Varianza muestral	*var*()	*sum*((**x**-*mean*())^2)/(*length*()-1)

Fuente: elaboración propia.

Al emplear la función *var*(), en una matriz (m por n), el valor resultante es una matriz de covarianzas de (n por n); más adelante, en el apartado para matrices y luego en las aplicaciones se verá el resultado de la matriz de covarianzas.

2.1.2.1. Vectores lógicos

En el capítulo 1, operando con las operaciones básicas, se introdujo la noción de los operadores de comparación y los operadores lógicos, cuya aplicación será necesaria en la manipulación y análisis de vectores lógicos.

En el lenguaje de R, se pueden hacer manipulaciones de vectores que

contengan valores lógicos, *TRUE* o *FALSE*, Verdadero y Falso, respectivamente, aunque se pueden abreviar con “T” o “F”; en R no hay palabras reservadas como valor, por eso es recomendado trabajar con los valores *TRUE* o *FALSE*.

Estos no son funciones, sino valores, los cuales son generados al cumplirse o no con una condición y asigna los valores a un nuevo vector que contiene los valores falsos o verdades. Por ejemplo, si el vector **y** cuenta con los valores -1, 4, 0 y 7, se quiere saber cuál de los valores es mayor o igual a cuatro.

```
> y<-c(-1,4,0,7)
>ylogic<-y>=4
>ylogic
[1] FALSE  TRUE FALSE  TRUE
```

La operación crea otro vector **ylogic** con los valores que cumplan con la condición de ser mayor o igual a cuatro. El nuevo vector tiene los valores lógicos, misma longitud y mismo orden que el vector **y.** En el ejemplo, los valores segundo y cuarto cumplen con la condición de ser mayores o igual a cuatro, por lo tanto son verdaderos *TRUE*, en caso contrario, son falsos *FALSE*.

Esta estructura lógica utiliza la simbología de mayor que “>”, menor que “<”, mayor o igual que “>=”, menor o igual que “<=”, igual a “==”, no igual a “!=”.

En el uso de la terminología de la lógica matemática, conociendo que un valor puede ser verdadero o falso, también existen los operadores lógicos descritos en la tabla II.

Suponiendo que **x** y **y** están compuesto por los valores (2, 3, 0, -5) y (-1, 4, 0, 7) respectivamente, empleando el comparador igual “==”, se puede

calcular un vector lógico **z**, en el cual, el tercer valor cumple la condición de ser igual, por lo tanto en el vector **z** es el único valor verdadero.

```
>x
[1]  2  3  0 -5
>y
[1] -1  4  0  7
> z<-x==y
>z
[1] FALSE FALSE  TRUE FALSE
```

Siguiendo con el mismo ejemplo, se crea un vector **v** a partir de la proposición lógica de los valores de **x** mayores o igual a dos. De igual forma, un vector **w** con los valores de **y** menores de dos.

```
>x
[1]  2  3  0 -5
>y
[1] -1  4  0  7
> v<-x>=2
>v
[1]  TRUETRUE FALSE FALSE
> w<-y<2
>w
[1]  TRUE FALSE  TRUE FALSE
```

Ahora se tienen los vectores lógicos **v** y **w,** los cuales se pueden relacionar en forma binaria. Al crear un vector **r** de intersección, donde los valores son verdaderos si ambos son verdaderos; otro vector **s** donde se operan en unión, el valor de **s** es verdadero si al menos uno de los vectores **v** o **w** es verdadero.

```
> r<-v&w
>r
[1]  TRUE FALSE FALSE FALSE
> s<-v|w
>s
[1]  TRUE TRUE TRUE FALSE
```

Estas funciones de vectores, son especialmente útiles en la extracción de datos de matrices o listas de datos, tanto para crear *script* s, filtrar y depurar datos.

Por otro lado, los vectores lógicos pueden ser manipulados en operaciones aritméticas aprovechando la propiedad de los cálculos binarios, ya que *TRUE* puede interpretarse como 1 y *FALSE* como 0.

2.1.2.2. Valores perdidos

En ocasiones existen registros estadísticos que no contienen valor alguno debido a varios factores de disponibilidad de datos, la cual no equivale a cero, ya que cero de hecho es un valor; de ello, es importante declarar que el valor es "no disponible" o *not avaliable* en la traducción al inglés. Por lo tanto, en una escala de datos es numérico, poner valores ceros es incorrecto, porque el cero por definición es la ausencia de valor.

Por ejemplo, en control estadístico de calidad(CEC), hay muestras que por algún motivo no se pudieron tomar o tienen valores atípicos. En tal sentido es erróneo, como ya se mencionó colocar un cero "0"; lo correcto es mostrar el registro como *NA*, de esa forma los cálculos correspondientes se hacen sobre los valores válidos.

```
> z<-c(1:3,NA)
> x<-is.na(z); x
[1] FALSE FALSEFALSE  TRUE
```

En este ejemplo se asigna al vector **z** los valores del 1 al 3 y por último un valor *NA*; mediante la función *is.na*() se crea otro vector lógico **x**, del mismo tamaño de **z,** con los valores verdaderos si el correspondiente valor de **z** es un valor perdido, de lo contrario es falso.

Otro caso de valores perdidos sucede cuando en un vector o en una matriz existe un valor, el cual depende de una operación entre otros vectores que depende funcionalmente de una forma numérica indeterminada, tal como:

$$\frac{0}{0} \quad \frac{\infty}{\infty} \qquad \text{y} \qquad \infty$$

En términos de cálculo se sabe que el $\lim_{n \to o} x/n$ es igual a∞, R interpreta este valor y los anteriores por razones más prácticas que teóricas como un valor que no es un número, por lo tanto los señala como *not a number* (*NaN*), al igual que la función *is.na*() que también aplica para este caso, la función *is.nan*(), es exclusiva para los valores *not a number*.

2.1.2.3. Vectores de caracteres

Hasta este punto se conocen ya vectores numéricos en secuencia o simplemente una colección de valores sin relación; se ha visto vectores numéricos donde existen valores no disponibles y valores que no son números, así como también los vectores lógicos.

En algunos cálculos estadísticos, es común que se presente también la necesidad de operar con caracteres, cadenas de texto, etiquetas y de concatenar números y letras; en otras palabras, es tratar con variables cualitativas, tales como color, apariencia, origen, etc., por ejemplo:

```
> v<-c('aceptable','no aceptable')
>v
[1] "aceptable"    "no aceptable"
```

Se crea un vector **v**, con las etiquetas “aceptable” y “no aceptable”, encerradas con comillas simples (‘ ’), sin embargo también se puede usar la doble comilla (“ ”), en cualquiera de los dos casos al llamar al vector **v**, mostrará los valores con doble comilla. Empleando la función *paste*() concatena cadenas de texto con números, ejemplo:

```
> r<-paste(c('X','Y'), 1:6, sep="-"); r
[1] "X-1" "Y-2" "X-3" "Y-4" "X-5" "Y-6"
```

La sintaxis dice que se asigna a un vector **r**, dos cadenas de texto “X” y “Y”, a la secuencia de números del uno al seis, separados por un guion, de omitirse el argumento *sep*=, el separador hubiese sido un espacio en blanco, entonces lo que figure dentro del par de comillas será el separador, si no hay nada dentro de comillas la cadena de texto y números figuran sin espacio.

```
> r<-paste(c('X','Y'), 1:7, sep="")
>r
[1] "X1" "Y2" "X3" "Y4" "X5" "Y6" "X7"
```

Notar que la secuencia de números es un número par, al igual que la cantidad de etiquetas, por lo tanto el vector r tiene pares completos de números

y texto concatenado.

2.1.2.4. Vectores índices

En R, al igual que otros lenguajes y bases de datos, se pueden crear subconjuntos de los elementos de un vector, anexando al nombre del vector un vector índice señalado por estar entre paréntesis cuadrados " [ ] ".

Partiendo de la existencia de un vector **x** que contiene valores numéricos y valores perdidos (*missing values*) o *NA*, se puede hacer un vector **y**, extrayendo los valores numéricos y un vector **z** solo con los valores perdidos. Suponiendo que el vector **x** está definido como:

```
> x<-c(3,NA,-2,0,NA,NA,4,1,NA); x
[1]  3 NA -2  0 NANA  4  1 NA
```

Ahora se adjunta un vector **y** con los valores perdidos, por lo tanto es un vector de longitud menor al de **x.**

```
> y<-x[is.na(x)];y
[1] NANANANA
>length(y)
[1] 4
```

Sin embargo, es lógico y útil enfocarse en los valores efectivos y no en los perdidos. Entonces, se crea un vector índice **z**, con los valores numéricos de **x** multiplicado por un escalar, dando como resultado los valores del nuevo vector índice **z**.

```
> z<-(x*2)[!is.na(x)&x>0];z
```

```
[1] 6 8 2
```

Otra posibilidad es extraer los valores TRUE o FALSE del vector **x,** idénticos o diferentes a un valor específico. Por ejemplo, se desea obtener todos los valores mayores que igual o mayor que dos del vector **x**.

```
> x[x>=2]
[1] 2 3
```

Las anteriores operaciones abren tantas posibilidades de aplicaciones en el control estadístico de la calidad, ya que es posible declarar por medio de funciones cuál es el límite o par de límites (límite de control inferior y superior) para calcular el verdadero límite central de la muestra.

2.1.3. Matrices y *arrays*

Como se ha mencionado anteriormente, una matriz es la generalización de los vectores, con el atributo adicional, el de dimensión o *dim* y los *arrays* son matrices en términos sencillos, que contienen otras matrices.

2.1.3.1. Crear una matriz

Para crear una matriz es necesario definirla a partir de uno o varios vectores, inclusive de un escalar, definir el número de filas y el número de columnas en forma optativa cualquiera de los dos parámetros; declarar si los datos llenan las columnas (por defecto) o las filas primero (optativo) y por último fijar de manera optativa los nombres de las filas y columnas (por defecto, son numeradas) por medio los siguientes argumentos:

matrix(datos, nrow, ncol, byrow=FALSE, dimnames=NULL)

Notar el argumento *byrow*, el cual por defecto es FALSO, quiere decir que los datos llenarán la matriz de orden *m* x n, de columna en columna, de lo contrario, se llenan de fila en fila.

Por otro lado, si uno de los argumentos *nrow* o *ncol* no es proporcionado, R intentará inferirlo por la longitud del vector de datos y el otro parámetro existente. En el siguiente ejemplo se observa un vector de longitud 9 que se adapta perfectamente a una matriz **X** de 3 x 3.

```
> X<-matrix(c(2,1,0,-1,-3,2,4,0,3),nrow=3,byrow=TRUE), X
[,1] [,2] [,3]
[1,]    2    1    0
[2,]   -1   -3    2
[3,]    4    0    3
```

Mientras que la matriz **Y**, el vector tiene una longitud 3, pero el argumento *nrow* define la existencia de 2 filas, por lo tanto, el máximo de columnas será el próximo entero de $\frac{length}{nrow\ o\ ncol}$**:**

```
> Y<-matrix(c(1,4,-2),nrow=2,byrow=TRUE)
Warning message:
In matrix(c(1, 4, -2), nrow = 2, byrow = TRUE) :
  data length [3] is not a sub-multiple or multiple of the number of rows [2]
> Y
     [,1] [,2]
[1,]    1    4
[2,]   -2    1
```

Como resultado, el número de columnas debe de ser 2 y el dato que se agrega al cuarto lugar a llenar, es el primero hasta el iésimo que falte para llenar la matriz de *m* x *n*. Otra fuente para crear matrices consisten en el uso de las funciones *cbind*( ) y *rbind* ( ), para utilizar la función *matplot*( ) es necesario concatenar varias columnas con *cbind*( ) para integrar las matrices **x** y **y** como argumentos de la función.

2.1.3.2. Índice de una matriz y dimnames

Tal y como sucede con un vector, el índice de una matriz es la descripción de submatrices tal y como se explicó en la sección 1.4.2. No obstante, una vez asignada la lista de nombres a las filas y columnas, la funcionalidad del índice numérico de filas y columnas permanece intacta, pero ahora se puede indexar con nombres.

Asumiendo que la matriz X.t corresponde a un arreglo en el que las filas constituyen origen, y las columnas son el destino de un problema de transporte.

```
> X.t<-matrix(c(3,2,4,1,4,3,0,2,0,2,5,1,2,6,0,4),byrow=TRUE,ncol=4)
> X.t
     [,1] [,2] [,3] [,4]
[1,]    3    2    4    1
[2,]    4    3    0    2
[3,]    0    2    5    1
[4,]    2    6    0    4
> # Los nombres de filas y columnas se asignan por listas
> dimnames(X.t)<-list(paste("Origen",1:4,sep=" "),
+ paste("Destino",1:4, sep=" "))
> X.t
```

```
   Destino 1     Destino 2     Destino 3     Destino 4
Origen 1      3             2             4             1
Origen 2      4             3             0             2
Origen 3      0             2             5             1
Origen 4      2             6             0             4
> X.t[2,3]
[1] 0
> #Notar que los nombre deben escribirse entre comillas " "
> X.t["Origen 2", "Destino 3"]
[1] 0
```

Al citar columnas filas o columnas enteras, tan solo es necesario escribirlo con la sintaxis: matriz[fila,] o matriz[,columna]; al igual funcionan los nombres o *dimnames* entre comillas.

```
> X.t[,"Destino 4"]
Origen 1Origen 2      Origen 3      Origen 4
     1             2             1             4
> X.t["Origen 1",]
Destino 1     Destino 2     Destino 3     Destino 4
3             2             4             1
```

2.1.3.3. Operaciones con matrices

Para exponer un ejemplo práctico de las operaciones con matrices se hará uso del álgebra lineal aplicada a un modelo de regresión lineal, en ese sentido, se asume una matriz X con las variables explicativas o independientes y una matriz Y con la variable explicada o dependiente. El modelo lineal general se expresa como:

$$Y=X\beta+\varepsilon$$

Donde β es la matriz de coeficientes a calcular por método de mínimos cuadrados. Las matrices de datos se pueden anotar como:

$$\mathbf{Y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \qquad \mathbf{X} = \begin{bmatrix} 1 & x_{11} & x_{12} & \cdots & x_{1k} \\ 1 & x_{12} & x_{22} & \cdots & x_{2k} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_1 & x_{n1} & x_{n2} & \cdots & x_{nk} \end{bmatrix} \qquad \widehat{\boldsymbol{\beta}} = \begin{bmatrix} \hat{\beta}_0 \\ \hat{\beta}_1 \\ \vdots \\ \hat{\beta}_k \end{bmatrix} \qquad \boldsymbol{\varepsilon} = \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_n \end{bmatrix}$$

De ello, la ecuación de matrices de mínimos cuadrados es:

$$(\mathrm{X}'\mathrm{X})\hat{\beta} = \mathrm{X}'\mathrm{Y}$$

Se observa la matriz de coeficientes $(\mathbf{X}'\mathbf{X})$ de los estimadores de los mínimos cuadrados $\widehat{\boldsymbol{\beta}}_{\mathbf{0}}, \widehat{\boldsymbol{\beta}}_{\mathbf{1}}, \widehat{\boldsymbol{\beta}}_{\mathbf{2}} \ldots \widehat{\boldsymbol{\beta}}_{\mathbf{k}}$, mientras que $\mathbf{X}'\mathbf{Y}$es la matriz de constantes, de tal forma que trasponiendo $(\mathbf{X}'\mathbf{X})$se tiene que la solución a los mínimos cuadrados es:

$$\hat{\beta} = (\mathrm{X}'\mathrm{X})^{-1}\mathrm{X}'\mathrm{Y}$$

Haciendo uso de la información citada en la sección 1.9.2, en la que se establece una relación entre la distancia que recorre un vehículo antes de llegar al reposo partiendo de una velocidad inicial.

El resultado de $(\mathrm{X}'\mathrm{X})$ da como resultado la matriz:

$$(\mathbf{X}'\mathbf{X}) = \begin{bmatrix} n & \sum_{i=1}^{n} x_i \\ \sum_{i=1}^{n} x_i & \sum_{i=1}^{n} x_i^2 \end{bmatrix}$$

```
> Y<-cbind(cars$dist)
> X<-cbind(cars$speed,1)
```

Notar que se captó la información a una lista, luego se extrae a dos matrices (X y Y) mediante la función *cbind*( ). En el caso de la matriz **X** se agrega la columna 1, con la constante 1 para todas las filas.

Para obtener el producto $(\mathbf{X}'\mathbf{X})$es necesario usar la sintaxis de R para la transpuesta de una matriz, se utiliza la función *t*() y la simbología para la multiplicación de matrices en el sentido lineal es %*%.

```
> t(X)%*%X
     [,1]   [,2]
[1,] 13228  770
[2,]   770  50
```

El producto $\mathrm{X}'\mathrm{Y}$ da como resultado:

```
> t(X)%*%Y
        [,1]
[1,]  38482
[2,] 2149
```

La matriz inversa $(\mathrm{X}'\mathrm{X})^{-1}$ se logra mediante la función *solve*( ) y la condición necesaria es que el sistema lineal deba de ser cuadrado.

```
> solve(t(X)%*%X)
            [,1]          [,2]
[1,]  0.000729927    -0.01124088
[2,] -0.011240876     0.19310949
```

Con los anteriores resultados, se está listo para aplicar $\hat{\beta} = (X'X)^{-1}X'Y$ y así determinar los coeficientes del modelo mediante la regresión lineal.

```
> solve(t(X)%*%X)%*%(t(X)%*%Y)
  [,1]
[1,]       3.932409
[2,]       -17.579095
```

Para comprobar el valor de los estimadores, se utiliza la misma información de con la función *lm*( ), los resultados son los siguientes:

```
> reg<-lm(Y~X[,1])
>Call:
lm(formula = Y ~ X[, 1])
Residuals:
Min              1Q          Median          3Q          Max
-29.069          -9.525          -2.272          9.215          43.201
Coefficients:
            Estimate        Std. Error      t value         Pr(>|t|)
(Intercept)     -17.5791        6.7584          -2.601          0.0123 *
X[, 1]          3.9324          0.4155          9.464           1.49e-12 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
Residual standard error: 15.38 on 48 degrees of freedom
Multiple R-squared:  0.6511,    Adjusted R-squared:  0.6438
F-statistic: 89.57 on 1 and 48 DF,  p-value: 1.49e-12
```

Los estimadores $\hat{\beta}_1$ =-17,58 y $\hat{\beta}_0$ = - 3,9324, coinciden con los calculados con *lm(*). Otras operaciones con matrices se calculan por medio de las siguientes funciones:

Tabla XVI. **Operaciones con matrices**

Descripción	**Función *f( )* o símbolo**
Determinante de una matriz X	*det*
Inversa de una matriz X	*solve*
Dimensiones de una matriz X	*dim*
Producto de la matriz A por B	A%*%B
Convertir un data.frame a matriz	data.matrix
Matriz diagonal	*diag*
Máximo valor de una matriz	*max.col*
Calcular la normal de una matriz	*norm*
Calcular la suma de columnas	*rowsum*
Producto cruz	*crossprod y tcrossprod*
Calcular los eigenvalores o eigenvectores	*eigen*
Matriz triangular superior o inferior	*upper.tri*

Fuente: elaboración propia con ayuda de R <help(base)>.

2.1.4. Listas

La lista como objeto de datos en R, viene siendo un arreglo ordenado de componentes; cada componente es a su vez un objeto y tiene su propio tipo de dato.

En el caso de las listas, cada objeto puede tener una longitud distinta, inclusive nula o *NULL*. Ya que esos componentes tienen las mismas características de un vector, pueden indexarse con el mismo método de los objetos vistos como lista$componente$[n].

Para crear una lista se utiliza la función *list*( ) en la cual es posible alimentar la información de otros objetos ya creados. Lo anterior implica que se pueden hacer listas de listas al concatenarlas con la función *c*( ) y cada una como argumento, ver el siguiente ejemplo:

```
> cc<-list(Actividades=c("Control de Proceso", "Causas Asignables"),
+ resultados=c("Aceptable", "No Aceptable"),
+ valor=seq(0,1,0.25), usuario="Inspector AM")
> class(cc)
[1] "list"
> cc
$Actividades
[1] "Control de Proceso" "Causas Asignables"
$resultados
[1] "Aceptable"    "No Aceptable"
$valor
[1] 0.00 0.25 0.50 0.75 1.00
$usuario
[1] "Inspector AM"
```

Ahora se crea una lista **cc2** de actividades para otro usuario “Inspector PM” y se concatenan ambas en la lista **ccGeneral**.

```
> ccGeneral<-c(cc,cc2)
> ccGeneral
$Actividades
[1] "Control de Proceso" "Causas Asignables"
$resultados
[1] "Aceptable"    "No Aceptable"
$valor
[1] 0.00 0.25 0.50 0.75 1.00
```

```
$usuario
[1] "Inspector AM"
$Actividades
[1] "Control de Proceso" "Causas Asignables"
$resultados
[1] "Aceptable"    "No Aceptable"
$valor
[1] 0.00 0.25 0.50 0.75 1.00
$usuario
[1] "Inspector PM"
```

2.1.5. Hoja de datos (*data.frame*)

En el capítulo 1 se anotaron ampliamente varias formas de crear una *data.frame* a partir de información proveniente de archivos de texto, hojas, electrónicas y sitios en internet. De hecho, todo indica que es la forma más habitual de la estructura de información cuando es un arreglo de variables.

Estos objetos son las estructuras que se asemejan a los paneles de datos, similares a las matrices, por ello, es en efecto una colección de vectores, con nombre a cada vector y a cada fila como atributos (*names* y *row.names,* respectivamente). Cada columna representa una sola variable y cada fila es una sola observación.

Uno de aspectos particulares, es que todas las variables deben de tener la misma longitud (*length*), pero cada columna puede diferir de las otras en el tipo de dato (numérico, lógico, factor, carácter).Para acceder a una variable la sintaxis es *data.frame*$*variable*, pero al no conocer o recordar los nombres del objeto, se utiliza la función *names*(objeto) para visualizarlos o para asignar.

Mediante la notación conocida ya como índice, es posible cambiar el nombre de una sola variable. Por ejemplo, en el siguiente ejemplo se cambia o asigna el nombre “color” a la columna 3 del *data.frame* de nombre “info”.

```
names(info)[3]<-“color”
```

Al igual que una matriz, una hoja de datos o *data.frame*, tiene la facultad de permitir nombrar las filas, ya que las columnas de hecho tienen nombre. Para este efecto se especifica el nombre de las filas por medio del argumento *row.names*, que puede ser el nombre de la variable o el número de la columna que contiene los nombres o por medio de la función *row.names*( ) a la cual se le asigna un vector del tipo *character*.

En el siguiente ejemplo, se diseña un marco muestral de una manufactura de hilos, de cinco colores distintos, dos tamaños y dos turnos de fabricación para determinar si cumplen con ciertos atributos vinculados al material de empaque.

```
> vcol<-c("rojo","amarillo","azul","negro","blanco")
> scol<-sample(vcol,10,replace=TRUE)
> scol
 [1] "azul"   "rojo"   "negro"  "negro"  "negro"  "blanco" "rojo"   "negro"
 [9] "rojo"   "negro"
tam<-sample(c("Grande","Mediano","Pequeño"),10,replace=TRUE)
> tur<-sample(c("AM","PM",10,replace=TRUE)
> muestra<-data.frame(Turno=tur,Tampaño=tam,Color=scol)
> muestra
  Turno Tampaño      Color
1    PM Pequeño      azul
2    PM Pequeño      rojo
```

```
3    PM  Grande      negro
4    AM  Grande      negro
5    PM  Grande      negro
6    AM  Pequeño     blanco
7    AM  Pequeño     rojo
8    PM  Pequeño     negro
9    AM  Grande      rojo
10   AM  Pequeño     negro
> names(muestra)
[1] "Turno"   "Tampaño" "Color"
```

Como se puede apreciar, en el marco muestral existen tres variables, una de ellas, su nombre “Tampaño” debería estar correctamente escrito como “Tamaño”, en ese sentido se aplica lo descrito anteriormente para cambiar el nombre de dicha variable.

```
> names(muestra)[2]<-“Tamaño”
> names(muestra)
[1] "Turno"  "Tamaño" "Color"
```

Notar el uso de la función *sample*( ) que calcula una muestra de tamaño (*size*), con o sin reemplazo (*replace*) a partir de un vector de datos **x**.

Sintaxis: sample(x,size,replece=FALSE)

2.1.6. Factores

Un factor (*factor*) es objeto vectorial usado para especificar una agrupación discreta de otro vector del mismo tamaño (*length*). En el siguiente ejemplo se asume un programa de control de calidad de una muestra de 28

hilos, en los cuales se seleccionó en un experimento sin reemplazo y los hilos de colores que salieron defectuosos por color están agrupados en el vector “defecto”.

```
>defecto<-c("blanco","negro","azul","rojo","blanco","verde","rojo",
+"blanco","negro","verde","azul","amarillo","naranja","blanco",
"rojo","verde","azul","blanco","blanco","azul","naranja",
+"verde","blanco","blanco","negro","azul","blanco","negro")
>summary(defecto)
   Length     Class      Mode
       28 character character
```

Ahora se procede a crear el vector factor llamado “defectof” con la función *factor*(vector) y luego obtener el listado resumido de los nombres de los valores; usar la función *levels*(vector).

```
>defectof<-factor(defecto)
>levels(defectof)
[1] "amarillo" "azul"     "blanco"   "naranja"  "negro"    "rojo"     "verde"
>summary(defectof)
amarillo     azul   blanco  naranja    negro     rojo    verde
       1          5         9          2          4          3       4
```

Los niveles encontrados en los objetos tipo factor, son interpretados como números enteros. Los que resultan son una secuencia de los niveles de los valores de las cadenas de texto, ordenadas en orden alfabético por defecto; sin embargo es un argumento optativo dentro de la función.

```
> as.integer(defectof)              #Orden alfabético
 [1] 3 5 2 6 3 7 6 3 5 7 2 1 4 3 6 7 2 3 3 2 4 7 3 3 5 2 3 5
```

```
> defectof<-factor(defecto, ordered=is.ordered(defecto),
+ exclude=c("blanco"))          #
> defectof
 [1] <NA>    negro   azul    rojo    <NA>    verde   rojo    <NA>
 [9] negro   verde   azul    amarillo naranja <NA>   rojo    verde
[17] azul    <NA><NA>    azul    naranja verde   <NA><NA>
[25] negro   azul    <NA>    negro
Levels: amarillo azul naranja negro rojo verde
> as.integer(defectof)
 [1] NA  4  2  5 NA  6  5 NA  4  6  2  1  3 NA  5  6  2 NA NA  2  3  6 NA NA  4
[26]  2 NA  4
```

El objeto factor tiene características diferentes basadas en los datos del vector original, pero ahora se puede observar mediante la función *summary*( ) el resumen de los valores ordenados.

La función *gl*( ) produce un factor con un patrón especificado de acuerdo con los argumentos tales como: *n* define el número de niveles; *k* es el número de repeticiones; *length* es la longitud del resultado; *n*, *k* y *length* son enteros.

```
> x.f.2<-gl(3,2,21)
> x.f.2
 [1] 1 1 2 2 3 3 1 1 2 2 3 3 1 1 2 2 3 3 1 1 2
Levels: 1 2 3
> class(x.f.2)
[1] "factor"
```

Con la función *expand.grid*( ) se pueden concatenar factores, vectores o listas para crear objetos tipo *data.frame*, ejemplo:

```
>muestra<-expand.grid(dinterno=c(0.1,0.11,0.12),          turno=c("AM","PM"),
```

largo=c(10,20))

> muestra

dinterno turno largo

1 0.10 AM 10

2 0.11 AM 10

3 0.12 AM 10

4 0.10 PM 10

: : : :

11 0.11 PM 20

12 0.12 PM 20

En tal caso se observa que la hoja de datos o *data.frame* de nombre **muestra** está integrada por la combinación de tres diámetros internos, combinados en dos turnos de producción y con dos largos distintos. Esta función tiene utilidad para combinaciones de aditivas con una longitud igual a:

$$L = \prod_{i=1}^{n} l_i$$

Es decir, es el producto de la longitud de los vectores, listas o factores incluidos, en este caso es$3 \times 2 \times 2 = 12$.

> dim(muestra)

[1] 12 3

2.1.7. Funciones internas

Las funciones internas en R también son objetos, aunque no se tratan de objetos de información, las funciones permiten el eficiente tratamiento de datos, entre ellas están las funciones como *tapply*, *apply, lapply, sapply.*

2.1.7.1. Función *tapply*( )

Esta función en especial es útil para el tratamiento estadístico de datos, como los objetos *factor*( ). Esta función aplica una función a los subconjuntos de un vector. Por ejemplo, se pretende saber el promedio de peso (masa) en kilogramos de los hilos encontrados como defectuosos, la lista de los valores corresponde a cada valor del vector **defectof**.

```
> peso<-c(2,1.8,2.1,1.8,2.01,1.95,1.8,1.985,1.85,
+ 2.03,2.05,1.95,1.98,2.01,1.9,2.0,1.99,2.3,2.4,
+ 1.93,1.99,2.05,2.4,2.0,1.98,1.94,2.3,2.0)
>pesoprom<-tapply(peso,defectof,mean)
>summary(peso)
Min. 1st Qu.Median    Mean 3rd Qu.    Max.
1.800   1.947   1.995   2.018   2.035   2.400
>pesoprom
amarillo azul   blanco  naranja    negro     rojo   verde
1.950000 2.002000 2.156111 1.985000 1.907500 1.833333 2.007500
```

Se obtuvieron los valores del vector **peso** mediante la función *summary*( ), pero se desea conocer los valores de los promedios en kilogramos, correspondientes a cada color de hilo defectuoso de la muestra de 28 unidades.

Aunque el tamaño de la muestra es pequeño, se puede notar que existe cierta vinculación del peso promedio (*mean*) con algunos colores.

```
resultado<-tapply(origen, factor, mean)
```

Sin embargo, la función promedio o media (*mean*) es una de tantas que se pueden emplear en la función *tapply*( ), no obstante se pueden construir otras

funciones mediante *function*(x).

```
>pesopromedio<-function(x)(sum(x)/length(x))
```

Construida la función "*pesopromedio*" que es igual a la función *mean*, se puede emplear el argumento FUN de *tapply*( ).

```
> pesoprom2<-tapply(peso, defectof, pesopromedio)
> pesoprom2
amarilloazul  blanco  naranja   negro    rojo   verde
1.950000 2.002000 2.156111 1.985000 1.907500 1.833333 2.007500
```

2.1.7.2. Función *apply*( )

La función *apply*( ), regresa un vector resultante al aplicar una función al margen de una matriz, la cual de no serlo, R lo coerciona a una matriz aplicando implícitamente la función *as.matrix*( ) al objeto. El vector resultante se calcula haciendo uso de las funciones internas y otras definidas por el usuario, tal y como se describió con la función *tapply*( ).

La sintaxis de la función es: apply(X, MARGIN, FUN).

Donde X es la matriz o arreglo; *MARGIN* es el argumento que define que siendo 1 la función se aplica a las filas; 2 para las columnas; c(1,2) indica que la función se aplica a filas y columnas; por último, *FUN* es la función que se aplica para obtener el vector resultante.

Esta función es sumamente útil cuando se necesite hacer cálculos en todas las filas y las columnas del objeto de datos.

En el siguiente ejemplo, se integran 10 grupos de muestras de 5 cada una, conteniendo el dato de pesos en kilogramos de una sustancia utilizada como emulsionante en ciertos procesos químicos, la cual es empacada en presentaciones estándar de 10 kilogramos. Para lo cual se desea conocer el rango de variación en cada grupo de muestra, así como el promedio en kilogramos.

Este ejercicio es un típico ejemplo del tema de control estadístico de la calidad, que se desarrollará con mayor detalle en el capítulo 3.

Los pesos son generados por una función de distribución uniforme por medio de la función *runif*( ) y con la función *rnorm*(n, media), en la que *n* es el tamaño de la muestra y *mean* es la media de la que alrededor se generan los valores. A medida que la muestra n crece la media de los valores generados tienden a la media.

```
> pesos<-cbind(X1=runif(10,9.885,10.1),X2=runif(10,9.885,10.1),
+ X3=runif(10,9.885,10.1),X4=runif(10,9.885,10.1),
+ X5=runif(10,9.885,10.1))
> # Calcular el promedio de cada fila
> peso.prom<-apply(pesos,1,mean); peso.prom
[1]  9.986707  9.987317  9.956908  9.991038  9.964957
[6] 10.015092  9.937202 10.015675  9.986008  9.960183
> # Calcular el máximo y mínimo de cada fila
> max.rango<-apply(pesos,1,max)
> min.rango<-apply(pesos,1,min)
> # Calcular el rango de la fila
> peso.rango<-max.rango-min.rango
> resultado<-cbind(pesos, peso.prom, peso.rango)
> round(resultado,3)
```

```
        X1     X2     X3     X4     X5     peso.prom  peso.rango
 [1,]   9.901  9.968  9.983  10.087 9.994  9.987      0.186
 [2,]   9.919  10.050 9.972  9.914  10.082 9.987      0.169
 [3,]   9.961  9.895  9.961  9.939  10.028 9.957      0.133
 :      :      :      :      :      :      :          :
[10,]   9.905  10.089 9.923  9.999  9.886  9.960      0.203
> plot(c(1:10),round(resultado[,6],2),type="b",
+ main="Gráfico X", xlab="Muestra",
+ ylab="Pesos en Kg.",col="blue")
abline(h=mean(resultado[,6]),col="red")
```

Como resultado de ello, es posible plotear por ejemplo los promedios de las muestras alrededor de la gran media $\bar{\bar{X}}$.

2.1.7.3. Funciones *lapply*( ) y *sapply*( )

En el caso de la función *lapply* aplica una función (FUN) como argumento a un vector o lista (X) y retorna una lista como resultado.

Sintaxis: lapply(X, FUN, …)

Los demás argumentos de la función a aplicar se denotan por los tres puntos "…" al final de la sintaxis. Recordar que la sintaxis aparece ampliamente especificada en la ayuda de R; únicamente las funciones que permiten ampliar los argumentos, presentan los tres puntos.

La función *sapply*, tiene por objetivo simplificar los resultados de la anterior función. Por ejemplo se tiene una lista de valores que describen mediciones de pesos en gramos de varias muestras del mismo producto de consumo masivo, pero de distinto tamaño.

Un objeto tipo lista es la disposición de datos adecuado para el ejemplo y los valores son generados por medio de la función *rnorm*( ).

```
> weigth.gr<-list(Lunes=rnorm(20,120),
+ Martes=rnorm(10,120),
+ Miercoles=rnorm(50,120))
> lapply(weigth.gr,mean)
$Lunes
[1] 120.4539
$Martes
[1] 119.8539
$Miercoles
[1] 119.9685
> # Simplificando los resultados con la función sapply
> sapply(weigth.gr,mean)
Lunes          Martes         Miercoles
120.4539       119.8539       119.9685
```

Esta función es muy importante para los cálculos relacionados con listas, con la necesidad técnica de generar listas de acuerdo con los resultados. Utilizando los datos del ejercicio anterior y llevando las primeras dos muestras de tamaño cinco como elementos de una lista.

```
> lpesos<-list(n1=as.numeric(pesos[1,]),n2=as.numeric(pesos[2,]))
> lpesos
$n1
[1]  9.900996  9.967926  9.983455 10.087109  9.994047
$n2
[1]  9.918627 10.049944  9.971633  9.913900 10.082482
> lapply(lpesos,mean)
$n1
```

```
[1] 9.986707$n2
[1] 9.987317
```

2.2. Sintaxis y elementos de lenguaje R

Dentro de la lógica de R, como lenguaje orientado a objetos, se han revisado hasta acá, objetos de datos, los objetos funciones y en forma práctica se ha hecho uso de los objetos de lenguaje, como las expresiones, llamadas y nombres (*expressions*, *calls* y *names,* respectivamente). También se ha empleado ya la clase de un objeto; ahora es turno de los métodos.

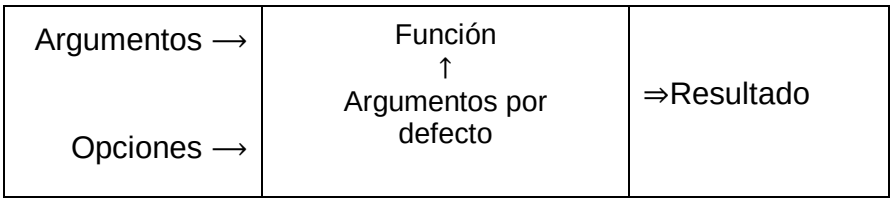
El método dentro del lenguaje es básicamente una función específica para efectuar cálculos específicos en objetos de una clase específica, esto lo hace un lenguaje orientado a objetos. Por ejemplo se tiene la función genérica *mean y* se desea conocer cuáles son los métodos en orden aplicados a la clase de objeto *data.frame*:

```
> methods(mean,data.frame)
[1] mean.Date     mean.default  mean.difftime mean.POSIXct  mean.POSIXlt
> mean.Date
function (x, ...)
structure(mean(unclass(x), ...), class = "Date")
<bytecode: 0x0a4f6a80>
<environment: namespace:base>
> class(mean.Date)
[1] "function"
```

Cada método es a su vez una función en sí y tiene especificado un ambiente o *environment* en el cual la función debe ser evaluada. En el caso del método *mean.default*, al llamarlo desde el *prompt*, R muestra el código de

programa que se ejecuta bajo el siguiente esquema:

Figura 48. **Esquema de sintaxis de las funciones**



Fuente: elaboración propia.

La siguiente función, *mean*, al llamarla desde el *prompt*, despliega, la función interna que cumple sí y solo sí se cumple con el ambiente y los argumentos necesarios para su interpretación.

```
> mean.default
function (x, trim = 0, na.rm = FALSE, ...)
{
    if (!is.numeric(x) && !is.complex(x) && !is.logical(x)) {
        warning("argument is not numeric or logical: returning NA")
        return(NA_real_)
    }
    if (na.rm)
        x <- x[!is.na(x)]
    if (!is.numeric(trim) || length(trim) != 1L)
        stop("'trim' must be numeric of length one")
    n <- length(x)
    if (trim > 0 && n) {
        if (is.complex(x))
            stop("trimmed means are not defined for complex data")
        if (anyNA(x))
```

```
            return(NA_real_)
        if (trim >= 0.5)
            return(stats::median(x, na.rm = FALSE))
    lo <- floor(n * trim) + 1
        hi <- n + 1 - lo
        x <- sort.int(x, partial = unique(c(lo, hi)))[lo:hi]
    }
      .Internal(mean(x))
    }
    <bytecode: 0x0a4f74f0>
    <environment: namespace:base>
```

Notar: el método de la función, al final llama a la función interna llamada *mean*. Las funciones internas son interpretadas directamente por R.

2.2.1. Expresiones

Los objetos del modo *expression* son una colección de una o más declaraciones en la sintaxis del lenguaje. Se comportan de forma especial, ya que sus declaraciones son analizadas pero no evaluadas. La forma para evaluar una expresión es mediante la función *eval*( ). En el capítulo uno pudo apreciarse que se han utilizado ya los objetos tipo *expression* para evaluar una distribución de probabilidad continua y escribir una expresión simbólica dentro de una gráfica.

A su vez, puede contener a una o más expresiones, que se comportan como una lista y sus componentes como tal deben de ser accedidos como una lista. Por ejemplo, se plantea el cálculo de la sumatoria de los elementos de un vector y luego calcular la media de los valores mayores que 5.

```
> x<-runif(15,3,10)
```

```
> ex.1<-expression(x,mean(x[x>5])); ex.1
expression(x, mean(x[x > 5]))
> ex.1[[1]]
x
> ex.1[[2]]
mean(x[x > 5])
> eval(ex.1)
[1] 6.974142
```

Otra forma útil en matemáticas y el cálculo es creando expresiones a las que se les asigna un valor numérico, para luego hacer uso de las función de primera derivada de funciones simples con *D*(expr,name) donde *expr* es la expresión a derivar y *name* es la variable respecto de la que calculará la derivada.

```
#Se declaran las variables
> x<-3;exp(x)->y; x
[1] 3
>y
[1] 20.08554
>x+log(y)
[1] 6
>ec<-expression(2*x-1/(x+3*y)); ec
expression(2 * x - 1/(x + 3 * y))
# Evaluar la expresión
>eval(ec)
[1] 5.984191
# Primera derivada parcial respecto a x
>D(ec,"x")
2 + 1/(x + 3 * y)^2
# Segunda Derivada parcial respecto a y
```

```
>D(D(ec,"x"),"y")
-(2 * (3 * (x + 3 * y))/((x + 3 * y)^2)^2)
```

2.2.2. Declaraciones y ciclos

Como en otros lenguajes de programación, el código y la sintaxis del flujo de control es muy similar. Entiéndase en este caso como la declaración IF – THEN - ELSE, la cual se asume un comportamiento específico si la condición es verdadera y otro si la condición es falsa.

La sintaxis para ejecutar una declaración se auxilia de las llaves “{ }” para encerrar las expresiones antes de que el lenguaje interprete las condiciones.

La sintaxis es:

```
If (condición) {
  conjunto de expresiones
}
```

La combinación de *if* con *else* tiene una amplia ventaja, ya que se incorporan declaraciones en cadena.

```
If (condición) {
  conjunto de expresiones
} else {
conjunto de expresiones
}
```

Los ciclos permiten a un programa repetir la ejecución de comandos, de ellos, se revisará por orden de ideas a los ciclos para volver la declaración “if”.

2.2.2.1. Ciclo *for*

El ciclo o bucle “for” se basa en la necesidad de calcular valores por iteraciones. En ella se declara un valor inicial y una condición de control final. Habitualmente se utiliza una variable de control “i”, no obstante a criterio del programador puede llamarse con otro nombre “no reservado”

La forma general o sintaxis del ciclo *for* es:

```
for (la variable en secuencia) {
  conjunto de expresiones
  }
```

Por ejemplo, se desea conocer la sumatoria de los números pares entre cero y 100.

```
par<-2
for (i in seq(0,100, by=2)){
  par=par + i
  }
> par
[1] 2552
```

2.2.2.2. Ciclo *while*

El ciclo *while* es una declaración que se cumple las veces que sean necesarias hasta que la condición asignada deje de ser verdadera, de lo contrario el ciclo se repite.

Forma general o sintaxis del ciclo *while*:

```
while (condición) {
  conjunto de expresiones
  }
```

Se se desea conocer los números impares menores que 15 se escribe el siguiente código.

```
x<-0; y<-0
while(y<15){
  print(y[y>0])
  y<-2*x+1
  x<-x+1
  }
```

El resultado es una secuencia de valores escalares de cada expresión que calcula un número entero impar.

```
numeric(0)
[1] 1
[1] 3
[1] 5
[1] 7
[1] 9
[1] 11
[1] 13
```

Ahora en el siguiente ejemplo se calculan los valores del cálculo de interés compuesto, con un capital K inicial de Q1, una tasa r del 10% a un plazo de 10 años, los resultados alimentan un vector K.

```
r<-0.1; K<-1; n<-10
```

```
while (length(K)<n+1){
  Ki<-length(K)
  K1<-K[1]*(1+r)^(Ki)
  K<-c(K,K1)
  }
> round(K,2)
[1] 1.00 1.10 1.21 1.33 1.46 1.61 1.77 1.95 2.14 2.36 2.59
```

Como se puede apreciar, los valores parten de n=0 hasta n=10, para cumplir con la función matemática para el interés compuesto:

$$K_n = K_0(1 + r)^n$$

2.2.2.3. Ciclo *repeat*

El ciclo *repeat* tiene como lógica la de comprobar la declaración al final del cuerpo del ciclo, si esta es cierta, se continúa con el resto del programa, de lo contrario, se finaliza.

```
repeat (condición) {
  conjunto de expresiones
  if (condición) {break}
  }
```

Para ejemplo de este ciclo, se calcula los primeros cuatro números pares:

```
x<-0; y<-0
repeat {
  print(y[y>0])
  y=2*x
```

```
  x=x+1
  if (x>5) {break}
  }
numeric(0)
[1] 2
[1] 4
[1] 6
[1] 8
```

2.2.3. Contador

En los trozos de código o *script* s que se presentarán posteriormente, será necesario utilizar ciclos que comparen los valores resultantes como los promedios de las muestras para compararlos con los límites de control y en el caso más simple, eliminarlos de la primera lista e iteración, para calcular los nuevos valores de los límites de control en el control estadístico de la calidad.

En casos más complejos se leerán los posibles patrones en los resultados para argumentar que el proceso está o no fuera de control. En el siguiente ejemplo se genera una serie de números respecto de una media y normalmente distribuidos; dicho arreglo se ajusta para efectos didácticos a una carta $\bar{X}$ en control estadístico de calidad y el promedio será el gran promedio o promedio de promedios o $\bar{\bar{X}}$.

Para calcular los límites de control se utiliza la tabla de factores para límites de control en el anexo 1, para muestras hasta un tamaño *n* de 10 y las fórmulas para límite de control para variables.

Los vectores de las medias y los rangos son los siguientes:

```
> X<-round(runif(25,455,465),1)
```

```
> R<-round(runif(25,1,10),1)
```

El tamaño *n* de la muestra se asume de 5 y la cantidad *m* de grupos tomados es de 25. Los datos corresponden a pesos netos en gramos de un producto de consumo masivo. Se desea conocer qué cantidad de muestras o puntos en la gráfica $\bar{\bar{X}}$ están fuera de los límites de control para definir que el proceso está fuera de control. Los rangos *R* de cada grupo de muestra fueron generados aleatoriamente.

La tabla de los factores, se lee también en R, para guardarla en la memoria como otro objeto para prescindir de hacer búsquedas manuales:

```
> factor<-read.table("clipboard", header=TRUE)
```

El siguiente cálculo corresponde a los límites de control que servirán de criterio en *script* del contador.

```
> n<-5
> LCS<-expression(mean(X)+factor$A2[n-1]*mean(R))
> LCI<-expression(mean(X)-factor$A2[n-1]*mean(R))
> LC<-expression(mean(X))
> eval(LCS)      # Calcular límite de control superior
[1] 463.1635
> eval(LCI)      # Calcular límite de control inferior
[1] 456.9965
> eval(LC)              # Calcular límite central
[1] 460.44
```

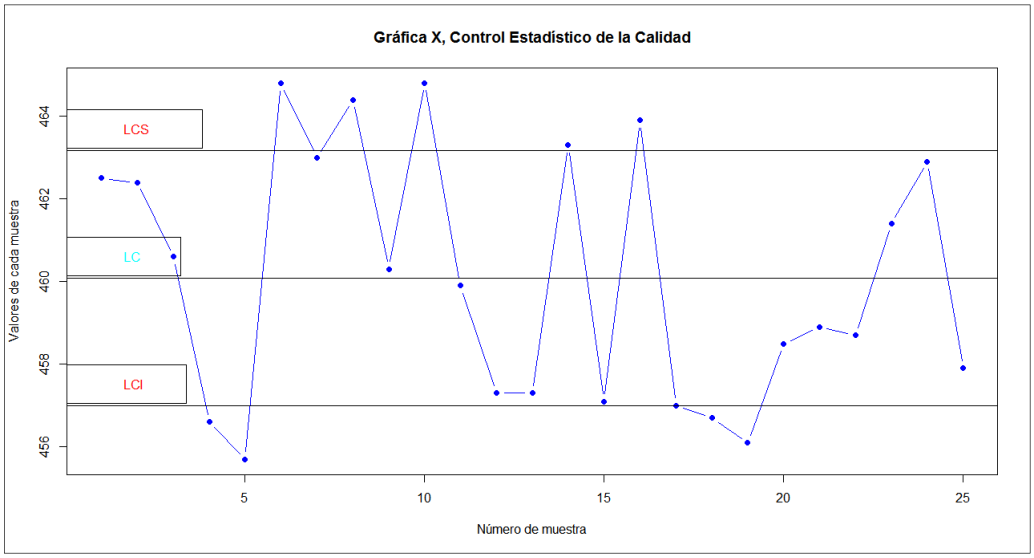
La forma manual de localizar qué puntos están fuera de los límites de control es:

```
> X[X > eval(LCS)]
[1] 464.8 464.4 464.8 463.3 463.9
> X[X < eval(LCI)]
[1] 456.6 455.7 456.7 456.1
```

Pero se necesita que el ciclo contador lea línea por línea y compare el valor del vector X con los límites de control y cuente la cantidad de muestras que están dentro de dichos límites de control y de haber uno fuera, que pinte que el proceso está fuera de control. Bajo los valores mostrados en los resultados anteriores, se espera que el proceso esté “fuera de control” apoyado visualmente de:

```
# Script  para Gráfica X
plot(X, xlab= "Número de muestra", ylab="Valores de cada muestra",
main="Gráfica X, Control Estadístico de la Calidad",type="b", col="blue",
ylim=range(X), pch =19)
abline(h= c(eval(LCS), eval(LCI), eval(LC)))
legend(0,eval(LCI)+1,"LCI", text.col="red")
legend(0,eval(LCS)+1,"LCS", text.col="red")
legend(0,eval(LC)+1,"LC", text.col="cyan")
```

Figura 49. **Gráfica de la carta $\overline{X}$**



Fuente: elaboración propia, empleando RGui.

En el siguiente *script* se aprecia que por medio del pseudocódigo del ciclo *for*, se compara muestra por muestra y se separan las que están bajo control de las que no lo están:

```
xma<-numeric(); xme<-numeric(); xnu<-numeric()
for (i in 1:length(X)){
  if (X[i] > eval(LCS)) {
          xma<-c(xma,X[i]); warning(X[i],
" Es superior al Límite de Control Superior")
  } else {
          if (X[i] < eval(LCI)) {
          xme<-c(xme,X[i]); warning(X[i],
" Es inferior al Límite de Control Inferior")
          } else {
                  xnu<-c(xnu,X[i]); warning(X[i], " Está bajo control")
```

```
                }
        }
 }
```

```
> warnings()
Warning messages:
1: 462.5 Está bajo control
2: 462.4 Está bajo control
3: 460.6 Está bajo control
4: 456.6 Es inferior al Límite de Control Inferior
5: 455.7 Es inferior al Límite de Control Inferior
6: 464.8 Es superior al Límite de Control Superior
7: 463 Está bajo control
8: 464.4 Es superior al Límite de Control Superior
: :          :
24: 462.9 Está bajo control
25: 457.9 Está bajo control
> xma
[1] 464.8 464.4 464.8 463.3 463.9
> xme
[1] 456.6 455.7 456.7 456.1
```

Notar que ahora existen dos vectores con los valores que están fuera de control en **xma** y **xme**, por otro lado el vector **xnu** que contiene los las muestras que están dentro de los límites de control, no obstante, el resultado dice que el proceso está fuera de control. Las muestras de **xma**, coinciden con la instrucción X[X > eval(LCS)], lo cual, en apariencia es mucho más práctico, pero a medida que se cuenta con objetos como una matriz o un *data.frame*, la situación no se vuelve tan sencilla, sin considerar que hay que hacer un análisis más profundo de los datos que están dentro de los límites, en función de que hay que detectar ciertos patrones no aleatorios.

Con un código en borrador, similar al practicado, es posible dicha comparación vertical entre valores, ya que la instrucción X[X > eval(LCS)] es una comparación horizontal.

2.2.4. Función y argumento

Durante todo el desarrollo previo a este inciso, se ha hecho uso de funciones, unas de ellas son completamente originarias del paquete base de R, mientras que otras corresponden a los paquetes no incluidos en el núcleo base.

El comportamiento en común es que todas tienen al menos un argumento, mientras que las más complejas observadas hasta aquí, tienen la capacidad de darle versatilidad a los resultados numéricos y gráficos, que dependerá del dominio de las mismas.

Una función, estructuralmente hablando, es una colección de comandos y expresiones que utilizan una colección finita de argumentos para darle ejecutar una tarea específica, es decir, deriva en un comportamiento específico.

Agregado a ello, es completamente posible crear nuevas funciones a partir de la siguiente sintaxis.

```
Función.nombre <- function (argumentos){
conjunto_de_expresiones
return (respuesta)
}
```

Como se observó anteriormente, la sintaxis de las funciones está especificada para adquirir determinado método, de acuerdo con el objeto de datos que va a tratar.

Empleando el ejemplo visto en 0, respecto del cálculo de un monto K_n a partir de un cK_0, con una tasa de interés compuesto r en un periodo de n unidades de tiempo, la función matemática que describe a dicho monto es $K_n = K_0(1 + r)^n$.

```
Finanzas.K<-function(r,K,n){
  while (length(K)<n+1){
  Ki<-length(K)
  K1<-K[1]*(1+r)^(Ki)
  K<-c(K,K1)
  }
  return(K[1:n+1])
}
```

Se supone un capital inicial K = 1000, una tasa de interés compuesto del 25 % anual; calcular el capital al final de tres años.

```
> Finanzas.K(r=0.25,K=1000,n=3)
 [1] 1250.000 1562.500 1953.125
> Finanzas.K(0.25,1000,n=3)
[1] 1250.000 1562.500 1953.125
> Finanzas.K(K=1000,n=3,r=0.25)
 [1] 1250.000 1562.500 1953.125
```

La respuesta es una lista del capital o monto acumulado partiendo del año uno al tres al final del periodo. Al no definir el primer dato, la función hubiera dado como primer dato el año cero, es decir, cuando no hay acumulación. Notar también el orden de los argumentos, los cuales cuando son llamados por su no importa en qué orden se indiquen; si los argumentos no son llamados explícitamente, sí es necesario especificarlos en el orden originario.

2.2.5. Depuración (*debugging*)

La depuración consiste en encontrar y reparar algunos errores lógicos y determinar por qué están sucediendo. Las indicaciones de que algo no está funcionando correctamente, por lo general están asociadas con:

- Mensajes o *message* que se dan en el momento de ejecución de las funciones.

- Advertencias o *warning* que se presentan al final de la ejecución de la función, que representa que algo no está funcionando correctamente, pero no es un error fatal y es generado por la función *warning*( ).

- Errores o *error* que ya representa un problema fatal y es la señal de que la ejecución de la función se detendrá; se genera mediante la función *stop*( ).
- Condiciones o *condition* que indica que algo inesperado puede ocurrir; estas condiciones pueden ser creadas a propósito y son de carácter genérico.

Un ejemplo de una advertencia se observó anteriormente, cuando se creó una matriz de dimensión 2 x 2, con tres datos.

```
> Y<-matrix(c(1,4,-2),nrow=2,byrow=TRUE)
Warning message:
In matrix(c(1, 4, -2), nrow = 2, byrow = TRUE) :
  data length [3] is not a sub-multiple or multiple of the number of rows [2]
> Y
```

```
     [,1] [,2]
[1,]    1    4
[2,]   -2    1
```

La advertencia, indica que la longitud del vector de los datos (3) no es submúltiplo del número de filas (2). Esta situación no detuvo la ejecución de la creación de una matriz, más bien llenó la matriz fila por fila, empezando de nuevo con el primer dato del vector para completar una longitud igual a 2 X 2. En otro ejemplo se observa una función que identifica si un número es entero.

```
es.entero<-function(n){
  if(n - round(n,0)== 0)
            print("n es un número entero")
  else
            print("n no es número entero")
  invisible(n)
  }
> es.entero(10)
[1] "n es un número entero"
> es.entero(pi)
[1] "n no es número entero"
> es.entero(x)
[1] "n es un número entero"
Warning message:
In if (n - round(n, 0) == 0) print("n es un número entero") else print("n no es
número entero") :
the condition has length > 1 and only the first element will be used
```

La ejecución de la función se efectuó pintando el resultado, sin embargo advierte que la condición empleada tiene una longitud mayor a uno, no obstante empleará el primer elemento únicamente, esto es debido a que **x** es una matriz.

En el siguiente ejemplo se contempla la posibilidad de emplear objetos como vectores o matrices para depurar aviso de advertencia:

```
es.entero<-function(n){
  if(length(n)>1)
        print("n es un vector de longitud mayor a uno")
  else if (n - round(n,0)== 0)
        print("n es un número entero")
  else
        print("n no es número entero")
  invisible(n)
  }
> es.entero(x)
[1] "n es un vector de longitud mayor a uno"
> es.entero(pi)
[1] "n no es número entero"
> es.entero(2)
[1] "n es un número entero"
```

Es necesario considerar los siguientes cuestionamientos, para depurar cualquier función:

- ¿Cuál fue el dato de entrada? ¿El dato de entrada es del tipo correcto?
- ¿Cómo fue hecha la llamada? ¿Son los argumentos correctos?
- ¿Qué era lo que se esperaba?
- ¿Qué fue lo que se obtuvo?
- ¿Qué tan diferente es lo que se obtuvo de lo que se esperaba?
- ¿Es posible reproducir el problema?
- ¿Los datos de argumento de la función son dinámicos o estáticos?

El lenguaje se apoya en funciones como *traceback*( ), *debug*( ), *browser*( ), *trace*( ) y *recover* ( ).

- *Traceback*: es la función que muestra la serie de funciones después de que un error ocurre, no hace nada si no hay ningún error. Es útil cuando una función llama a otra función y esta a su vez a otra.

```
> plot(z)
Error in plot(z) : object 'z' not found
> traceback()
1: plot(z)
```

Acá muestra que en la función implícita *mean*( ), no puede ejecutarse debido a que el objeto z, no existe.

- *Debug*: pone una bandera de modo "*debug*" o de depuración que permite ir línea por línea a través de la ejecución de la función, es decir muestra todo el código o *script* del programa.
- *Browser*: suspende la ejecución de una función donde quiera que sea llamada y pone la función en modo "*debug*". Debe escribirse en forma explícita en el *script* de la función.

```
> debug(lm)
> lm(a~x)
debugging in: lm(a ~ x)
debug: {
   ret.x <- x
   ret.y <- y
…
 if (!qr)
```

```
        z$qr <- NULL
    z
}
Browse[2]> n
debug: ret.x <- x
Browse[2]> n
debug: ret.y <- y
…
Browse[2]> n
Error in model.frame.default(formula = a ~ x, drop.unused.levels = TRUE) :
invalid type (list) for variable 'a'
```

La función *lm*( ), necesita de dos argumentos, la variable dependiente y la independiente relacionadas de la forma (y ~ x); no obstante la variable o argumento **a** no es del tipo correcto. A llegar a la última pieza del código, R muestra en el *prompt* "Browse[2]>" que indica que está en el modo *debug* y está navegando entre línea y línea del código, para pasar a la siguiente; se escribe "n" de *next* o siguiente, hasta que encuentra la línea que provoca el error, en este caso es donde indica que el objeto **a** no es del tipo correcto de dato.

Para salir del modo *debug* se utiliza la función *undebug*(FUN). Una vez puesto en estado de *debugging*, es necesario especificar cuál es el siguiente paso a analizar, por ello, al escribir "c" en el *prompt* "Browse[2]>", se ejecuta el resto de la función sin parar: al escribir "Q", es para salir del modo *debug*.

- *Trace:* permite insertar un código de depuración dentro de una función en un lugar específico, sin necesidad de cambiar la función en sí.

- *Recover*: permite modificar el comportamiento del error, de tal forma que es posible navegar entre la serie de funciones.

2.3. Creación de paquetes

La creación de paquetes resulta ser el producto tangible de la aportación y consolidación de conocimientos útiles para otros usuarios. Sin embargo, resulta ser para el mismo programador de R un reto más, ya que implica mejorar la calidad del código del paquete y sus funciones e incluir ejemplos claros para su comprensión.

Esta sección es el resumen de los pasos necesarios y estándares, para crear un paquete. Al decir estándar, es debido a que la mayoría de usuarios y primeros programadores no son “programadores de profesión”, por ello no se incluye el uso de otros lenguajes de programación como C + + o Fortran+ y por lo tanto no se necesitan de compiladores *GNU*.

Uno de los conocimientos necesarios previos a la creación de paquetes es el uso de la función *library*( ), la cual toma algunas medidas adicionales cuando el paquete *methods* es adjunto. Por otro lado, la función *require*( ), llama a un paquete previamente instalado. Los paquetes pueden hacer uso de las funciones genéricas como redefinir funciones en otros paquetes, contenidas en el núcleo base, para ser consideradas como genéricas.

2.3.1. Instalación de Rtools y compiladores

El paquete Rtools, es en sí un paquete necesario para construir nuevos paquetes o modificar el núcleo base de R. Es necesario para construir paquetes, cuando se trabaja en R instalado en sistemas operativos de MS Windows. Los demás programas o recursos adicionales, permiten al usuario construir un paquete desde la línea de comando.

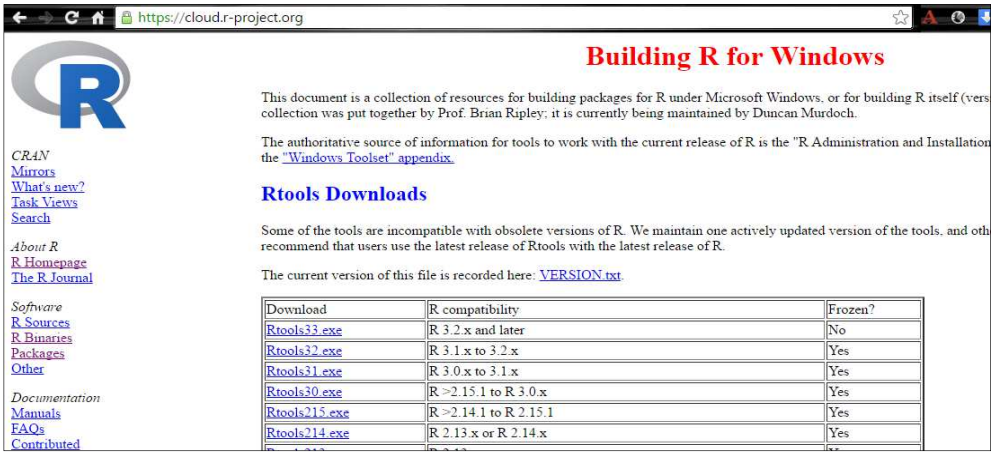
2.3.2.1. Paquete *Rtools*

El paquete *Rtools* es una herramienta que se utiliza para compilar paquetes programados, inclusive de varios lenguajes; por eso el programa es construido como un GCC o *GNU Compiler Collection*.

Para la instalación se inicia partiendo de la ruta que muestra la instalación de R. la buscar entre los recursos “R Binaries”, según la figura 50; se procede a descargar la versión estable “Rtools32.exe” que es compatible con la versión 3.2.x y posterior de R. Al instalar *Rtools*, mostrará la opción de cambiar la ruta de la variable de entorno ‘Path’ como:

PATH=c:\Rtools\bin;c:\Rtools\gcc-4.6.3\bin;<otros>

Figura 50. **Vínculo para descarga de instalador de *Rtools***

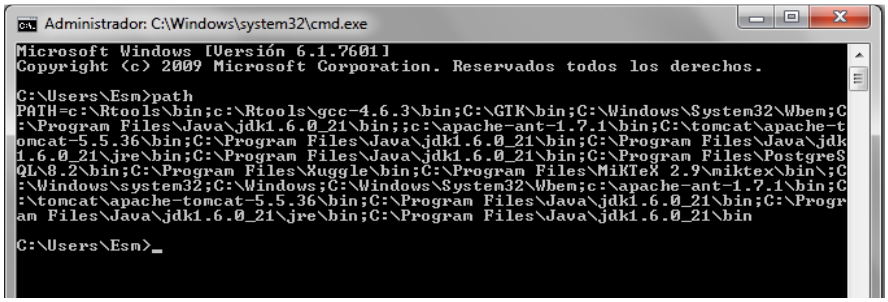


Fuente: R Project. https://cloud.r-project.org/.Consulta: agosto de 2016.

Se recomienda dejarla tal y como aparece, pero tiene que coincidir con la carpeta de instalación que también es modificable. En la línea de comando se

escribe “path” para listar las variables de entorno y luego se muestran todas las variables que se necesitan.(ver figura 51).Es importante verificar que entre las variables de entorno se encuentre la ruta del ejecutable de R. Una forma de constatar es ejecutar R desde la línea de comando es escribir “C:\Users\Esm>R”.

Figura 51**.** **Variable de entorno *Path* en la línea de comando**



Fuente: toma de pantalla línea de comando de MS Windows.

Al ejecutar el comando R, convierte a la línea de comandos en la consola de R; No obstante, suele suceder que no exista la variable de entorno de R, para ello es necesario crearla, por ejemplo, una variable de nombre:

“PATH” con la ruta[12]: C:\Program Files\R\R-3.3.1\bin\i386

La misma variable de entorno debe de utilizarse para incluir a *Rtools*, *LaTeX* y el compilador HTML. Todo lo anterior es en absoluto necesario; en el caso de no contar con *R Studio*, no obstante para efectos de alcance en el presente texto, se auxilia la plataforma gráfica de *R Studio*[13].Con anterioridad se mencionaba el entorno *RGui,* ya que es en efecto un entorno gráfico, en

[12] XML.https://www.java.com/es/download/help/path.xml.

[13]http://cran.r-project.org/doc/manuals/R-admin.html#The-Windows-toolset.

cambio al llamar a R desde la línea de comando, todas las funciones y herramientas que se necesiten, deben escribirse.

Figura 52. **Ejecución de R desde la línea de comando**

Fuente: toma de pantalla línea de comando MS Windows

2.3.2.2. Paquete *LaTex*

Para crear un paquete, es necesario utilizar el editor de documentación *LaTex*, ya que se necesita de la documentación con alta definición tipográfica y que permita la edición de expresiones matemáticas. No obstante no es el único editor, pero permite estandarizar la presentación de la documentación de los paquetes. El sitio para descargar el editor *LaTex* es http://www.miktex.org, dar clic en “Download” para descargar “Basic MiKTeX” (figura 53).

2.3.2.3. Plataforma Microsoft HTML Help Workshop

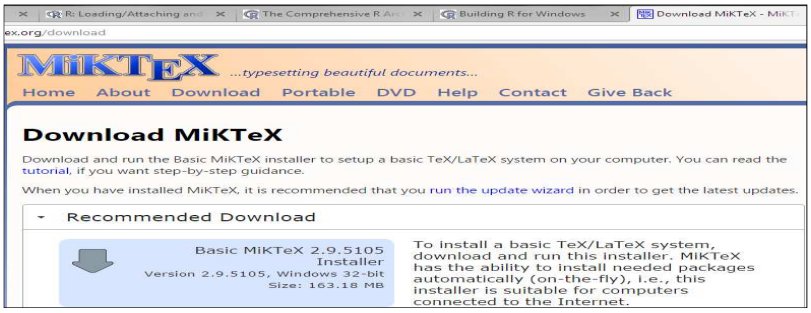
Permite compilar los archivos de ayuda para para el paquete. El sitio para

su descarga es:

http://www.microsoft.com/en-us/download/details.aspx?id=21138

Aunque suele estar ya instalado en Windows, no está por demás comprobar su instalación. Los archivos a bajar deben de incluir necesariamente el *htmlhelp.exe*. Asimismo se necesita de *R Studio*.

Figura 53. **Sitio para descarga de editor *LaTex***



Fuente: *Miktex.* http://www.miktex.org. Consulta: abril de 2016.

2.3.2. Creación de la estructura de archivos y descripción del paquete

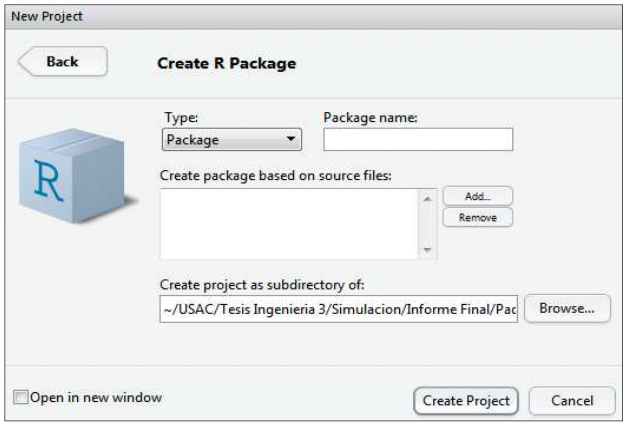
Previo a la creación de un paquete nuevo es necesario limpiar todos los objetos en memoria, por ello se escribe la siguiente instrucción:

```
>rm(list=ls())
```

Luego, mediante el uso de *R Studio*, es más versátil la creación de proyectos de nuevos paquetes. La secuencia inicia entrando el menú “File” opción “New Project”; se selecciona la opción de “New Directory”. Luego dar clic

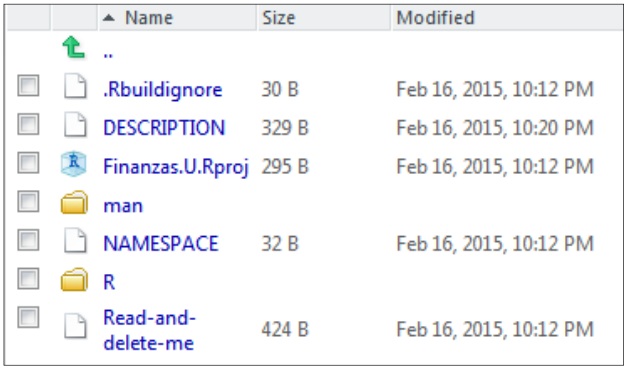
a la opción de “R Package”.Luego hay que nombrar el nuevo paquete en la casilla “Package name” y dar clic a “Create Project”. Esta secuencia de pasos crea la carpeta del nuevo paquete en el directorio de trabajo (el cual se define previamente o se puede definir en “Browse”), en cuyo contenido se crean por defecto las subcarpetas que contienen los archivos de definición.

Figura 54. **Nuevo paquete en *R Studio***



Fuente: toma de pantalla de *R Studio* 0.99.902.

Figura 55. **Carpetas de nuevo paquete**



Fuente: toma de pantalla de *R Studio* 0.99.902.

El contenido de cada carpeta se detalla a continuación:

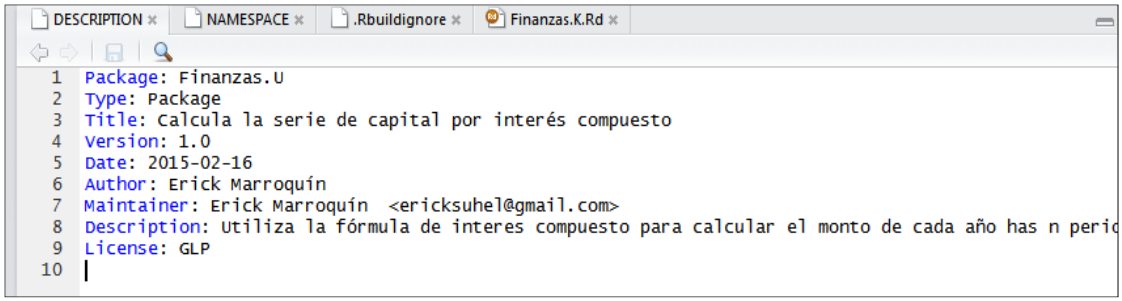
Tabla XVII. **Carpetas de trabajo de nuevo paquete**

Carpeta	**Descripción**
DESCRIPT ION	Describe las características principales del paquete
NAMESPACE	Carga los paquetes necesarios y exporta las funciones de los mismos
R	Funciones del nuevo paquete (Archivos .R)
man	Documentación o ayuda del paquete
src	Código en un lenguaje bajo nivel
data	Bases de datos optativas para ejemplos demostrativos del paquete
Read-and-delete-me	Indicaciones necesarias para la creación del paquete

Fuente: elaboración propia.

Una vez que existen los archivos y carpetas es necesario editar el archivo *DESCRIPTION*, aprovechando la interface gráfica, solo con un clic, tal y como aparece en la figura 56.

Figura 56. **Edición archivo descripción del paquete**



Fuente: toma de pantalla de *R Studio* 0.99.902.

Lo más importante de un paquete son las funciones que define para ser utilizadas, estas a su vez provienen de archivos tipo ".R", los cuales son creados la mayoría de las veces en R, mediante la secuencia:

- Menú *File* (archivo)
  - New *script* (Nuevo *Script*)
    - Escribir el *script* con la o el conjunto de funciones para el paquete nuevo
      - ✓ <Guardar los cambios>

Por defecto, el *script* es almacenado en la carpeta R del proyecto creado. El siguiente paso consiste en llenar la documentación del nuevo paquete.

2.3.3. Documentación de las funciones

Este paso es el más complejo, puesto que requiere mayor detalle, que implica construir bajo una estructura estándar tal y como la que obtiene invocando la ayuda en R. La documentación está en la estructura de archivos del paquete en la capeta "man", un archivo de extensión Rd para cada función

y datos adjuntos y un nombrado [nombre paquete]-*package*.Rd para todo el paquete. La documentación se crea en un formato compatible con Tex/LaTex y cada documentación de cada función contiene al menos los siguientes campos:

Tabla XVIII. **Estructura de la documentación de funciones**

Campo	**Descripción**
name	Nombre de la función utilizada en la ayuda
alias	Nombre de la función utilizada para otras funciones relacionadas
title	Nombre corto de lo que hace la función
description	Descripción a mayor detalle de los datos de entrada y el resultado deseado de la función
usage	Es la sintaxis de la función y sus argumentos
Arguments items	Es la definición de cada argumento
details	Alguna aclaración necesaria para el funcionamiento correcto de la función
author	Autor de la función y/o el paquete
note	Notas alternativas y opcionales
References	Referencias documentales o bibliográficas
seealso	Referencias sobre otras funciones u otros paquetes
examples	Ejemplo(s) del uso y especificación de la función las cuales pueden ser copiados y ejecutados en la consola de R

Fuente: elaboración propia.

En el transcurso de editar la documentación, tanto para cada función o

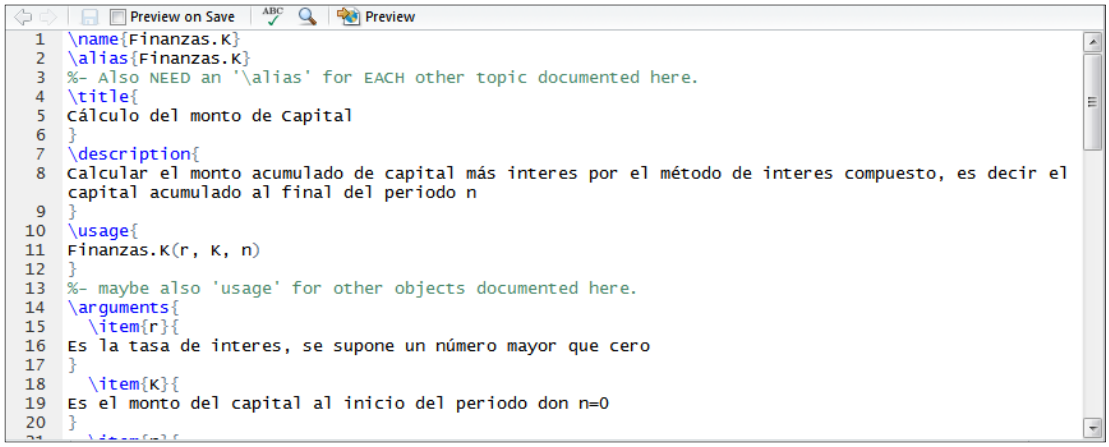
datos, es necesario chequear constantemente su validez por pasos por medio del Menú *Build*, luego la opción *Check Package*; la ventana de *Build* muestra los posibles errores del paquete vinculados a errores de falta de depuración o *debug*; la inclusión de texto no correspondiente al código ASCII.

El formato de Tex/LaTex no permite la inclusión de caracteres fuera del formato ASCII, por lo que se recomienda evitar las tildes y los caracteres tales como: ñ,$,#,_,<,> o |. Otras consideraciones para evitar errores de compilación se describen a continuación:

- Evitar los espacios entre un nombre y llave que abre "{" y la llave que cierra "}".
- La cadena de texto "\cr" se usa para insertar una nueva línea al final.
- La cadena de texto "\tab" inserta un tabulador a continuación.
- Llenar los campos de forma incremental para ir depurando los errores de forma más inmediata.
- La ausencia de llenar determinados campos no es en sí una fuente de advertencia o error, únicamente es señalado como nota.

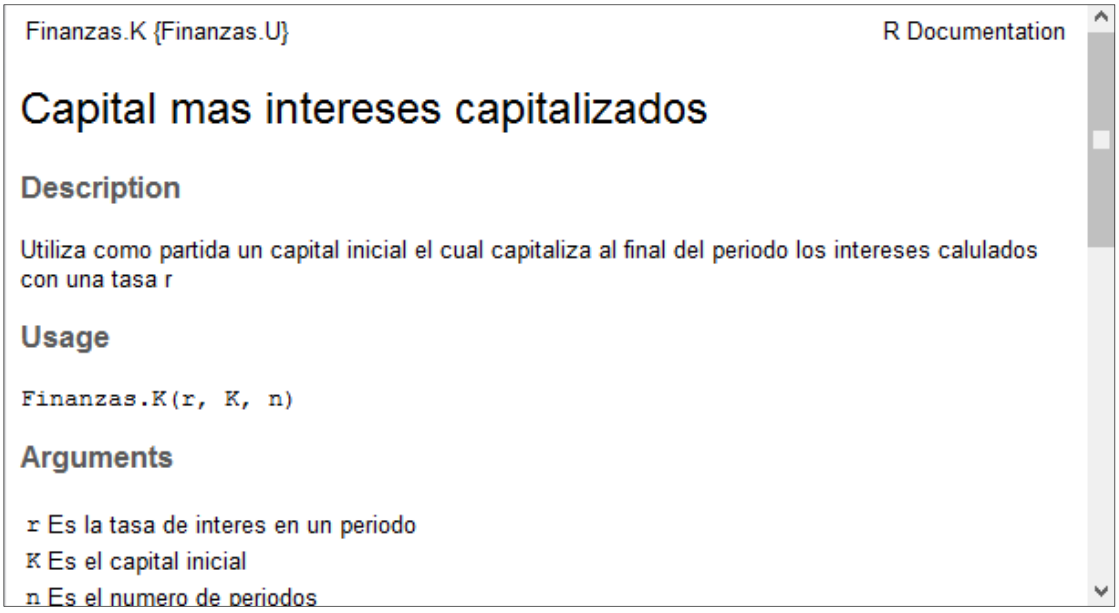
Una vez depurada la documentación se deben guardar los cambios. En *R Studio* existe un ícono de guardar del lado superior izquierdo, luego a la izquierda hay otro ícono para visualizar el borrador de la documentación en las tablas siguientes:

Figura 57. **Edición de la documentación**



Fuente: toma de pantalla de *R Studio* 0.99.902.

Figura 58. **Vista preliminar de documentación**



Fuente: toma de pantalla de *R Studio* 0.99.902.

2.3.4. Generación de manual del usuario

El resultado final será un archivo en formato HTML, conteniendo el nombre de la función, alias, título, descripción, uso, descripción de todos los argumentos de la función, detalles, valores, referencias, notas, "Ver además" y ejemplos.

Esta actividad con *R Studio* es por demás automática, ya que una vez corregidos todos los errores posibles mediante *Check*, se accede a la opción *Built & Reload*, que construye de nuevo todos los archivos depurados y carga el paquete de nuevo a memoria, equivalente a library(paquete). En el ejemplo utilizado se aplica la función para el cálculo de monto de capital con la capitalización de intereses. Los argumentos empleados en el ejemplo fueron de una tasa de interés del 25%, con un capital inicial de Q4 000,00 a cinco años; el resultado es el siguiente:

```
> Finanzas.K(.25,4000,5)
[1]  5000.000  6250.000  7812.500  9765.625 12207.031
```

Con la ayuda de *R Studio*, la creación de los manuales se hizo con base en la documentación para cada función y uno en general para el paquete completo, es decir, se parte de los archivos .Rd.

El *vignette* de cada función y del paquete completo consiste en un archivo en formato PDF que crea el proceso de construcción final del paquete fuente o Source Package. De la misma forma en el mismo menú de *Build* o construir, se genera el paquete binario o *Binary Package*. La mayoría de paquetes instalados en R, tienen *vignettes* en formato PDF, los cuales son optativos, ya que el mismo proceso de creación del paquete binario crea la documentación en

formato HTML, que a su vez es accedida por medio de la ayuda de R.

2.3.5. Creación final y distribución del paquete

La creación de un paquete con *R Studio*, resulta mucho más práctica para los efectos de este texto, por lo tanto una vez construido o compilado como se le conoce en el mundo de la programación, este puede ser probado en la versión binaria instalándolo desde R con la opción:

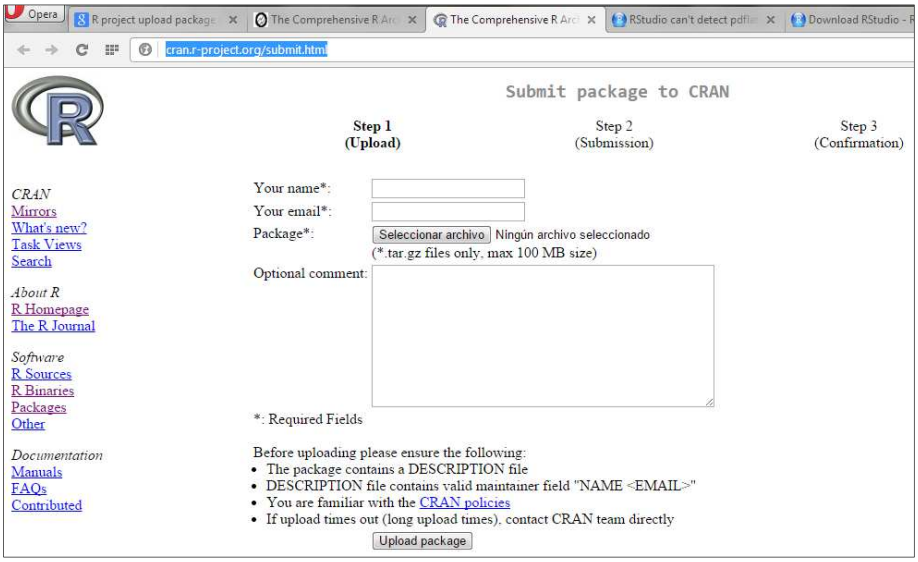
- Packages
  - Install package(s) from local zip files..

```
> utils:::menuInstallLocal()
package ‘Finanzas.U’ successfully unpacked and MD5 sums checked
> # Comprobación de que el paquete Finanzas.U funciona
> Finanzas.K(.34,1000,3) # 34% de interés, Q.1000 de capital inicial
> # y tres años para el cálculo.
[1] 1340.000 1795.600 2406.104
```

El último paso descrito en la consola de R, es una comprobación de que el paquete funciona. Otra comprobación es ver los archivos de ayuda; si lo anterior funciona ya es posible cargar el paquete para su distribución. La dirección que se utiliza es https://*CRAN*.r-project.org/submit.html.

En el sitio, subir el paquete, subir el archivo *.tar.gz generado mediante la opción *Build Source Package*; una vez, cargado el archivo, ya está disponible en el *CRAN* y los motores de búsqueda están la capacidad de localizarlo gracias a la documentación del mismo.

Figura 59. **Subir paquetes al *CRAN***



Fuente: CRAN. https://*CRAN*.r-project.org/submit.html. Consulta: febrero de 2016.

3. APLICACIÓN AL CONTROL ESTADÍSTICO DE LA CALIDAD

3.1. Introducción al control estadístico de calidad

Es importante dar una perspectiva del concepto de calidad para comprender el desarrollo de este capítulo así como el conocimiento general de todo el instrumental técnico del texto, puesto que, al mencionar el concepto de calidad, es en efecto más amplio que el simple tema del control estadístico de la calidad, que viene siendo la expresión cuantitativa de las variaciones de cualquier proceso.

Pero entonces, ¿Que es calidad?, para citar una definición breve, “la calidad es la satisfacción del cliente”[14].

En la misma referencia, se opta por complementar la definición del término “calidad” con las implicaciones con el entorno del “cliente”, ya que hay clientes internos y externos, por otro lado, la satisfacción del cliente viene dada por las “características del producto” y la “falta de deficiencias”, a su vez, esta última condición se refiere a la calidad de conformancia.

Las implicaciones de todo lo anterior, está vinculado con la función de calidad, tan similar a una espiral de crecimiento integro de toda empresa y su relación con los clientes internos y externos es decir con el mercado mismo, con muchas actividades para lograr los objetivos de calidad, que dan como resultado mayor rentabilidad y retroalimentación continua.

[14]JURAN, Joseph. *Análisis y planeación de la calidad*. p. 3.

De ello resulta que en el proceso del cumplimiento de los objetivos de calidad, se despliega en consecuencia la función de calidad y el proceso para administrar dichos procesos es necesario la planeación de la calidad, el control de calidad y mejoramiento continuo de la calidad.

En tal sentido, el aporte de este texto interviene en forma técnica en los aspectos técnicos y pedagógicos del control de calidad.

No obstante, sigue siendo aún un concepto muy amplio, ya que entre todas las actividades relacionadas, este texto está enfocado principalmente en el aspecto sensorial de la variabilidad, lo que es posible por medio de la inspección. La inspección a su vez, es entendida como el proceso de verificación del cumplimiento de las especificaciones establecidas.

Dentro de la actividad de inspección, se determinan las características de calidad, es decir, las variables que se pueden medir, sean estas cuantitativas o cualitativas. En el mismo proceso, se incluye el estudio de materias primas, el análisis de causa y efecto de los productos defectuosos y como resultado, crear nuevos estándares y especificaciones.

Entre los métodos de inspección, se encuentran las planillas de inspección, las distribuciones de frecuencias, las gráficas de control, los diagramas de dispersión y los muestreos de aceptación. En otras referencias, se incluyen siete herramientas en el control estadístico de calidad o C.E.C., las cuales son:

- Histograma
- Lista de comprobación
- Diagrama de Pareto

- Diagrama de causa y efecto o diagrama de Ishikawa
- Diagrama de concentración de defectos
- Diagrama de puntos
- Gráficas de control

En los capítulos anteriores, se han mostrado referencias generales y prácticas de varias de las herramientas descritas. En el presente capitulo se enfoca sobre el desarrollo de las gráficas de control y temas con vínculos estrechos al CEC.

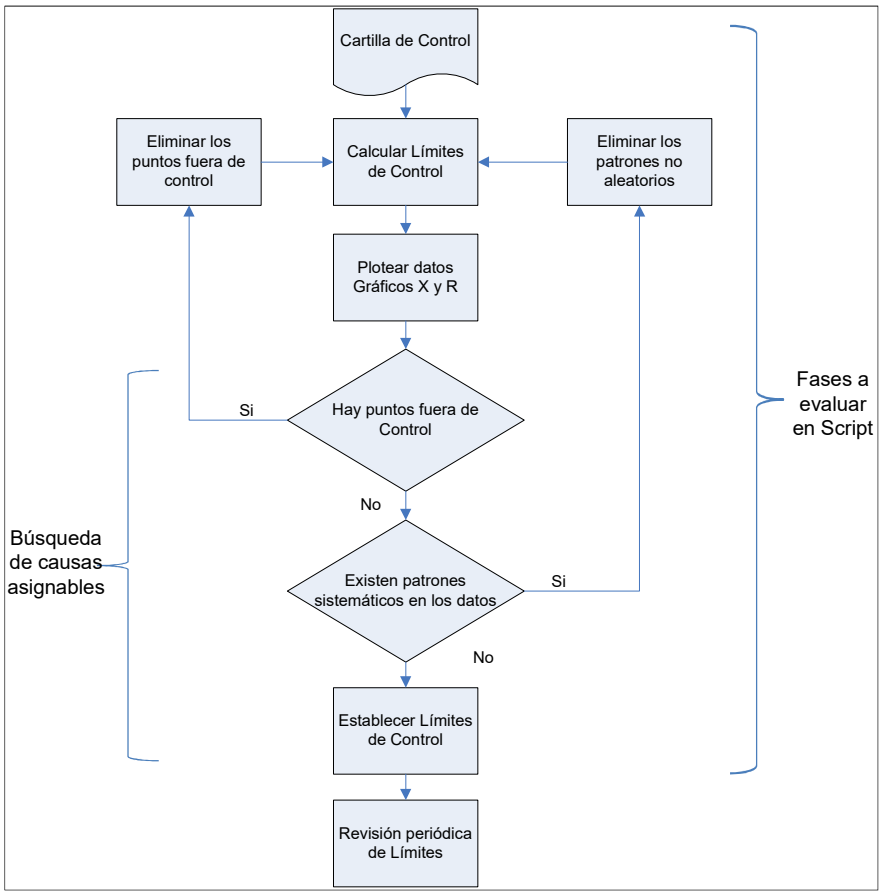
3.2. Script para cartas de control estadístico de calidad, paquete XRSCC

Lo que el paquete XRSCC pretende hacer es simular el mismo proceso por el cual se calcula todos los parámetros necesarios para analizar el nivel de variabilidad del proceso en función de variables cuantitativas y cualitativas.

En detalle, se pretende que en el caso de la gráfica $\bar{X}$, a partir de información ya existente y tabulada en archivos externos, la función lea dicha información, la traslade a la clase de objeto necesario para ejecutar la función y emplear secuencialmente todas las funciones relacionadas *xrs_gr*( ), X_it( ), R_it( ) y we_rules( ), sin embargo, en el paquete diseñado se ha adjuntado información de ejemplo.

El procedimiento a seguir se resume en el siguiente esquema:

Figura 60. **Proceso de aplicación de los gráficos de control**



Fuente: elaboración propia, empleando el programa Visio.

No obstante, interviene en tal proceso calcular la capacidad del proceso, en la medida de lo posible, detectar los corrimientos en las cartas de control, lo que está relacionado con el análisis del riesgo a cometer el error tipo I o el error tipo II y el tema de la curva de operación.

A propósito del error tipo I, es el riesgo de rechazar un proceso cuando en realidad está bajo control; mientras que el error tipo II, es el riesgo de aceptar un proceso cuando en realidad está fuera de control.

La figura 60 se ajusta a lo que recomiendan los expertos, ya que la fase I del proceso es para aplicar las gráficas de control y determinar los límites de control, mientras que a fase II es básicamente la acción de monitorear[15].

3.2.1. Gráfica Shewhart $\bar{X}$, R y S, para control de variables

El inicio metodológico de los siguientes párrafos se fundamenta en varios supuestos o parámetros prácticos a saber:

- No hay restricción alguna sobre el tamaño *m* de la muestra, no obstante una muestra grande se asemeja a una inspección total, lo que representa un costo elevado de la calidad

- Aunque es posible trabajar con grupos de tamaño *n* mayores a 10, pero por motivos prácticos, se trabaja sobre grupos de muestras entre el rango de $2 \leq n \leq 10$

- El formato de almacenamiento de información de muestras más habitual para la mayoría de usuarios es en hojas electrónicas.

- Se supone la existencia de un riesgo para cometer el Error Tipo II en el proceso de la selección de los límites de control.

- Los límites de especificación son dados y sirven de argumentos.

- No se conoce la verdadera desviación estándar, "σ", únicamente se emplea el estimador insesgado "$\hat{\sigma}$".

[15] MONTOGOMERY, Douglas. *Introduction to statistical quality control*. p. 168.

- Hay un factor *k* de corrimiento de la media de μ_0 a μ_1y un riesgo β de no detectar el corrimiento entre cada iteración.

- La metodología no contempla el estudio de las causas asignables, pero sí la detección de los patrones sistemáticos, que se asumen son síntomas cuantitativos de causas asignables. Las causas asignables principales son: inapropiado ajuste y calibración de máquinas, error del operario y defectos en el material.

Los requisitos que se deben considerar para la utilización del paquete final, por ejemplo: se tiene una muestra de 100 grupos, con cinco muestras cada uno (m = 100, n = 5). Se trata de una bebida envasada, en cuya etiqueta del envase se declara que tiene un contenido neto de 500 ml (objeto **vol_sample**)

De acuerdo con la norma COGUANOR 34 039 de “Etiquetado de productos alimenticios para consumo humano”, que manifiesta las “tolerancias para los productos, en forma líquida, que se comercialicen en volumen”, según la sección 6.1.6, cuadro 1, los volúmenes declarados de 251 cm^3 hasta 1000 cm^3, tienen según la norma una tolerancia del ±2% del volumen declarado.

Las normas COGUANOR utilizan el Sistema Internacional de Unidades o SI, por lo que un cm^3 es equivalente a un mililitro o ml. En tal sentido, se presume que por norma, el contenido líquido de las bebidas envasadas no debe exceder los límites entre 490 y 510 ml; no obstante, dichos límites no pueden considerarse como límites de especificación; sin embargo suelen tomarse como parámetros de comparación. Acá se considera un hipotético intervalo entre 494 y 506 ml como límites de especificación.

3.2.1.1. Inicio y código de la función xrs_gr para control de variables

La función está diseñada para el inicio del proceso de determinar los límites de control y crea una lista con toda la información original y la procesada, la cual puede almacenarse en otro objeto que tendría “lista” como clase.

- Cargar el paquete: en efecto, el paquete debe de estar previamente instalado, ya sea en línea a través de un *CRAN* o de la instalación manual a partir de un archivo zip local.

```
> library(XRSCC)
```

- Definición de función y argumentos:la función tiene como único argumento un objeto que contenga m filas y n>1 columnas, sin importar el nombre de las variables, que se espera sean elementos del mismo tipo, registros de la misma variable. Para ejemplo se toman los datos adjuntos al paquete, **vol_sample** o desde una hoja electrónica. En MS Windows se utiliza el argumento “clipboard”, en Linux se usa *Glipper* para la utilización de los datos cargados en memoria.

```
> data(vol_sample)
># o desde una hoja electrónica
>x<-read.table("clipboard", header = TRUE)
```

La función valida si el argumento está definido, de lo contrario su ejecución se detiene.

```
xrs_gr<-function(X){
  # Validar la existencia del objeto con las muestras
  if (missing(X)){
    stop("No hay muestras para leer, No sample to read")
  }
```

Validados los argumentos, procede a calcular internamente las dimensiones del marco de datos (m x n), seguido del promedio **X.prom**, el rango **X.range** y la desviación estándar **X.s** por fila.

```
else {
    x<-X
    m<-nrow(x)
    n<-ncol(x)
    X.prom<-apply(x,1,mean)
f.rango<-function(x){
      f.rango.p<-range(x)
      return(f.rango.p[2]-f.rango.p[1])
    }
    X.range<-apply(x,1,f.rango)
    X.S<-apply(x,1,sd)
```

Para el cálculo de los límites de control de la carta $\bar{X}$, R y S se necesita de las tablas tal y como están en el anexo 1, los datos están alojados en el objeto **factor.a** que está asociado con el paquete que se carga internamente en la ejecución de la función. Los límites están identificados como LCS para el límite superior, LCI para el límite inferior y LC para el límite central o promedio; cada límite viene seguido del tipo de gráfica a que pertenece, por ejemplo LCS.X, es el límite de control superior de la carta $\bar{X}$.

Notar que se utilizan expresiones (*expression*) para ser evaluados

posteriormente, ya que la misma expresión puede cambiar de valor al cambiar por ejemplo el número de filas, ya que esto afecta el promedio.

```
data(factor.a)
#   Limites de control grafica X
LCS.X<-expression(mean(X.prom) + factor.a$A2[n-1] * mean(X.range))
LCI.X<-expression(mean(X.prom) - factor.a$A2[n-1] * mean(X.range))
LC.X<-expression(mean(X.prom))
#   Limites de control grafica R
LCS.R<-expression(mean(X.range)*factor.a$D4[n-1])
LCI.R<-expression(mean(X.range)*factor.a$D3[n-1])
LC.R<-expression(mean(X.range))
#   Limites de control grafica S
LCS.S<-expression(mean(X.S)*factor.a$B4[n-1])
LCI.S<-expression(mean(X.S)*factor.a$B3[n-1])
LC.S<-expression(mean(X.S))
```

El valor de los factores utilizados está en función de la tabla, cuyos valores van desde 2 hasta 25; por ello, la indexación de los datos va al valor n-1 y la columna o factor A2, D4, D3, B4 o B3.

- Diseño de gráficas de control: esta parte de la función, está apoyada en la programación de funciones internas dentro del ambiente general de la función *xrs_gr*( ), por ejemplo: La función *plot.X*() que evalúa expresiones en función de del tamaño de la muestra y utiliza los argumentos y atributos de la función interna *plot*( ), de la cual además, se utilizan las opciones para presentar gráficas de bajo nivel como “abline” y “text”. Previo se define el marco sobre el cual se ordenan las gráficas a mostrar.

```
# Marco de Graficas
mat<-matrix(1:4,2,2,byrow=TRUE)
layout(mat)
layout.show(length(1:4))
# Mostrar un histograma
hist.X<-function(x=Xprom,breaks="Sturges"){
    hist(X.prom, breaks = breaks,
xlab="Valores", ylab="Frecuencia",
        main="Histograma de los promedios")
}
# Script  para Gráfica X
plot.X<-function(x=X.prom,type="b",col="blue",pch =19){
plot(x=x, xlab= "Numero de muestra", ylab="Valores de cada muestra",
        main="Grafica X, Control Estadistico de la Calidad",type=type,
col=col,
ylim=c(min(eval(LCI.X), min(X.prom)), max(eval(LCS.X), max(X.prom))),
        xlim=c(-0.05*m, 1.05*m), pch = pch)
    abline(h= c(eval(LCS.X), eval(LCI.X), eval(LC.X)),col="lightgray")
    text(c(rep(1,3),rep(7,3)), rep(c(eval(LCS.X),eval(LC.X),eval(LCI.X)),2),
        c(c("LCS = ","LC = ","LCI = "), c(round(eval(LCS.X),2),
                                round(eval(LC.X),2),
                                round(eval(LCI.X),2))), col="red")
}
```

En el anterior párrafo se definió la función que diagrama el histograma y la carta $\bar{X}$; la misma lógica es seguida por las funciones para la carta R y S.

- Cálculo de la información de salida: esta empieza desde el cálculo de los límites de control; ahora la función compara cada valor de **X.prom** con los límites y asigna los valores que permanecen dentro de los límites, calcula un subconjunto de los datos originales **x.1,** e identifica el número

de fila de los que están fuera **X.f.** Lo mismo para los valores de carta R.

```
X.pos <- which(X.prom > eval(LCI.X) & X.prom < eval(LCS.X))
x.1<-x[X.pos,]
X.fs<- which(X.prom >= eval(LCS.X))
X.fi<- which(X.prom <= eval(LCI.X))
X.f<-c(X.fs,X.fi)
R.pos <- which(X.range > eval(LCI.R) & X.range < eval(LCS.R))
X.range.1<-X.range[R.pos]
S.pos <- which(X.S > eval(LCI.S) & X.S < eval(LCS.S))
```

Por otro lado, define en forma binaria los resultados de la comparación entre el número de filas original y el número de filas que están bajo control; si esta última es menor que m, el resultado es “1”, de lo contrario es “0”; en forma análoga los resultados para la carta R y S.

```
bin.X<-if(length(X.pos)< m){
    bin.X<-1
  } else {
    bin.X<-0
  }
  bin.R<-if(length(R.pos)< m){
    bin.R<-1
  } else {
    bin.R<-0
  }
  bin.S<-if(length(S.pos)< m){
    bin.S<-1
  } else {
    bin.S<-0
}
```

El toda la información calculada y almacenada dentro del ambiente de la función se lleva a una estructura de modo lista; cada elemento de dicha lista es un resultado o una forma de expresarse. Previo a eso se llama a ejecutar las funciones gráficas.

```
hist.X()
plot.X()
plot.R()
plot.S()
# Datos de salida
structure(list("in.control" = X.pos,
"R.in.control" = R.pos,
            "out.control" = X.f,
            "Iteraciones" = 1,
"data.0"= x,
            "data.1"= x.1,
            "data.r.1" = X.range.1,
            "bin" = c(bin.X, bin.R, bin.S),
            "LX"= c("LCI"=eval(LCI.X),  "LC"=eval(LC.X),"LCS"=eval(LCS.X)),
"LR"= c("LCI"=eval(LCI.R), "LC"=eval(LC.R), "LCS"=eval(LCS.R)),
            "LS"= c("LCI"=eval(LCI.S), "LC"=eval(LC.S), "LCS"=eval(LCS.S)),
            "Limites Grafica X" = c("LCI.X"=eval(LCI.X),
"LC.X"=eval(LC.X),"LCS.X"=eval(LCS.X)),
            "Limites Grafica R" = c("LCI.R"=eval(LCI.R), "LC.R"=eval(LC.R),
"LCS.R"=eval(LCS.R)),
            "Limites Grafica S" = c("LCI.S"=eval(LCI.S), "LC.S"=eval(LC.S),
"LCS.S"=eval(LCS.S)),
            "Conclusion del proceso"= c(if(length(X.pos)< m){
              print("Proceso fuera de Control en Grafica X")
            } else {
              print("El proceso esta bajo control en Grafica X")
```

```
}, if(length(R.pos)< m){
print("Proceso fuera de control en Grafica R")
                } else {
                  print("El proceso esta bajo control en Grafica R")
}, if(length(S.pos)< m){
print("Proceso fuera de control en Grafica S")
                } else {
                  print("El proceso esta bajo control en Grafica S")
                  }))
}
```

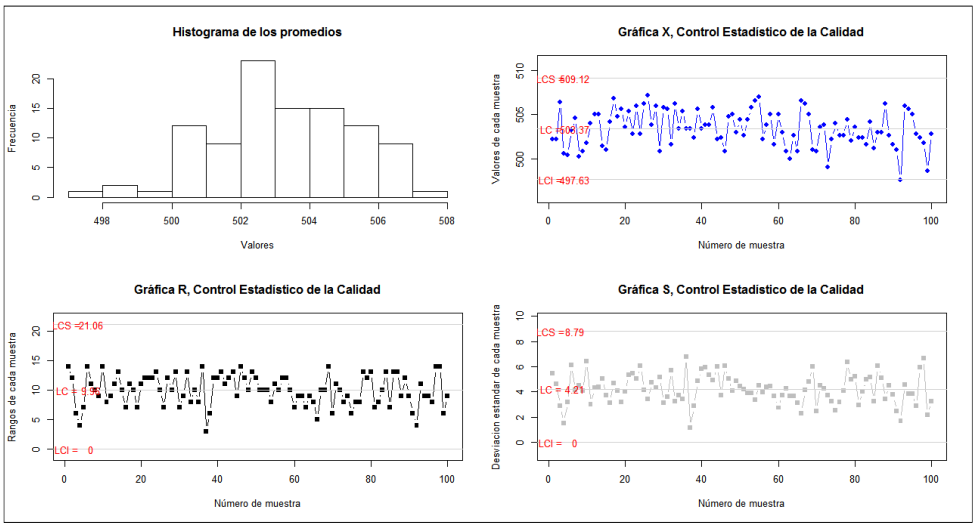
La ultima llave del anterior párrafo es la que cierra el código de la función *xrs_gr*( ).

- Ejecución de la función y almacenaje de resultados: como estaba previsto, al ejecutar la función en la consola principal de R, utilizando los datos de ejemplo adjuntos, los resultados son los siguientes:

```
> resultados1<-xrs_gr(vol_sample)
[1] "Proceso fuera de Control en Grafica X"
[1] "El proceso esta bajo control en Grafica R"
[1] "El proceso esta bajo control en Grafica S"
```

La figura 61 es presentada en pantalla en forma automática al mismo tiempo que los resultados sobre el “control” de en las cartas $\bar{X}$, R Y S.El ejemplo dice que en el objeto **resultados1**menciona que la carta $\bar{X}$ está fuera de control, porque al menos un promedio de fila **X.prom,** está fuera de los límites de control.

Figura 61. **Gráficas de control por variables**



Fuente: elaboración propia, empleando paquete XRSCC.

Para extraer más información de los resultados, se escribe directamente el nombre del objeto de resultados (tipo lista) en la consola principal y mostrará todos los resultados de salida especificados en la estructura dentro de la función. Sin embargo para efectos prácticos, por ejemplo si se desea saber cuáles son los límites de control de la carta $\bar{X}$, se escribe el nombre del objeto seguido del símbolo "$" y el vector que contiene dicha información.

```
> resultados1$LX
    LCI        LC        LCS
497.6271  503.3740  509.1209
```

Notar que los valores de los límites de control de la carta $\bar{X}$ son los de prueba, ya que el proceso está fuera de control. En tal sentido, el siguiente paso es utilizar la función *X_it*( ), la cual utiliza esa información de salida (no la original), ya que en la lista **resultados1** lleva implícito los valores que están

fuera de control, para eliminarlos en la siguiente iteración y calcular los nuevos límites de control. Lógicamente los nombres de los vectores no son fáciles de deducirlos; por ello, se recomienda listar los nombres de la siguiente forma:

```
> names(resultados1)
 [1] "in.control"          "R.in.control"        "out.control"
 [4] "Iteraciones"         "data.0"              "data.1"
 [7] "data.r.1"            "bin"                 "LX"
[10] "LR"                  "LS"                  "Limites Grafica X"
[13] "Limites Grafica R"   "Limites Grafica S"   "Conclusion del proceso"
```

Por lo tanto, se pueden extraer los valores que están fuera de control, únicamente como referencia, utilizando la vectorización a partir de los resultados, lo que indica que la fila 92, en promedio está fuera de los límites de control, ya que el promedio es 497,6, justo por debajo del límite inferior (497,62).

```
> vol_sample[resultados1$out.control,]
   n1    n2    n3    n4    n5
92    500   496   496   498   498
```

3.2.1.2. Iteración y código de la función X_it para control de variables

El objeto **resultados1** contiene información útil de la clase que utiliza la función *X_it*( ), cuyo propósito es depurar las líneas del marco de datos original cuyos promedios están fuera de los límites de control y calcular los nuevos límites de control; luego los resultados se ajustan al formato de la primera iteración con la función *xrs_gr*( ). El orden de la iteración viene almacenado en el vector **Iteraciones**, por ejemplo. Al ejecutar la función *X_it*( ) por primera vez,

el valor será igual a dos.

```
> resultados1$Iteraciones
[1] 1
```

A diferencia de *xrs_gr*( ) la presente función estima únicamente los resultados para la carta $\bar{X}$, aunque de forma implícita para la carta R para su posterior utilización. En las siguientes líneas de código se detallan las diferencias de la anterior función, lo demás funciona exactamente igual, sin considerar los nombres que asigna internamente a los objetos, ya que la lista de salida tiene la misma estructura y nombres.

Respecto de los argumentos, ya se definió que el único válido para la función es la lista de resultados provenientes de xrs_gr( ). De lo contrario, la ejecución se detiene. Existe la posibilidad de que se ejecute la función con un argumento con clase válida (lista), pero en estructura no tiene los nombres válidos; por ello, como siguiente paso, valida por medio del vector **bin** si el primer valor correspondiente a la carta $\bar{X}$, es igual a 1, que indica que el proceso está fuera de control, aun en la iteración anterior, si el valor es igual a cero, se detiene.

```
X_it<-function(prev.results){
# Validar la existencia del objeto con los resultados previos
if (missing(prev.results)){
   stop("No elementos para iteracion, No elements for iteration")
   } else {
     if(prev.results$bin[1]==0){
stop("El proceso ya esta bajo control, The process is already under control")
       } else {
```

El siguiente paso consiste en vectorizar el subconjunto de las filas que están bajo control estadístico. Seguido de ello, los cálculos (código) de los límites de control están definidos a la función anterior.

```
x.1<-prev.results$data.1
        m.1<-nrow(x.1)
        n.1<-ncol(x.1)
        X.prom.1<-apply(x.1,1,mean)
        f.rango.1<-function(x.1){
          f.rango.p.1<-range(x.1)
          return(f.rango.p.1[2]-f.rango.p.1[1])
}
          ……
```

Los resultados de abajo, muestran que el proceso estudiado ya está bajo proceso en la segunda iteración, tanto en la carta $\bar{X}$ como en la carta R, tal y como lo refleja lafigura 62, en la que ya no figura ningún punto fuera de los límites de control.

```
> resultados2<-X_it(resultados1)
[1] "El proceso esta bajo control en Grafica X"
[1] "El proceso esta bajo control en Grafica R"
```

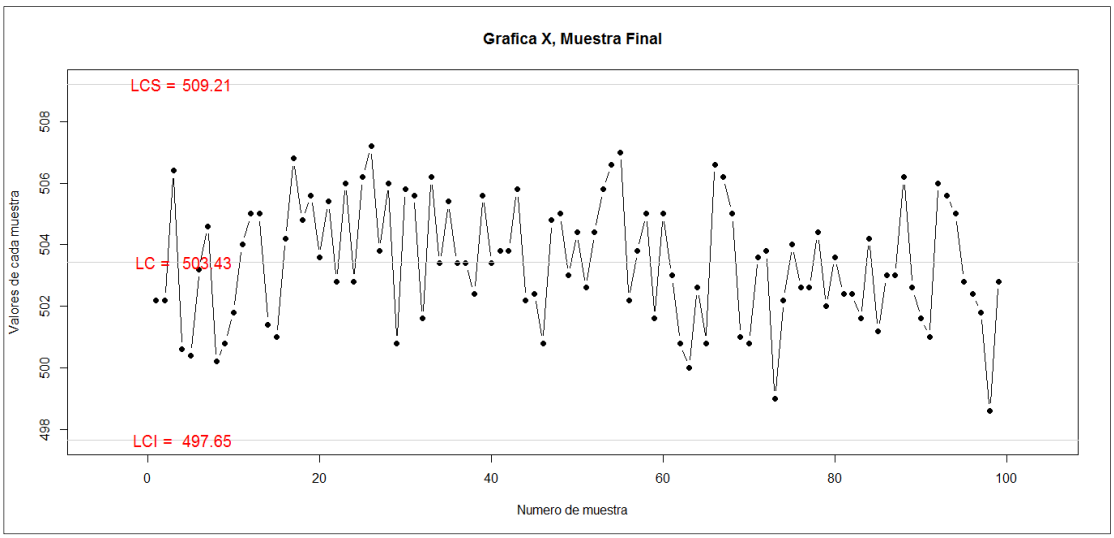
La comparar los límites de control de resutlados1 y el de **resultados2**, se puede apreciar el nuevo valor de los límites, así como el leve corrimiento de la media (límite central). Dichos límites, corresponden a los valores naturales.

```
> resultados1$LX
LCI             LC              LCS
497.6271        503.3740        509.1209
> resultados2$LX
```

LCI	LC	LCS
497.6507	503.4323	509.2140

Si después de (*i – 1*) iteraciones, el proceso está bajo control en la carta $\bar{X}$, (**bin** = 0,0,0), al intentar la iteración *i*, el proceso de ejecución se detiene, ya que la función no tiene elementos que evaluar.

Figura 62**.** **Gráfica $\bar{X}$, segunda iteración**



Fuente: elaboración propia, empleando paquete XRSCC.

3.2.1.3. Iteración y código de la función R_it para control de variables

El objetivo de practicar una iteración más, es depurar los registros que en rango están fuera de los límites de control para la carta R, ya que ambas cartas deben de estar bajo control[16]. La misma estructura se aplicó en resultados

[16] SMALL, Bonnie B. *Statistical quality control handbook*. p. 152.

previos; el mismo tipo de lista se utiliza como argumento para ejecutar la función *R_it*( ) y como es de esperar, devuelve una estructura idéntica pero con valores depurados.

Internamente, en las iteraciones previas, los resultados llevan consigo el vector **R.in.control,** el cual es el vector con los números de línea cuyos rangos están bajo control. El mismo vector se utiliza en esta función para calcular de nuevo los límites de control de la carta $\bar{X}$, como de la carta R. Si el proceso ya está bajo control (**bin** = 0,0,0) la función no se ejecuta, en cambio sí (**bin** = 0,1,0), las siguientes líneas son ejecutadas:

```
R_it<-function(prev.results){
# Validar la existencia del objeto con los resultados previos
if (missing(prev.results)){
   stop("No elementos para iteracion, No elements for iteration")
 } else {
   if(prev.results$bin[2]==0){
     stop("El proceso ya esta bajo control, The process is already under control")
   } else {
     x.0<-prev.results$data.1
     R.pos.0<-prev.results$R.in.control
x.1<-x.0[R.pos.0,]
     m.1<-nrow(x.1)
n.1<-ncol(x.1)
     X.prom.1<-apply(x.1,1,mean)
     f.rango.1<-function(x.1){
       f.rango.p.1<-range(x.1)
…….
```

El filtro de los datos en esta función cambia de **X.pos** a **R.in.control** pero afectando el mismo marco de datos. Adelante los cálculos son los mismos que

la función *X_it*( ), de tal forma que se obtiene la gráfica de la carta X y los límites de control de $\bar{X}$ y de R.

Con el anterior ejemplo, ya se conoce que el proceso está bajo control en la carta R, por ello se recurre a los datos adjuntos **qqsugar,** cuya definición está documentada en la ayuda del paquete.

```
> resultados1<-xrs_gr(qqsugar)
[1] "El proceso esta bajo control en Grafica X"
[1] "Proceso fuera de control en Grafica R"
[1] "Proceso fuera de control en Grafica S"
> resultados2<-X_it(resultados1)
Error in X_it(resultados1) :
  El proceso ya esta bajo control, The process is already under control
> resultados2<-R_it(resultados1)
[1] "El proceso esta bajo control en Grafica X"
[1] "El proceso esta bajo control en Grafica R"
```

Del anterior ejercicio, se pueden mencionar los siguientes aspectos:

- Inicialmente el proceso está bajo control en la carta $\bar{X}$, no así en la carta R y S, por lo tanto, en general el proceso sigue estando fuera de control (**resultados1**).

- En la segunda iteración (**resultados2**), el proceso ya está bajo control en la carta $\bar{X}$ y R.

- Por lo tanto los límites de control tienen un corrimiento, ver los resultados siguientes:

```
> resultados1$LX
      LCI        LC         LCS
 99.91013  100.96495  102.01977
> resultados2$LX
      LCI        LC         LCS
 99.97994  100.96989  101.95984
```

3.2.2. Gráfica p, nc, c y u para control por atributos

El siguiente conjunto de códigos están diseñados para hacer control estadístico de la calidad en función de atributos, es decir, aquellas características que no pueden ser medias por valores continuos, como el peso, longitud, volumen, entre otros; en tal sentido están diseñados para el conteo de ciertos atributos o falta de ellos. El conteo de los atributos debe figurar en efecto entre ciertos límites para que el proceso se considere como dentro o fuera de control.

3.2.2.1. Gráfica p, proporción de los “no conformes”, función p_gr

Se dice que tiene la característica de “defectuoso” o “no defectuoso” y más reciente, se utiliza en forma alternativa “no conforme” y “conforme”, respectivamente. Por definición, se acopla matemáticamente a una distribución binomial, dado que hay una proporción de los “no conformes” descrita por:

$$\bar{p} = \frac{D_i}{n}$$

Donde “*Di*”, es el número de los “no conformes”, la media está dada por $\mu = \bar{p}$ y la varianza por: $\sigma^2 = \frac{\bar{p}(1-\bar{p})}{n)}$, alternativamente $(1 - \bar{p})$ es la proporción de

los artículos “conformes”. El propósito al final es para probar lo siguiente:

- Hipótesis nula $H_0: p = p_0$
- Hipótesis alternativa $H_1: p \neq p_0$

Significa entonces que p_0 es un valor objetivo y debe de ser tan pequeño como sea posible.

Al igual que con las gráficas $\bar{X}$, R y S, no se profundiza en teoría subyacente a este tema, más que en la metodología estadística para calcular los parámetros consistentes en los límites de control:

- Límite de control superior LCS = $\bar{p} + 3\sqrt{\frac{\bar{p}(1-\bar{p})}{n}}$
- Límite de control central LC = $\bar{p}$
- Límite de control inferior LCI = $\bar{p} - 3\sqrt{\frac{\bar{p}(1-\bar{p})}{n}}$

Los datos de entrada o argumentos están conformados por dos variables necesarias, la primera consiste en el tamaño “n” de la muestra, lo suficientemente grande para detectar al menos un artículo “no conforme”; la segunda es la frecuencia de ocurrencia de los artículos “no conformes” o “D”.

Considerar un ejemplo en un proceso de sellado de un químico industrial altamente corrosivo, por lo que es necesario que cada envase de 60 ml esté herméticamente cerrado con un sello de aluminio que se adhiere a mano y se funde a la boca del envase PET (politereftalato de etileno)por medio de una máquina de sellado por inducción de calor de microondas.

No obstante, se está en una fase temprana de este nuevo método de

sellado, ya que el anterior método era rudimentario y ocasionaba que hasta un 10 % de los envases tuviera fuga; en tal sentido, se decidió cambiar de método de sellado, poniendo como meta que se redujera a la mitad la proporción de los envases “no conformes”. En la tabla XIX están los datos de muestras de tamaño 100, tomados de 80 turnos de trabajo. Se espera en tal caso que haya al menos 1% de “no conformes” y un máximo de 5%.

Se pide establecer si el proceso está bajo control e identificar las muestras que en particular salen de los límites de control; luego de ello recalcular los límites de prueba. Notar que la muestra tiene un ancho constante, dispuesto de tal forma para efectos prácticos.

En las siguientes párrafos se desarrolla la función *p_gr*( ) y *P_it*( ) con las que se establecen los límites de control para la gráfica p.

- Cargar el paquete: en efecto, el paquete debe de estar previamente instalado, ya sea en línea a través de un *CRAN* o de la instalación manual a partir de un archivo zip local.

```
> library(XRSCC)
```

- Definición de función y argumentos: la función tiene como único argumento un objeto que contenga m filas y una columna, no importando el nombre de la variable, que corresponde al número de defectuosos en una muestra **n** de tamaño fijo. El objeto donde están almacenados los datos debe de ser de clase data frame o una matriz; si se cuenta con un vector, es necesario transformarlo a cualquiera de las clases indicadas. Para el ejemplo, se toman los datos adjuntos al paquete, **vol_sample** o desde una hoja electrónica, siempre que la variable de estudio esté en la

primera columna.

Tabla XIX. **Datos ejemplo, proceso envasado**

Datos para Gráfica p, Número de envases mal sellados "no conformes", muestra "n" = 100

Muestra	Di	Muestra	Di	Muestra	Di	Muestra	Di
1	1	21	7	41	4	61	3
2	9	22	10	42	1	62	3
3	12	23	4	43	3	63	2
4	3	24	7	44	6	64	2
5	5	25	2	45	3	65	4
6	6	26	2	46	4	66	2
7	3	27	2	47	4	67	3
8	4	28	3	48	3	68	3
9	10	29	1	49	0	69	4
10	9	30	3	50	1	70	4
11	1	31	3	51	9	71	2
12	3	32	2	52	3	72	6
13	3	33	4	53	5	73	8
14	2	34	4	54	6	74	5
15	4	35	5	55	3	75	3
16	1	36	3	56	5	76	3
17	1	37	1	57	2	77	2
18	9	38	2	58	4	78	1
19	3	39	6	59	2	79	4
20	11	40	1	60	1	80	2

Fuente: elaboración propia.

```
> data(vol_bottles)
>#  o desde una hoja electrónica
>x<-read.table("clipboard", header = TRUE)
```

La función valida si el argumento está definido, de lo contrario su ejecución se detiene, lo mismo para fijar el tamaño **n** de la muestra utilizada.

```
p_gr<-function(D,n){
  if (missing(D)){
    stop("No hay muestras para leer, No sample to read")
  } else {
if (missing(n)){
     stop("Debe fijar un ancho de la muestra para calcular la proporcion")
} else {
       p.0<-D[,1]/n
       if(max(p.0) >= 1){
stop("No puede existir una proporcion mayor a la unidad")
        } else {
```

Validados los argumentos, procede a calcular internamente las proporciones en **p.0** y a validar si alguna de las proporciones es mayor a uno, puesto que se supone que no hay un número mayor de defectuosos al tamaño de la muestra.

Seguido inicia el cálculo de los límites de control con las fórmulas ya indicadas; en el caso del límite inferior, fija en cero, si el límite calculado es menor que cero. Con tales cálculos la función ya está en capacidad de interpretar que proporciones **p.0** están fuera de los límites de control, de tal forma que de haber un número de líneas de **p.0** bajo control, menor al número de filas **m** original, la función interpreta que el proceso está fuera de control y el resultado se expresa en forma binaria en el resultado **bin** de “1” para “fuera de control”, “0” para “bajo control”.

```
m <-nrow(D)
        LCS.p.0<-expression(mean(p.0)+3*sqrt((mean(p.0)*(1-mean(p.0)))/n))
        LCI.p.0<-expression(mean(p.0)-3*sqrt((mean(p.0)*(1-mean(p.0)))/n))
        LC.p.0<-expression(mean(p.0))
```

```
        if (eval(LCI.p.0)>0){
          LCI.p.0<-eval(LCI.p.0)
          } else {
            LCI.p.0 <- 0
            }
        p.pos<-which(p.0 >= eval(LCI.p.0)  & p.0 < eval(LCS.p.0))
        p.1<-p.0[p.pos]
        p.fi.0<-which(p.0 < eval(LCI.p.0))
p.fs.0<-which(p.0 >= eval(LCS.p.0)) # Solo valores positivos
bin.p<-if(length(p.pos)< m){
          bin.p<-1
          } else {
            bin.p<-0
}
```

- Definición de la gráfica p:esta parte de la función está apoyada en la programación de funciones internas dentro del ambiente general de la función *p_gr*( ), por ejemplo: la función *plot.p*( ) que evalúa expresiones de acuerdo con el tamaño de la muestra y utiliza los argumentos y atributos de la función interna *plot*( ), de la cual además, se utilizan las opciones para presentar gráficas de bajo nivel como “abline” y “text”.

```
plot.p<-function(P=p.0,type="b",col="blue",pch =19){
plot(P, xlab= "Numero de muestra",
             ylab="Proporcion de los no conformes de cada muestra",
             main="Grafica P, Control Estadistico de la Calidad",
type=type, col=col,   ylim=c(eval(LCI.p.0)-mean(p.0)*0.05,
                  max(eval(LCS.p.0)*1.1, max(p.0)*1.1)),
             xlim=c(-0.05*m, 1.05*m), pch = pch)
          abline(h= c(eval(LCS.p.0), eval(LCI.p.0),
                  eval(LC.p.0)),col="lightgray")
```

```
            text(c(rep(1,3),rep(7,3)),
                rep(c(eval(LCS.p.0),eval(LC.p.0),eval(LCI.p.0)),2),
                c(c("LCS = ","LC = ","LCI = "),  c(round(eval(LCS.p.0),3),
                   round(eval(LC.p.0),3), round(eval(LCI.p.0),3))),
col="red") }
          plot.p()
```

- Cálculo de la información de salida: la información que se ordena en la estructura es muy similar a la de salida en la función *xrs_gr*( ). Ya que deja los resultados de la primera iteración para ser utilizados en otra iteración o para la referencia inmediata, por medio de índices.

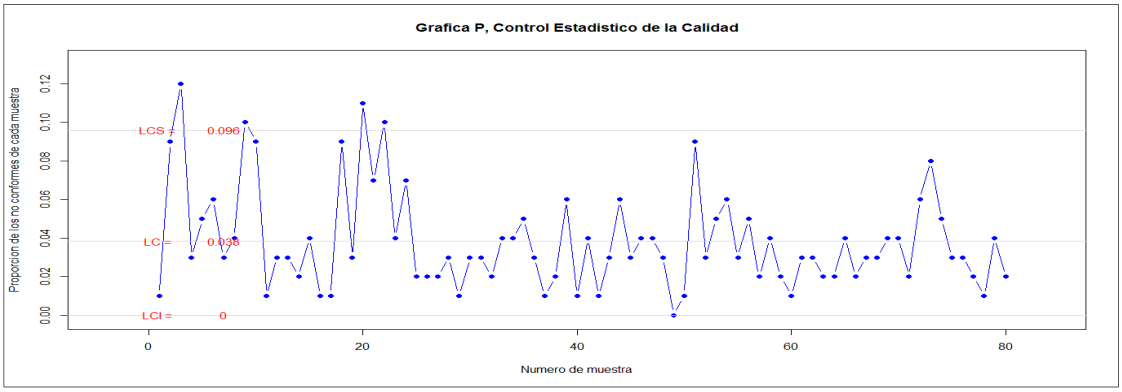
```
structure(list("in.control" = p.pos,  "out.control"= c(p.fi.0,p.fs.0),
"Iteraciones" = 1,"data.n"=n, "data.0"= D,
"data.1"= p.1, "bin" = bin.p,
                    "Limites de Control Grafica p" =
                    c("LCI.p"=eval(LCI.p.0),"LCS.p"=eval(LCS.p.0),"LC.p"=eval(LC.
                    p.0)),
                   "Conclusion del proceso"= c(if(length(p.pos)< m){
                     print("Proceso fuera de Control en Grafica p")
                     } else {
                       print("El proceso esta bajo control en Grafica p")
                       }))
        }
     }
  }
}
```

- Ejecución de la función y almacenaje de resultados: los resultados se pueden apreciar al ejecutar la función con los datos de ejemplo adjuntos ver la figura 63.

```
> data(bottles)
> r1<-p_gr(bottles, n=100)
[1] "Proceso fuera de Control en Grafica p"
```

Como ya se indicó, los puntos fuera del límite superior deben de ser filtrados para calcular los verdaderos límites de control. Analizando visualmente el comportamiento de las muestras en la gráfica P, se aprecia que hay mayor variabilidad entre las muestras 1 y 25, esto puede deberse a que corresponden a determinado turno, operario, empaque de distinto proveedor, entre otros. Por lo tanto esa variabilidad asociada a causas asignables, debe de ser eliminada para calcular los verdaderos límites de control.

Figura 63. **Proporción de los no conformes**



Fuente: elaboración propia, utilizando paquete XRSCC.

Sin la necesidad simular procedimientos más complejos, se advierte que la sola existencia de puntos más allá de $\pm 2\sigma$, implica señales fuertes de que el proceso puede no estar bajo control[17], tal y como se muestra en la figura 63.

[17] MONTOGOMERY, Douglas. *Introduction to statistical quality control*. p. 159.

Por lo tanto, el uso de límites de advertencia incrementa la sensibilidad de la gráfica de control, ya que sugiere un posible corrimiento de los límites de control.

Por último, es necesario recordar que el riesgo de no detectar el corrimiento de la muestra, es inversamente proporcional al tamaño de la misma y esta a su vez es el resultado del diseño previo del experimento para la recolección de la muestra; en tal sentido, puede que la muestra varíe de tamaño de acuerdo con el lote de producción.

- Ejecución de la función de iteración *P_it*( ):los resultados almacenados en la lista **r1** señalan claramente que el proceso se encuentra fuera de control, en tal sentido es necesario establecer los verdaderos límites de control con la función de iteración *P_it*( ). La función comprueba si los resultados anteriores, utilizados como argumento en la presente, están o no bajo control; de estarlo, el proceso se detiene; de lo contrario, tiene por objeto eliminar las filas cuyas proporciones calculadas **p.0,** están fuera de los límites de control; luego el procedimiento es exactamente igual a la función *p_gr*( ), sumando un dígito al número de iteración corrida y estableciendo el valor binario en **bin,** para confirmar si el proceso está fuera de control o bajo control.

```
P_it<-function(prev.results){
  if (missing(prev.results)){
    stop("No elementos para iteracion, No elements for iteration")
} else {
  if(prev.results$bin[1]==0){
    stop("El proceso ya esta bajo control, The process is already under control")
  } else {
    p.0<-prev.results$data.1
```

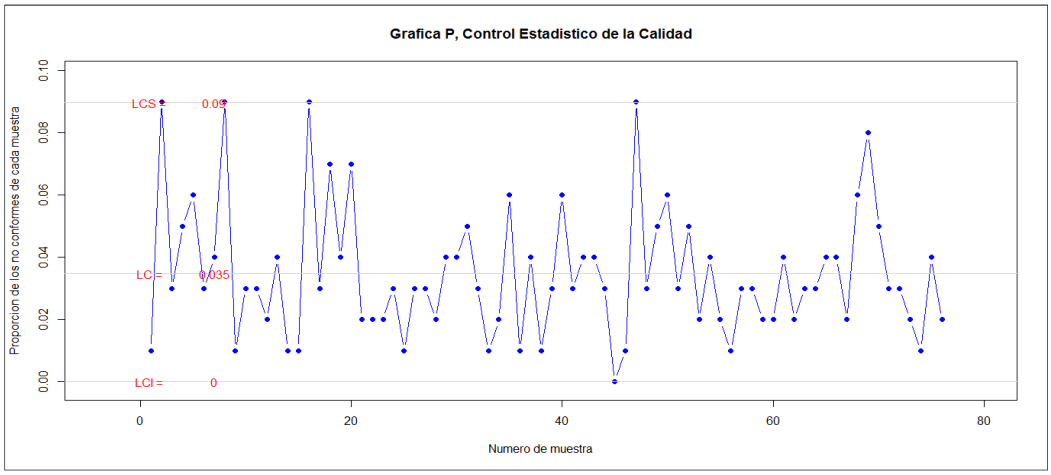
```
    m <-length(p.0)
n <-prev.results$data.n
```

Al ejecutar la función *P_it*( ), la figura 64 muestra claramente una reducción en el rango de las proporciones; no obstante, el proceso permanece fuera de control, por lo que se ejecuta una nueva iteración. Por último, se comparan los límites de control de cada iteración.

```
> r2<-P_it(r1)
[1] "Proceso fuera de Control en Grafica p"
> r3<-P_it(r2)
[1] "El proceso esta bajo control en Grafica p"
> r1$"Limites de Control Grafica p"
    LCl.p          LCS.p          LC.p
0.00000000     0.09600496    0.03837500
> r2$"Limites de Control Grafica p"
    LCl.p          LCS.p          LC.p
0.00000000     0.08967060    0.03473684
> r3$"Limites de Control Grafica p"
    LCl.p          LCS.p          LC.p
0.00000000     0.08500247     0.03211268
```

Observar el límite de control superior LCS.p en la tercera iteración en **r3**, que tiene un valor en porcentaje del 8,5 % de defectuosos, mucho mayor al 5 % máximo que se delimitó en el enunciado.

Figura 64. **Gráfica *p*, segunda iteración**



Fuente: elaboración propia, utilizando paquete XRSCC.

3.2.2.2. Gráfica np, número de los “no conformes”, función np_gr

Muy similar a la gráfica “p”, la de control de atributos “np”, identifica el número de los “no conformes” sí y solo sí todas las muestras son del mismo tamaño; no obstante, también es posible el análisis mediante tamaño de muestras variables.

Las fórmulas necesarias para calcular los límites de control son:

- Límite de control superior LCS = $n\bar{p} + 3\sqrt{n\bar{p}(1-\bar{p})}$
- Límite de control central LC = $n\bar{p}$
- Límite de control inferior LCI = $n\bar{p} - 3\sqrt{n\bar{p}(1-\bar{p})}$

Las diferencias entre la gráfica de la carta “n” y la “np” de hecho son mínimas; únicamente que en esta última las cifras que se expresan en la gráfica no son proporciones sino cantidades de artículos “no conformes”, por lo que se opta por exponer más los resultados que la programación subyacente.

Tomando el ejemplo del inciso anterior (tabla XIX), se procede bajo la siguiente secuencia:

```
> library(XRSCC)
> r1<-np_gr(bottles, n=100)
[1] "Proceso fuera de Control en Grafica np"
> r1$out.control
[1]  3  9 20 22
> bottles[r1$out.control,]
[1] 12 10 11 10
> r1$"Limites de Control Grafica np"
  LCI.np   LCS.np    LC.np
0.000000 9.600496 3.837500
```

Notar que entre los argumentos también se necesita el tamaño constante de la muestra “n”. De paso, se muestra el vector de las líneas que están fuera de control, y con los mismos resultados se listan los valores de dichas líneas, los que pueden compararse también en la figura 65 con los límites de control de prueba. Por lo tanto, se necesita de la función *NP_it*( ) que calcula las siguientes iteraciones necesarias hasta encontrar los verdaderos límites de control.

Como en las funciones anteriores, la función *NP_it*( ), valida el único argumento necesario, que es el objeto que contiene la lista de los resultados anteriores, en este caso es **r1**, del cual comprueba si el proceso ya tiene el

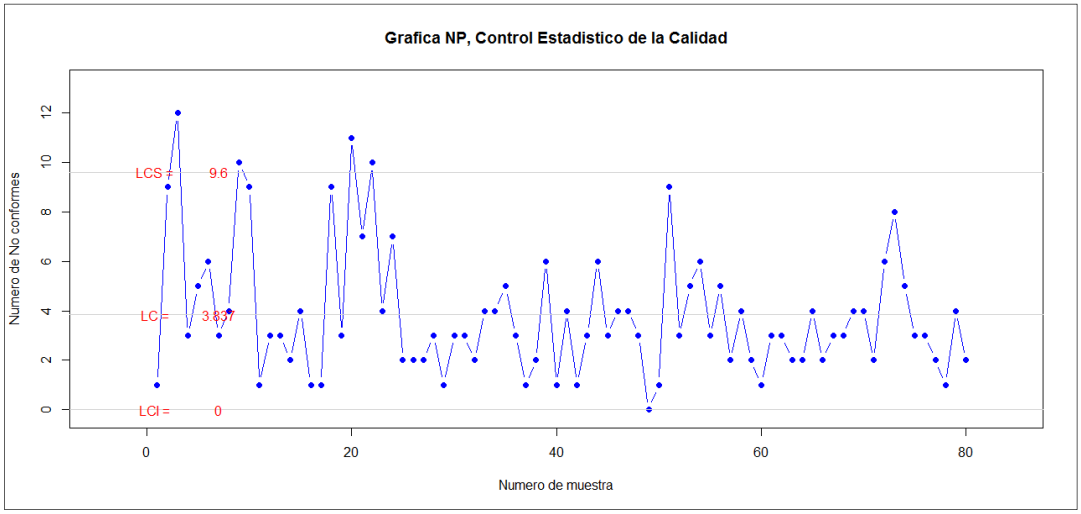
valor binario de “0” para “bajo control” y “1” para fuera de control; por lo tanto se detiene si los cálculos son innecesarios. Los pasos a seguir son:

```
> r2<-NP_it(r1)
[1] "Proceso fuera de Control en Grafica np"
> r3<-NP_it(r2)
[1] "El proceso esta bajo control en Grafica np"
> r3$"Limites de Control Grafica np"
  LCI.np   LCS.np    LC.np
0.000000 8.419999 3.166667
```

El resultado final al calcular los límites de prueba se muestra en la figura 66, que refleja la misma tendencia de los datos pero expresados en unidades. Tanto en la gráfica “p” como en la gráfica “np”, puede apreciarse que en las proporciones y el número de los no conformes, el límite de control superior es francamente elevado, respecto de los límites de especificación, ya que se espera un 5% de proporción de “no conformes” y en el caso de la gráfica “np”, equivale a 5 unidades por el tamaño de la muestra.

En tal sentido, examinando los puntos a lo largo de la muestra, existe cierto patrón de volatilidad en el extremo izquierdo, mientras que los datos de la muestra 25 en adelante, tienen, en términos estadísticos, una distribución normal. Para comprobar la estabilidad de los límites se procede a vectorizar los datos como se ha expuesto, para comparar los límites de control.

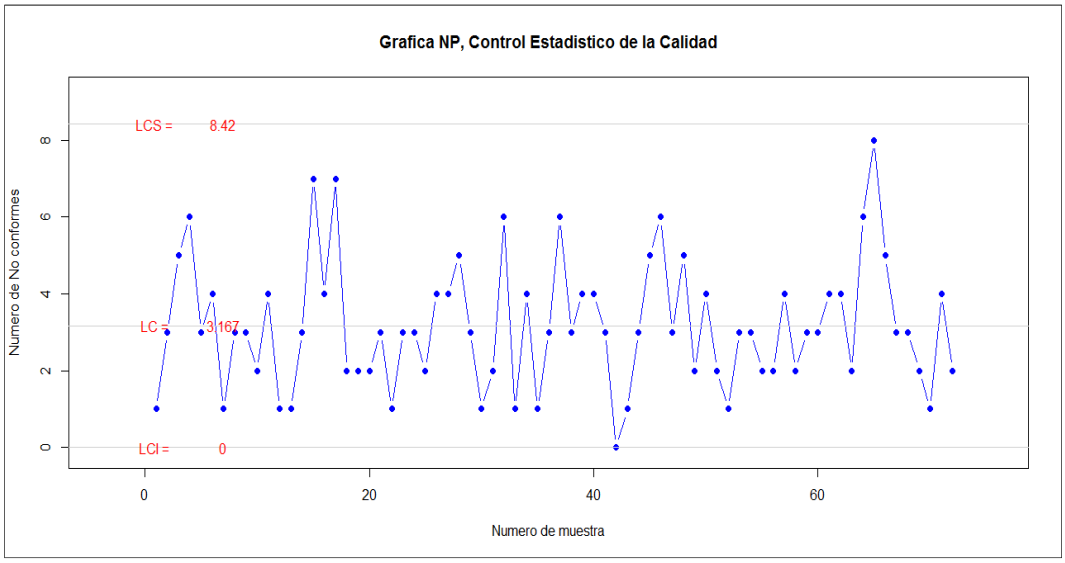
Figura 65. **Gráfica “np”, número de "no conformes"**



Fuente: elaboración propia, utilizando paquete XRSCC.

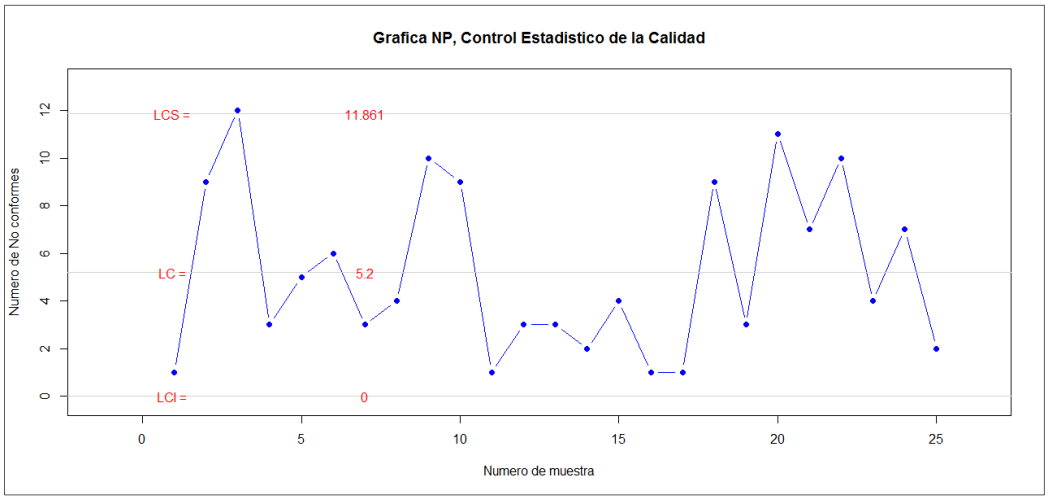
```
> r1.25<-np_gr(as.data.frame(bottles[1:25,]), n=100)
[1] "Proceso fuera de Control en Grafica np"
> r1.25.2<-NP_it(r1.25)
[1] "El proceso esta bajo control en Grafica np"
> r1.25.2$"Limites de Control Grafica np"
   LCI.np    LCS.np     LC.np
0.000000 11.403143  4.916667
> r26.80<-np_gr(as.data.frame(bottles[26:nrow(bottles),]), n=100)
[1] "Proceso fuera de Control en Grafica np"
> r26.80.2<-NP_it(r26.80)
[1] "El proceso esta bajo control en Grafica np"
> r26.80.2$"Limites de Control Grafica np"
  LCI.np   LCS.np    LC.np
0.0          8.319651 3.111111
```

Figura 66. **Gráfica "np", tercera iteración**



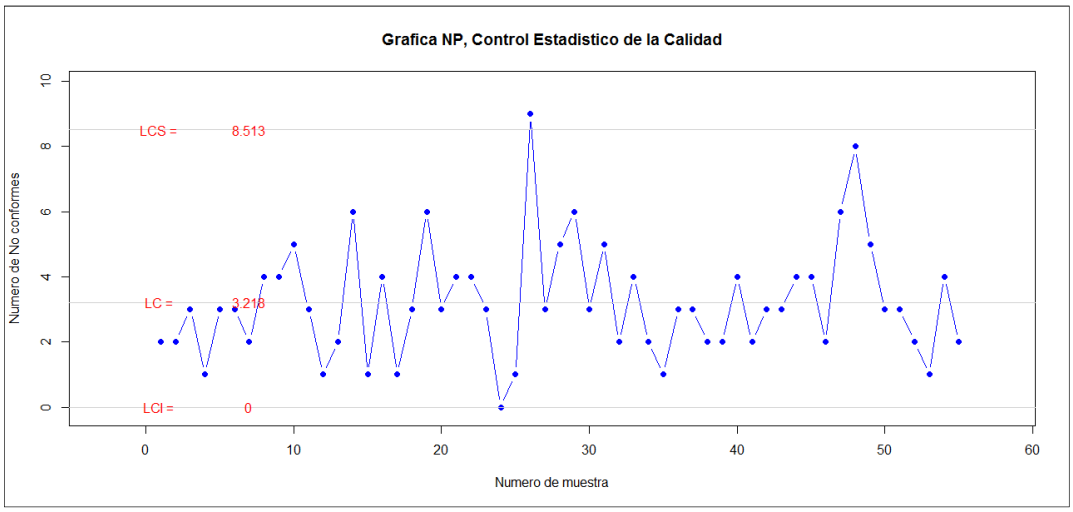
Fuente: elaboración propia, utilizando paquete XRSCC.

Figura 67. **Gráfica "np", grupo del 1 al 25**



Fuente: elaboración propia, utilizando paquete XRSCC.

Figura 68. **Gráfica "np", grupo del 26 al 80**



Fuente: elaboración propia, utilizando paquete XRSCC.

Comparando ambas gráficas, puede notarse la diferencia significativa entre los límites superiores del objeto **r1.25** y **r26.80**. En ambos casos, la primera iteración señala que el proceso está “fuera de control”, mientras que en la segunda el proceso ya es calificado como “bajo control”. El límite de control superior del primer grupo es francamente elevado, mientras que el correspondiente al segundo se asemeja a la muestra total. Esto confirma que a medida que el tamaño de la muestra crece los límites tienden a los límites de control natural; debido a ello se hizo mención de la normalidad del segundo grupo de la muestra.

3.2.2.3. Gráfica u, número de inconformidades por unidad, función u_gr

En esta gráfica para control de atributos, se calculará sobre el supuesto de que el tamaño n_i de la muestra es variable; de esa forma se incrementa la

complejidad a medida que se desarrolla el texto. No obstante, la complejidad de los cálculos, no implica cambios de fondo a lo ya desarrollado en las funciones anteriores. Notar que para el presente caso se describe de forma indistinta entre “no conformidades” e “inconformidades” y respectivas conjugaciones.

Las fórmulas necesarias para calcular los límites de controles son:

- Límite de control superior LCS = $\bar{u} + 3\sqrt{\frac{\bar{u}}{\bar{n}}}$
- Límite de control central LC = $\bar{u}$
- Límite de control inferior LCI = $\bar{u} - 3\sqrt{\frac{\bar{u}}{\bar{n}}}$

Los datos de entrada están conformados por tres cifras necesarias; la primera consiste en el tamaño “n_i” de la muestra, lo suficientemente grande para detectar al menos una “inconformidades”; la segunda es el número de inconformidades encontradas u_i en cada muestra. Ver el siguiente ejemplo:

Un proceso de manufactura de prendas de vestir deportivas requiere de inspección, puesto que se han observado “no conformidades” en distintas prendas y en lotes de producción; se desea conocer por medio de un muestreo aleatorio sobre los distintos turnos y líneas de producción el número de inconformidades tales como: costuras sueltas, cierres de cuello mal ajustados, etiqueta mal cosida, entre otras. El propósito final de esta inspección es aplicar los correctivos a las causas asignables. Los datos del muestreo están tabulados en los datos adjuntos **clothes2** u observar la tabla XX.

Para determinar los límites de control mediante la función *u_gr*( ) y posteriormente con *U_it*( ) (sí y solo sí es necesario)los verdaderos límites de

control para el promedio de inconformidades por unidad se procede bajo la siguiente secuencia.

```
> library(XRSCC)
> r1<-u_gr(clothes2)
[1] "Proceso fuera de Control en Grafica U"
> r1$`Limites de Control Grafica U`
   LCI.u    LCS.u     LC.u
1.643497 2.876564 2.260031
```

La función *u_gr*( ) en particular necesita que el objeto donde está almacenada, tenga los nombres específicos de las variables como “d” para el número de defectos encontrados en la muestra de tamaño “n”; ante cualquier diferencia a lo estipulado en nombre y cantidad de variables (2), la ejecución se detiene. De igual forma, la función valida si un tamaño de muestra entre las “m” filas, es igual cero; de serlo, la ejecución se detiene.

Al igual que las funciones anteriores, el dictamen sobre el estado del proceso se calcula a partir de la vectorización de los puntos que están fuera de los límites de control. En el ejemplo, la lista **r1**, de la primera iteración contiene la información necesaria para las iteraciones que sean necesarias para encontrar los límites naturales del proceso. Los puntos que están fuera de control son los siguientes:

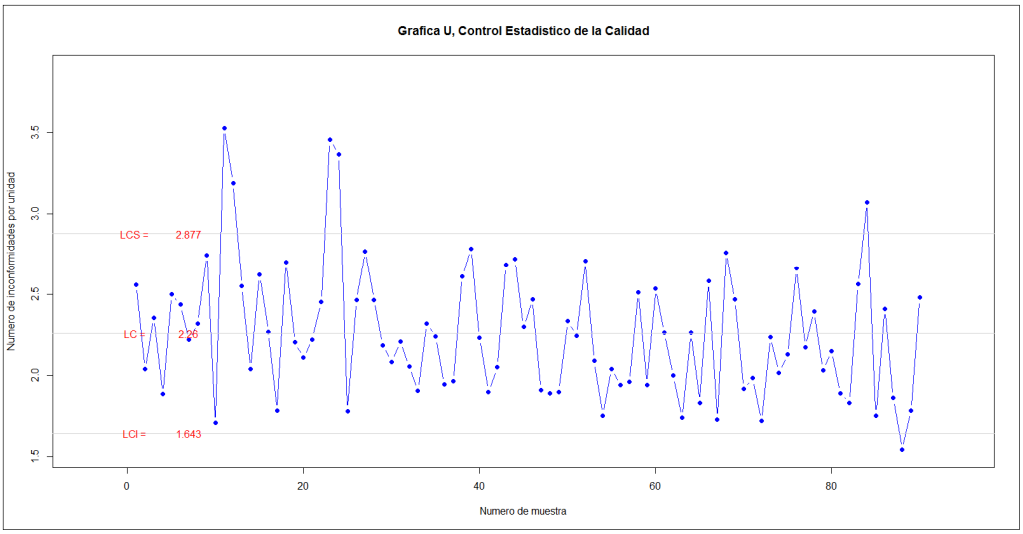
```
> r1$out.control
[1] 88 11 12 23 24 84
```

Tabla XX. **Datos de entrada, gráfica u**

Número de inconformidades por unidad de prenda de vestir											
n	**d**	**n**	**d**	**n**	**d**	**n**	**d**	**n**	**d**	**n**	**d**
50	128	60	136	53	117	49	121	61	138	53	141
50	102	60	107	55	113	55	105	55	110	57	124
45	106	60	162	54	103	55	104	50	87	61	146
44	83	59	130	50	116	60	114	53	120	60	122
40	100	55	116	50	112	60	140	54	99	60	129
39	95	54	120	55	107	61	137	55	142	55	104
50	111	51	125	55	108	62	168	52	90	54	99
50	116	55	190	54	141	55	115	50	138	41	105
55	151	55	185	60	167	49	86	60	148	42	129
55	94	55	98	60	134	50	102	61	117	53	93
38	134	56	138	60	114	50	97	60	119	56	135
37	118	56	155	60	123	50	98	50	86	58	108
40	102	56	138	60	161	51	128	55	123	57	88
50	102	59	129	50	136	53	103	55	111	51	91
50	131	59	123	50	115	56	142	54	115	48	119

Fuente: elaboración propia.

Figura 69. **Gráfica "u", primera iteración**

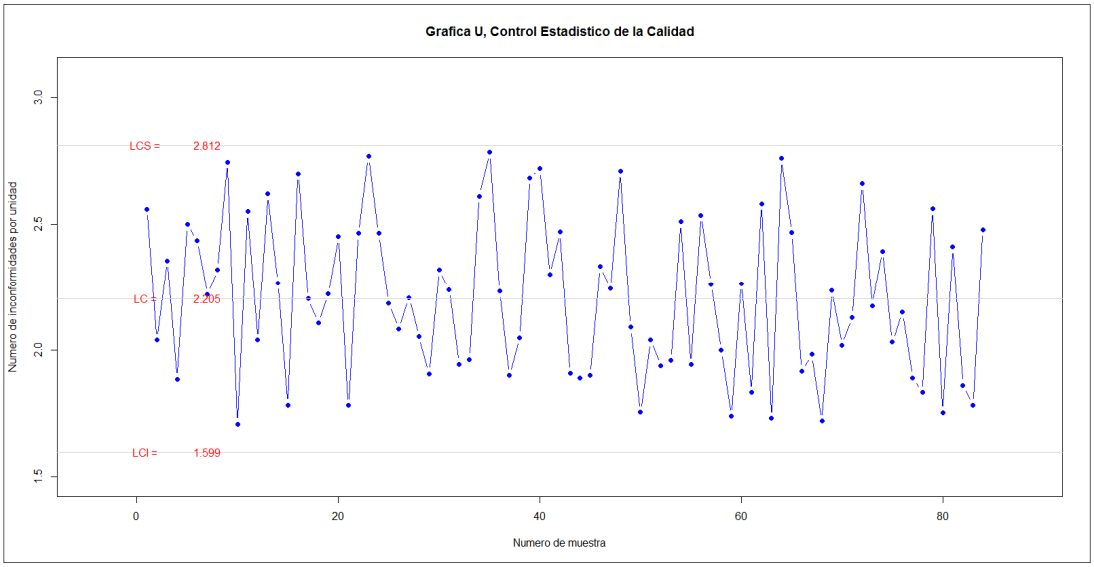


Fuente: elaboración propia, utilizando paquete XRSCC.

Derivado de la gráfica anterior y por los resultados expresados y almacenados en **r1**, en la segunda iteración se consigue obtener los límites naturales de control después de depurar aquellos datos mostrados en la página anterior.

```
> r2<-U_it(r1)
[1] "El proceso esta bajo control en Grafica U"
> r2$`Limites de Control Grafica U`
   LCI.u    LCS.u     LC.p
1.598862 2.811943 2.205402
```

Figura 70. **Gráfica "u", segunda iteración**



Fuente: elaboración propia, utilizando paquete XRSCC.

3.2.2.4. Gráfica c, número de inconformidades por unidad, función c_gr

Se puede decir que la gráfica “c” es una variante de la gráfica “u” y viceversa, puesto que en la gráfica “c” se consideran muestras del mismo tamaño; lo que implica que solo es una columna de datos de entrada y el tamaño de la muestra se define como un argumento constante en la función *c_gr*( ), seguido de la iteración con *C_it*( ), lo límites de control se calculan como:

- Límite de control superior LCS = $\bar{c} + 3\sqrt{\bar{c}}$
- Límite de control central LC = $\bar{c}$
- Límite de control inferior LCI = $\bar{c} - 3\sqrt{\bar{c}}$

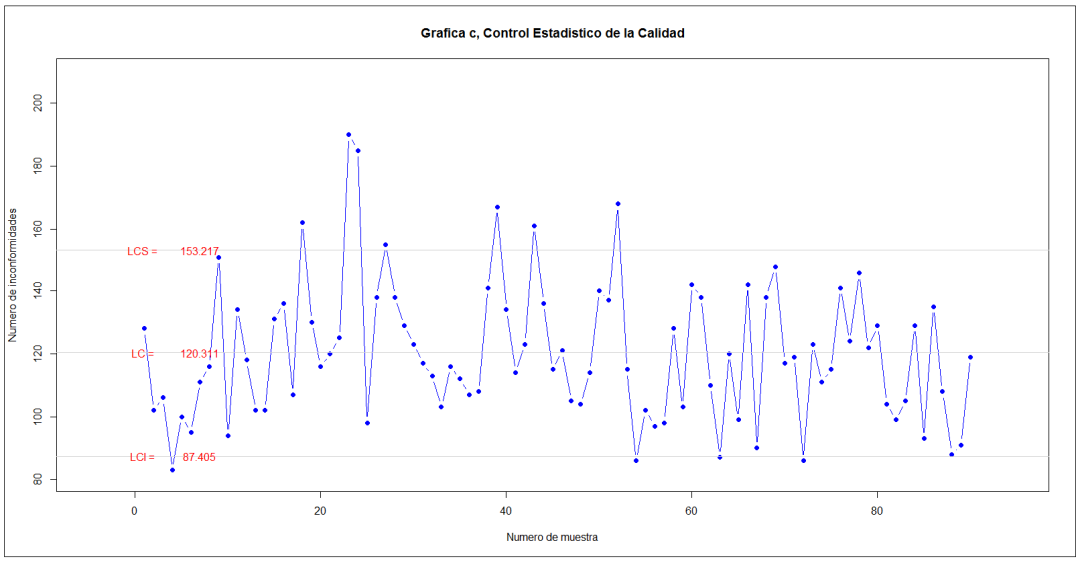
El límite de control central o $\bar{c}$, se calcula como:

$$\bar{c} = \frac{1}{N}\sum_{i=1}^{n} c_i$$

Adaptando el enunciado de la gráfica u, los datos de entrada para el presente tema, vectorizando únicamente el número de defectos de cada muestra; el número **N** de subgrupos es el número de líneas, usando la columna “d” de la tabla XX se tiene:

```
> C<-clothes2$d
> r1<-c_gr(as.data.frame(C))
[1] "Proceso fuera de Control en Grafica c"
```

Figura 71. **Gráfica "c", primera iteración**



Fuente: elaboración propia, utilizando paquete XRSCC.

Mientras empleando los datos **clothes** adjuntos en el paquete XRSCC, se tienen los siguientes resultados:

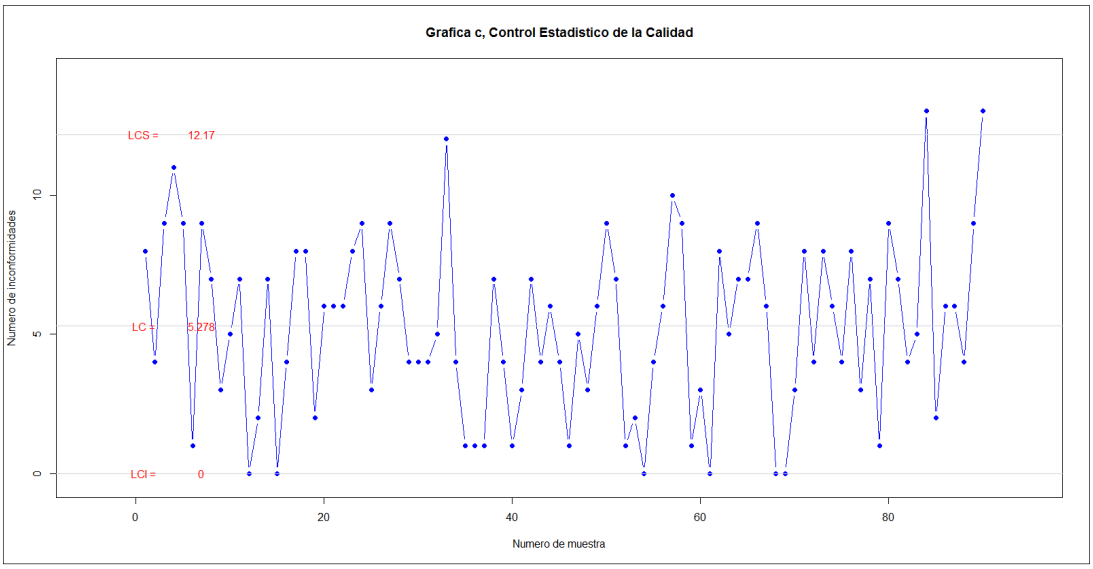
```
> r1b<-c_gr(clothes)
[1] "Proceso fuera de Control en Grafica c"
```

En ambos casos, existen puntos fuera de los límites de control, por lo tanto se utiliza la función de iteración *C_it*( ),en ambos casos y las iteraciones necesarias.

```
> r2<-C_it(r1)
[1] "Proceso fuera de Control en Grafica c"
> r2b<-C_it(r1b)
[1] "Proceso fuera de Control en Grafica c"
```

```
> r3<-C_it(r2)
[1] "El proceso está bajo control en Grafica c"
> r3$`Limites de Control Grafica c`
    LCI.c     LCS.c          LC.c
 84.78196 149.75651 117.26923
> r3b<-C_it(r2b)
[1] "El proceso esta bajo control en Grafica c"
> r3b$`Limites de Control Grafica c`
    LCI.c     LCS.c          LC.c
 0.000000 11.746596  5.022989
```

Figura 72. **Gráfica "c", primera iteración datos *clothes***

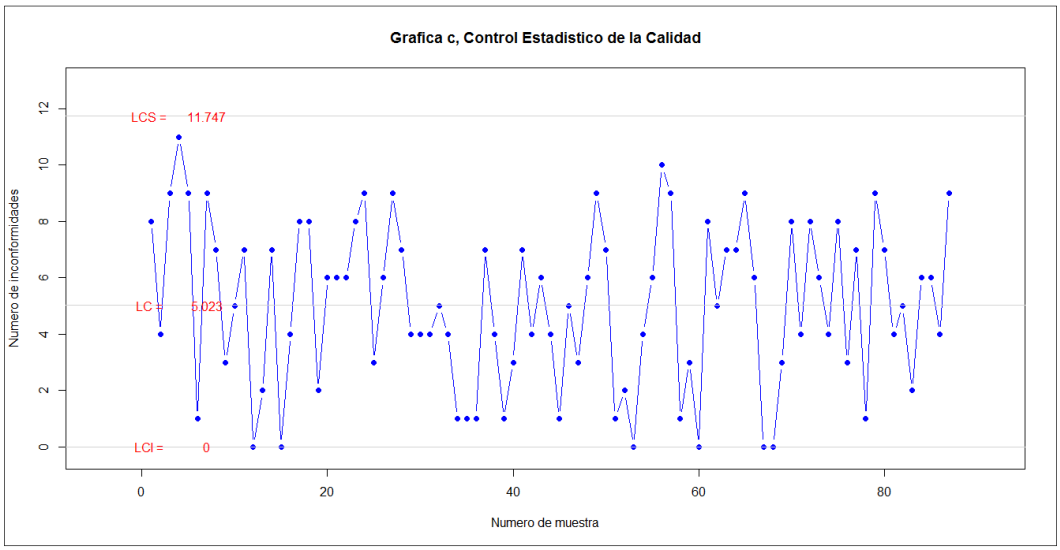


Fuente: elaboración propia, utilizando paquete XRSCC.

En la tercera iteración, en ambos casos se obtiene un proceso controlado; en lo que puede notarse claro un diferencia de dimensiones entre los respectivos límites de control, que dependerán del tipo de proceso que se trate, es decir, unos con mayor complejidad que otros; no es lo misma cantidad de

defectos que se pueden encontrar en la fabricación de una computadora que la cantidad de defectos en la fabricación de bolsas de agua pura, aunque el presente se trate de en ambos casos de manufactura de ropa.

Figura 73. **Gráfica "c", tercera iteración, datos *clothes***



Fuente: elaboración propia, utilizando paquete XRSCC.

3.2.3. Capacidad del proceso

Hasta el anterior inciso, se observó en algunos casos en forma indirecta el mayor o menor rango entre los límites de control y los límites de especificación, sin embargo para comprobar la capacidad del proceso es necesario utilizar el parámetro sigma $\left(\hat{\sigma} = \frac{\bar{R}}{d_2}\right)$; no obstante, a menudo no se conoce su verdadero valor, por lo que se calcula un estimador insesgado.

Para el caso de la gráfica $\bar{X}$ se utiliza $\hat{\sigma} = \frac{\bar{R}}{d_2\sqrt{n}}$, en cualquier caso el análisis se fundamenta en la función:

$$C_p = \frac{LES - LEI}{6\hat{\sigma}}$$

En la que $6\hat{\sigma}$ es el rango de $\pm 3\hat{\sigma}$ de la media μ, es decir que define el rango entre el límite natural de control inferior y el superior. De ello, son bastantes los autores que advierten la diferencia conceptual entre los límites naturales de control y los límites de especificación, la cual radica en que los últimos son fijados de acuerdo con el cumplimiento de una norma, metas de calidad o simplemente pruebas, mientras que los límites naturales de control son los que resultan de la variabilidad natural o aleatoria del proceso. Sin embargo, la razón de capacidad del proceso" C_p" es una aproximación del uso del "rango de especificación". Entonces es claro ver que:

- C_p> 1, los límites naturales están dentro de los límites de especificación
- C_p< 1, los límites naturales sobre pasan a los límites de especificación
- C_p= 1, los límites naturales de control son iguales a las especificaciones

En este procedimiento se emplean los límites de especificación como argumentos, así como los resultados obtenidos al encontrar los límites naturales de control después de ejecutar la función *xrs_gr* ( ), *X_it* ( ) y *R_it* ( ) y que el proceso está centrado entre los límites de especificación. Por último, la razón derivada $\hat{p}$ o C_{pk} es el porcentaje de uso de los límites de especificación mediante:

$$\hat{p} = \left(\frac{1}{C_p}\right) \times 100$$

El proceso para los cálculos es de baja complejidad, sin embargo se agrega el aspecto gráfico, llevando los límites a figurar bajo una curva normal de probabilidades, asumiendo desde luego normalidad $(x \sim N(\mu, \sigma^2))$, en la distribución de los datos.

La curva de la función de densidad está definida por la siguiente expresión es:

$$fd = \frac{1}{\hat{\sigma}\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\hat{\sigma}}\right)^2}$$

Los datos de entrada o argumentos para obtener la capacidad del proceso por variables son *Cp_X*(prev.results, LES, LEI, mu).

Tabla XXI. **Argumentos función Cp_X**

Argumento	**Descripción**
prev.results	Es las lista de resultados con la carta X y R bajo control
LES, LEI, mu	Vectores de tamaño uno, con el límite superior, inferior de especificación y la media esperada en el proceso.

Fuente: elaboración propia.

Los resultados obtenidos mediante la siguiente secuencia de pasos:

```
> data(vol_sample)
> r1<-xrs_gr(vol_sample)
[1] "Proceso fuera de Control en Grafica X"
[1] "El proceso esta bajo control en Grafica R"
[1] "El proceso esta bajo control en Grafica S"
> r2<-X_it(r1)
[1] "El proceso esta bajo control en Grafica X"
[1] "El proceso esta bajo control en Grafica R"
```

```
> cp<-Cp_X(r2, LES=510, LEI=490, mu=500)
[1] "Los limites naturales sobrepasan  los limites de especificacion"
```

Estos resultados muestran que se está utilizando un 29,2% más de los límites de especificación. Lo que conlleva al análisis de la variabilidad implícita del proceso, persiguiendo que el porcentaje del uso de los límites de especificación sea menor que el 100%; los resultados se presentan en la figura 74.Por otro lado, se puede emplear el estimador insesgado en función de la media del proceso, omitiendo el argumento **mu** (ver figura 75).

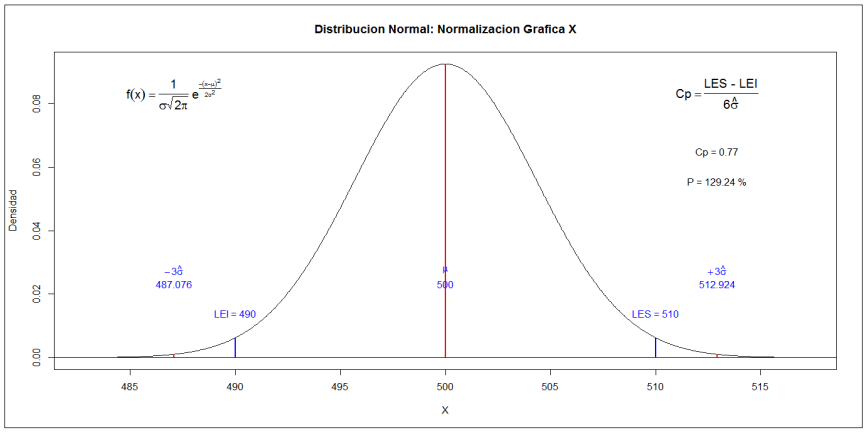
```
> r3<-Cp_X(r2,LES=510, LEI=490)
[1] "Los limites naturales estan dentro de los limites de especificacion"
```

Notar que se han normalizado resultados para que bajo la curva aparecieran en forma simétrica respecto de la media, la cual utilizó la “cantidad consignada” es decir de 500 ml, identificada con el parámetro μ o “mu”. Las líneas de los límites naturales, se encuentran $\pm 3\hat{\sigma}$ de la media, mientras que el otro par de líneas visibles corresponde a límites de especificación. Dichas líneas pueden verse no simétricas respecto de la media, por ello, para medir la capacidad del proceso, debe utilizarse el indicador C_{pk} que se calcula de la siguiente forma:

$$C_{pk} = \frac{\min(LES - \mu,\ \mu - LEI)}{3\sigma}$$

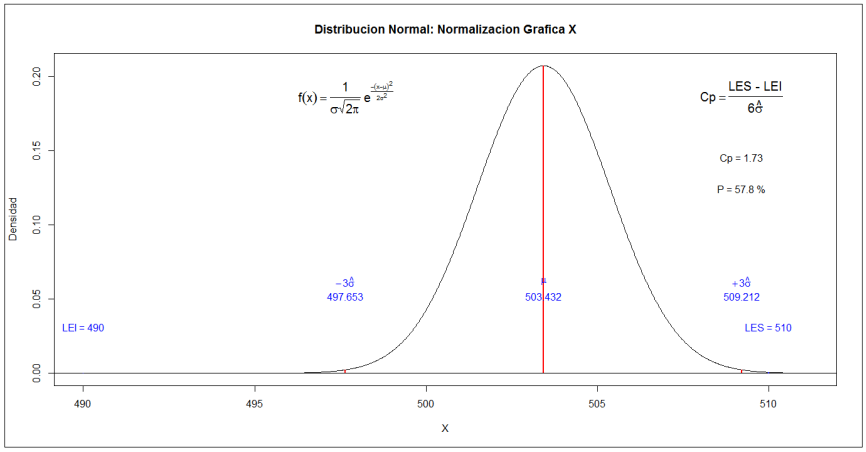
Ya que se basa en el supuesto que cuenta con los dos límites de especificación, pero la media $\bar{\bar{X}}$ calculada a partir de los datos, está descentrada o no está de forma equidistante entre los límites de especificación.

Figura 74. **Capacidad del proceso**



Fuente: elaboración propia, utilizando paquete XRSCC.

Figura 75. **Capacidad del proceso, calculada con estimado insesgado**



Fuente: elaboración propia, utilizando paquete XRSCC.

Se ha visto que el criterio indiscutible sobre el que se considera que un proceso está fuera de control, es cuando uno o más puntos exceden los límites establecidos. No obstante, los comportamientos “no estocásticos”,

estadísticamente reflejan una variabilidad mayor en contraste con los procesos compuestos, mayormente por vaivenes aleatorios.

Por otro lado, hay valores atípicos, en las series de datos, que junto con patrones o rachas, pueden estar ligados a causas asignables e incluso a errores de medición y conteo, tal es el caso para control por variables y por atributos, respectivamente.Los patrones atípicos o grupos racionales dentro de series de datos para el control estadístico de calidad, deben de ser extraídos para calcular los verdaderos límites de control, para evitar la comisión del error tipo II al aceptar lotes o procesos que en realidad están fuera de control.

Una de las actividades estratégicas previo a la recolección de los datos, es sin duda el diseño del plan de muestreo. El diseño permite capturar el núcleo mismo del comportamiento aleatorio, en el caso de la carta X, resulta en un comportamiento que tiende a ajustarse al supuesto de normalidad.

Sin embargo, hay condiciones muy particulares en las que el comportamiento de las variables presentes o atributos no se presentan como un fenómeno aleatorio, cuando son comparadas con el espectro que encierra los límites de control llamado como “Seis Sigma”. Los ya mencionados valores atípicos y rachas en particular dentro de los límites de control, se dan bajo las reglas *Western Electric* (signos de que el proceso se encuentra fuera de control) y deben de ser investigadas dentro de las causas asignables.

Los límites naturales que definen hasta ± 3 desviaciones estándar, observados de forma breve con anterioridad, son los que separan tres zonas, en las cuales se definen las reglas *Western Electric*.

Las reglas *Western Electric* fueron diseñadas por el comité especializado

de la división de manufactura de la compañía con el mismo nombre. Todo el contenido completo se refiere a asegurarse de que todo trabajador o ingeniero pudiera interpretar las cartas de control de la misma forma. En tal sentido, las reglas no consisten en sí en un agregado técnico al instrumental estadístico, sino un conjunto de reglas diseñadas para servir como criterio para interpretar las cartas de control.

La primera regla *Western Electric* ya ha sido interpretada y puesta en práctica en el contenido de este mismo capítulo, la cual señala que un proceso se encuentra fuera de control si al menos un punto de la carta está fuera de los límites de control o en este caso, más allá de la zona A, por lo que al utilizar la función se espera que no existan indicios de la primera regla, debido a que se recomienda utilizar los objetos o listas de resultado con los límites naturales de control. Para el análisis de las reglas *Western Electric,* es necesario definir el intervalo que cubre cada zona, las mismasestán definidas en la tabla XXII.

Tabla XXII. **Zonas de alerta, reglas Western Electric**

Zona	**Intervalo que cubre**
A	Desde los límites $\pm 2\,\sigma$ a $\pm 3\,\sigma$, es decir hasta el límite superior o inferior.
B	Desde la línea que limita $\pm 1\,\sigma$ a $\pm 2\,\sigma$
C	De la línea de central de control o la media a $\pm 1\,\sigma$

Fuente: elaboración propia.

Como se observa, al dividir la gráfica de la carta de control en zonas delimitadas por múltiplos de la desviación estándar, se obtiene un rango numérico de tres, que va del límite superior o inferior y el límite central. Las reglas a analizar son especificadas en la tabla XXIII.

En efecto, las reglas *Western Electric* complementan las reglas *Shewhart*. Las primeras observan los patrones no aleatorios, mientras que las últimas observan variabilidad, no obstante, al eliminar patrones sistemáticos, se espera que el proceso manifieste por lo tanto variabilidad natural, por ende los límites de control.

Tabla XXIII. **Reglas de sensibilización Western Electric**

Regla	**Descripción**
1*	Uno o más puntos fuera de los límites de control
2	Dos de cada tres puntos consecutivos en la zona A
3	Cuatro de cada cinco puntos consecutivos más allá de la zona C
4	Una corrida de ocho puntos consecutivos de un lado de la línea de control central
(*) Regla ya revisada	

Fuente: MONTOGOMERY, *Douglas. Introduction to statistical quality control*. p. 166.

Para efectos demostrativos se emplean únicamente las descritas en la tabla XXII, sinembargo más adelante se analizan con otras herramientas otras reglas referidas a los siguientes aspectos:

- Un comportamiento inusual o no aleatorio en los datos y presenciao ausencia de autocorrelación.

- Tendencia marcada o no estacionaria en el tiempo

Para ponerlo en un contexto gráfico, se emplea la gráfica de la carta $\bar{\bar{X}}$, vista en el inciso 0, ampliando los resultados y mostrando las zonas que deben verificarse de las reglas *Western Electric*, de allí se debe hacer un análisis de las reglas descritas en la tabla XXII.

La función *we_rules*( )tiene por objetivo concluir si el proceso está “bajo” o “fuera” de control con la existencia de violación a cualquiera de las reglas *Western Electric* uno a la cuatro.

El único argumento necesario para ejecutar la función es el objeto que contiene la lista de los resultados previos después de que el proceso sea calificado bajo control en la última iteración, ya que se utiliza el límite central del rango o **LCR** y los límites de control de la carta $\bar{\bar{X}}$ (LCS, LCI y LC).

Para calcular sigma a partir de utiliza $\hat{\sigma} = \frac{\bar{R}}{d_2}$, se procede a incorporar el factor corrector para la muestra, figurando por último como $\hat{\sigma} = \frac{\bar{R}}{d_2\sqrt{n}}$, en el que $\sqrt{n}$ es la raíz del tamaño de cada subgrupo.

```
we_rules<-function(prev.results){
  if (missing(prev.results)){
    stop("No elementos para evaluar")
    } else {
      data(factor.a)
      n.1<-ncol(prev.results$data.1)
      LCR<-as.numeric(prev.results$LR[2])
X.sigma<-expression(eval(LCR)/((factor.a$d2[n.1-1])*sqrt(n.1)))
```

El valor acumulado de cada zona, es detallado en la tabla XXIV, se enumera según el intervalo [-3,3] y se crean en las siguientes líneas de la función.

Tabla XXIV. **Orden acumulado de las zonas de alerta**

Zona	**Valor**
Más allá de la zona + A	7
+ A	6
+ B	5
+ C	4
- C	3
- B	2
- A	1
Más allá de la zona – A	0

Fuente: elaboración propia.

```
limites.zona <- seq(-3, 3)
    # Crea el conjunto de zonas para cada punto de la carta X
zonas <- sapply(limites.zona,
                function(i) {
                  i * rep(eval(X.sigma), m.1)
})
    zonas<-zonas + eval(LC.X)
    colnames(zonas) <- paste("zona", -3:3, sep="_")
    x.zonas <- rowSums(X.prom.1 > zonas)
```

Donde se crean siete columnas con los valores de LC + [- 3$\hat{\sigma}$, 3 $\hat{\sigma}$] en **m** filas; comparando cada promedio de fila con cada columna, el resultado es 1

para “mayor que”, y cero “0” cuando es menor. Cada valor de la anterior comparación se suma para almacenarse internamente en **X.zonas**; dicho objeto resulta en una cadena de valores según la tabla XXIV. Las cadenas son validadas en un ciclo para cada regla:

```
   for(i in 1:m.1){
    # Cualquier valor mas alla +/- 3 sigmas
v1 <- x.zonas[i]
    regla1 <- ifelse(any(v1 == 7), 1,
ifelse(any(v1 == 0),1, 0))
    # Dos de cada tres puntos en la zona A
v2 <- x.zonas[max(i-3, 1):i]
    regla2 <- ifelse(sum(v2 == 6) >= 2, 1,
                ifelse(sum(v2 == 1) >= 2, 1, 0))
# Cuatro de cada cinco puntos mas alla de la zona C
v3 <- x.zonas[max(i-5, 1):i]
    regla3 <- ifelse(sum(v3 >= 5) >= 4, 1,
                ifelse(sum(v3 <= 2) >= 4, 1, 0))
# Ocho puntos consecutivos de un lado de la
    # linea de control central
    v4 <- x.zonas[max(i-8, 1):i]
regla4 <- ifelse(all(v4 >= 4), 1,
                ifelse(all(v4 <= 3), 1, 0))
}
```

En las anteriores líneas queda cuantificada de forma binaria la existencia con “1” o inexistencia con “0” de la violación a cualquiera de las cuatro reglas, cuyo resultado general se manifiesta en la estructura final y resumen de los resultados.

```
    bin.w<- if(sum(c(regla1,regla2,regla3,regla4))> 0){
```

```
bin.w<-1
       } else {
         bin.w<-0
}
     # Ejecutar grafico
     dev.off()
     plot.Xzones()
     structure(list("Resultados del analisis" = if(bin.w>0){
       print("El proceso esta fuera de control por las Reglas Western Electric")
       print("Las siguientes reglas tienen al menos un grupo que viola la regla")
       c("Regla 1"= regla1,
         "Regla 2"= regla2,
         "Regla 3"= regla3,
         "Regla 4"= regla4)
     } else {
       print("El proceso esta bajo control por las reglas Western Electric")
     }))
   }
}
```

Siguiendo el ejemplo con los datos de sacos de azúcar pesados en libras, se tiene en primer lugar un proceso bajo de control en la carta X, no así, en la carta R, por ello se recurre a la iteración *R_it*( ); en seguida se ejecuta la función *we_rules*( ), en tal orden de ideas,se espera que no haya violación a la regla 1.

```
> data(qqsugar)
> r1<-xrs_gr(qqsugar)
[1] "El proceso esta bajo control en Grafica X"
[1] "Proceso fuera de control en Grafica R"
[1] "Proceso fuera de control en Grafica S"
> r2<-R_it(r1)
```

```
[1] "El proceso esta bajo control en Grafica X"
[1] "El proceso esta bajo control en Grafica R"
> we_rules(results2)
[1] "El proceso esta fuera de control por las Reglas Western Electric"
[1] "Las siguientes reglas tienen al menos un grupo que viola la regla"
$`Resultados del analisis`
Regla 1 Regla 2 Regla 3 Regla 4
 0       1            0      0
```

Se puede apreciar que hay cadenas que violan la regla 2 o la existencia de al menos una cadena de 2 de cada tres puntos en la zona A, es decir dos puntos consecutivos de tres, aproximándose al límite de control inferior o superior después de $\pm 2\hat{\sigma}$, respectivamente.

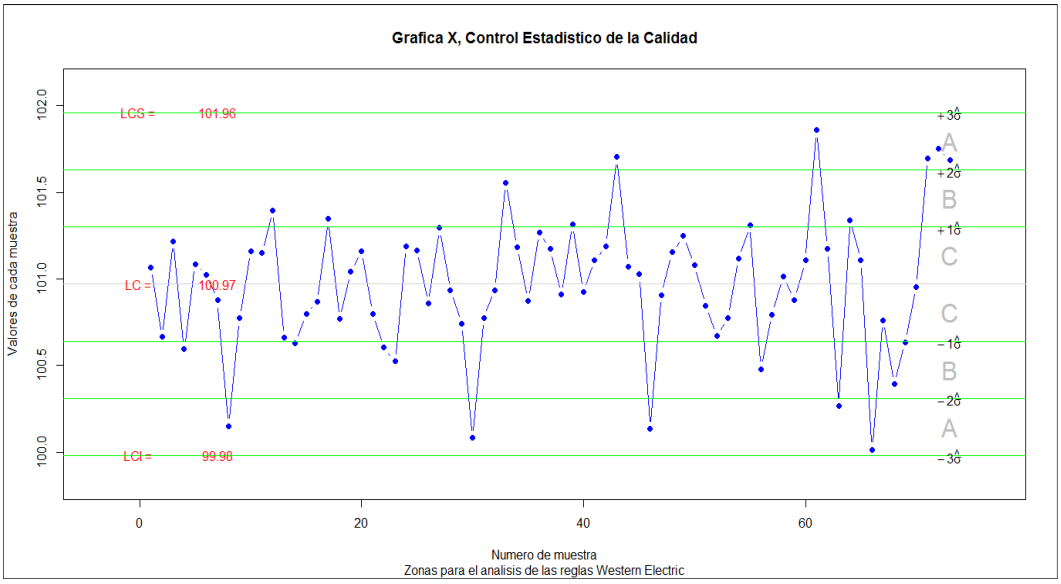
Los patrones no aleatorios se pueden apreciar en la figura 76, los que figuran en los últimos puntos de la derecha; hay tres puntos en la zona + A que son argumento suficiente para concluir que se ha violado la regla dos, por lo tanto, el proceso no está bajo control.Al eliminar estos registros o filas, se espera de hecho que los límites de control se vean modificados.

3.2.4. Curva de operación

La capacidad de las gráficas $\bar{X}$ y R para detectar un corrimiento en el proceso de calidad es descrita por la curva característica de operación o simplemente curva característica de operación (*OC Curve*)[18]. La función de la CO se basa en la relación entre el corrimiento de κ veces la desviación estándar σ, para que la media μ sea μ_1 es decir $\mu_1 = \mu + k\sigma$ en tal sentido existe un riesgo β de no ser detectado ese corrimiento.

[18] MONTOGOMERY, Douglas*. Introduction to statistical quality control*. p. 217.

Figura 76. **Gráfica $\bar{X}$, con zonas de alerta**



Fuente: elaboración propia, utilizando paquete XRSCC.

Dado que se asume la normalidad de la media si $\bar{x} = \mu$

$$\because \bar{x} \sim N(\mu, \sigma^2)$$

Entonces se define el riesgo acumulado a partir de que L es el número de desviaciones estándar de la media, por lo general $L = 3$ (3 sigmas). De tal manera que el riesgo se calcula como:

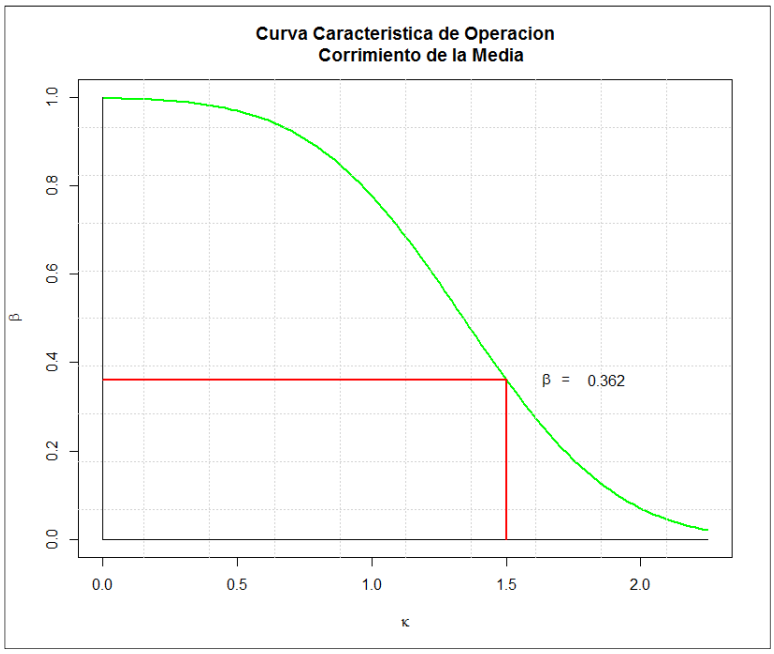
$$\beta = \Phi\left(L - \kappa\sqrt{n}\right) - \Phi\left(-L - \kappa\sqrt{n}\right)$$

Donde Φ representa la distribución normal acumulada y β está directamente relacionadacon el corrimiento κ y con el tamaño de la muestra n.

El riesgo β varía en forma inversa al tamaño de la muestra y al factor de corrimiento de la media. Dicha relación es trasladada a una gráfica donde se visualiza la función matemática $f(\kappa, n) = \beta$, con todo el código completo de la función *Beta.X*( ) contenida en el paquete XRSCC, la cual utiliza el principio de la función ya diseñada, pero ampliando el ambiente para devolver tanto la curva de operación como la gráfica del ancho medio de la corrida. Los argumentos necesarios son: “k” que es el factor de “k” desviaciones estándar respecto de la media y “n” que es tamaño de la muestra; en todo caso el tamaño del grupo.

```
>Beta.X(k=1.5,n=5)
$beta
[1] 0.3616312
$ARL
[1] 1.566493
> Beta.X(k=1.5,n=3)
$beta
[1] 0.6561299
$ARL
[1] 2.908075
> Beta.X(k=0.9,n=5)
$beta
[1] 0.8383105
$ARL
[1] 6.184692
> Beta.X(k=0.9,n=3)
$beta
[1] 0.9252269
$ARL
[1] 13.37379
```

Figura 77. **Curva característica de operación con n=5 y k=1,5**



Fuente: elaboración propia, utilizando paquete XRSCC.

La figura 77 demuestra que el riesgo de no detectar el corrimiento κ de la media igual a 1,5 desviaciones estándar y con una muestra igual a 5, es igual a 36,2 %. Asumiendo un mismo tamaño de muestra, para un factor de corrimiento menor, el riesgo de no detectarlo es 83,8 %.

En el siguiente ejemplo y para efectos demostrativos, se ejecuta una función directamente en la línea de comando, la cual devuelve un vector con todos los 100 valores de “k” comprendidos entre 0 y 5, para obtener la curva completa. En la figura 78 se detalla la CO a sendos valores de n = 5,7,10 y 12, lo cual señala que a medida que n crece, el riesgo β de no detectar el corrimiento disminuye.

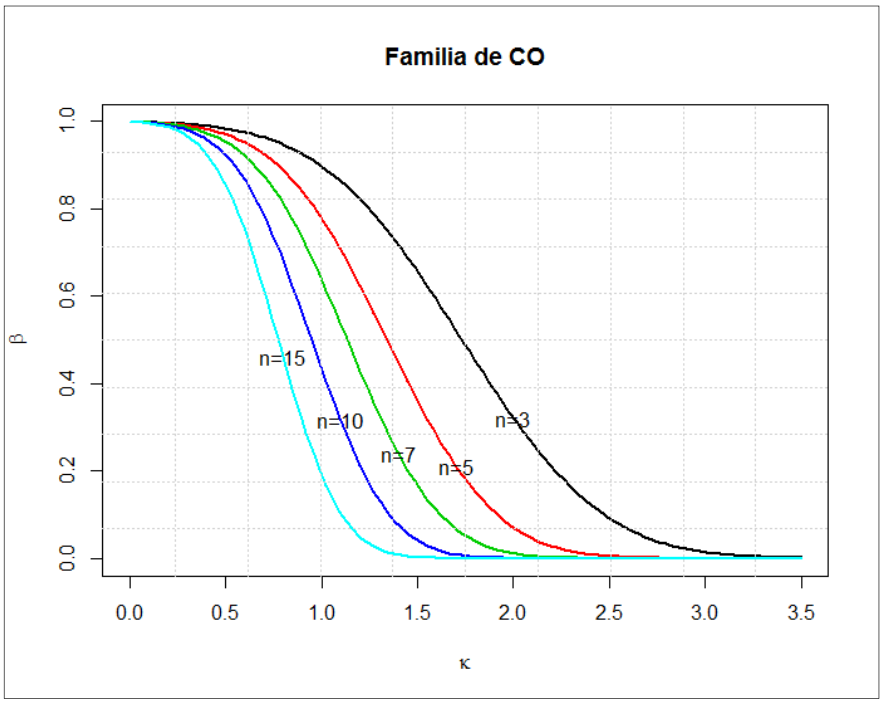
```
beta.x1<-function(k,n,L=3){
   phi1 <- pnorm(c(L-k*sqrt(n)), mean=0, sd=1, lower.tail=TRUE)
   phi2 <- pnorm(c(-L-k*sqrt(n)), mean=0, sd=1, lower.tail=TRUE)
   beta.1<-phi1-phi2
   return(beta.1)
}
```

Uniendo los resultados de n=3,5,7,10 y 15 en una matriz, para luego plotearlos juntos en la misma gráfica con la siguiente secuencia de instrucciones.

```
k<-seq(0,5,length.out=100)
# Aplicar la función Beta.X a distintos valores de n
Beta.n3<- beta.x1(k,n=3)
Beta.n5<- beta.x1(k,n=5)
Beta.n7<- beta.x1(k,n=7)
Beta.n10<- beta.x1(k,n=10)
Beta.n15<- beta.x1(k,n=15)
# Integrar los resultados en una matriz
Beta.mat<-cbind(Beta.n3,Beta.n5,Beta.n7,Beta.n10,Beta.n15)
#    Plotear la matriz
matplot(k, Beta.mat, type = "l", lty = 1, lwd = 2, lend = par("lend"),
     pch = NULL,
     col = 1:6, cex = 2, bg = NA,
     xlab = expression(kappa), ylab = expression(beta),
xlim = NULL, ylim = NULL,
main ="Familia de CO", add = FALSE, verbose = getOption("verbose"))
# Agrega opciones de graficas de bajo nivel
grid(10, 10, lwd = 0)
text(c(0.8,1.1,1.4,1.7,2), c(Beta.X(k=0.8,n=15), Beta.X(k=1.1,n=10),
  Beta.X(k=1.4,n=7),Beta.X(k=1.7,n=5),Beta.X(k=2,n=3)),
```

paste("n=", c(15,10,7,5,3), sep="", collapse=NULL), cex=1.1)

Figura 78. **Curva CO, a diferentes valores de *n***



Fuente: elaboración propia, empleando RGui.

La figura 78 muestra la tendencia a reducirse el riesgo β de no detectar un corrimiento de la media a medida que crece el tamaño de la muestra a niveles iguales de corrimiento, identificado por factor κ o "kappa" desviaciones estándar de la media μ original. Por ejemplo, a $\kappa = 1$, con una muestra de n=3, el riesgo β es aproximadamente del 89,8 %, mientras que con una muestra estándar de n=5, el riesgo es aproximadamente del 77,8 %; si se incrementa el tamaño de la muestra a n=15, el riesgo disminuye drásticamente a 19,1 %.

No obstante, el razonamiento matemático puede ser tentador a

incrementar el tamaño de la muestra para evitar la “no detección” de corrimientos, sin embargo hay que poner en la balanza la factibilidad económica de muestras mayores, ya que el diseño del muestreo debe contemplar posiblemente pruebas destructivas y en efecto, se invierte más tiempo y personal involucrado en la inspección. Como se mencionó, β representa la probabilidad de no detectar un corrimiento en la siguiente muestra, mientras que $1-\beta$ es la probabilidad de detectar el corrimiento; esto se calcula utilizando la misma función *Beta.X*( ) de la siguiente forma:

```
> 1-Beta.X(k=1,n=15)$beta
[1] 0.8086639
> 1-Beta.X(k=1,n=5)$beta
[1] 0.222454
```

Es decir que el 80,9 % de probabilidad de detectar el corrimiento se calcula como la diferencia entre 1 y β; mientras que para un tamaño de muestra de n=5, la probabilidad de detectar un corrimiento de $\kappa = 1$ desviaciones estándar es del 22,3 %.

Observar la terminación “$beta”, para extraer el valor β de los resultados que la función *Beta.X*( ) devuelve, ya que también calcula el ancho medio de la corrida o *Average Run Length* en inglés, cuyas siglas ARL se suelen utilizar para calcular el número de muestras tomadas antes de detectar un corrimiento en la media:

$$ARL = \frac{1}{1-\beta}$$

Dado que: $1-\beta = \alpha$

$$\because \quad ARL = \frac{1}{\alpha}$$

Utilizando los cálculos previos cuando (n=5 y $\kappa = 1$):

```
> alfa<-1-Beta.X(k=1,n=5)$beta; alfa
 [1] 0.2225
ARL<-1/alfa;ARL
[1] 4.494382
```

Quiere decir que se estima que se deben tomar un poco más de cinco muestras para detectar el corrimiento $\kappa = 1\sigma$; mientras que con una muestra de n=15, el resultado es que se necesita un poco más de una para detectar el corrimiento.

```
>ARL<-1/alfa; ARL
[1] 1.236552
```

La función *Beta.X*( ) devuelve ambos valores, β y ARL, sin embargo desarrolla gráficas para ambos indicadores.

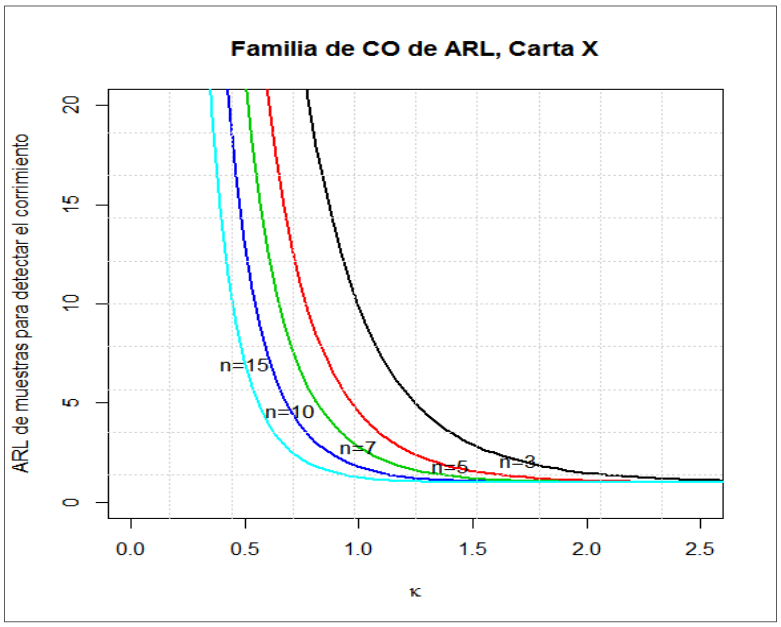
```
> Beta.X(k=1,n=5)
$beta
[1] 0.777546
$ARL
[1] 4.495312
```

Con un tamaño de muestra igual a cinco y esperando un factor de corrimiento igual a 1, el riesgo de no detectar dicho corrimiento es igual a 77,7 %, mientras que se necesitan al menos 4,49 corridas para detectarlo.

La siguiente función fue diseñada para efectos de cálculo de la figura 79, para distintos valores de “n”. Trazando una línea vertical imaginaria en “k” es igual a 1, se puede comparar que el valor de ARL crece a medida que el tamaño de la muestra es menor.

```
ARL.X<-function(k,n,L=3){
  phi1 <- pnorm(c(L-k*sqrt(n)), mean=0, sd=1, lower.tail=TRUE)
  phi2 <- pnorm(c(-L-k*sqrt(n)), mean=0, sd=1, lower.tail=TRUE)
  return(round(1/(1-(phi1-phi2)),2))
  }
```

Figura 79. **Curva CO, ARL para la carta $\overline{X}$**



Fuente: elaboración propia, empleando RGui.

Los vectores para cada valor de “n” se agregan en forma de matriz, tal y como se hizo con la figura 78.En conclusión, el indicador ARL es inversamente proporcional al factor de corrimiento, si el corrimiento es grande, se necesitan

menos muestras para detectarlo.

Si el tamaño de la muestra es grande, un mismo factor de corrimiento se detecta en menor número de muestras.

3.2.5. Relación NCA y NCL (productor – consumidor)

La gráfica de la curva CO observada, corresponde a una carta X, la cual es una de muchas aplicaciones y variantes en control estadístico de la calidad, ya que también se le vincula a los planes de muestreo de aceptación, relacionando el riesgo del consumidor y el del productor. Por un lado, existe un riesgo para el productor de que sea rechazado el lote completo cuando es aceptable. Mientras que hay un riesgo inherente al consumidor derivado del plan específico de muestreo, de aceptar lotes que son rechazables. Esta relación se adapta funcionalmente al error tipo I y tipo II, respectivamente.

La relación entre ambos riesgos tiene su conexión a partir del diseño de un plan de muestreo de aceptación, tema que abrirá el instrumental técnico y de análisis para el contenido del capítulo 4, respecto de los planes de muestreos de aceptación. Se dice que "α" es el riesgo del productor para que un lote “aceptable” sea rechazado, por lo tanto la probabilidad de aceptación de un lote “aceptable” es $Pa = 1 - \alpha$.En el caso del consumidor, existe el riesgo de aceptar lotes “rechazables” identificado por β. En ambos casos, el riesgo se asocia a la proporción de los no conformes, al tamaño de la muestra, al número de aceptación c, entre otros.

El nivel de calidad aceptable (NCA) o *aceptancing quality level* se define como el porcentaje máximo de los “no aceptables”, en tal sentido, la probabilidad de aceptar lotes con un NCA determinado es $Pa = 1 - \alpha$. De modo

que el productor desea disminuir al máximo el riesgo α de rechazo. En consecuencia busca trabajar con NCA muy bajos, menores a 1 % para que la probabilidad de aceptación sea bastante alta. Mientras que el consumidor acepta o rechaza lotes, considerando un nivel de calidad límite (NCL) o *limiting quality level*, el cual es el nivel de calidad que se considera como no satisfactorio. Tanto el NCA como el NCL son un porcentaje de los artículos no conformes del lote y por lo tanto tienen como función juzgar un lote para inspección.

Para ponerlo en perspectiva, la curva CO funciona como instrumento que despliega el poder discriminatorio del plan de muestreo. Por ejemplo, en la figura 80, cabe destacar que en este tipo de curvas, no hay área bajo la curva para identificar probabilidades acumuladas, mientras que trazando una línea vertical desde la curva hacia el eje "x" o *p,* en este caso se trata la probabilidad de aceptar un lote, mientras que una línea desde el eje "x" hacia la curva, es la probabilidad de rechazo. Cada configuración de tamaño específico de muestra y número máximo de rechazo *c* es una curva diferente.

Mientras que el tamaño del lote *N*, es lo suficientemente grande, tal como $N \geq 10n$; la distribución probabilística de Pa se ajusta a una distribución binomial, por lo que se calcula como:

$$Pa = P\{d \leq c\} = \sum_{d=0}^{c} \frac{n!}{d!\,(n-d)!} p^d (1-p)^{n-d}$$

Donde:

- c es el número máximo de artículos defectuosos en plan de muestreo.
- d es el número de artículos defectuosos en la muestra aleatoria, mientras

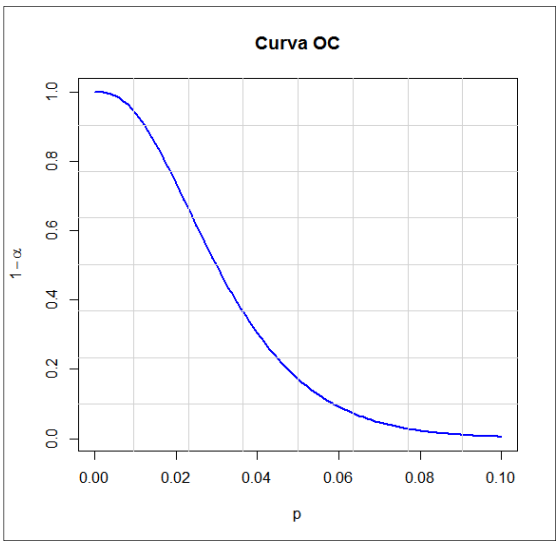
que la probabilidad asociada un número d en específico es $\mathrm{Pa} = \mathrm{P}\{\mathrm{defectuosos}\} = \frac{\mathrm{n!}}{\mathrm{d!(n-d)!}}\mathrm{p}(1-\mathrm{p})^{\mathrm{n-d}}$, caso contrario es una probabilidad acumulada.

- n es el tamaño de la muestra aleatoria.

- p es la proporción o porcentaje de los no conformes, que bajo la visión del productor y consumidor es NCA y NCL, respectivamente.

Ejemplo: considerar un plan simple de aceptación con una muestra aleatoria de 120 artículos de un lote donde $N \gg n$ y el número d máximo de artículos defectuosos en la muestra debe de ser de 4, en tal sentido, se desea conocer la probabilidad de rechazar el lote, dado que el producto ha fijado un NCA = 0,75 %.

Figura 80. **Curva característica de operación**



Fuente: elaboración propia, empleando RGui

```
> sum(dbinom(0:4,120,0.0075))
[1] 0.9977837
pbinom(0:4,120,0.0075)
[1] 0.4051929 0.7726223 0.9378267 0.9869302 0.9977837
> 1-sum(dbinom(0:4,120,0.0075))
[1] 0.002216278
> 1-pbinom(0:4,120,0.0075)
[1] 0.594807076 0.227377725 0.062173344 0.013069775 0.002216278
```

En conclusión, la probabilidad de aceptación del lote bajo un plan simple es aproximadamente el 99,8 % con un NCA del 0,75 %, mientras que riesgo α de rechazar es del 0,2 %.

Considerando ahora la posición del consumidor con un NCL del 4 %, se tiene el siguiente resultado:

```
>cumsum(dbinom(0:4,120,0.04))
[1] 0.007456722 0.044740333 0.137172619 0.288658865 0.473282728
> pbinom(0:4,120,0.04)
[1] 0.007456722 0.044740333 0.137172619 0.288658865 0.473282728
```

Es decir, hasta un 47,3% tiene de probabilidad de ser aceptado, considerando un NCL del 4%.

Notar el uso indistinto de la función *dbinom*( ) como una suma acumulada o la función *pbinom*( ) para calcular los mismos resultados.

Por otro lado, es necesario notar que dada una distribución binomial corresponde a lo que se conoce como una curva tipo B, donde $N \gg n$, que quiere decir que N es mucho mayor a n, de lo contrario se trata de

una curva característica de operación tipo A, donde la distribución asociada es la hipergeométrica donde la probabilidad de aceptación está dada por la ecuación especificada en forma de combinaciones por los elementos entre llaves:

$$P(a) = \frac{\begin{bmatrix} N-m \\ n-d \end{bmatrix}\begin{bmatrix} m \\ d \end{bmatrix}}{\begin{bmatrix} N \\ n \end{bmatrix}}$$

Pero a medida que crece N, las curvas A y B son muy similares, por ello, en los enunciados se asume que N es una población bastante grande. Sin embargo, estos ejemplos son considerando relaciones independientes.

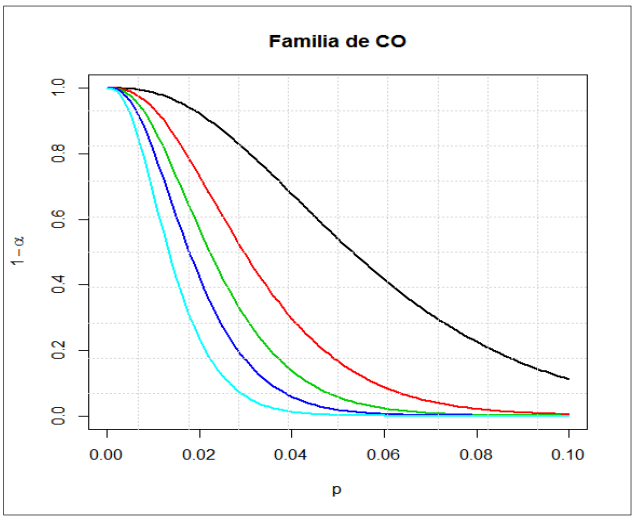
En la práctica, los niveles de calidad son consensuados, incluso el riesgo del productor es fijado por el consumidor, para calcular el tamaño de la muestra y el número c de aceptación.

En el capítulo siguiente se aborda esa situación, en la que se busca el tamaño de la muestra y el número de aceptación, dados un NCA y NCL; en el presente se quiere demostrar únicamente la probabilidad de aceptar un lote de productos cuando ya está previsto un nivel de calidad. Es decir, diseñar una curva OC específica para *n* y *c* conocidos. Ahora es necesario combinar varias curvas específicas, primero a distintos tamaños de muestra “n”, luego a distintos número de rechazo. El *script* necesario para la anterior gráfica es el siguiente:

```
# Aplicar la función pbinom a distintos valores de n
# Dado c=2
n50<-pbinom(2,50,seq(0,0.1,by=0.001))
n90<-pbinom(2,90,seq(0,0.1,by=0.001))
n120<-pbinom(2,120,seq(0,0.1,by=0.001))
```

```
n150<-pbinom(2,150,seq(0,0.1,by=0.001))
n200<-pbinom(2,200,seq(0,0.1,by=0.001))
# Fijar en una matriz la secuencia de
# los valores de los "no conformes"
p.mat<-matrix(rep(seq(0,0.1,by=0.001),5),ncol=5,byrow=FALSE)
# Integrar los resultados en una matriz
OC.mat<-cbind(n50,n90,n120,n150,n200)
# Plotear ambas matrices
matplot(p.mat, OC.mat, type = "l", lty = 1, lwd = 2, lend = par("lend"),
    pch = NULL,
    col = 1:6, cex = 2, bg = NA,
    xlab = "p", ylab = expression(1- alpha),
xlim = NULL, ylim = NULL,
main ="Familia de CO", add = FALSE, verbose = getOption("verbose"))
# Agrega opciones de graficas de bajo nivel
grid(10, 10, lwd = 0)
```

Figura 81. **Familias de curvas CO, con muestra variable**

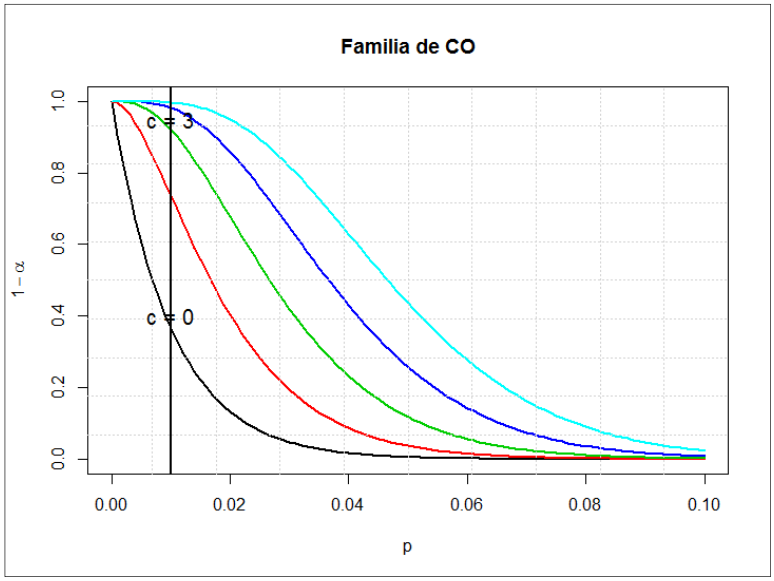


Fuente: elaboración propia, empleando RGui.

Las curvas CO con n grande son más cercanas al origen, quiere decir que a medida que crece la muestra, la probabilidad de aceptación es menor, lo cual tiende al escenario de una inspección del 100 %, como resultado, dado un nivel de calidad donde se rechace el lote.

A medida que la muestra es pequeña, la probabilidad de aceptar se incrementa considerablemente, pero a su vez, crece el riesgo de aceptar lotes que deberían ser rechazados. Por ejemplo, qué sucede cuando el nivel de calidad es del 0,02 o 2 %; la probabilidad de aceptar el lote cuando n = 200 es del 23,5 %, en contraste, cuando la muestra es n = 90, la probabilidad de ser aceptado es de 73,1 %. Entonces se observa los estratégico y sensible que resulta para un plan de muestreo el tamaño de la muestra. No obstante, cuando lo que se varía es el número de máximo de aceptación o “c”, se visualiza la siguiente gráfica.

Figura 82. **Curvas CO, con número de aceptación variable**



Fuente: elaboración propia, empleando RGui

En el caso “ideal” de c=0, es una trampa visual para la mayoría de planificadores de muestras. Al incrementarse el número de aceptación “c”, la probabilidad de aceptación es mayor; va desde 36,6 % en c=0, hasta un 98,2 % en c = 3.

```
> pbinom(0:4,100,0.01)
[1] 0.3660323 0.7357620 0.9206268 0.9816260 0.9965677
```

A medida que la proporción de los defectuosos “*p*” crece, decrece la probabilidad de ser aceptados.

```
> pbinom(0:4,100,0.03)
[1] 0.04755251 0.19462212 0.41977508 0.64724921 0.81785481
```

Es por ello que también es muy sensible para el diseño del plan de muestreo, la selección del número de aceptación.

En conclusión, la relación NCA – NCL depende del diseño del plan del muestreo de aceptación, ya que las combinaciones posibles originan un menor o mayor riesgo de rechazo de lotes aceptables o aceptación de lotes no aceptables. Sin embargo, los planes de muestreo parten del riesgo a que se está dispuesto asumir; como resultado de ello se obtiene el tamaño de la muestra y el número de aceptación; dicha lógica es la que se desarrolla más adelante, con el auxilio de algunos paquetes de R.

En el siguiente capítulo se aborda con mayor profundidad y por orden de ideas la relación NCA – NCL con el paquete AcceptanceSampling, el cual en particular, calcula directamente el tamaño de la muestra y el número de aceptación mediante la función *find.plan*(PRP,CRP,type=”binomial”).

Se menciona dicha referencia, en función de aproximar y calcular el único par que genera la curva característica de operación que satisface las posiciones y los riesgos que tanto el productor como el consumidor están dispuestos a asumir.

Un ejemplo sencillo para dicha función es que se conoce que un productor desea contar con un 95% de probabilidad de aceptación de sus lotes, esperando que el nivel de calidad aceptable que proporciona sea del 3%. Mientras que el consumidor está dispuesto a soportar un nivel máximo de calidad como no satisfactorio de 4%, por lo que asume un riesgo de aceptar lotes no satisfactorios bajo este nivel de calidad del 10%. Por lo tanto, es necesario encontrar el tamaño de la muestra y el número de aceptación que calcule la curva característica de operación que satisfaga ambas posiciones.

Notar los argumentos de la función *find.plan*( ), que necesita la posición del productor (PRP) y la posición del consumidor (CRP), ambos consistentes en un vector que especifique NCA o NCL y la probabilidad de aceptación 1- α y β, respectivamente; además se opta por la distribución binomial para ajustar la curva tal y como se han calculado las gráficas de las CO.

De tal forma que se especifica de la siguiente forma:

```
> library(AcceptanceSampling)
> find.plan(PRP=c(0.03,0.05), CRP=c(0.04,0.1), type="binomial")
$n
[1] 57
$c
[1] 0
$r
[1]1
```

El resultado anterior dice que se necesita de una muestra de 57 artículos, en la que no debe de existir ninguno no satisfactorio para aceptar el lote; dicha condición es la relación NCA - NCL en la misma curva característica de operación4.

4. MUESTREOS DE ACEPTACIÓN

4.1. Muestreo de aceptación

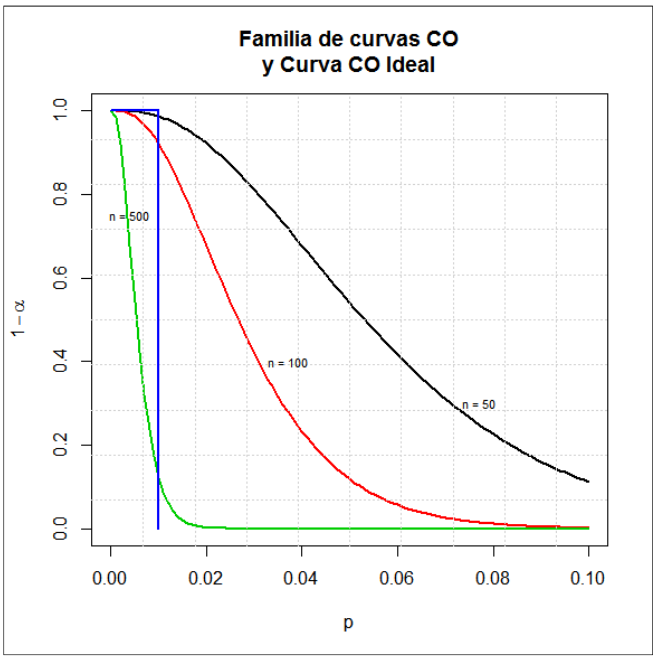
En el capítulo se utilizó la relación que implica una curva característica de operación (CO), en la cual se observó el método para calcular la probabilidad de aceptación de un lote en particular mediante el análisis de una muestra, dadas las circunstancias como el NCA y NCL o niveles de calidad que son de interés para el productor y consumidor, respectivamente.

En tal sentido, una curva o una familia de curvas CO tiene distinto comportamiento a diferentes tamaños de muestra, ya que suelen acercarse más a la curva "ideal", tal y como se observa en la figura 83. En el caso de la curva ideal, se aprecia que para un nivel de calidad dado, por ejemplo 0,01 o 1 %, el producto desearía que siempre le fueran aceptados los lotes cuya calidad fuera mejor que el NCA y que acepten los lotes con un nivel de calidad, pero al NCA el 0 % de las veces[19]. En dicha gráfica se aprecia con color azul, pero dicho comportamiento, únicamente se lograría con el 100 % de inspección.

Mientras que una curva típica, tal como sucede cuando n = 50,100 o 500, entre mayor sea la muestra, más se acerca a la curva ideal. El siguiente *script*, define la anterior gráfica.

[19] KRAJEWSKI, Lee J. *Administración de operaciones: estrategia y análisis*. p. 288.

Figura 83. **Curva CO ideal y diferentes tamaños de muestra**



Fuente: elaboración propia, empleando RGui

```
# Fijar en una matriz la secuencia de
# los valores de los "no conformes"
p.mat<-matrix(rep(seq(0,0.1,by=0.001),3),ncol=3,byrow=FALSE)
# Integrar los resultados en una matriz
n50<-pbinom(2,50,seq(0,0.1,by=0.001))
n100<-pbinom(2,100,seq(0,0.1,by=0.001))
n200<-pbinom(2,500,seq(0,0.1,by=0.001)
OC.mat<-cbind(n50,n100,n500)
# Plotear ambas matrices
matplot(p.mat, OC.mat, type = "l", lty = 1, lwd = 2, lend = par("lend"),
      pch = NULL,
      col = 1:6, cex = 2, bg = NA,
      xlab = "p", ylab = expression(1- alpha),
```

```
xlim = NULL, ylim = NULL,
main ="Familia de curvas CO
  y Curva CO Ideal", add = FALSE, verbose = getOption("verbose"))
# Agrega opciones de graficas de bajo nivel
grid(10, 10, lwd = 0)
# Líneas que hacen la curva ideal
segments(x0=0.01,y0=0,x1=0.01,y1=1,col="blue",lwd=2)
segments(x0=0,y0=1,x1=0.01,y1=1,col="blue",lwd=2)
# Texto para indicar el tamaño de la muestra
text(0.004,0.75,"n = 500",cex=.7)
text(0.037,0.4,"n = 100",cex=.7)
text(0.077,0.3,"n = 100",cex=.7)
```

Anteriormente se ha discutido que en una curva típica con una muestra grande (por ejemplo n = 500), disminuye considerablemente la probabilidad de aceptación para los NCA muy grandes, entonces señala la capacidad de discriminación del plan de muestreo. En contraste, muestras más reducidas, la probabilidad de aceptación es mayor a un mismo NCA; no obstante, se incrementa el riesgo de aceptar lotes rechazables.

En tal sentido, se puede apreciar que un plan de muestreo de aceptación o la definición del mismo, es un instrumento que puede generar cierto tipo de controversias entre productor y consumidor.

En consecuencia, primero son negociados y fijados los niveles de calidad NCA y NCL y luego calculado el tamaño de la muestra, posteriormente analizado el conjunto de riesgos del productos y los riesgos del consumidor $\alpha\ y\ \beta$, respectivamente.

El hecho de calcular el tamaño de la muestra y el número de aceptación, es un paso más enfocado al extenso de tema de los muestreos de aceptación. Un muestreo de aceptación es un proceso por el cual se calcula una muestra que se levanta entre un lote de producción, con el objetivo de aceptar o rechazarlo; no juzga en ningún momento su calidad, por lo que se limita a dictaminar.

La muestra, es una porción finita y determinada de la población; en este caso se habla del lote a dictaminar. Las características de la muestra sirven para estimar los verdaderos parámetros del lote en cuestión. Por eso, a través de la muestra se puede aceptar o rechazar un lote completo.

En los muestreos de aceptación intervienen los niveles de calidad que tienen estrecha relación con proporciones sugeridas o fijadas sobre los artículos “no conformes”.

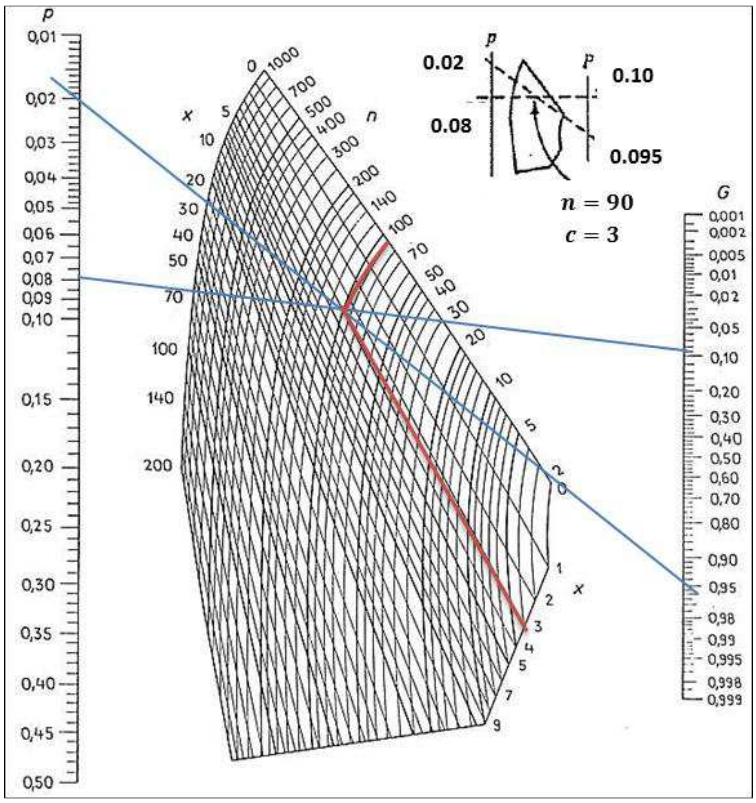
Existen varias técnicas y/o estándares para diseñar el plan del muestreo de aceptación. Los métodos más complejos y conocidos fueron producto de estándares ampliamente utilizados como los MIL, STD y Dodge Romig. Sin embargo estos fueron en su oportunidad producto de un diseño riguroso de experimentos.

Como resultado de ello, se cuenta hoy en día con una secuencia de pasos para calcular el tamaño de la muestra; asimismo, también hay paquetes en R para simular la misma secuencia de pasos. La relación de la curva característica de operación muestra la probabilidad de ser aceptado determinado lote de productos, debido al plan de muestreo o al diseño del mismo. La figura 84 es una técnica para determinar el tamaño “n” de la muestra y el número “c” de aceptación basado en un nomograma. Esta herramienta se

desarrolló resolviendo el siguiente sistema de ecuaciones simultáneas no lineales.

$$1-\alpha=\sum_{d=0}^{c}\frac{n!}{d!(n-d)!}p_1^d(1-p_1)^{n-d} \qquad \text{y} \qquad \beta=\sum_{d=0}^{c}\frac{n!}{d!(n-d)!}p_2^d(1-p_2)^{n-d}$$

Figura 84. **Nomograma de Larson**



Fuente: MONTOGOMERY, Douglas. *Introduction to statistical quality control*. p. 658.

Donde α, es el riesgo del productor,β es el riesgo del consumidor,n es el tamaño de la muestra, c es el número máximo aceptación,d es número de los no conformes donde $(d \leq c)$; p_1 y p_2 son los puntos correspondientes a los niveles de calidad, NCA y NCL; respectivamente, al productor y consumidor.

En la figura 84 se pueden observar dos líneas que cruzan de un punto de la escala izquierda a la escala del lado derecho, de p_1 a $(1-\alpha)$ y de p_2 a β.

Suponiendo que $p_1 = 0{,}02$ y $(1-\alpha) = 0{,}95$ y $p_2 = 0{,}08$ y $\beta = 0{,}1$, es necesario cruzar ambas líneas y el punto de intersección es la solución al sistema de ecuaciones; en el ejemplo $n = 90$ y $c = 3$, que corresponden al tamaño de la muestra y número de aceptación, respectivamente, se satisfacen ambos niveles de calidad (NCA y NCL).

Esto es una primera aproximación de un muestreo de aceptación simple, partiendo de la curva característica de operación. En otras palabras, los datos de entrada son los puntos sobre la curva característica de operación, la cual debe de ajustarse a la combinación precisa de los valores de n y c.

4.2. Índices de calidad

En el inciso anterior se ha discutido ya de forma introductoria el tema de los índices de calidad, mencionados como proporciones o niveles de calidad. Es de suma importancia ver a los índices NCA y NCL como un par de complementariedad mutua, de la relación entre los intereses del productor y consumidor, respectivamente. Dichas proporciones son, en efecto un índice, debido a que son de hecho un porcentaje. Debido a la complejidad que reviste un plan de muestreo para satisfacer ambos intereses de forma completa, en consecuencia, se buscan altas probabilidades de aceptar lotes “buenos” y bajas probabilidades de aceptar lotes “malos”, dicho en forma coloquial.

Dentro del diseño del plan de muestreo de aceptación, un paso de carácter estratégico es establecer, no calcular, los índices de calidad, los cuales junto con la probabilidad de aceptar los lotes “buenos”, se traducen en el

tamaño de la muestra y el número de aceptación del lote. Como resultado de lo anterior, la relación de intereses productor – consumidor se mantiene en equilibrio, no obstante, el riesgo en contra de los intereses de cada uno subsiste.

4.2.1. Nivel de calidad aceptable

Desde el capítulo anterior se ha trabajado ya en el concepto del índice de calidad llamado nivel de calidad aceptable en sus siglas NCA o AQL en inglés, el cual es conocido por el nivel de calidad del productor expresado como el porcentaje máximo de unidades que no cumplen con la calidad especificada. El riesgo de que sea rechazado un lote bajo el NCA se expresa por α, mientras que la probabilidad de ser aceptado se nata por "$1-\alpha$". Los valores más habituales para el riesgo α son de 0,1 o 0,05 y son un nivel de calidad de referencia, por lo que el productor tratará de trabajar por debajo del NCA para minimizar el riesgo de que un lote sea rechazado.

4.2.2. Nivel de calidad límite

Todos los lotes que pasan este índice deben ser rechazados, debido a que a este nivel de calidad los lotes se consideran como no satisfactorios. En la práctica y simulación con para los planes *Dodge Romig* se le conoce también como LPTD (*lot tolerance percent defective*) o porcentaje defectivo tolerado del lote.

Al ser un nivel de calidad “no satisfactorio” la probabilidad de aceptar un lote debe ser muy baja, en el orden del 0,05 al 0,10; esta probabilidad es designada por β “beta” y cuantifica el riesgo del consumidor al existir de hecho una probabilidad “no nula” de aceptar un lote “no satisfactorio”.

4.2.3. Calidad de salida promedio

Conocido en inglés como AOQ (*Average Outgoing Quality*), el índice de calidad de salida promedio(CSP) es la calidad promedio de varios lotes después de un proceso de inspección. En otras palabras es el registro posterior de haber implementado un plan de muestreo por aceptación. Mientras que el límite superior es el límite de calidad de salida promedio(LCSP) que representa el peor posible promedio de calidad que resultará de rectificar el plan de inspección[20]. Mientras que el LCSP o AOQL, viene dado por punto máximo del vector de valores AOQ.

```
> AOQL<-max(AOQ); AOQL
[1] 0.01538161
```

Es decir que en promedio, no se tendrá un nivel de calidad pero al 1,54%; sin embargo, por ser promedio, no implica que no existan puntos peores o mayores a 1,54% de defectuosos; este índice se presenta en una cadena larga de lotes, considerando los valores de una muestra de 89 y un número de aceptación de 2(ver la figura 85).La función utilizada para calcular el AOQ se deriva de:

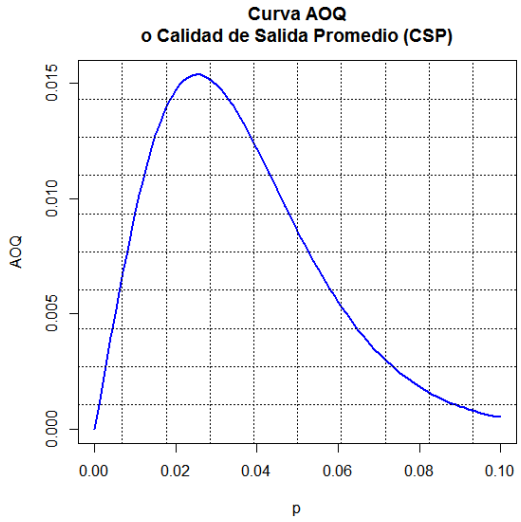
$$AOQ = \frac{P_a p(N-n)}{N}$$

Pero debido a que los lotes llegan a ser mucho mayor que la muestra, la función se calcula como:

$$AOQ = P_a$$

[20]MONTOGOMERY, Douglas. *Introduction to statistical quality control*. p. 661.

Figura 85. **Curva AOQ – CSP**




```
# AOQ O CSP
# Dado c=2, n=89
p<-seq(0,0.1,by=0.001)
  Pa<-pbinom(2,89,p)
  AOQ<-p*Pa
#Plotear AOQ vrs p
plot(p, AOQ, type = "l", lty = 1, lwd = 2,
  lend = par("lend"),
  pch = NULL,
  col = "blue", cex = 2, bg = NA,
  xlab = "p", ylab = "AOQ",
  xlim = NULL, ylim = NULL,
main ="Curva AOQ
  o calidad de salida promedio (CSP)")
# Agrega opciones de graficas de bajo
nivel grid(10, 10, lwd = 0)
```

Fuente: elaboración propia, empleando RGui

4.2.4. Inspección total promedio

Es la cantidad total de inspección que requiere el plan, considerando que si un lote sale rechazado se inspecciona el 100 %. Es decir a porcentajes altos de defectuosos la muestra se incrementa. Es conocida en inglés como *Average Total Inspection* o ATI o en español como ITP. La función para calcular el ATI es:

$$ATI = n + (1 - P_a)(N - n)$$

Una curva se construye a varios valores de la proporciones p de defectuosos, dado por ejemplo un lote $N = 1\ 000$ unidades, una muestra n de

89 y un número c de aceptación igual a 2.

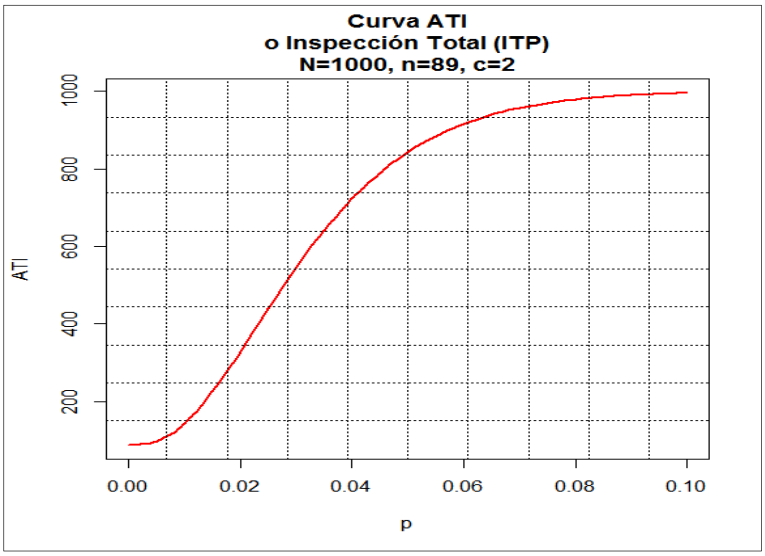
```
# ATI o ITP
# Dado c=2, n=89, N= 1000
p<-seq(0,0.1,by=0.001)
N<-1000
n<-89
c<-2
Pa<-pbinom(c,n,p)
ATI<-n+(1-Pa)*(N-n)
#Plotear ATI vrs p
plot(p, ATI, type = "l", lty = 1, lwd = 2,
  lend = par("lend"),
  pch = NULL,
  col = "red", cex = 2, bg = NA,
  xlab = "p", ylab = "ATI",
  xlim = NULL, ylim = NULL,
  main ="Curva ATI
  o Inspección Total (ITP)
  N=1000, n=89, c=2")
# Agrega opciones de graficas de bajo nivel
grid(10, 10, lwd = 0,col="black")
```

Puede apreciarse en la figura 86que, a mayores proporciones o porcentajes de defectuosos, el tamaño de la muestra se incrementa; partiendo de que n= 89, se incrementa la necesidad de inspeccionar una muestra mayor con el propósito de evitar aceptar lotes defectuosos. Por ejemplo, para una proporción de defectuosos de 0,04, se necesita una muestra en promedio de 723 unidades para evitar aceptar un solo lote defectuoso.

```
> ATI<-n+(1-pbinom(c,n,0.04))*(N-n); ATI
```

[1] 722.9117

Figura 86. **Curva de inspección total promedio o ATI**



Fuente: elaboración propia, empleando RGui

4.2.5. Curvas de operación

Las curvas de operación hasta acá revisadas, corresponden a la relación obtenida de un plan de muestreo de aceptación simple. En la práctica existen diversas técnicas de muestreo de aceptación, en las que se contempla el muestreo doble o múltiple.

La extensión del plan de muestreo a doble o múltiple, responde a situaciones de incertidumbre en el primer plan de muestreo de aceptación. Por lo que un segundo o subsecuentes son necesarios para dictaminar con la aceptación o rechazo de un lote, recordando que no evalúa la calidad del mismo.

Los parámetros necesarios para dictaminar un lote son:

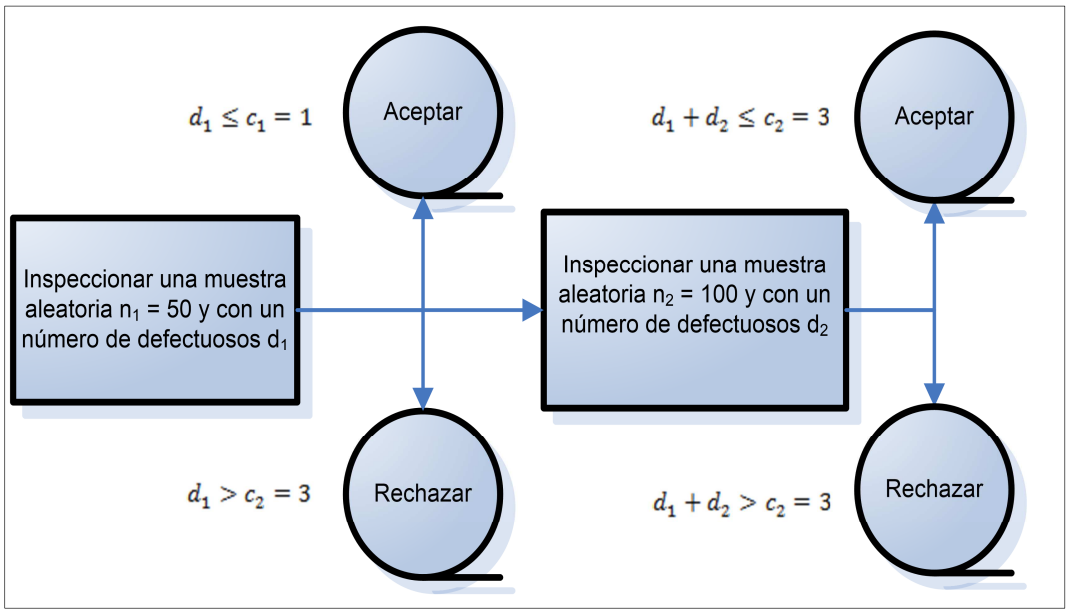
- n_1 Tamaño de la primera muestra
- c_1 Número de aceptación
- n_2 Tamaño de la segunda muestra
- c_2 Número de aceptación en para ambas muestras
- d_1 Número de defectuosos en la primera muestra
- d_2 Número de defectuosos en la segunda muestra

Es decir, que si el número de defectuosos d_1 de la primera muestra es menor o igual al número de aceptación c_1, se acepta; si es mayor que c_2, se rechaza; pero si es menor o igual a c_2, se recurre a un segundo muestreo de tamaño n_2, donde si$d_1 + d_2 \leq c_2$, se acepta, pero si es mayor, se rechaza.

El muestreo de aceptación doble implica a veces la reducción de los costos de inspección, pero en algunas circunstancias puede que se requiera de una inspección total, inclusive contando el mismo grado de protección que ofrece un muestreo de aceptación simple. No obstante, los muestreos simples y dobles ofrecen los mismos riesgos de aceptar o rechazar lotes de una calidad específica.

Esto implica que la curva característica de operación combinada de la primera muestra y la segunda en un muestreo doble, tiende a ser idéntica a la que presenta un muestreo de aceptación simple.

Figura 87. **Esquema muestreo y tabla de aceptación doble**



Fuente: MONTOGOMERY, Douglas. *Introduction to statistical quality control*. p. 663.

De esa forma, es posible simular la curva característica de operación; conociendo las fórmulas de la tabla XXV de las posibles combinaciones de los números d de defectuosos, se resuelve con el *script* posterior a la misma.

Tabla XXV. **Muestreo de aceptación doble**

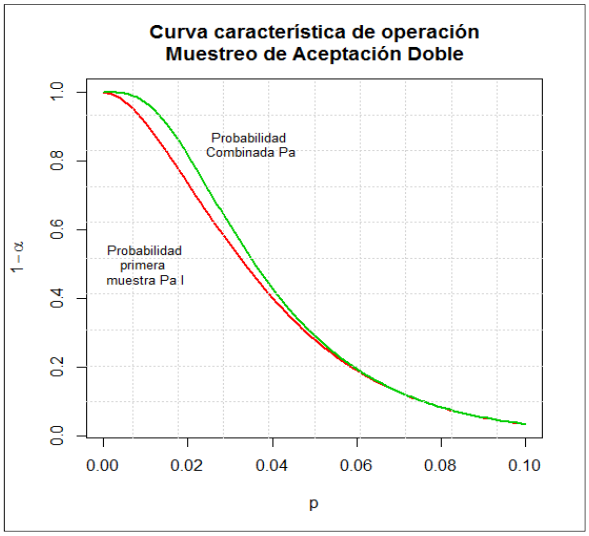
Muestra	**Valores**	**Fórmula**
I $c_1 = 1, n_1 = 50$	d_1 =0 o 1	$P_a^I = P\{d_1 \leq c_1\} = \sum_{d=0}^{c_1} \frac{n_1!}{d_1!\,(n_1 - d_1)!} p^{d_1}(1-p)^{n_1 - d_1}$
II $c_2 = 3, n_2 = 100$	d_1 =2 y d_2 =0 o 1	$P\{d_1 = 2\} = \frac{n_1!}{d_1!\,(n_1 - d_1)!} p^{d_1}(1-p)^{n_1 - d_1}$ $P\{d_2 \leq 1\} = \sum_{d=0}^{d_2} \frac{n_1!}{d_2!\,(n_2 - d_2)!} p^{d_2}(1-p)^{n_2 - d_2}$
	d_1 =3 y d_2 =0	$P\{d_1 = 3\} = \frac{n_1!}{d_1!\,(n_1 - d_1)!} p^{d_1}(1-p)^{n_1 - d_1}$ $P\{d_2 = 0\} = \frac{n_1!}{d_2!\,(n_2 - d_2)!} p^{d_2}(1-p)^{n_2 - d_2}$
		$P_a^{II} = P\{d_1 = 2\} \times P\{d_2 \leq 1\} + P\{d_1 = 3\} \times P\{d_2 = 0\}$
P_a		$P_a = P_a^I + P_a^{II}$

Fuente: elaboración propia.

```
# Fijar en una matriz la secuencia de los valores de los "no conformes"
p.mat<-matrix(rep(seq(0,0.1,by=0.001),2),
  ncol=2,byrow=FALSE)
# Integrar los resultados en una matriz
prob.x<-seq(0,0.1,by=0.001)
Pa1<-pbinom(1,50,prob.x)
Pa2<-dbinom(2,50,prob.x)*pbinom(1,100,prob.x)+
  dbinom(3,50,prob.x)*dbinom(0,100,prob.x)
Pa<- Pa1 + Pa2
OC.mat<-cbind(Pa1,Pa)
```

```
# Plotear ambas matrices
matplot(p.mat, OC.mat, type = "l", lty = 1, lwd = 2,
  lend = par("lend"),
  pch = NULL, col = 2:3, cex = 2, bg = NA,
  xlab = "p", ylab = expression(1- alpha),
xlim = NULL, ylim = NULL,
main ="Curva característica de operación
  Muestreo de Aceptación Doble", add = FALSE,
  verbose = getOption("verbose"))
# Agrega opciones de graficas de bajo nivel
grid(10, 10, lwd = 0)
  text(0.035,0.85,"Probabilidad
Combinada Pa",cex=.8)
  text(0.01,0.5,"Probabilidad
primera
muestra Pa I",cex=.8)
```

Figura 88. **Curva CO, muestreo de aceptación doble**



Fuente: elaboración propia, empleando RGui

La figura 88 muestra la proximidad entre ambas curvas, es decir la aproximación de las probabilidades de aceptación de un determinado lote. En tal sentido, estadísticamente hablando, esta relación apoya la idea de la factibilidad de emplear los muestreos dobles o múltiples, ya que implica una reducción del costo económico por muestras mayores en muestreos simples.

4.2.6. Paquete AcceptanceSampling

El paquete *AcceptanceSampling* es un compendio de herramientas para evaluar planes de muestro de aceptación. Dichas herramientas se pueden ejecutar con los métodos de R mayor a la versión 2.4.0. La metodología del paquete se fundamenta en aspectos con más contenido probabilístico, ya que se basa en la distribución hipergeométrica, binomial (curvas tipo A y B, respectivamente); además, está definida la distribución de Poisson, para los planes de muestreo de aceptación por atributos.

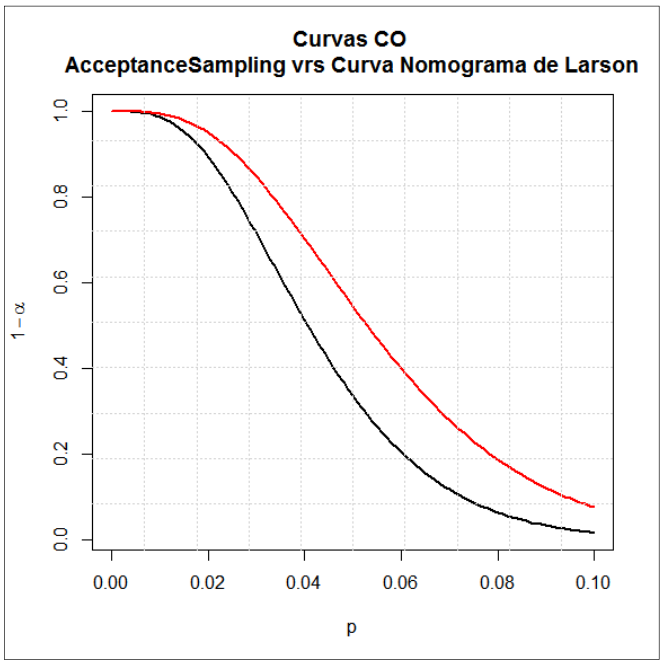
En el caso de una distribución binomial, se puede calcular el plan de muestreo mediante la función *find.plan*( ), la cual utiliza como argumentos el NCA y la probabilidad de aceptación del lote $(1-\alpha)$, descritos en forma vectorial con el argumento PRP o *Producer Risk Point*; por otro lado, también en forma vectorial descrito como el CRP o *Consumer Risk Point*, se detalla el NCL y la probabilidad de rechazo de un lote aceptable "β".

Por ejemplo regresando al ejemplo sobre el nomograma de *Larson*, en el que PRP = ($NCA = p_1 = 0{,}02, (1-\alpha) = 0{,}95$) y por otro lado, se definió que CRP = ($\beta = 0{,}1$, NCL= $p_2 = 0{,}08$).Asumiendo que $N \gg n$, la función se define de la siguiente forma:

```
> find.plan(c(0.02,0.95),c(0.1,0.08),type="binomial")
$n
[1] 69
$c
[1] 3
$r
[1] 4
```

Por lo tanto, el plan de muestreo de aceptación calcula una muestra de n=69, con un número de aceptación c=3 y un número de rechazo del lote r=4. En contraste con el resultado del nomograma, el tamaño de la muestra fue de 90 artículos;(ver la comparación de los resultados en la siguiente gráfica seguido del *script* para la misma.

Figura 89. **Comparación curvas CO, de dos métodos**



Fuente: elaboración propia, empleando RGui

```
# Fijar en una matriz la secuencia de
# los valores de los "no conformes"
p.mat<-matrix(rep(seq(0,0.1,by=0.001),2),ncol=2,byrow=FALSE)
# Integrar los resultados en una matriz
n90<-pbinom(3,90,seq(0,0.1,by=0.001))
n69<-pbinom(3,69,seq(0,0.1,by=0.001))
OC.mat<-cbind(n90,n69)
# Plotear ambas matrices
matplot(p.mat, OC.mat, type = "l", lty = 1, lwd = 2, lend = par("lend"),
    pch = NULL,
    col = 1:6, cex = 2, bg = NA,
    xlab = "p", ylab = expression(1- alpha),
xlim = NULL, ylim = NULL,
main ="Curvas CO
AcceptanceSampling vrs Curva Nomograma de Larson", add = FALSE, verbose
= getOption("verbose"))
grid(10, 10, lwd = 0)
```

Los resultados resultan ser importantes para efectos de análisis de cobertura de riesgos debido al plan. La metodología del nomograma de Larson resulta ser una buena aproximación (curva de color negro), mientras que la utilización del paquete *AcceptanceSampling*, arroja desde luego una muestra de tamaño menor que implica que a niveles de calidad iguales, la probabilidad de aceptación es mayor. El efecto de una muestra mayor, mientras las demás variables son constantes, es que se acerca a la curva ideal; por ello se sabe que la curva con una muestra mayor está más hacia el origen.

La diferencia de resultados entre ambos métodos se relaciona con la utilización de métodos de cómputo más precisos con el paquete respecto de la aproximación del método gráfico del nomograma de Larson.

Otra forma de ver el problema de selección del plan es cuando se asume un tamaño de muestra y los vectores PRP y el CRP. De esa forma con la función *assess*( ), se calcula la probabilidad de aceptación con los criterios preespecificados con los que deben de ajustarse de forma estrecha, tanto para el PRP como para el CRP. En el caso del PRP, se asume una probabilidad de aceptación de 0,95, mientras que el resultado calculado por la función es de 0,95031105, lo que implica que los resultados muestren el aviso: *"Plan CAN meet desired risk point(s*):", que quiere decir que el plan puede reunir los puntos de riesgos propuestos.

```
> assess(OC2c(69,3), PRP=c(0.02, 0.95), CRP=c(0.1, 0.08))
Acceptance Sampling Plan (binomial)

            Sample 1
Sample size(s)      69
Acc. Number(s)       3
Rej. Number(s)       4
Plan CAN meet desired risk point(s):
Quality   RP P(accept) Plan     P(accept)
PRP         0.02          0.95    0.95031105
CRP         0.10          0.08    0.07623423
```

En contraste, el plan obtenido por el nomograma de Larson proporciona una diferencia evidente entre los criterios pre especificados y la verdadera probabilidad de aceptación, por lo que quiere decir que no se ajusta sobre la misma curva característica de operación y se confirma con el aviso *"Plan CANNOT meet desired risk point(s*):" que dice que el plan NO puede reunir los puntos de riesgo propuestos, ya que el PRP se propuso un 0,95; pero se calculó un 0,8933, mientras que el CRP se propuso un 0,08, pero el resultado calculado es de 0,01688. Esto confirma las diferencias técnicas entre ambas

curvas CO de la figura 89.

```
> assess(OC2c(90,3), PRP=c(0.02, 0.95), CRP=c(0.1, 0.08))
Acceptance Sampling Plan (binomial)
            Sample 1
Sample size(s)      90
Acc. Number(s)       3
Rej. Number(s)       4
Plan CANNOT meet desired risk point(s):
        Quality   RP P(accept) Plan P(accept)
PRP        0.02         0.95     0.89325241
CRP        0.10         0.08     0.01688065
```

Por último, la función *f_CO.NCA.NCL*( ) en el paquete Planesmuestraen lagráfica la curva característica de operación, en la que se puede observar el par de resultados que son satisfechos por la única combinación de muestra y número de aceptación. Los argumentos son los siguientes:

Tabla XXVI. **Argumentos función f_CO.NCA.NCL**

Argumento	**Descripción**
NCA	Nivel de calidad aceptable
NCL	Nivel de calidad límite
N	Tamaño de la muestra
C	Número de aceptación

Fuente: elaboración propia.

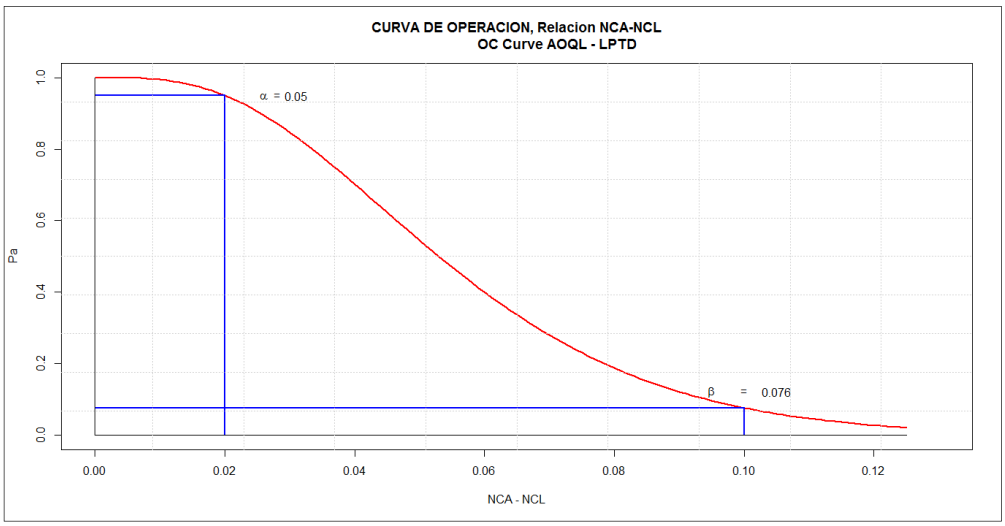
Se toman en consideración los resultados y argumentos empleados en la función *find.plan*( ):

```
> find.plan(c(0.02,0.95),c(0.1,0.08),type="binomial")
$n
[1] 69
$c
[1] 3
$r
[1] 4
```

Se tiene un NCA = 0,02, NCL = 0,1, n= 69, c=3, por lo tanto la curva que representa dicha situación es la siguiente:

```
> f_CO.NCA.NCL(NCA=0.02,NCL=0.1,n=69,c=3)
  c       n       NCA   NCL     beta          alpha
[1,] 3    69      0.02  0.1     0.07623423    0.04968895
```

Figura 90. **Relación NCA – NCL**



Fuente: elaboración propia, empleando RGui

La función calcula ambos porcentajes de riesgo, para el productor el

riesgo α de que sea rechazado el lote es de 5 %, mientras que el riesgo β del consumidor de aceptar un lote rechazable es de 7,6 %, aproximadamente a 8 %; ambas probabilidades aparecen en la figura 90 y confirman en forma inversa lo que la función *find.plan*( ) calcula.

Es preciso mencionar que para que ambos esquemas coincidan, debe calcularse la muestra por medio de la distribución binomial.

4.2.6.1. Las clases y objetos definidos

Los planes dentro del paquete *AcceptanceSampling* están definidos por clases, ya que el muestreo simple es definido como *2-class attribute inspection plans* que solo contempla dos posibilidades a saber: aceptable o no aceptable.

Mientras que la clase de planes dobles corresponde a *3-class attribute inspection*, implica clasificar cada ítem como aceptable, marginal o no aceptable. Por otro lado está definida la clase para los muestreos de aceptación por variables, donde la característica de calidad es mesurable como *inspection by variables plans***.** Las clases de objetos vinculados al paquete son **OC2c**y **OCvar**, para los planes *2-class attribute inspection plans* e *inspection by variables plans* respectivamente[21].

4.2.6.2. Los planes OC2c

Este tipo de plan adquiere variantes por el tipo de distribución probabilística definida. Ya que puede ser binomial, hipergeométrica o de Poisson. Los parámetros están definidos en la siguiente tabla:

[21] KIERMEIER, Andreas. *Visualising and assessing acceptance sampling.* p. 5.

Tabla XXVII. **Paquete AcceptanceSampling, argumentos planes OC2c**

Distribución **Argumento**	**Binomial**	**Hipergeométrica**	**Poisson**
n: vector con los tamaños de las muestras	√	√	√
c: Vector con los números de aceptación	√	√	√
r: Vector con los números de rechazo	√	√	√
type: La distribución en que se basa el plan	√	√	√
pd: vector de la proporción de defectuosos	√	√	√
N: tamaño del lote		√	

Fuente: KIERMEIER, Andreas. *Visualising and assessing acceptance sampling*. p.7.

Respecto de los argumentos de los planes **OCvar,** estos son definidos bajo el supuesto de normalidad, los cuales son los siguientes:

Tabla XXVIII. **Argumentos función planes OCvar**

Argumento	**Descripción**
N	Tamaño de la muestra, vector de tamaño 1
K	La constante de aceptabilidad, vector de tamaño 1

Continuación de la tabla XXVIII

Argumento	**Descripción**
s.type:	Cualquiera de las cadenas de caracteres "*known"* o "*unknown*", confirmando si se conoce la desviación estándar de la población.
pd	La proporción de los defectuosos.

Fuente: KIERMEIER, Andreas. *Visualising and assessing acceptance sampling*. p.8.

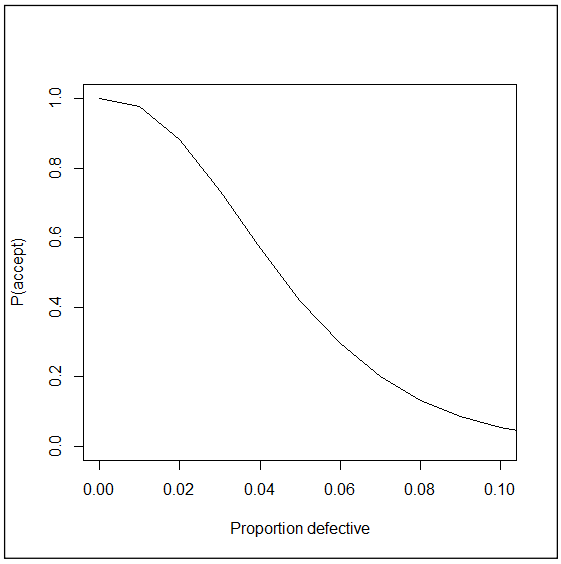
El argumento "pd" puede tratarse de la proporción estandarizada de la diferencia entre el límite superior o inferior o ambos, respecto de la media dividido entre la desviación estándar. Por el momento el análisis se centra en los planes **OC2c**. Para definir un plan **OC2c** tan solo es necesario definir el tamaño "n" de la muestra y el número de aceptación "c" con la función *OC2c*( ); si se omite el tipo de distribución, por defecto, el método que emplea la función es la distribución binomial. Por ejemplo un plan donde el tamaño de la muestra es igual a 60 y el número de aceptación es igual a 2, se define como:

```
> x1.plan<-OC2c(60,2); x1.plan
Acceptance Sampling Plan (binomial)
            Sample 1
Sample size(s)      60
Acc. Number(s)       2
Rej. Number(s)       3
```

Los resultados se pueden asignar a otro objeto y plotearlo de forma directa (figura 91), ya que lleva implícita la distribución binomial y emplea las funciones del núcleo base o de otros paquetes para gráficas. Sin embargo

resulta complejo plotear varias curvas de operación bajo las funciones gráficas descritas.

Figura 91. **Curva OC, definida por *AcceptanceSampling***



Fuente: elaboración propia, con paquete *AcceptanceSampling*.

```
> plot(x1.plan, type = “l”, xlim = c(0,0.1))
```

4.3. *Script* para muestreo de aceptación por atributos

Existen paquetes que simulan el cálculo de planes de muestreo de aceptación por medio de distribuciones de probabilidad discreta, en tanto que en los siguientes párrafos se obtiene el plan por medio de la lectura de las tablas para cada tipo plan.

4.3.1. Sistema Dodge Romig

Este tipo de plan se basa en el supuesto que no importando que un lote sea rechazado, este termina siendo inspeccionado al 100 %; por lo tanto las unidades de esos lotes rechazados, calificados como defectuosos o inconformes, son sustituidos por otros aceptables[22], por ello son planes de inspección de rectificación.

Con el propósito de dar protección de lotes de baja calidad, existen cuatro tablas, dos para tipo de muestreo simple o doble, en función de los índices NCL y del límite de calidad de salida promedio (LCSP, LPTD y AOQL, en inglés, respectivamente).

De acuerdo con las tablas comúnmente disponibles para efectos prácticos y didácticos[23], se diseñó una función que extrae los datos del plan de inspección basados en el LCSP por un lado y el LPTD, por otro. Se asume de antemano que se conoce el promedio del proceso, cuyo índice se extrae del promedio del porcentaje de los artículos “no conformes” de al menos 25 lotes; no obstante, a menudo no hay dicho historial, por lo que una vez obtenida cierta información del porcentaje.

Los planes Dodge – Romig basados en el LCSP y LPTD, fueron diseñados y calculados para que se minimice la inspección total; esto es sumamente útil para procesos de productos aún no terminados o dentro de la planta[24]. Los datos de entrada para la función *f_dodge.romig.simple*( ) son los siguientes:

---

[22] JURAN, Joseph. *Análisis y planeación de la calidad*. p. 475.

[23] DODGE et al., *Sampling inspection tables, single and double sampling*. p.45.

[24] MONTOGOMERY, Douglas. *Introduction to statistical quality control*. p. 682.

Tabla XXIX. **Argumentos función f_dodge.romig.simple**

Argumento	**Descripción**
N	Tamaño del lote, debe de ser un número entero mayor que dos, por lo que es obligatorio especificarlo al igual que los demás argumentos de la función.
plan	Tipo de plan, ya sea “AOQL” o “LPTD”.
p	Proporción promedio de los “no conformes”

Fuente: elaboración propia.

Declarados los argumentos de la función, a continuación se establecen las condiciones para verificar la existencia de los argumentos necesarios para que sea posible ejecutar la función, de lo contrario, esta se detiene.

```
f_dodge.romig.simple<-function(N,plan,p){
# Encontrar el número de linea de intervalo del tamaño del lote
  # Atendiendo al tipo de plan AOQL 0.03 O LPTD 0.01
  if (missing(N)){
    # si no encuentra el lote, para la función
    stop("El lote debe ser igual o mayor que 2")
  } else {
```

De allí en adelante hasta el final, la función dependerá de otros condicionantes en cadenas sucesivas.

Por ejemplo, depende de los valores del tamaño del lote para interpolar el valor correcto, el caso extremo es cuando el lote es superior al valor máximo de las tablas del anexo 2, atendiendo al tipo de plan, AOQL o LPTD.

```
# De lo contrario carga las series lotes
load(file = "lot_size_DR.rda")
# Filtra las series de lotes de acuerdo al plan declarado
plan_lot<-lot_size_DR[lot_size_DR$plan==plan,]
# Si el plan es AOQL
   if (plan == "AOQL"){
     # Compara si el lote existe dentro de los limites superiores de
     # las series filtrada de lotes
     if (any(plan_lot$N == N)){
       # De ser cierto encuentra el intervalo especifico
lot_interval<-findInterval(N,as.integer(levels(as.factor(plan_lot$N))))
} else {
       # De lo contrario compara si es mayor a cualquier valor
       # para asignarle el mayor de los límites superiores
       # de lo contrario lo busca y lo ubica en el lugar i + 1
if (N > max(as.integer(levels(as.factor(plan_lot$N))))){
         lot_interval<-findInterval(N,as.integer(levels(as.factor(plan_lot$N))))
} else {
         lot_interval<-findInterval(N,as.integer(levels(as.factor(plan_lot$N))))+1
}
     }
     plan2<-"LPTD"
   } else {
     # Repite el proceso anterior con la variante del tipo de plan
     # Para el proceso si no esta definido cualquiera de los dos planes
if (plan == "LPTD"){
       if(any(plan_lot$N == N)){
         lot_interval<-findInterval(N,as.integer(levels(as.factor(plan_lot$N))))
       } else {
         if (N > max(as.integer(levels(as.factor(plan_lot$N))))){
           lot_interval<-findInterval(N,as.integer(levels(as.factor(plan_lot$N))))
         } else {
```

```
            lot_interval<-findInterval(N,as.integer(levels(as.factor(plan_lot$N))))+1
}
        }
        plan2<-"AOQL"
```

En el cálculo del intervalo del lote en cualquiera de los dos planes, resulta útil emplear la indexación del *data frame* puesto que hay valores repetidos y la función *findInterval*( ) soporta interpolar intervalos sí y solo sí no están repetidos y si el vector referido está ordenado; por ese motivo se declaran como enteros “integers” los niveles o *levels* posibles del *data frame***plan_lot.**

Se suma la unidad para ajustar aquellos lotes que estarían entre los límites superiores de la tabla. Si no está definido ningún tipo de plan, el proceso para. Una vez fijado el intervalo correcto se fija el tamaño de lote, para luego indexar el plan junto con los demás argumentos.

```
} else {
        stop("Debe de definir plan como AOQL o LPTD")
      }
    }
    # Fija el tamaño de lote exacto segun las series
lot_fix<-plan_lot[lot_interval,1]
```

Al igual que el lote, la proporción **p** de los “no conformes”, es interpolada por el mismo mecanismo ya descrito, pero vinculado a su propia tabla de límites superiores.

```
# Busca y fija la proporcion de los no conformes de acuerdo a la tabla
    # Al igual que con el tamaño de lote, compara y fija el intervalo correcto
    if (missing(p)){
```

```
    stop("Debe definir una proporción promedio de no conformes")
} else {
    load(file = "ap_DR.rda")
    if(any(ap_DR[plan] == p)){
      p_interval<-findInterval(p,ap_DR[,plan])
    } else {
      if (p > max(ap_DR[,plan])){
        p_interval<-findInterval(p,ap_DR[,plan])
      } else {
        p_interval<-findInterval(p,ap_DR[,plan])+1
}
    }
```

Con los valores fijados (terminación “fix” en los vectores), se procede a indexar el plan correcto.

```
# con el intervalo se fija el valor p de la tabla encuentra la serie de valores con p
de table  y el lote de la interpolación
p_fix<-ap_DR[p_interval,plan]
    plan_lot_p_fix<-plan_lot[plan_lot$p==p_fix,]
    plan_lot_n<-plan_lot_p_fix[plan_lot_p_fix$N==lot_fix,]
}
```

La clase de los objetos indexados es heredada desde el *data frame* **plan_lot**, por lo tanto, la extracción de los valores para la presentación se basa en la misma sintaxis.

```
# Presentar resultados
   structure(list("plan" = c("n"=eval(as.vector(plan_lot_n$n)),
                             "c"= eval(as.vector(plan_lot_n$c)),
"p"= eval(p_fix)),
```

```
                "Argumentos" = c("Lote" = as.integer(N),"Tipo de plan" = plan,
                        "Porcentaje Promedio de no conformes"= p_fix*100),
                "Resultados" = c("Muestra" =as.vector(plan_lot_n$n),
                        "Numero de Aceptacion" = as.vector(plan_lot_n$c),
                        "Numero Rechazo" = as.vector(plan_lot_n$c)+1,
plan2"=(plan2,as.vector(plan_lot_n$LPTD._AOQL)))))
}
}
```

Hasta la última llave descrita en el párrafo anterior, está definido el ambiente de la función *f_dodge.romig.simple*( ). Ahora ya es posible ejecutarla con el siguiente ejemplo:

"Se conoce que el promedio de los no conformes de varios lotes analizados previamente es 1,7 %; se desea conocer el plan de inspección de por variables que minimice los costos de inspección de un lote de 5 000 unidades con base en el tipo AOQL." Usando la función en la línea de comando de R, se declaran los argumentos descritos en el enunciado.

```
> f_dodge.romig.simple(N=5000, plan="AOQL", p=0.017)
$plan
     n      c      p
  "85"    "4" "0.018"

$Argumentos
 Lote      Tipo de plan    Porcentaje Promedio de no conformes
"5000"      "AOQL"                          "1.8"

$Resultados
 Muestra          Numero de Aceptacion  Numero Rechazo  plan21  plan22
 "85"              "4"                                "5"      "LPTD"   "9.5"
```

La muestra resultante es de 85 artículos y el número de aceptación es 4, por lo tanto el número de rechazo en un muestreo simple es 5. Por último, en contraparte el índice LPTD para este tipo de muestreo es de 9,5 %.

Si se cambiara entre los argumentos de la función a un plan LPTD, se tienen los siguientes resultados:

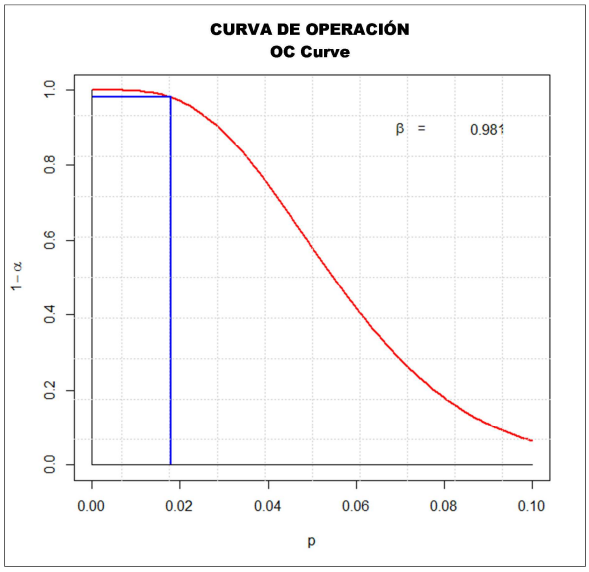
```
> f_dodge.romig.simple(N=5000, plan="LPTD", p=0.095)
$plan
      N     c         p
 "1120"  "7"      "0.005"

$Argumentos
  Lote     Tipo de plan    Porcentaje Promedio de no conformes
 "5000"  "LPTD"                    "0.5"

$Resultados
  Muestra          Numero de Aceptacion Numero Rechazo plan21    plan22
   "1120"          "7"                        "8"           "AOQL"      "0.31"
```

Se puede notar que según la tabla para un plan LPTD, el porcentaje máximo es 0,5 % por lo que toma los valores correspondientes a dicho bloque de datos. En este caso puede apreciarse la proporción elevada de la muestra respecto del tamaño del lote, aproximadamente el 22,4 %, por lo que el costo de inspección es elevado debido a que el promedio de la proporción “p” de los “no conformes” es grande; por ello el consumidor no estaría dispuesto a correr el riesgo con muestras pequeñas.

En adicional, notar en la figura 92 y 93, en las que se escribe sin acento la palabra “OPERACIÓN”, ya que la compilación del programa del paquete no

acepta caracteres especiales.

Figura 92. **Curva CO, plan AOQL método Dodge Romig**



Fuente: elaboración propia, con paquete Planesmuestra.

Dichos escenarios son comparables en función de la probabilidad de rechazo o aceptación de un lote analizado en una curva de operación; a continuación dos curvas de operación ploteadas mediante la función *f_DR.CO*( ) o con *f_CO.plan*( )con los argumentos “n” como la muestra, “c” como el número de aceptación y “p” como la proporción de los “no conformes”.

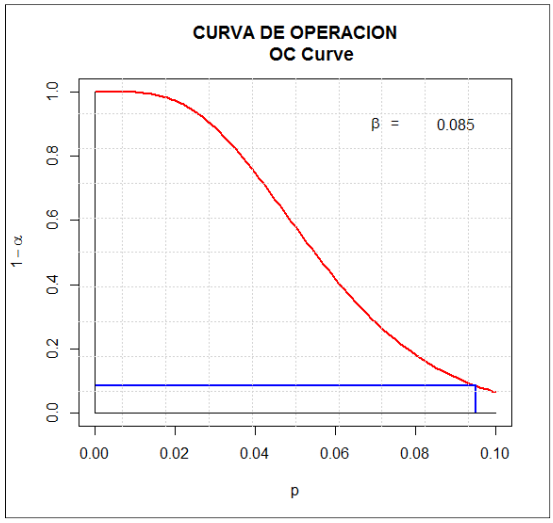
Para la segunda se utiliza el vector **plan** implícito en los resultados de salida de la función *f_dodge.romig.simple*( ). Los argumentos “n” y “c”, provienen de los resultados del plan con la función *f_dodge.romig.simple*( ), así que tendrán su efecto dependiendo del tipo de plan definido (AOQL o LPTD).

Puede apreciarse en la figura 92 una probabilidad “beta” de aceptar el lote

de 98,1%. Mientras que para el plan LPTD, la figura 93 indica que la probabilidad es ser aceptado un lote “rechazable” es de 8,5 %; lo que constituye el riesgo para el consumidor.

```
> f_DR.CO(n=85, c=4, p=0.095)
  c       n       p       beta
[1,]      4       85      0.095   0.08457451
```

Figura 93. **Curva CO, plan LPTD método *Dodge Romig***



Fuente: elaboración propia, con paquete Planesmuestra.

Es muy importante notar que en la metodología *Dodge Romig* el resultado de la función *f_DR.CO*( ), se basa en la distribución binomial o sea curva tipo “B”, por lo tanto no es apta para lotes pequeños; en tal caso, es recomendado usar un código que calcule probabilísticamente los resultados con la distribución hipergeométrica o tipo “A”.

4.3.2. Sistema MIL STD 105E

Basado en el NCA, el MIL STD 105E o ANSI ASQ C Z 1.4 es un sistema de muestreo de aceptación por atributos. El índice NCA como porcentaje máximo de los no conformes o defectuosos, puede verse también como el número máximo de las "inconformidades" por cada 100 unidades y en ambos casos, existen los valores correspondientes del plan de muestreo atendiendo a:

- Nivel de calidad aceptable (NCA)
- Tamaño del lote (N)
- Tipo de inspección: normal, rigurosa o reducida
- Tipo de muestreo: simple, doble o múltiple
- Nivel de Inspección: I, II, III o los especiales S-1, S-2, S-3 y S-4.

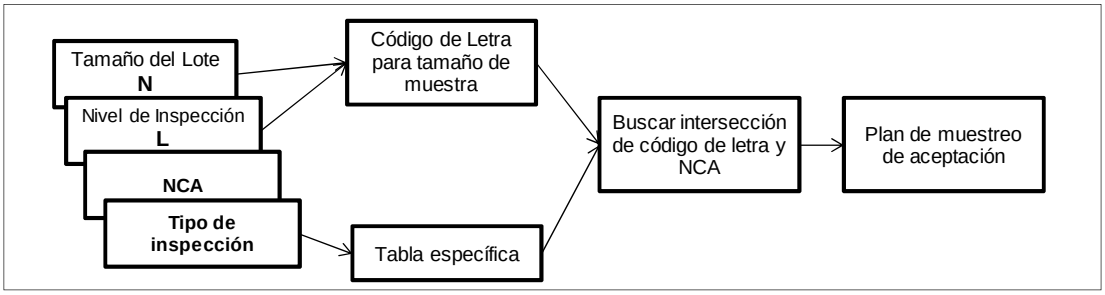
Respecto del tipo de inspección, habitualmente se inicia con "Normal" y dependiendo del récord posterior del productor, se traslada el procedimiento para determinar el plan de inspección a "Rigurosa" en el caso de que sea más recurrente el número de lotes rechazados; de lo contrario, se opta por el tipo "Reducida", cuando el número de lotes rechazados se reduce excepcionalmente.

El nivel de inspección empleado habitualmente es el II, al bajar al nivel I el plan de muestreo se reduce casi a la mitad[25], mientras que en el nivel III se incrementa casi al doble. Los niveles de inspección especiales se emplean para tamaños de muestra reducidos. La figura 94 esquematiza los pasos para determinar el plan de muestreo, los cuales sirven para el *script* de la función *f_milstd105e*( ).

[25] JURAN, Joseph. *Análisis y planeación de la calidad*. p. 470.

Las tablas que utilizan dicho procedimiento están detalladas en el anexo; dicha información se adaptó a un objeto tipo *data frame* para facilitar la llamada y lectura por índices por parte de la función *f_milstd105e*( ). La información se segmentó en los archivos: lote_size.Rda, code_letter.Rda, NCA_valuesR, milstd105eplans.Rda, para interpolar el tamaño del lote, el código de letra y el plan de muestreo de aceptación específico.

Figura 94. **Procedimiento del método MIL STD 105E**



Fuente: elaboración propia, empleando Word.

Los argumentos para ejecutar la función *f_milstd105e*( )se detallan en la siguiente tabla.

Tabla XXX. **Argumentos función f_milstd105e**

Argumento	**Descripción**
N	Tamaño del lote, debe de ser un número entero mayor que dos, por lo que es obligatorio especificarlo al igual que los demás argumentos de la función.
L	Nivel de inspección
NCA	Nivel de calidad aceptable

Continuación de la tabla XXX

Argumento	**Descripción**
type	Tipo de inspección normal = “n”, rigorosa = ”t” y reducida = “r”, las abreviaturas provienen de su nombre en inglés *normal, tightened* y *reduced,* respectivamente.

Fuente: elaboración propia.

El texto completo del *script* de la función *f_milstd105e*( ) se encuentra en apéndice 3, pero en los siguientes párrafos se desarrolla la lógica y el cálculo de los resultados del plan, atendiendo a los argumentos.

En primer lugar es necesario crear la propia función y fijar los argumentos necesarios, fijando la condición que al faltar un argumento, la función se detiene con aviso *stop*( ). La falta de un argumento se evalúa con *missing*( ).

```
f_milstd105e<-function(N,L,NCA,T){
  if (missing(N)){
    stop("El lote debe ser igual o mayor que 2")
    }
```

De lo contrario, se debe cargar el archivo “lot_size.rda” para interpolar con la función *findInterval*( ) el valor del tamaño del lote dentro de la tabla de lotes, es decir, determinar qué posición en la tabla tiene, para luego asignar el número de línea y cruzarlo con el código de nivel de inspección “L” y encontrar el código de letra para el plan, cargando el archivo “code_letter.rda”.

```
else {
     load(file = "lot_size.rda")
```

```
        lot_interval<-findInterval(N,lot_size[,1])+1
      rm(lot_size)
       if (missing(L)){
stop("Es necesario el argumento nivel o L")
} else {
                load(file = "code_letter.rda")
              code_l<-code_letter[lot_interval,L]
rm(code_letter)
          }
```

El argumento NCA es buscado e interpolado dentro de la tabla de valores, definidos en el anexo 2, y lo asigna a la variable interna.

```
if (missing(NCA)){
          stop("Es necesario el argumento NCA")
          } else {
               load(file = "NCA_values.rda")
               if (any(NCA==NCA_values)){
NCA2<-NCA
                 } else {
                      NCA_interval<-findInterval(NCA,NCA_values)
                      NCA2<-NCA_values[NCA_interval]
                     }}
```

El último argumento también es validado, para después asignar valores de texto con el nombre de cada tipo de inspección.

```
if (missing(type)){
          stop("Es necesario el argumento type del tipo de inspección")
}
         if(type=="n"){
          T_ins<-"Normal"
```

```
        } else {
              if(type=="r"){
                T_ins<-"Reducida"
} else {
                      T_ins<-"Rigurosa"
                      }
                }
```

Siendo todos los argumentos válidos, se lee el archivo “milstd105eplans.rda” donde están todos los tipos de planes y los valores que componen un plan de inspección en particular. Luego la búsqueda de los valores es cuestión de recurrir a hacer subconjuntos de la información y asignarlos a los vectores de resultados.

```
load(file = "milstd105eplans.rda")
code1<-milstd105eplans[milstd105eplans$code_letter==as.vector(code_l),]
T1<-code1[code1$T==type,]
NCA3<-T1[T1$NCA==NCA2,]
c<-NCA3$c
muestra<-NCA3$n
```

Hasta este punto la función ha terminado de hacer las búsquedas y cálculos necesarios, de allí es necesario presentar el resumen de los argumentos y los resultados en forma visible mediante la función *print*( ). Notar en la siguiente pieza de código la creación interna de vectores con los valores de los argumentos y resultados, cuyos nombres son asignados para ser visualizados al ejecutar la función, así también se ha empleado la función *as.vector*( ) para transformar el resultado de clase “factor” del código de letra.

```
argumentos_nombres<-c("Lote","Tipo de Inspeccion", "Nivel de Inspeccion",
```

```
                                "Nivel de Calidad Aceptable")
        argumentos_plan<-c(N,T_ins,L,NCA)
# Objeto con los parametros del plan
        resultados_nombres<-c("Codigo Letra",
                                "Nivel de Calidad Aceptable", "Muestra",
                                "Numero de Aceptacion", "Numero de Rechazo")
        resultados_plan<-c(as.vector(code_l),NCA2,muestra,c,c+1)
# Integrar los resultados en un data frame
        names(argumentos_plan)<-argumentos_nombres
        names(resultados_plan)<-resultados_nombres
# Presentar resultados
        print(argumentos_plan)
        print(resultados_plan)
         }
}
```

Un ejemplo del uso de la función con el enunciado siguiente: “Se cuenta con un lote de 11 mil piezas que el productor pone a disposición para su inspección. Dado que, esta es la primera vez en que se establecen relaciones comerciales con dicha empresa, se piensa en que la inspección debe de ser del tipo rigurosa, aceptando un máximo de 30 piezas defectuosas de la muestra. Con base en la metodología MIL STD 105E se tiene contemplado un nivel de inspección “I”. Se pide determinar el plan de inspección.

Haciendo uso de la función *f_milstd105e*( ) los resultados son los siguientes:

```
> f_milstd105e(N=11000,L="I",type="t",NCA=30)
Tamaño Lote         Tipo de Inspección
              "11000"               "Rigurosa"
Nivel de Inspección      Nivel de Calidad Aceptable
```

```
"I"                          "30"
          Código Letra  Nivel de Calidad Aceptable   Muestra
               "K"              "25"                  “125”
Número de Aceptación Número de Rechazo
"18"                          "19"
```

Notar que se regresan los argumentos y los resultados con nombre. En este caso, es necesario levantar una muestra de 125 unidades, de las cuales 18 es el número máximo de aceptación; si hubiera 19 unidades no conformes, el lote completo se rechaza, bajo el supuesto de una inspección simple. Otro ejemplo podría tratarse de un lote 1 500 unidades, para lo cual, el cliente fija un NCA igual a 0,05, suponiendo un tipo de inspección normal y un nivel de inspección “estándar”, es decir, nivel “II”, los resultados son los siguientes:

```
> f_milstd105e(N=1500,L="II",NCA=0.05,type="n")

          Tamaño Lote        Tipo de Inspección
               "1500"             "Normal"

Nivel de Inspección Nivel de Calidad Aceptable
               "II"               "0.05"

          Codigo Letra Nivel de Calidad Aceptable
               "K"               "0.04"

          Muestra      Número de aceptación        Número de rechazo
          "125"              "0"                        "1"
```

Con dichos valores y lo aprendido en el capítulo anterior es posible utilizarlos como argumentos para elaborar una curva característica de operación o CO.

```
f_DR.CO<-function(c,n,NCA){
if (missing(NCA)){
    stop("El nivel de calidad aceptable debe de ser definido,
    para calcular la probabilidad de aceptación")
    }
 if (missing(n)){
    stop("La muestra debe de ser un número entero y positivo")
    }
 if (missing(c)){
    stop("El numero de aceptación debes ser entero e igual o mayor que cero")
    }
 beta.x<-pbinom(c,n,NCA)
prob.x<-seq(0,0.1,by=0.001)
 n.1<-pbinom(c,n,prob.x)
 plot(prob.x, n.1, type = "l", lwd = 2,
     col = 2, cex = 2, bg = NA,
     xlab = "p", ylab = expression(1- alpha),
xlim = c(0,0.1), ylim = c(0,1),
     main = "Curva de Operación")
# Agrega opciones de graficas de bajo nivel
segments(x0=0.0,y0=beta.x,x1=p,y1=beta.x,col="blue",lwd=2)
 segments(x0=p,y0=0.0,x1=p,y1=beta.x,col="blue",lwd=2)
 segments(x0=-0.0,y0=0,x1=0.1,y1=0,col="black",lwd=1)
 segments(x0=-0.0,y0=0,x1=0,y1=1,col="black",lwd=1)
 text(0.07,0.9, expression(paste(beta)),cex = 1, col="black")
 text(0.075,0.9, expression(" = "),cex = 1, col="black")
text(0.09,0.9, round(beta.x,3),cex = 1, col="black")
 grid(10, 10, lwd = 1)
 structure(cbind("c"=c, "n"=n, "p"=p, "beta"=beta.x))
}
```

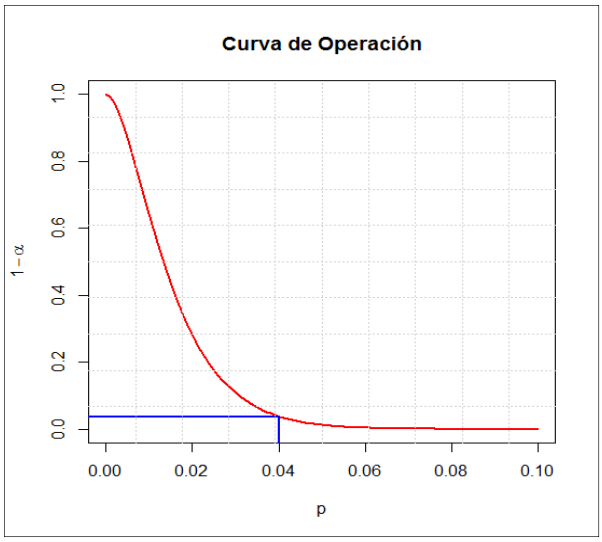
Un ejemplo basado en la anterior función, es tomando los resultados del

plan calculado. Con n = 125, c = 0 y un NCA = p= 0,04. La gráfica de la CO muestra la curva probabilística y los segmentos de línea señalan el valor de la probabilidad de aceptación del lote, asumiendo un lote lo suficientemente grande más que la muestra “n”, es decir, del tipo de curva “B” (distribución binomial), la función *f_DR.CO*( ) necesita los argumentos recién descritos.

```
> f_DR.CO(c=0,n=125,p=0.04)
   c       n       p       beta
[1,] 0     125     0.04    0.006080008
```

El valor de NCA se declaró como “p” una proporción, dentro de la función; únicamente aplican para proporciones menores a la unidad. La probabilidad de rechazo para el productor de que sea rechazado el lote es de 0,6 %. Por tal razón está la línea perpendicular al eje 1 menos alfa muy cercana a cero (figura 95).

Figura 95. **Curva CO de acuerdo al plan determinado**

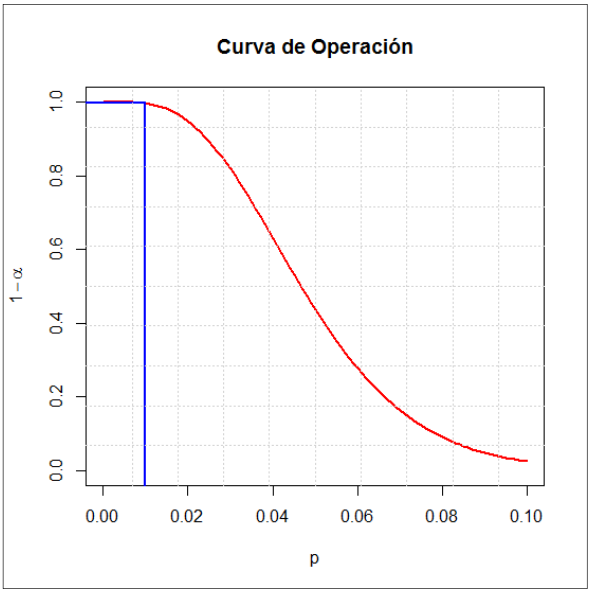


Fuente: elaboración propia, con paquete Planesmuestra.

En otras palabras, el riesgo de rechazar el lote para el productor es muy alto. Por el contrario con el siguiente ejemplo, la probabilidad de aceptar el lote se incrementa considerablemente con un NCA más bajo (ver figura 96), confirma el razonamiento de que a mayor NCA, mayor probabilidad de rechazo y la curva tiende a tener mayor convexidad al origen.

```
> f_milstdE.CO(c=4,n=100,NCA=0.01)
> pbinom(4,100,0.01)
[1] 0.9965677
```

Figura 96. **Curva CO, con un mayor NCA**



Fuente: elaboración propia, con paquete Planesmuestra.

4.4. *Script* para muestreo de aceptación por variables

La diferencia fundamental de los muestreos por variables de los que se dan por atributos, es que tomando una medida física de la muestra de los

artículos a inspeccionar, ya sea destructiva o no, se calculan índices de dicha muestra, se compara con un “valor permisible” y se acepta o rechaza un lote. Es decir, en estos sistemas de muestreo por variables se compara con un valor que corresponde a una escala de una variable, mientras que en los muestreos por atributos, se compara con una cantidad entera de elementos.

4.4.1. Sistema MIL STD 414

Por medio del sistema MIL STD 414, en la versión militar del ANSI ASQ C Z 1.9, es definido el valor “permisible”; se extrae de las tablas del anexo 2, que dependen del tipo de plan, ya sea “normal” o “riguroso” los cuales están incluidos en dicha tabla. Este valor es comparado con el índice estandarizado de la diferencia de un límite de especificación y la media de la muestra.

Quiere decir que los valores estandarizados se comparan de la siguiente forma: Sí $\frac{S-\bar{X}}{s} \geq k,$ el lote se acepta, de lo contrario se rechaza; donde S es límite superior; y $\frac{L-\bar{X}}{s} \leq k$, el lote se acepta; de lo contario se rechaza, donde I, es el límite de especificación inferior.

El sistema MIL STD 414 es muy similar a su contraparte en MIL STD 105E para el muestreo por atributos, ya que hay que definir previamente un “tipo de inspección” y un “nivel de inspección” (anexo 2). Por lo que el *script* necesario para ejecutar dicha consulta es bastante aproximado al visto en la sección anterior.

Para hacer funcional la búsqueda del plan de inspección (código de letra, muestra y el valor k), la tabla es llevada a un arreglo tipo *data frame* con un total de 433 líneas para buscar el plan adecuado, así como encontrar el valor fijo de

la tabla de los valores de NCA o AQL en inglés, se interpola el valor propuesto con los valores de la tabla y el tamaño del lote para encontrar el código de letra entre los niveles de inspección.

En los siguientes párrafos se desarrolla la función *f_milstd414*( ) que encuentra el plan de inspección por variables adecuadas en función de los siguientes argumentos. En tal sentido, los datos de entrada o argumentos son:

Tabla XXXI. **Argumentos función f_milstd414**

Argumento	**Descripción**
N	Tamaño del lote, debe de ser un número entero mayor que dos, por lo que es obligatorio especificarlo al igual que los demás argumentos de la función.
L	Nivel de inspección
NCA	Nivel de calidad aceptable
type	Tipo de inspección normal = “n”, rigorosa = ”t”

Fuente: elaboración propia.

Puede observarse la similitud de los argumentos con el método del sistema MIL STD 105E; la diferencia es que no se obtiene un número de rechazo, más bien un índice de comparación, tal y como sucede con las pruebas de hipótesis.

Otra característica fundamental es que por tratarse de variables, la distribución de probabilidades es normal, en comparación con los planes de inspección por atributos que van desde la hipergeométrica, Poisson y binomial,

por tratarse de distribuciones discretas.

Para encontrar el plan de inspección se utilizan objetos de datos conteniendo las tablas, las cuales son leídas al ejecutarse la función.

Los datos están contenidos en los archivos lot_size.milstd14.Rda, code_letter.milsted414.Rda y k_plnas.milstd414.Rda.

Primero, es necesario asignar al objeto la clase *function* con los argumentos ya definidos. Luego se establece la validez de la existencia del lote y la validez del tamaño, el cual como mínimo debe de ser mayor que tres; siguiendo con la fijación del intervalo del lote observado para fijar el tamaño del lote según los intervalos de la tabla.

```
f_milstd414<-function(N,L,NCA,type){
# Encontrar el intervalo del lote si este existe
  if (missing(N)){
    # si no encuentra el lote, para la funcion
    stop("Debe definir un Lote para continuar")
  } else {
    if (N<3){
      stop("El lote debe ser igual o mayor que 3")
    } else {
    # De ser cierto encuentra el intervalo especifico
load(file = "lot_size.milstd414.rda")
        lot_interval<-findInterval(N,lot_size.milstd414$N)
        lot_fix<-lot_size.milstd414[lot_interval,1]
```

Continúa con la verificación de la existencia del nivel de inspección, parando la función si no existe. Luego carga el catálogo de letras para encontrar

la letra en función de la “fila”, que corresponde al mismo número de fila del intervalo del lote.

```
if (missing(L)){
        stop("Debe definir un nivel de inspección")
} else {
         load(file = "code_letter.milstd414.rda")
# Asigna la letra segun el intervalo del lote
code_letter<-as.vector(code_letter.milstd414[lot_interval,L])
}
```

La existencia del argumento del tipo de inspección (normal o rigurosa) es verificada en la función; sí es “falsa”, detiene la ejecución de la función, de lo contrario carga en forma tabular todas las tablas a la memoria, indexa y filtra por medio de los argumentos la fila que contiene el plan de inspección.

```
if (missing(type)){
        stop("Debe definir un tipo de inspección")
} else {
         load(file = "k_plans.milstd414.rda")
# Indexa todos los planes y filtra de acuerdo al plan
NCA_T<-k_plans.milstd414[k_plans.milstd414$T==type,]
# Indexa el objeto anterior y lo filtra de acuerdo a la letra
NCA_T_c.l<-NCA_T[NCA_T$code_letter==code_letter,]
# Busca el intervalo del NCA
         NCA_interval<-findInterval(NCA,NCA_T_c.l$NCA)
   # Decide si el intervalo es menor al menor del NCA de tabla
   # para aproximarlo al primero, de lo contrario la fijacion del
   # NCA de tabla es normal y los valores mayores se quedan en el
   # valor maximo de tabla
        if (NCA < min(NCA_T_c.l$NCA)){
```

```
            NCA_fix<-NCA_T_c.l$NCA[1]
          } else {
            NCA_fix<-NCA_T_c.l$NCA[NCA_interval]
          }
    # Indexa y filtra la unica fila de acuerdo al NCA de tabla
          milstd414_plan<-NCA_T_c.l[NCA_T_c.l$NCA==NCA_fix,]
```

El último paso es nombrar todos los argumentos y resultados, asignarlos como nombres a los vectores con los respectivos datos de entrada y resultados, los cuales son presentados al final.

```
if(type=="n"){
            T_plan<-"Normal"
} else {
            T_plan<-"Riguroso"
          }
        }
    #
    # Consolida los resultados en vectores con nombres
    #
    argumentos_nombres<-c("Tamaño Lote","Plan",
                          "Nivel de Inspeccion","NCA Inicial")
    argumentos_plan<-c(as.integer(N),T_plan,L,NCA)
    resultados_nombres<-c("Codigo Letra","NCA Tabla","Muestra","k")
    resultados_plan<-c(code_letter,NCA_fix,milstd414_plan$sample,
                               milstd414_plan$k)
    names(argumentos_plan)<-argumentos_nombres
    names(resultados_plan)<-resultados_nombres
    # Presentar resultados
print(argumentos_plan)
    print(resultados_plan)
```

```
    }
   }
  }
```

Como ejemplo, se tiene el siguiente enunciado: “Una compañía toma la decisión de adquirir con un nuevo proveedor, un producto químico “A” empacado en unidades de 50 ml junto con otro producto químico “B” en la misma presentación, fabricado internamente. Por razones de ventajas competitivas, el producto “B” será fabricado y empacado de acuerdo con las especificaciones exigidas. No obstante, las regulaciones locales exigen ciertos parámetros como el producto “A” que al mezclarse con “B” produce una emulsión que emana gases ligeramente tóxicos. Dicha toxicidad, depende en gran medida del porcentaje de ingrediente activo presente en el producto.

Las regulaciones dictan que dicho porcentaje no debe de ser menor del 4,7 % y no mayor del 5,1 %. Para la compañía, la seguridad del consumidor es uno de sus principios, por lo que busca que el proveedor le garantice al menos el porcentaje máximo. El proveedor garantiza proporcionar un nivel de calidad aceptable (NCA) no mayor al 1 %.

El tamaño regular de los lotes es de 1 200 unidades; no obstante, se acordó que se harán al principio pruebas destructivas con una muestra para determinar si son aceptados o rechazados los lotes, al obtener los estudios de laboratorio.

Por iniciar relaciones comerciales con el nuevo proveedor, se elige un tipo de inspección rigurosa y un nivel de inspección “III” y se acuerda adoptar el sistema MIL STD 414 para obtener el plan de inspección”. Se pide obtener el tamaño de la muestra y el factor de corrimiento *k*.

```
> f_milstd414(N=1200,NCA=1,L="III",type="t")

Tamaño Lote          Plan    Nivel de Inspección      NCA Inicial
      "1200"          "Riguroso"          "III"                "1"

Código letra          NCA tabla    Muestra        k
      "I"          "1"                "25"          "1.98"
```

Los resultados señalan que la muestra debe de ser de 25 unidades y el factor de corrimiento *k* es igual a 1,98. No obstante, es preciso comprobar si al hacer el análisis de laboratorio el lote debe de aceptarse o rechazarse, haciendo uso de una nueva función *f_milstd414.test*( ) que utiliza los resultados del plan de inspección por variables como argumento.

Suponiendo que las 25 muestras tienen los siguientes resultados porcentuales:

4,7 5,1 4,9 4,9 4,8 4,9 4,9 4,8 4,8 4,7 4,7 4,9 4,8
4,9 4,6 4,8 4,9 5,1 4,8 5,0 5,0 4,7 5,0 5,0 4,8

Estos se almacenan en un objeto de nombre **x** para mayor facilidad de manipulación de los argumentos de la función:

```
> x<-read.table("clipboard")
```

La definición de la función *f_milstd414.test*( ) se basa en los siguientes argumentos según la tabla XXXII. La definición de la función inicia con la verificación de la “no existencia” del vector de valores de los resultados de la muestra.

```
f_milstd414.test<-function(x,k,S,Limite,L){
if(missing(x)){
   stop("Debe de haber una referencia valida de los datos de muestra")
} else {
   .x<-x[,1]
   med_x<-mean(.x)
}
```

Tabla XXXII. **Argumentos función f_milstd414.test**

Argumento	**Descripción**
N	Tamaño del lote; debe de ser un número entero mayor que dos, por lo que es obligatorio especificarlo al igual que los demás argumentos de la función.
k	Factor de corrimiento estandarizado de la tabla.
S	Desviación estándar, si no se conoce, se omite el argumento y se calcula de la muestra.
Límite	Límite de especificación para estandarizar, superior “S” o inferior “I”.
L	Valor del límite.

Fuente: elaboración propia.

Al verificarse la existencia de los datos de la muestra, se calcula el promedio y la desviación estándar si la misma no es conocida:

```
if(missing(S)){
      S<-sd(.x)
} else {
 S<-S
  }
```

Es necesario también verificar que hay un límite, de lo contrario, el proceso se detiene, para luego efectuar el mismo tipo de verificación con el factor de corrimiento *k*, que de acuerdo con los resultados del último enunciado es igual a 1,98. De allí en adelante es cuestión de tomar los argumentos, calcular el valor crítico de *k*, compararlo con el valor de la tabla y tomar una decisión si aceptar el lote o rechazarlo.

```
if(missing(k)){
   stop("Falta el argumento de corrimiento k")
} else {
     if(missing(Limite)){
       stop("Falta especificar si es limite superior o inferior, S o I")
     } else {
       if(missing(L)){
stop("Falta especificar el valor del limite de especificacion")
} else {
          k_critico<-(L - med_x)/S
          if(Limite=="S"){
            if(k_critico >= k){
print("Aceptar el lote")
              } else {
                print("Rechazar el lote")
}
          } else {
            if(k_critico <= k){
print("Aceptar el lote")
              } else {
                print("Rechazar el lote")
              }
          }
        }
```

```
    }
  }
}
```

Los resultados de los argumentos del presente ejercicio expresados en la función arrojan los siguientes resultados:

```
> f_milstd414.test(as.data.frame(x2),k=1.98,Limite="S", L=5.1)
[1] "Rechazar el lote"
> f_milstd414.test(as.data.frame(x2),k=1.98,Limite="I", L=4.7)
[1] "Aceptar el lote"
```

Los resultados dictaminan por separado a cada límite de especificación. El promedio de la muestra indica que el porcentaje del ingrediente activo está en promedio a 4,86 % con una desviación estándar de 0,1290994, siendo el límite superior de especificación de 5,1%; en tal sentido el k crítico se calcula en 1,85, comparándolo con el valor del plan de 1,98; la conclusión sin dudas es rechazar el lote.

En cambio para el límite inferior de 4,7 %, la conclusión es aceptar el lote, ya que el “k” crítico es menor que el valor proveniente del plan.

5. APLICACIÓN A LAS SERIES DE TIEMPO Y PRONÓSTICOS

5.1. Introducción a las series de tiempo

Una serie de tiempo es una secuencia de datos ordenada en forma cronológica. A diferencia de una serie de corte transversal que mide una o varias variables en un instante de tiempo determinado, la serie de tiempo implica una perspectiva de un fenómeno cualquiera a través del tiempo, sin importar la unidad de medida empleada.

Al final, la utilidad de analizar las series de tiempo es entender el comportamiento en determinados momentos de una variable y tener los elementos de cálculo suficientes para levantar pronósticos.

En economía, *marketing*, medioambiente, demografía e ingeniería, el estudio las series de tiempo son un conjunto de técnicas y una estrategia para proyectar hacia el futuro y entender los vaivenes del pasado.

En este capítulo se desarrolla más de forma introductoria la comprensión exhaustiva de las series de tiempo en control de calidad y los pronósticos industriales, para hacer hincapié en los paquetes ya existentes y compartidos en el *CRAN*, aplicables para tales propósitos, incluyendo algunas pruebas paramétricas y no paramétricas que son importantes para la detección de ciertos síntomas, como la presencia de una tendencia en una muestra para el control de calidad por variables, que en tal caso, equivale a un corrimiento de la media y los límites naturales de control.

Por otro lado, en los pronósticos industriales, implica un instrumental que permite adecuar un modelo a los datos y detectar los componentes de una serie de tiempo, que como es sabido, en la mayoría de los casos, las series tienen un componente estacional (*E*), uno de tendencia (*T*) y el componente aleatorio expresado como (*I*):

$$X_t = T_t + E_t + I_t$$

El subíndice "sub t" significa el tiempo o periodo y corresponde a valores discretos en el tiempo[26]; de ser continuos los valores, la notación es igual a:

$$X(t) = T(t) + E(t) + I(t)$$

Esta aclaración puede ser crítica en determinado momento, ya que constantemente se trata con series de tiempo cuyos datos son continuos o en su defecto discretos, tal y como sucede con las cartas de control.

De las series de tiempo se conocen dos tipos, las estacionarias y las no estacionarias.

- Estacionarias: una serie de tiempo es estacionaria cuando a través del tiempo es estable; dicha condición se cumple cuando la media y la varianza son constantes. Además, de ser estacionaria se dice que es estocástica y que es como tal, porque además de una media y varianza constante en el tiempo, "el valor de la covarianza entre dos periodos depende solo de la distancia o rezago entre estos dos periodos y no del tiempo en el cual se calculó la covarianza"; en resumen es un proceso estocástico débilmente estacionario.

[26] GUJARATI, Damodar.*Econometría*. p. 740.

5.1.1. Definición matemática de la serie de tiempo

La anterior definición de serie de tiempo estacionaria, está planteada matemáticamente bajo los siguientes principios probabilísticos:

Media	$E(X_t) = \mu$
Varianza	$\text{var}(X_t) = E(X_t - \mu)^2 = \sigma^2$
Covarianza	$\gamma_k = E[(X_t - \mu)(X_{t+k} - \mu)]$

En contraposición a la definición matemática anterior, una serie de tiempo no estacionaria se refiere a la que no cumple con la definición matemática anterior; su aparición en cualquier contexto no implica un defecto, más bien su descubrimiento implica cambio en la perspectiva de los datos y el modelo especificado. Las series de tiempo no estacionarias a su vez, se fundamentan en el modelo de caminata aleatoria, la cual puede ser con deriva o sin deriva, en particular el modelo de caminata aleatoria o *Random Walk*, es conocido como un proceso de raíz unitaria.

La anterior definición resulta importante para comprender y detectar una situación de no estacionariedad; un ejemplo típico es mediante la prueba de raíz unitaria. Una serie de tiempo no estacionaria, observada en términos de control estadístico de calidad, implica corrimiento de la media o del límite central. Un corrimiento en la media, no implica necesariamente un problema técnico en el proceso sino una advertencia de que hay cambios notorios.

5.1.2. Definición de serie de tiempo en R

En el capítulo 2, se dio a conocer el objeto de R conocido como serie de tiempo o *ts,* del cual los argumentos que destacan se describen a continuación:

Tabla XXXIII. **Argumentos objeto ts**

Argumento	**Descripción**
Data	Es el objeto conteniendo la serie de los valores ordenados, puede ser un vector o una matriz, de ser un data.frame tiene que interpretarse como una matriz.
Start	Es un vector de longitud dos, que contiene el año y mes inicial, ambos tienen que ser números enteros, sin embargo puede tratarse únicamente del año (longitud uno).
End	Un vector de longitud dos, que contiene la fecha final de la serie, especificada de la misma forma que el inicio.
frequency	Es el número de observaciones por unidad de tiempo.

Fuente: elaboración propia.

Por ejemplo, se tiene una serie de tiempo consistente en un arreglo de meses y años de ventas en unidades de baterías para automóvil de 12 voltios.

Trasladando los datos a un objeto de R, se necesita de tratamiento previo a convertirse en serie de tiempo como tal. Es conveniente mencionar que al copiar datos como la tabla XXXIV, es necesario hacerlos sin formatos como los marcos, ya que pueden ocasionar errores en la lectura de datos, en este caso, únicamente se copian los datos, sin encabezados (columna); por ello el argumento "*header* = FALSE".

Tampoco es necesario copiar el nombre del mes (fila), ya que en la definición del objeto "ts", se asignan automáticamente los encabezados y nombres de fila omitidos.

Tabla XXXIV. **Serie de tiempo, ventas de baterías**

Mes / Año	**2010**	**2011**	**2012**	**2013**	**2014**	**2015**
Enero	122	111	99	101	112	118
Febrero	81	81	78	82	93	115
Marzo	100	153	143	100	112	121
Abril	166	112	113	165	148	-
Mayo	98	88	100	101	133	-
Junio	71	66	72	74	78	-
Julio	201	218	227	232	238	-
Agosto	189	170	200	204	209	-
Septiembre	72	67	98	101	110	-
Octubre	66	72	68	73	82	-
Noviembre	90	78	81	82	98	-
Diciembre	181	202	199	209	234	-

Fuente: elaboración propia.

```
> baterias<-read.table("clipboard",header=FALSE,
na.strings="NA",sep="\t",dec=".")
```

Observar que el último año detalla únicamente los primeros tres meses, por lo que hay valores no disponibles o “NA”.

```
> baterias
   V1    V2    V3    V4    V5    V6
1 122   111    99   101   112   118
2  81    81    78    82    93   115
3 100   153   143   100   112   121
4 166   112   113   165   148    NA
5  98    88   100   101   133    NA
6  71    66    72    74    78    NA
7 201   218   227   232   238    NA
8 189   170   200   204   209    NA
```

```
9   72    67      98      101     110     NA
10  66    72      68      73      82      NA
11  90    78      81      82      98      NA
12 181   202     199     209     234      NA
```

Ahora, es necesario llevar el objeto **baterías** a la forma vector para luego adaptarlo al objeto de clase “ts”.

```
> baterias<-as.numeric(as.matrix(baterias))
> baterias<-ts(baterias,start=c(2010,1),end=c(2015,3),frequency=12)
> baterias
```

```
     Jan  Feb  Mar  Apr  May  Jun  Jul  Aug  Sep  Oct  Nov  Dec
2010 122   81  100  166   98   71  201  189   72   66   90  181
2011 111   81  153  112   88   66  218  170   67   72   78  202
2012  99   78  143  113  100   72  227  200   98   68   81  199
2013 101   82  100  165  101   74  232  204  101   73   82  209
2014 112   93  112  148  133   78  238  209  110   82   98  234
2015 118  115  121
```

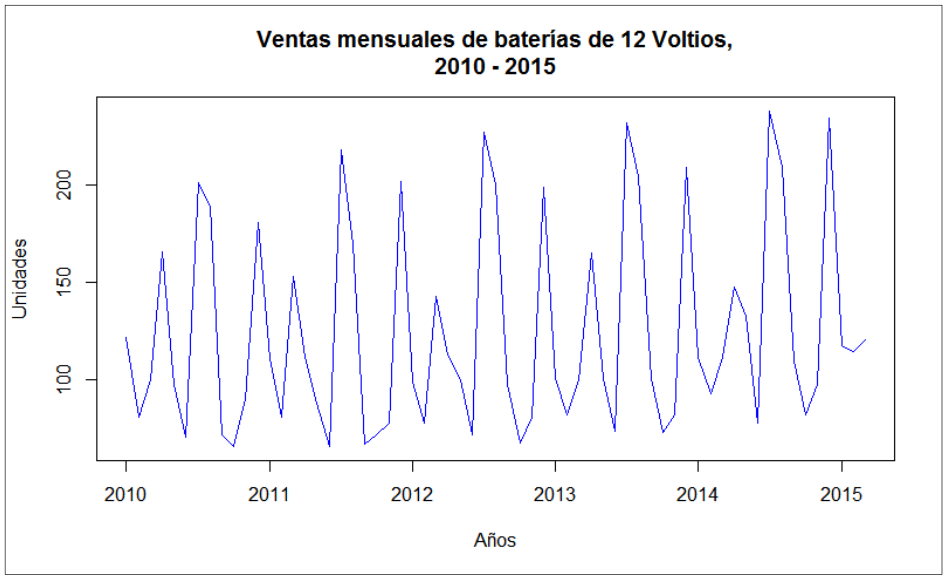
Ya creado el objeto **baterías** como una serie de tiempo, se puede diagramar un gráfico de los valores de ventas en unidades respecto del tiempo.

```
> plot(baterias,main="Ventas mensuales de baterías de 12 Voltios,
+ 2010 - 2015", xlab= "Años", ylab="Unidades", col="blue")
```

En la figura 97se observa cierto comportamiento cíclico con un periodo anual en las ventas y una leve tendencia al crecimiento. En tal ejemplo, se puede deducir una aplicación con enfoque al pronóstico de ventas y por lo tanto en producción, si fuera el caso. Por ello, detectar la presencia de una tendencia es importante.

Para análisis de un proceso, la tendencia es como se ha mencionado más una advertencia de cambios en el proceso, por lo que se espera que la misma sea un valor tendiente a cero.

Figura 97. **Serie de tiempo de ventas mensuales de baterías**



Fuente: elaboración propia, con base en la función plot(*ts*).

El siguiente ejemplo corresponde a un tipo de serie de tiempo con datos del tipo económico, por ejemplo, se tiene el tipo de cambio de referencia diario según el Banco de Guatemala. La definición del tipo de cambio de referencia se resume en el "promedio ponderado de los tipos de cambio correspondientes a la totalidad de las operaciones de compra y de venta de divisas realizadas por las instituciones que constituyen el Mercado Institucional de Divisas, establecido a las 18:00 horas del mismo día hábil bancario al que corresponda"[27].

```
> tc<-ts(tc[,2],start=c(2015,1,1), frequency=365); tc
```

[27] Banco de Guatemala, <www.banguat.gob.gt>. [Consulta: septiembre de 2015].

Time Series:

Start = c(2015, 1)

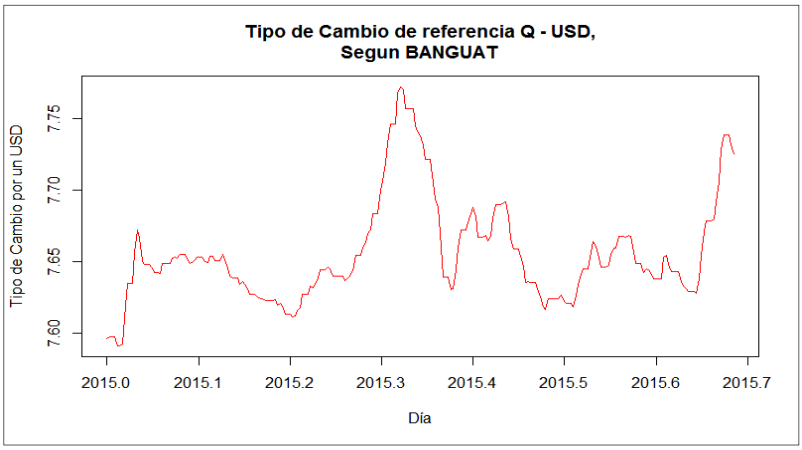
End = c(2015, 251)

Frequency = 365

[1] 7.59675 7.59748 7.59748 7.59748 7.59133 7.59153 …..

[10] 7.63526 7.63526 7.65646 7.67197 7.66638 7.65046 …..

Figura 98. **Tipo de cambio de referencia a corto plazo**



Fuente: con datos obtenidos de Banguat, elaboración propia, con base en función plot(*ts*).

El siguiente ejemplo es una serie de la misma variable de un periodo de tiempo más amplio:

Time series:

Start = c(2010, 1)

End = c(2015, 253)

Frequency = 365

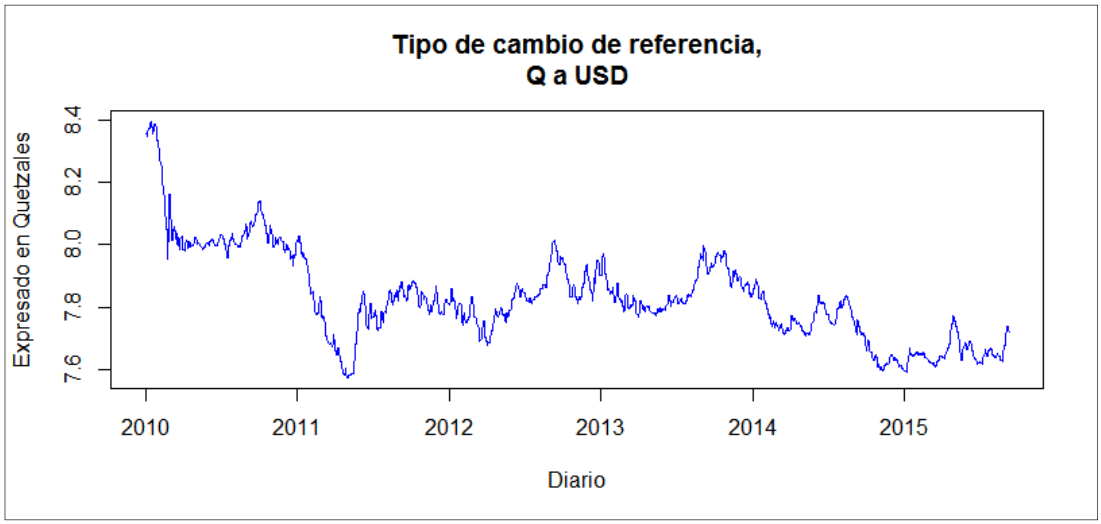
[1] 8.35439 8.35439 8.35439 8.34950 8.34451 8.36037 8.37178 ……

[10] 8.37356 8.37895 8.38522 8.39482 8.39003 8.37492 8.37492 ……

[19] 8.35503 8.36222 8.36858 8.38402 8.38402 8.38402 8.38662 ……

[28] 8.35386 8.33528 8.33528 8.33528 8.32093 8.30565 ……

Figura 99. **Tipo de cambio de referencia a largo plazo**



Fuente: elaboración propia, con base en función plot(*ts*), con datos obtenidos de BANGUAT.

La anterior serie de tiempo representa de forma adecuada las variaciones de una variable económica en el tiempo. En cuestiones de series de tiempo de ventas, producción o medición de algunas variables en control estadístico de la calidad, representan desde luego una herramienta inicial, en particular, si se tratase de corrimientos en la media, de la cual se ha aprendido con las herramientas y terminología de la literatura de control de calidad, las implicaciones de no detectar corrimientos o que implícitamente en la serie de tiempo se presenten rachas, se han mencionado ya con algunos criterios de las reglas Western Electric.

Las series de tiempo siguen un orden cronológico de muestras en el tiempo, no obstante, la muestras de algunas variables o atributos no necesariamente corresponden a valores dispuestos inmediatamente uno

después de otro, ya que dependerá del plan de muestreo que se adopte, es decir que pueden existir discontinuidades, registros vacíos o valores “NA”.

En los siguientes incisos se discuten las funciones propias del núcleo central de R y otras contenidas en paquetes, que se adaptan para trabajar con series de tiempo, en el sentido estricto de la palabra, para propósitos de pronóstico; por otro lado, las funciones y filtros pueden indicar que una determinada serie de tiempo en el sentido de muestras de datos para control de calidad por variables o atributos, tiene cierto comportamiento que se calificaría como sistemático o no aleatorio, tal es el caso cuando hay presencia de autocorrelación. El fenómeno de autocorrelación es evidente cuando hay señales ocultas bajo el ruido blanco, que se asemejan a oscilaciones armónicas

Lo anterior implica que no hay una variación natural de la variable bajo análisis, que implique que los límites de control calculados inicialmente en el control estadístico de calidad, se vean sesgados por oscilaciones atribuibles a causas asignables; por ello es necesario detectarlos y filtrar los valores que estén influenciados por valores temporales sucesivos.

5.1.3. Los rezagos y diferencias

Se dice que existen razones psicológicas, tecnológicas e institucionales para la existencia de los rezagos[28]. Un modelo con variables explicativas actuales y rezagadas se denomina modelo de rezagos distribuidos, los cuales no son parte de análisis en este texto; más bien se abordan aquellos modelos dinámicos en los cuales la variable dependiente está entre las variables explicativas rezagadas en el tiempo; este tipo es el modelo autorregresivo.

[28]GUJARATI, Damodar. *Econometría*. p. 622.

Por ejemplo, al calcular pronósticos industriales se observa que las cantidades vendidas de un determinado artículo, se asemejan a las que registró el mismo mes del año anterior. Sin importar la longitud del rezago *k;* un modelo autorregresivo simple se presenta de la forma siguiente:

$$X_t = \alpha X_{t-k} + u_t$$

Donde el rezago *k* es igual a 1 y u_t es el término de error, el cual quiere decir que las ventas en unidades X_t se ven explicadas por las ventas del periodo $t - 1$ más un término de error o aleatorio. En el siguiente tema de los modelos dinámicos se detalla aún más lo anteriormente expuesto; no obstante, es necesario estudiar las funciones con las que R permite tratar a los rezagos. Retomando el objeto de serie de tiempo **baterias**, se calcula su rezago de orden *k*= 1, mediante la función *lag*(*x,k*) cuyos argumentos principales son:

Tabla XXXV. **Argumentos función *lag***

Argumento	**Descripción**
x	Un vector, matriz u objeto de serie de tiempo simple o multivariable.
k	Es el número de rezagos por unidad de tiempo observados.

Fuente: elaboración propia.

Donde el número de rezagos puede ser especificado positivo o negativo; si *k* = 1, la serie se corre en una unidad de tiempo más temprano, es decir que el periodo *t* de la serie actual, se empareja con el periodo *t* + 1 de la serie con rezagos.

Si el rezago de serie actual se le aplica, se especifica en $k = -1$; el periodo t se empareja con el periodo $t - 1$ del rezago.

```
> bateria.k.1<-lag(baterias,1)
> bateria.k.1
      Jan  Feb  Mar  Apr  May  Jun  Jul  Aug  Sep  Oct  Nov Dec
2009                                                       122
2010  81  100 166   98   71  201  189  72   66   90  181  111
2011  81  153 112   88   66  218  170  67   72   78  202   99
2012  78  143 113  100   72  227  200  98   68   81  199  101
2013  82  100 165  101   74  232  204 101  73   82  209  112
2014  93  112 148  133   78  238  209 110  82   98  234  118
2015 115  121
> bateria.k.1.b<-lag(baterias,-1); bacteria.k.1.b
      Jan  Feb  Mar  Apr  May  Jun  Jul  Aug  Sep  Oct  Nov Dec
2010      122   81  100  166   98   71  201 189  72   66   90
2011 181  111   81  153 112   88   66  218 170  67   72  78
2012 202   99   78  143 113 100   72  227  200   98   68  81
2013 199  101   82  100 165 101   74  232 204 101   73  82
2014 209  112   93  112 148 133   78  238  209  110   82 98
2015 234 118 115  121
```

Para visualizar el efecto del rezago en $k = 1$ unidades de tiempo, se utiliza la función *ts.plot*(*…,gpars*), donde '….' es el conjunto de los objetos de tiempo a plotear y *gpars* es el conjunto de parámetros gráficos.

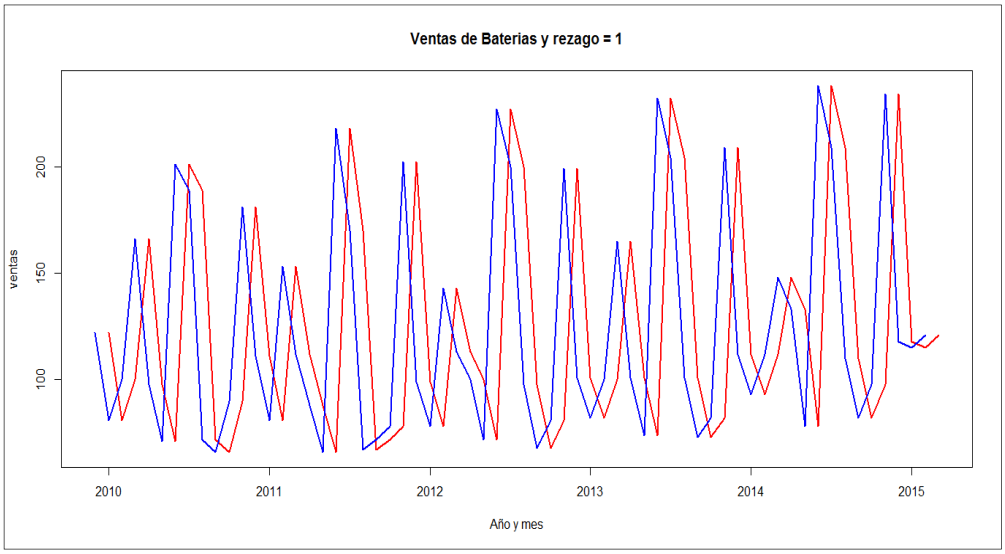
```
> ts.plot(baterias,bateria.k.1,gpars=list(xlab="Año y mes",
+ ylab="ventas",lwd=2, col=c(2,4),main="Ventas de Baterias y rezago = 1"))
```

Notar en la figura 100 el desfase entre la serie original y la de rezagadas en k = 1; la utilidad es baja con este orden de rezago, ya que se había observado que el componente cíclico de dicha serie es mayor a un mes.

En el siguiente ejemplo se muestra una serie de tiempo con un rezago de *k* = -12, donde se puede apreciar que en efecto la serie observada tiene cierto comportamiento igual al mismo mes del año anterior; notar la diferencia en el eje X, que representa la línea del tiempo, ya que en la serie con rezago, los datos se trasladan hasta el 2016.

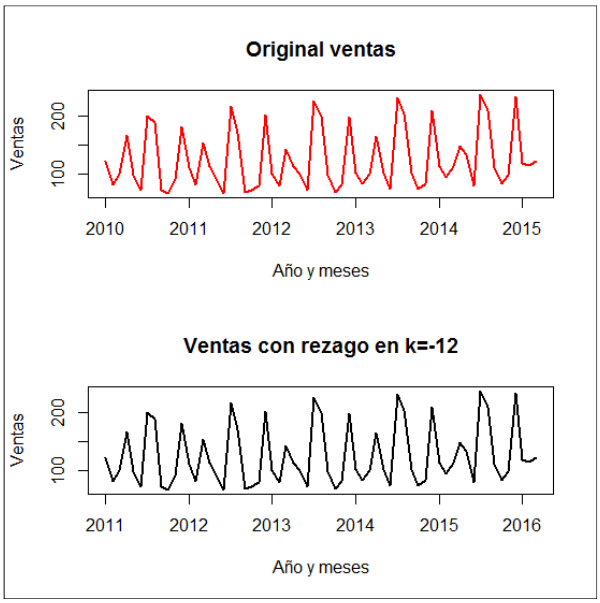
```
> ts.plot(baterias,bateria.k.12.b,gpars=list(col=c(2,4),ylab="Ventas",
+ xlab="Año y meses",main="Ventas y rezago en k=-12"))
```

Figura 100. **Gráfica de serie de tiempo y rezagos**



Fuente: elaboración propia, con base en función ts.plot().

Figura 101. **Serie de tiempo y rezago k = -12**



Fuente: elaboración propia, con base en la función ts.plot().

En el análisis de series de tiempo también se utilizan las diferencias entre el valor del periodo *t* y el valor del periodo *t* – *d*, cuya técnica se emplea para eliminar la autocorrelación u obtener una serie de tiempo estacionaria, cuando la serie original es no estacionaria.

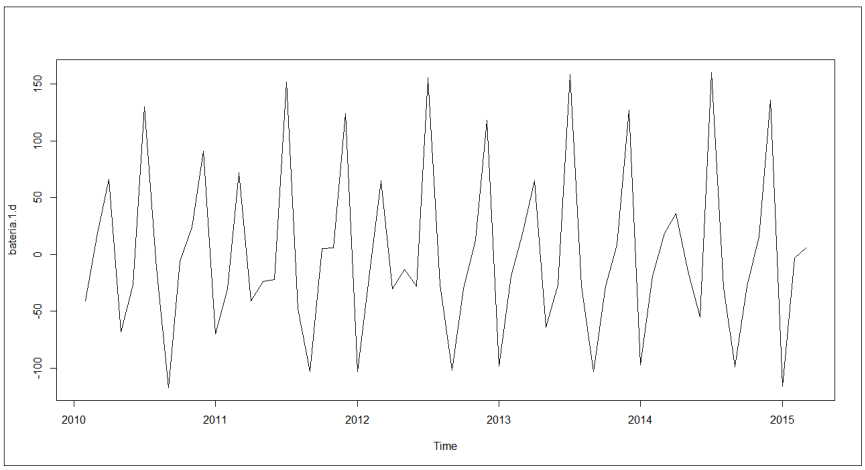
La función de R para este objetivo es *diff* (*x, lag, differences*), en el que el único argumento adicional a la anterior función es "*differences"* que especifica el orden *"d"* del modelo.

Si la primera diferencia hace de una serie de tiempo "no estacionaria" una "estacionaria" se dice que tiene un orden de integración "d" igual a 1.

Siguiendo con el ejemplo de ventas de baterías, se calcula la serie de primera y segunda diferencia como:

```
> bateria.1.d<-diff(baterias,differences=1); bacteria.1.d
  Jan  Feb  Mar  Apr  May  Jun  Jul  Aug  Sep  Oct  Nov  Dec
2010          -41   19   66  -68  -27  130  -12 -117   -6   24   91
2011  -70  -30   72  -41  -24  -22  152  -48 -103    5    6  124
2012 -103  -21   65  -30  -13  -28  155  -27 -102  -30   13  118
2013  -98  -19   18   65  -64  -27  158  -28 -103  -28    9  127
2014  -97  -19   19   36  -15  -55  160  -29  -99  -28   16  136
2015 -116   -3    6
> ts.plot(bateria.1.d)
```

Figura 102. **Serie de tiempo con primera diferencia**



Fuente: elaboración propia, con base en función ts.plot(*ts*).

5.1.4. Modelos dinámicos

Un modelo dinámico es una relación entre variables en distintos momentos de tiempo, al contrario de un modelo estático, que es formulado

considerando relaciones endógenas y exógenas, una de tales relaciones no es el aspecto temporal. Un modelo autorregresivo es un buen ejemplo de un modelo dinámico, en el que la influencia de los valores pasados es la principal relación que caracteriza al modelo.

5.1.4.1. Modelo AR

Una serie de tiempo en el sentido estricto para efectos de pronóstico puede verse representada por una variable aleatoria en una regresión lineal sobre sí misma rezagada en el tiempo, es decir que depende linealmente de sus valores anteriores.

$$X_t = \alpha X_{t-1} + u_t$$

En la ecuación anterior se dice que es un proceso autorregresivo de primer orden conocido también como esquema autorregresivo de primer orden de Markov, ya que la variable aleatoria X depende de su valor inmediato anterior; de nuevo es de primer orden o **AR(1)**, porque está rezagado en una unidad de tiempo. El término u_t es el ruido blanco o por sus siglas como IIDN, el cual es independiente e idénticamente distribuido, como una distribución normal, con media cero y varianza constante; se formula en términos probabilísticos como $u \sim IIDN(0, \sigma^2)$.

Por otro lado, los modelos AR, como modelos dinámicos manifiestan síntomas de autocorrelación por la misma definición funcional. No implican por ello un error, ya que existe el rezago en el tiempo entre causa y efecto. Este texto se centra en el análisis de series de tiempo expresadas como una variable aleatoria en el tiempo, por lo que también se deja fuera de análisis el posible problema de regresión espuria respecto de otra variable diferente al tiempo, es

decir los modelos de rezagos distribuidos. A continuación la forma general de los modelos dinámicos:

- Rezagos distribuidos $Y_t = \alpha + \beta_0 X_t + \beta_1 X_{t-1} + \beta_2 X_{t-2} + u_t$
- Autorregresivo $Y_t = \alpha + \beta X_t + \gamma Y_{t-1} + u_t$

Un proceso autorregresivo de orden **p**, como:

$$(Y_t - \delta) = \alpha_1 (Y_{t-1} - \delta) + \alpha_2 (Y_{t-2} - \delta) + \cdots + \alpha_p \left(Y_{t-p} - \delta\right) + u_t$$

Se dice que es un proceso AR(p) en el que el término u_t es el error aleatorio no correlacionado, con media cero y varianza constante, conocido también como el ruido blanco. Es decir que *Y* está expresado como desviaciones de su valor medio; la anterior expresión solo es posible si la serie de tiempo es estacionaria y los valores actuales únicamente dependen de los valores anteriores de Y; por eso no se observan otros regresores o coeficientes estimados de la variable dentro del modelo.

5.1.4.2. Modelo MA

Cuando el modelo se expresa en función de un promedio móvil de los términos de error presente y pasado se dice que siguen un proceso de media móvil de primer orden o **MA(1)**.

$$Y_t = \mu + \beta_0 u_t + \beta_1 u_{t-1}$$

Donde u sigue siendo el término de error o ruido blanco y μ es una constante. Al especificar un modelo de orden **q,** se dice que es un proceso **MA(q),** el cual es una combinación lineal de términos de error.

5.1.4.3. Modelo ARMA

Mientras que una combinación de los dos modelos anteriores, es mediante un modelo **ARMA(p,q)**, donde hay un orden **p** de términos autorregresivos y un orden **q** de términos de media móvil y el término θ representa una constante, por ejemplo, se tiene a $Y_t = \theta + \alpha_1 Y_{t-1} + \beta_0 u_t + \beta_1 u_{t-1}$, es decir que es un proceso autorregresivo y de media móvil.

5.1.4.4. Modelo ARIMA

Considerar el modelo $Y_t = \rho Y_{t-1} + u_t$, en el cual practicando la regresión se calcula que ρ es igual a 1 o estadísticamente significativo igual a 1, se dice que Y_t tiene una raíz unitaria, en tal caso la serie de tiempo es una caminata aleatoria. De forma alternativa, se puede escribir como:

$$\Delta Y_t = (\rho - 1) Y_{t-1} + u_t$$

$$\Delta Y_t = \delta Y_{t-1} + u_t$$

Donde Δ es el operador de la primera diferencia, es decir que equivale a $\Delta \mathrm{Y_t} = \mathrm{Y_t} - \mathrm{Y_{t-1}}$ y δ es igual a $(\rho - 1)$, por lo que si se espera que la serie de tiempo sea no estacionaria, δ debe de ser igual a cero, ya que ρ es igual a 1.

Es habitual practicar estos tipos de modelos para transformar una serie "no estacionaria" en estacionaria. Suponiendo que al ser diferenciada una vez la serie y la resultante es ahora estacionaria, se dice que la caminata aleatoria es integrada de orden 1 y sucesivamente las veces que se diferencien hasta obtener una serie estacionaria, se dice que es integrada de orden **d**, es decir es **I(d).**

En el caso de modelos que respondan a otros regresores y ambas series por separado sean caminatas aleatorias e integradas en el mismo orden d, se dice que las variables están cointegradas.

De lo anterior, una serie de tiempo “no estacionaria”, para hacerla estacionaria se debe diferenciar d veces y luego un modelo ARMA de orden (p,q) se dice comúnmente que la serie de tiempo original es ARIMA(p,d,q). Dicho técnicamente es una serie autorregresiva, integrada y de media móvil.

5.2. Paquetes en R para series de tiempo

En los incisos anteriores se introdujo el tema de las series de tiempo desde su acepción estadística y matemática, así como una breve noción de la definición de serie de tiempo como objeto de clase **“**ts**”** en R. No obstante, es preciso aclarar que hay todo una especialidad en la econometría que estudia las series de tiempo con mayor detalle.

Ahora es necesario emplear las series de tiempo y formular modelos sobre la base de los mismos para pronósticos, o para detectar de forma técnica algunos síntomas que desde la perspectiva de control de calidad o la de pronósticos industriales, son un paso de rigor para aportar garantía de la calidad y parsimonia del modelo que explique a los datos.

5.2.1. Ajuste de modelos

El ajuste de modelos, pretende aproximar un modelo matemático a los datos, para que este explique el comportamiento a largo plazo; también se determina la periodicidad de la serie (estacionalidad) y se comprueba la estacionariedad. Inicialmente se basa en el modelo aditivo. Otro modelo más

complejo y de uso habitual es el multiplicativo, además del modelo Log-Aditivo de las series de tiempo.

El modelo aditivo se formula como:

$$X(t) = T(t) + E(t) + A(t)$$

El componente de tendencia se identifica por $T(t)$, el componente estacional con $E(t)$ y el componente aleatorio con $A(t)$; la suma de los componentes equivale al valor observado $X(t)$.

Suponiendo que el componente estacional no está presente en $X(t) = T(t) + A(t)$, a tal modelo se le denomina "no estacional" y representa una variable en función de una tendencia en el tiempo $T(t)$, más un componente de ruido blanco $A(t)$. Al respecto de la tendencia, los métodos para estimarla son:

- Ajustar la función a un modelo lineal
- Suavizar los valores de la serie de tiempo
- Utilización de diferencias y ajustar a un modelo lineal

En efecto, existe una variedad más amplia de combinaciones a partir de modelos básicos; tal y como lo señala la tabla XXXVI; todos los modelos están en función del tiempo, lo diferente es la relación funcional que refleja la gráfica de la tendencia.

Ejemplo de formas funcionales de los modelos lineales se presentan en la tabla siguiente:

Tabla XXXVI. **Modelos lineales**

Modelo	Función de tendencia
Lineal	$T(t) = \alpha + \beta t$
Exponencial	$T(t) = \alpha e^{\beta t}$
Logarítmico	$T(t) = \alpha + \beta \ln(t)$
De potencia	$T(t) = \alpha t^{\beta}$
Recíproco	$T(t) = \alpha + \beta \frac{1}{t}$

Fuente: elaboración propia.

5.2.1.1. Modelos lineales de la tendencia

En relación con los modelos lineales, se hace uso de tabla XXXVI para entender cuál es la sintaxis para formular un modelo en R; en tal sentido se hace uso de la función *lm*( ) y del objeto "fórmula".

En primer lugar, se ajusta un modelo aditivo de la forma $T(t) = \alpha + \beta t$, donde se asume que la variable $T(t)$ es el objeto **X** donde se almacenan como un vector los valores de la serie de tiempo (en este caso es única variable) y t son los valores del vector **t** que almacena la cronología de los periodos observados de la serie de tiempo **baterias.** Observar en las siguientes líneas de resultado, la forma en que se especifica el modelo lineal: en este caso, X es proporcional al tiempo " X ~ t ".

```
> class(baterias)
[1] "ts"
> X<-as.vector(baterias)
```

```
> length(baterias)
[1] 63
> t<-1:length(baterias)
> bateria.lm.1<-formula(X~t)
> class(bateria.lm.1)
[1] "formula"
```

Con la formula **bateria.lm.1** recién expuesta, dentro de sus propiedades, se incluyen los valores de los coeficientes del modelo formulado.

```
> summary(lm(bateria.lm.1))
Call:
lm(formula = bateria.lm.1)
Residuals:
Min     1Q Median     3Q    Max
-57.49 -43.57 -21.15  43.53 104.51
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 112.5443    13.4247   8.383    9.66e-12 ***
t                 0.3808      0.3647    1.044    0.301
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
Residual standard error: 52.64 on 61 degrees of freedom
Multiple R-squared:  0.01756,   Adjusted R-squared:  0.00145
F-statistic:  1.09 on 1 and 61 DF,  p-value: 0.3006
```

Puede apreciarse que según el modelo $T(t) = \alpha + \beta t$ el valor de la pendiente β es igual a 0,38; con un coeficiente de determinación o bondad de ajuste es tan solo de 0,01756, por lo que se puede interpretar, que el modelo no se ajusta a los datos.

Un modelo alternativo, tomando el logaritmo de la variable **t**, es decir que la variable X es proporcional al logaritmo natural de t, " X ~ log(t) " especificado en R de la siguiente forma:

```
> bateria.lm.2<-formula(X~log(t))
> bateria.lm.2
X ~ log(t)
> summary(lm(bateria.lm.2))
Call:
lm(formula = bateria.lm.2)
Residuals:
Min     1Q Median     3Q    Max
-58.94 -45.09 -17.61  41.77 107.89
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  103.708     24.578   4.220    8.26e-05 ***
log(t)             6.589       7.416    0.888    0.378
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
Residual standard error: 52.77 on 61 degrees of freedom
Multiple R-squared:  0.01277,   Adjusted R-squared:  -0.003409
F-statistic: 0.7893 on 1 and 61 DF,  p-value: 0.3778
```

Sin embargo el R^2 sigue siendo bajo, por lo que se recomienda seguir experimentando con los modelos previstos en la tabla XXXVI, llevados a la forma del modelo aditivo.

La siguiente tabla resume los resultados con los coeficientes y el R^2 obtenido.

Tabla XXXVII. **Resultado de modelos lineales**

Modelo	Fórmula en R	Resultado
$\boldsymbol{T(t) = \alpha e^{\beta t}}$	log(X) ~ t	Coeficientes poco significativos, R^2 bajo.
$\boldsymbol{T(t) = \alpha t^{\beta}}$	log(X, 10) ~ log(t, 10)	Coeficientes poco significativos, R^2 bajo.
$\boldsymbol{T(t) = \alpha + \beta \frac{1}{t}}$	X ~ 1/t	No está linealizado.

Fuente: elaboración propia.

Ahora existen suficientes argumentos estadísticos para aseverar que no hay una tendencia definida bajo los modelos expuestos. Al menos linealmente no tiene ninguna bondad de ajuste. Sin embargo, algunos expertos recomiendan no despreciar los resultados y atender al R^2 mayor, el cual le corresponde al modelo $T(t) = \alpha + \beta t$. Esto implica que es posible calcular un pronóstico, al menos de corto plazo, utilizando los atributos heredados del objeto obtenido mediante la función *lm*( ), por medio de la función *predict*( ), la cual necesita un objeto tipo **lm** o "linear model".

```
> bateria.lm.1
X ~ t
> bateria.pre.1<-lm(bateria.lm.1)
> bateria.pre.1
Call:
lm(formula = bateria.lm.1)
Coefficients:
(Intercept)          t
  112.5443      0.3808
```

```
> class(bateria.pre.1)
[1] "lm"
> head(predict(bateria.pre.1))
1        2        3        4        5        6
112.9251 113.3059 113.6867 114.0675 114.4483 114.8291
```

Los anteriores resultados muestran el pronóstico de acuerdo con los periodos observados, por lo que se espera, dado el modelo, una línea recta. Para ello se hace coerción del vector obtenido como una serie de tiempo definida como la serie original **baterias**.

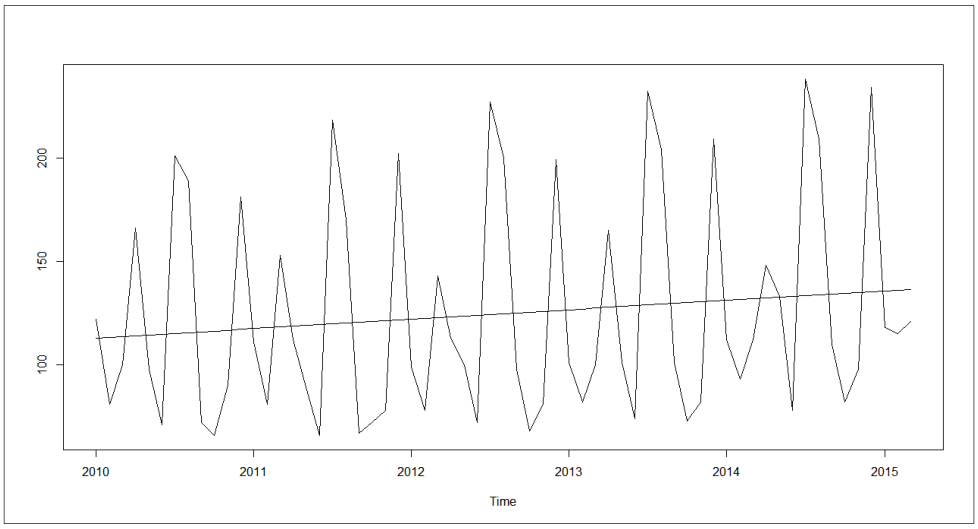
```
> p.baterias<-ts(predict(bateria.pre.1),start=c(2010,1),frequency=12)
> head(p.baterias)
     1        2        3        4        5        6
112.9251 113.3059 113.6867 114.0675 114.4483 114.8291
```

Ahora es necesario plotear ambas series para constatar el contraste del pronóstico con la serie original. La línea de tendencia $T(t)$ se establece como:

$$T(t) = 112{,}54 + 0{,}381t$$

Sin importar el grado de ajuste del modelo a los datos, la tendencia existe sin tomar en cuenta el grado de ajuste y variabilidad estacional respecto de la tendencia. Al ampliar el uso de los argumentos de la función *predict*( ), es posible hacer un pronóstico más allá de la serie de tiempo original; esto es posible mediante el argumento "*newdata*" el cual puede ser un objeto del tipo *data frame* para alojar los datos que tomará la variable explicativa; en este caso, los valores del tiempo *t* mayor a $t = n$.

Figura 103**.** **Serie de tiempo con pronóstico de ajuste lineal**



Fuente: elaboración propia, en base a función plot(*ts*).

En resumen, se han seguido los siguientes pasos para llegar hasta el anterior cálculo:

- Importar los datos externos de la serie de tiempo a un *data frame.*
- Crear el objeto de serie de tiempo definiendo el inicio y periodicidad.
- Graficar la serie de tiempo.
- Conocer y experimentar el funcionamiento de los rezagos.
- Crear un objeto de fórmula con el modelo a ajustar.
- Determinar los coeficientes del modelo con la función *lm*( ).
- Probar con otros modelos lineales para contrastar los resultados.
- Guardar en otro objeto los resultados de la regresión que más se ajusta o la seleccionada en otro objeto.
- Utilizar el objeto del modelo ajustado para pronosticar o trazar la “media” de la línea recta que identifica la tendencia por medio de la función *predict*( ).

- Graficar la serie original con la línea de tendencia.

Otra forma de calcular un criterio para el mejor modelo es utilizar la función *AIC*( ) que simula el *Akaike's Information Criterion*, el cual en esencia minimiza sus valores y el criterio final consiste en escoger aquel modelo que calcule el menor estadístico AIC, mientras que para el enfoque Bayesiano se utiliza el estadístico BIC. Citando a los anteriores modelos, se calcula el estadístico de la siguiente forma:

```
> t.bateria<-1:length(baterias)
> bateria.lm.1<-lm(baterias~t.bateria)
> class(bateria.lm.1)
[1] "lm"
> AIC(bateria.lm.1)
[1] 682.1631
> bateria.lm.2<-lm(baterias~log(t.bateria))
> AIC(bateria.lm.2)
[1] 682.469
```

Los resultados son muy similares, sin embargo, los resultados numéricos, pese a la aproximación entre sí, siguen constituyendo un criterio, por ello, según el estadístico AIC, el modelo lineal es el indicado para continuar con la descomposición de la serie de tiempo.

Restando el componente tendencia $T(t)$ según el modelo aditivo de $X(t) = T(t) + E(t) + A(t)$ a $X(t) - T(t) = E(t) + A(t)$, equivale a la simple sumatoria del componente estacional $E(t)$ más el componente aleatorio $A(t)$.

Esta operación se efectúa en R, vectorizando los valores de ambas series u objetos y obteniendo la diferencia $X(t) - T(t)$ o simplemente restando ambas

series de tiempo y asignando el vector resultante a un nuevo objeto. Los valores esperados de dicha operación se enfocan en que existan valores positivos y negativos alrededor de una media; a continuación los primeros valores de la serie y la clase del objeto resultante.

```
> E.baterias<-baterias - p.baterias
> head(E.baterias)
[1]   9.074901 -32.305908 -13.686716  51.932476 -16.448333 -43.829141
> class(E.baterias)
[1] "ts"
```

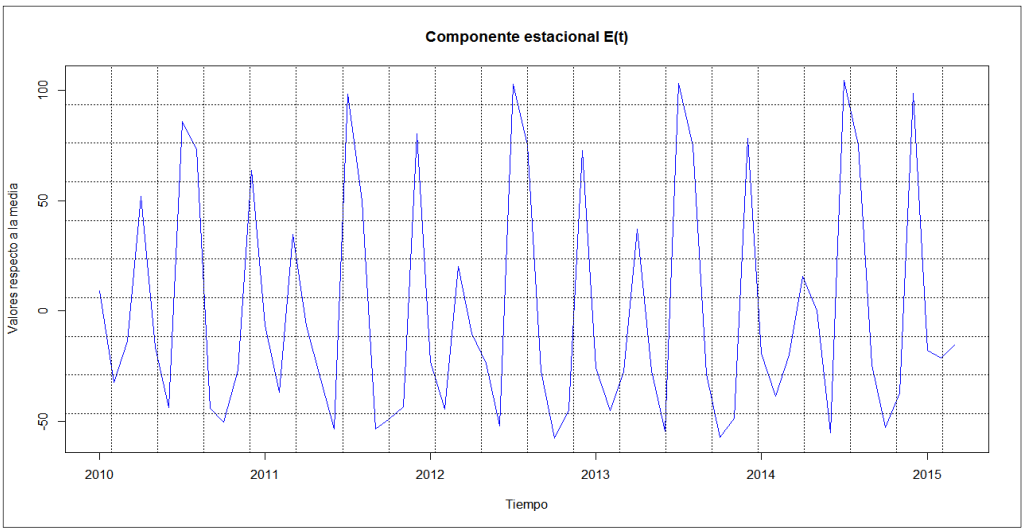
Una forma alternativa y más directa del anterior ejemplo es emplear la función *residuals*( ), la cual estima el residuo obtenido de los valores observados, menos los valores calculados de cualquier modelo. Para el cálculo es necesario contar con el objeto que contiene al modelo ajustado, es decir de clase “lm”.

```
> head(residuals(bateria.pre.1))
         1          2          3          4          5          6
 9.074901 -32.305908 -13.686716  51.932476 -16.448333 -43.829141
```

A continuación la gráfica que muestra el componente estacional y el error aleatorio.

```
> plot(E.baterias, main="Componente estacional E(t)",
+ xlab="Tiempo", ylab="Valores respecto a la media",
+ col=4)
> grid(20, 10, lwd = 0,col="black")
```

Figura 104. **Componente estacional de una serie de tiempo**



Fuente: elaboración propia, con base en función plot(*ts*).

El anterior ejercicio es una burda señal que en materia de series de tiempo, el coeficiente R^2 no será aproximado a la unidad, como lo que sucede en otros modelos con mayor grado de una relación determinística Sin embargo, un mejor ajuste puede lograrse mediante la suavización o *smoothing* de los valores de la serie de tiempo.

5.2.1.2. Suavización de las series de tiempo

Una técnica muy habitual para suavizar los datos es calculando un promedio de un subconjunto de datos de la serie, ya sea pasados o centrados respecto del valor presente " *t*" de la serie cronológica, lo que se obtiene es una serie MA, cuyas iniciales corresponden al vocablo inglés de *moving average* o promedio móvil. En R se utiliza la función *filter*( ), la cual permite calcular una serie de promedio móvil, entre otras funcionalidades. Por ejemplo, se tiene la serie de tiempo baterías y se desea calcular un promedio móvil de orden n = 3,

cuyo orden permite hacerlo centrado; por ello, entre los argumentos se utiliza el valor *sides* = 2, es decir, alrededor del rezago “0”.

```
> n<-3; ma3.baterias<-filter(baterias, rep(1/n,n), sides=2)
> ts.plot(baterias,ma3.baterias)
```

Puede observarse el uso del vector rep(1/n,n), el cual es en un factor con orden n = 3, que especifica las *n* veces que se suma el producto del valor contante 1/n por los valores de la serie original. Por ser un filtro que calcula la media móvil centrada (ver argumento *sides*=2), se pierde el mes inicial y el mes final de la serie de tiempo, por ejemplo en la serie de tiempo se tiene que las ventas de baterías para el mes de febrero 2010 fueron de 81 unidades, por medio de la media móvil, se observa que el promedio se calcula en el centro de la operación (122/3 + 81/3 + 100/3) = 101.

```
> baterias
        Jan  Feb  Mar  Apr  May  Jun  Jul  …
2010    122  81   100  166  98   71   201  …
```

En cambio, la serie de promedios móviles pierde el valor inicial y el final por no encajar en las operaciones, por eso los resultados reflejan los valores *NA*.

```
> ma3.baterias
        Jan  Feb   Mar   Apr   May   Jun   Jul   ...
2010    NA   101.0 115.7 121.3 111.7 123.3 153.7 …
```

La figura 105 muestra la superposición de la serie original y la de media móvil (MA); en esta última los sobresaltos y valles entre estos son menores debido al efecto de los promedios.

Respecto de la influencia del promedio móvil centrado, tiene que analizarse en función del modelo previsto, ya que para algunos es apropiado, mientras que para otros es óptimo promediar hacia atrás.

Ver el siguiente ejemplo en el que varía el argumento *sides* a un lado (*sides* = 1).

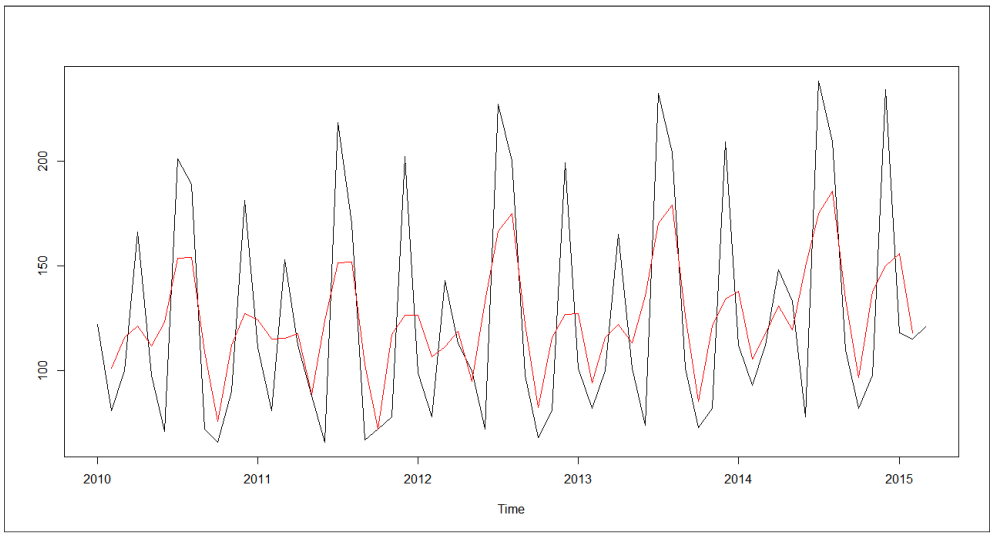
```
ma3.baterias2<-filter(baterias, rep(1/n,n), sides=1)
```

Como resultado se tiene una serie en la que debido al orden (n = 3) se pierden los primeros dos meses, no así el último, ya que el valor promedio móvil de un periodo determinado incluye al mismo (*t*), más dos meses anteriores hasta el periodo (*t* - 2), en marzo de 2010 el valor es (122/3 + 81/3 + 100/3) = 101.

```
> ma3.baterias2
Jan      Feb      Mar      Apr      May      Jun      Jul      …
2010      NA       NA   101.0 115.7    121.3    111.7    123.3        …
```

Notar el valor *NA* de los dos meses perdidos (enero y febrero de 2010), ya que el efecto que provoca llevar la serie de tiempo en promedio móvil hacia atrás, es que la gráfica siendo similar a la centrada, se traslada un periodo hacia adelante, observado en la figura 105.

Figura 105. **Media móvil sobre serie de tiempo original**



Fuente: elaboración propia, con base en función plot(*ts*).

Sin embargo también existe la técnica de la media ponderada, la cual se emplea para asignarle mayor peso a los valores recientes o a los más antiguos; la misma es conocida como *Weighted Moving Average* o MAP en español.

El objetivo es ponderar de forma diferente a los valores históricos; dichos pesos, la mayoría de las veces, son asignados bajo criterio. La mayor ponderación de valores recientes o en los valores antiguos depende del análisis previo de la gráfica de la serie de tiempo y también del criterio del analista.

En el siguiente ejemplo (figura 106) se asignan las ponderaciones *w* de la siguiente forma: $w_{t-1} = 0{,}2$, $w_t = 0{,}5$, $w_{t+1} = 0{,}3$; notar que en el vector de las ponderaciones, se especifica del más reciente al más antiguo; en este caso el mayor peso está asignado al periodo *t*; en otras palabras, está centrada la ponderación; notar además que la sumatoria de las ponderaciones es igual a

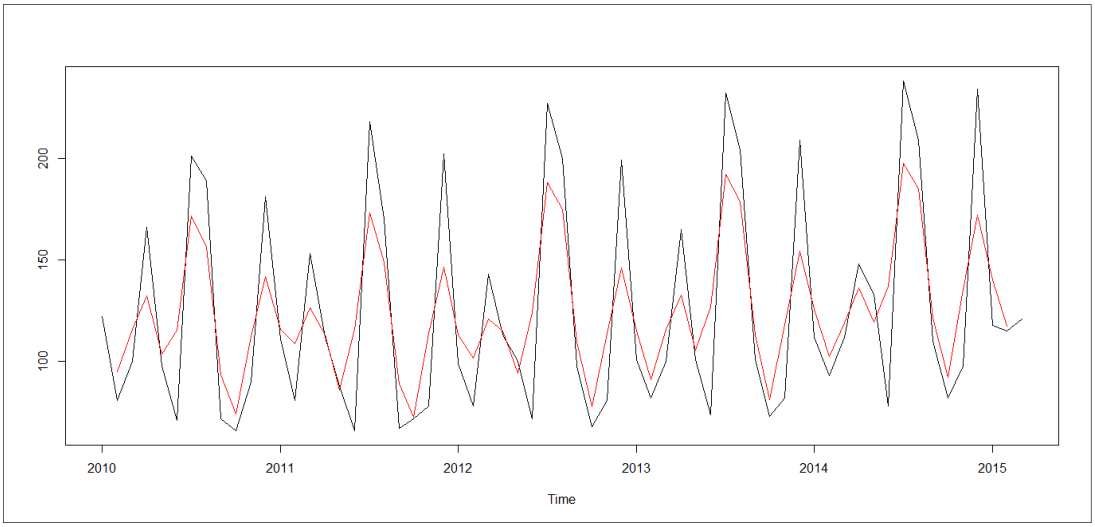
uno, lo que implica que el resultado de la sumatoria de los productos da como resultado un valor muy similar a los valores cercanos al periodo *t*.

```
map.baterias3<-filter(baterias, c(0.3,0.5,0.2), sides=2)
```

Puede apreciarse que la serie suavizada, la de promedio móvil ponderado, describe gráficamente menor amplitud de las variaciones en ambos sentidos.

Otra técnica es la conocida como la media móvil exponencial, de la cual los efectos de los valores más antiguos decrecen en forma exponencial.

Figura 106. **Promedio móvil ponderado central y serie original**



Fuente: elaboración propia, con base en función plot(*ts*).

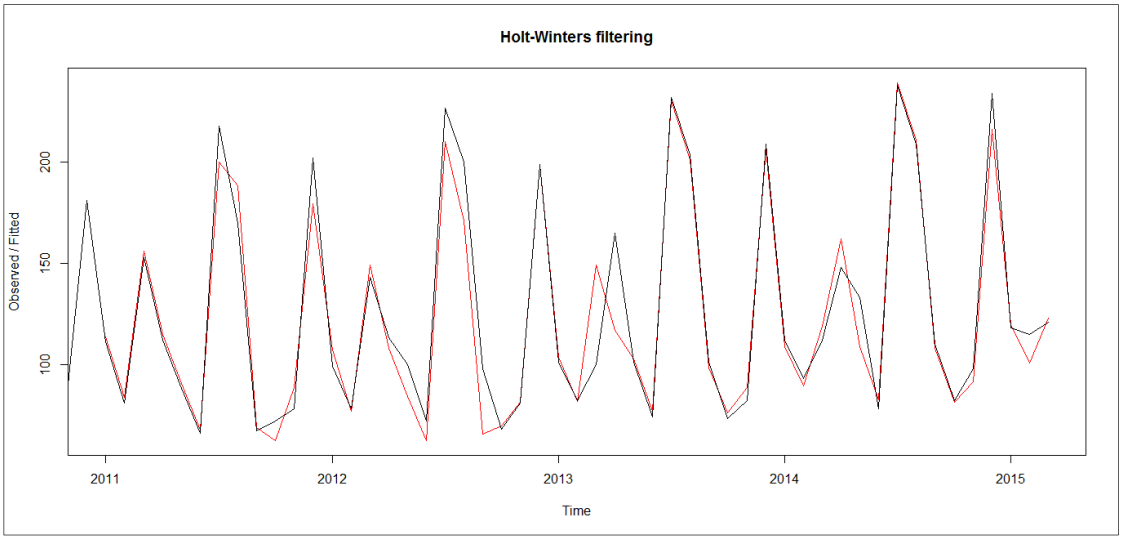
Una función que permite descomponer mediante el filtro Holt - Winters, el nombre de la función es *HoltWinters*( ); dicha función crea un objeto de clase "HoltWinters" con modo "lista" conteniendo el cálculo de los parámetros de suavizamiento "*alpha", "beta"* y *"gamma*" y por defecto, se calcula del modelo

aditivo, no obstante, es posible especificar un modelo multiplicativo.

En la figura 107 se observa al igual que en la anterior, el acompañamiento de la línea de la serie “suavizada” a la serie original. La primera no presenta los “picos” tan acentuados como en la segunda, por ello se utiliza el término suavizamiento. El mismo efecto se observa en la figura 106.

```
> baterias.hw<-HoltWinters(baterias)
> plot(baterias.hw)
```

Figura 107. **Serie suavizada con filtro Holt – Winters**



Fuente: elaboración propia, con base en función plot(*ts)*.

Entre los resultados del objeto **baterias.hw,** están implícitos los componentes del modelo aditivo; dicha información se puede extraer mediante la función *fitted*( ), la cual calcula cuatro series por medio de los parámetros obtenidos en el filtro Holt - Winters.

La primera serie es la calculada o “Xhat” que dice “X gorro” que se refiere a un valor calculado; el filtro toma al objeto suavizado como x, debido a que utiliza internamente la función “*call”* la cual retorna una función sin evaluar.A continuación está la serie de nivel o “*level*”, que viene siendo la media; luego el componente tendencia o “trend”; por último el componente estacional o “*season”,* ver figura 108.

```
> fitted(baterias.hw)
  xhat    level          trend          season
Jan 2011        113.78339     120.3718      -0.30026224   -6.288194
Feb 2011        83.50570      120.0390      -0.32845389   -36.204861
Mar 2011        156.12255     119.6812      -0.35383295   36.795139
…         ….     …..           ……           …..           ….
```

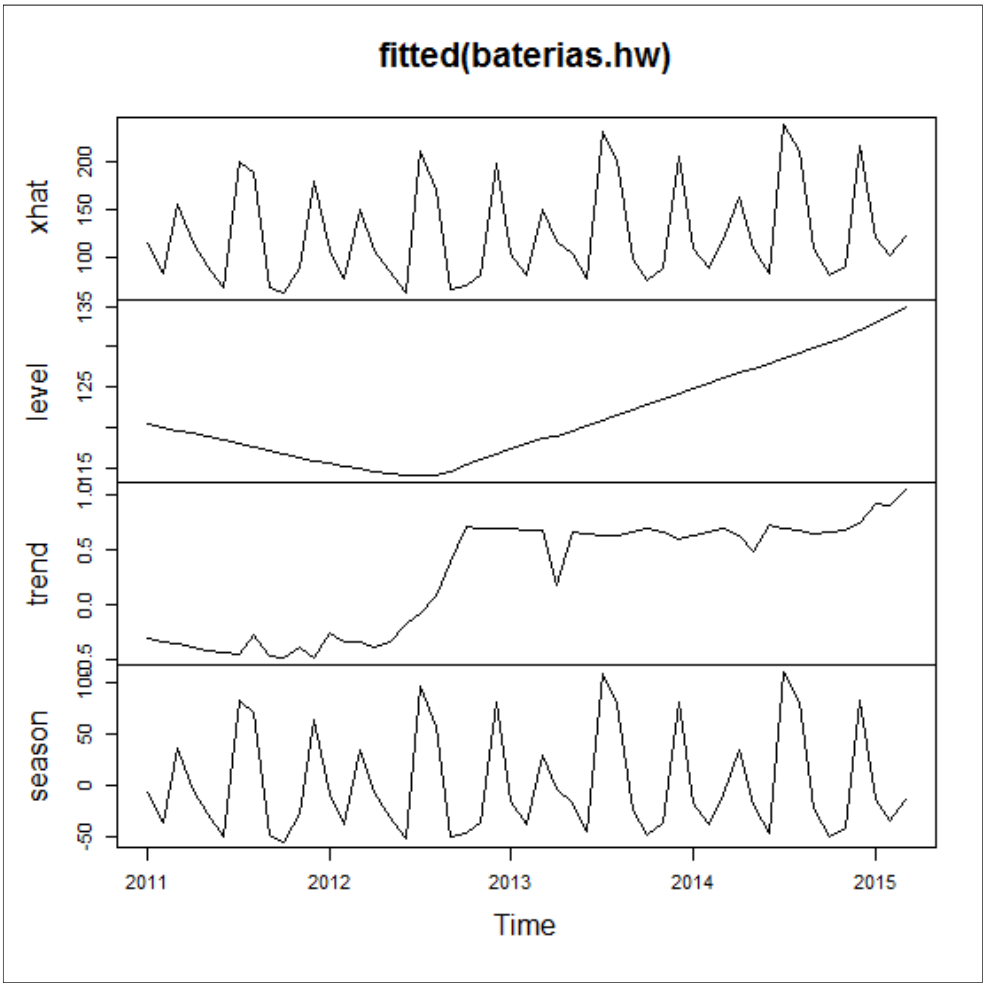
Los parámetros calculados se pueden extraer tal en forma de lista:

```
> baterias.hw$alpha
    alpha  0.01170107
```

En el objeto obtenido con *fitted*(baterías.hw), puede observarse que contiene el orden cronológico de la serie de tiempo, como tal; el objeto suavizado tiene una estructura mixta, ya que es una serie de tiempo “ts”, es una matriz “matrix”, en tal caso es una matriz de serie de tiempo “**mts**”.

```
> class(fitted(baterias.hw))
[1] "mts"    "ts"     "matrix"
```

Figura 108. **Descomposición del modelo ajustado, filtro Holt – Winters**



Fuente: elaboración propia, mediante función plot(*ts*).

Uno de los objetivos previstos para este capítulo es calcular pronóstico respecto de los datos observados; para tal efecto, empleando la metodología del filtro *Holt – Winters,* se aplican las funciones contenidas en el paquete *"forecast*". La función aplicable para continuar el desarrollo del presente tema es *forecast.HoltWinters*( ).

```
> p.baterias.hw<-forecast.HoltWinters(baterias.hw,h=12);  p.baterias.hw
Point      Forecast        Lo 80   Hi 80            Lo 95   Hi 95
Apr 2015      160.90590          142.76640      179.0454 133.16393 188.6479
May 2015       137.72564 119.58182    155.8695 109.97706 165.4742
…….              …..              ….              …          …        …..
Feb 2016      124.18822 105.56625    142.8102  95.70838 152.6681
Mar 2016      133.69882 114.94342    152.4542 105.01491 162.3827
> plot(p.baterias.hw)
```

La función genérica *forecast*( ) es utilizable para aplicarla a objetos de clase series de tiempo u objetos con clase de modelos de series de tiempo, como "*HoltWinters*" o "Arima", entre otros.

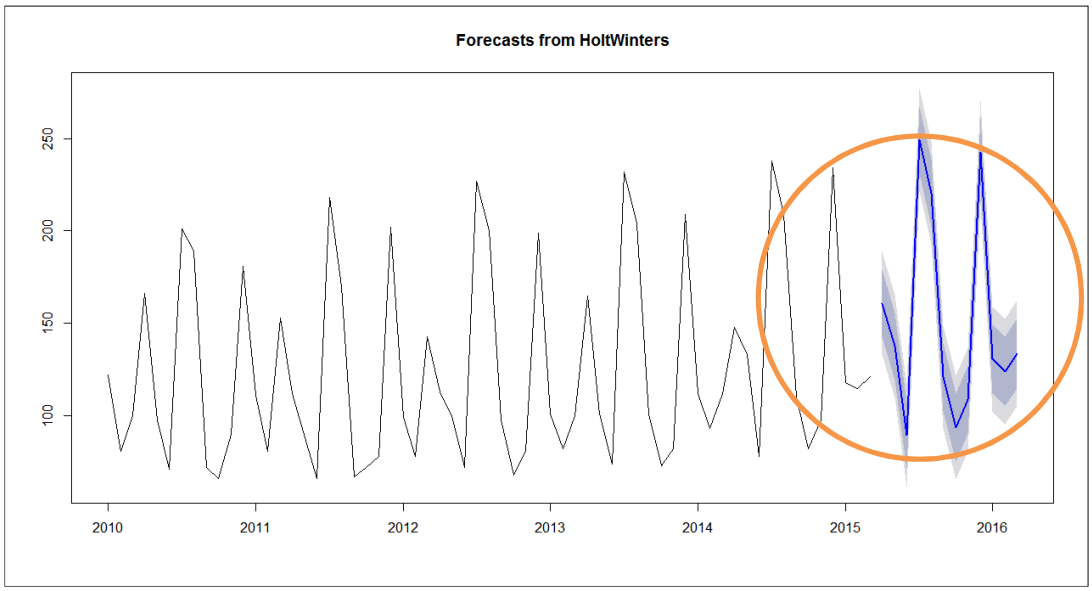
En la figura 109 del objeto **p.baterias.hw** se puede observar el uso de los niveles de confianza para los intervalos del pronóstico, los mismos se reflejan gráficamente en colores más tenues respecto del pronóstico (la línea sólida). Dentro de un círculo se ha enfatizado la zona donde figuran los intervalos de confianza.

Los intervalos de confianza son los márgenes en donde el pronóstico es estadísticamente confiable, dichos cálculos provienen de los propios intervalos de confianza de los parámetros del ajuste.

5.2.2. Filtros de desestacionalización

Hasta el inciso anterior se hizo uso de las funciones internas de R; no obstante es importante hacer ampliación del tema de la descomposición de las series de tiempo, mediante la aplicación de paquetes disponibles en el *CRAN*.

Figura 109. **Pronóstico a partir del modelo Holt – Winters**



Fuente: elaboración propia, con base en función plot(*ts*).

La utilización de los filtros es necesaria considerando lo complejo y laborioso de algunos algoritmos que simulan determinados modelos matemáticos ampliamente utilizados y que las causas que generan la estacionalidad son exógenas o que no están bajo control, pero que indudablemente influyen en la variable de estudio; por ello es necesario descomponer la serie de tiempo.

5.2.2.1. Descomposición de datos no estacionales

El filtro consiste en una serie que consta de la tendencia y del componente irregular, el cual se describe según el modelo aditivo como:

$$X(t) - E(t) = T(t) + A(t)$$

El paquete “TTR” contiene varias funciones que permiten calcular los promedios móviles al igual que con *filter*( ) con diferentes funciones.

Por ejemplo, la función *SMA*(X, n) calcula un promedio móvil simple. Para tales efectos, se considera únicamente el argumento “X”, que es el objeto que contiene la serie de tiempo; por otro lado, “n” es el número de periodos a promediar.

El resultado de la función es un objeto tipo vector numérico, por lo que es conveniente regresarlo a una serie de tiempo, si fuera necesario.

Con el ejemplo de la serie de tiempo de ventas mensuales de baterías de 12 voltios, se calcula el vector resultante.

```
> ma3.baterias.ttr<-SMA(baterias, n=3, offset=0)
> class(ma3.baterias.ttr)
[1] "numeric"
> ma3.baterias.ttr<-ts(ma3.baterias.ttr, start=c(2009,12), frequency=12)
```

Notar que se optó por correr la serie, a un mes anterior del inicio de la serie original, debido a que dicha función no tiene un argumento para centrar el promedio móvil; de esa cuenta, en este sentido resulta más eficiente la función *filter*( ), por la cantidad de pasos; no obstante el resultado debe de ser el mismo si en la función *filter*( )se argumenta el número de lados n = 1.

En el caso de los promedios ponderados, la función *WMA*(X,n,wts) calcula la serie suavizada y ponderada con el argumento “wts”. Es necesario recordar que la función hace un promedio ponderado hacia atrás, es decir, a un lado. Considerando los anteriores pesos $w_{t-2} = 0{,}2$, $w_{t-1} = 0{,}3$, $w_t = 0{,}5$, se tienen los siguientes resultados:

```
> wma.baterias.ttr<-WMA(baterias, n=3, wts=c(0.2,0.3,0.5))
> head(wma.baterias.ttr)
[1]    NA    NA  98.7 129.2 118.8  98.1
```

Al igual que el anterior ejercicio, es necesario regresar el resultado a un objeto de clase “ts”.

La actividad de calcular las series suavizadas con las anteriores funciones, no siempre da como resultado una serie que identifique claramente una tendencia más el componente aleatorio, por eso es habitual incrementar la amplitud del promedio, ya que en el ejercicio anterior únicamente se ha empleado un periodo de n = 3; no obstante puede ser de un orden mayor. Los resultados óptimos se presentan en la mayoría de veces a prueba y error.

En tal escenario, el objetivo es mostrar sobre el ejemplo un resultado que cumpla con el objetivo de reflejar la tendencia de la serie.

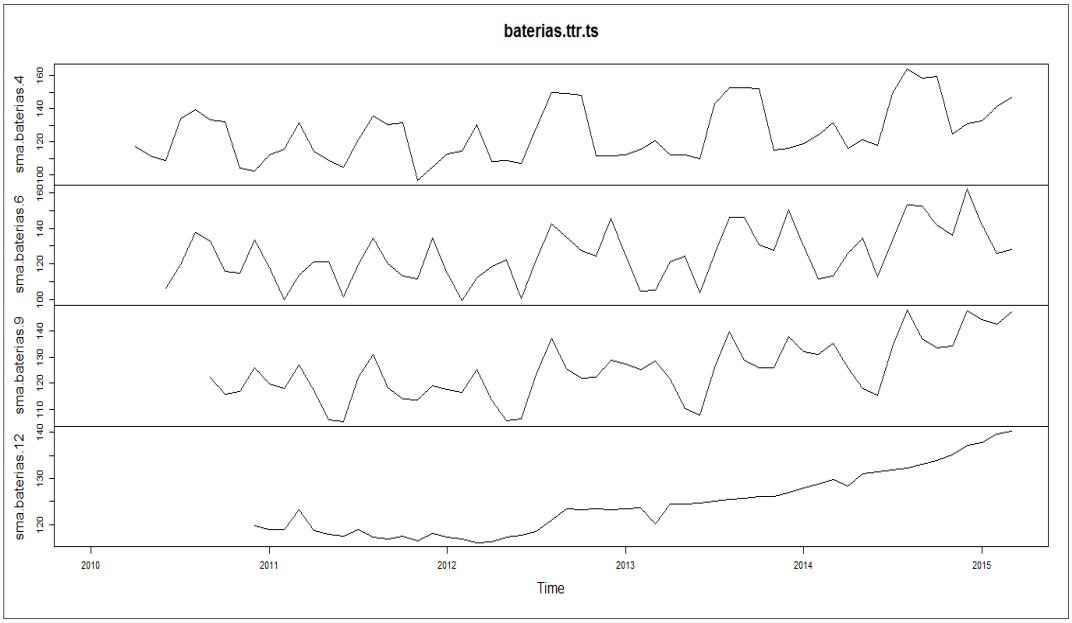
Considerando un promedio móvil simple y ampliando la cantidad de resultados con los periodos n= 4,6,9 y 12 meses, se tienen los siguientes resultados y gráficas.

```
> sma.baterias.4<-SMA(baterias, n=4)
> sma.baterias.6<-SMA(baterias, n=6)
> sma.baterias.9<-SMA(baterias, n=9)
> sma.baterias.12<-SMA(baterias, n=12)
> # Los resultados se llevan a un objeto de clase “mts”.
> baterias.ttr<-cbind(sma.baterias.4,sma.baterias.6,
+ sma.baterias.9,sma.baterias.12)
> baterias.ttr.ts<-ts(baterias.ttr,start=c(2010,1), frequency=12)
> class(baterias.ttr.ts)
```

```
[1] "mts"    "ts"     "matrix"
> baterias.ttr.ts
  sma.baterias.4 sma.baterias.6 sma.baterias.9 sma.baterias.12
Jan 2010           NA             NA             NA              NA
Feb 2010           NA             NA             NA              NA
Mar 2010           NA             NA             NA              NA
Apr 2010       117.25             NA             NA              NA
…                   …              …              …               …
```

Observar en la figura 110 que el promedio móvil con un periodo de 12 meses o en forma técnica MA(12), muestra una clara tendencia sin reflejar componentes estacionales. Dicho resultado es muy similar al obtenido mediante el filtro *Holt - Winters* en el componente de tendencia “trend” de la figura 108.

Figura 110. **Descomposición no estacional – tendencia**



Fuente: elaboración propia, con base en función plot(*ts*).

El resultado de la figura 111 indica claramente que la venta de baterías tiene una tendencia que va en franco crecimiento. Además, también se observan algunas variaciones en dicha gráfica debidas a factores de carácter aleatorio, por lo tanto se cumple con el modelo aditivo:

$$X(t) - E(t) = T(t) + A(t)$$

5.2.2.2. Descomposición total de una serie

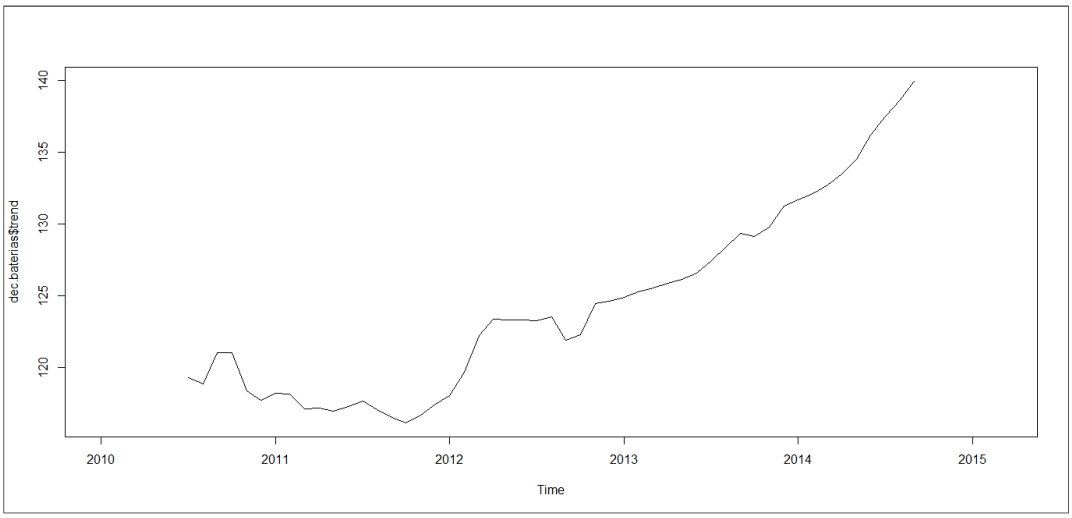
En R, la función *decompose*( ) es un filtro que descompone una serie a un modelo aditivo (por defecto) o multiplicativo. Para efectos de estudio observar el modelo aditivo:

$$X(t) = E(t) + T(t) + A(t)$$

El proceso de obtención de cada componente resulta francamente simple para calcular y manipular, obteniendo un objeto de clase “decomposed.ts”.

```
> dec.baterias<-decompose(baterias)
> class(dec.baterias)
[1] "decomposed.ts"
> mode(dec.baterias)
[1] "list"
> # Obtener grafica del componente tendencia
> plot(dec.baterias$trend)
```

Figura 111. **Serie de tiempo, componente tendencia**



Fuente: elaboración propia, con base en función plot(*ts*).

La comparación de los resultados recientes y los anteriores relacionados a la tendencia en la siguiente gráfica, notar la similitud del patrón del componente.

```
> mat<-matrix(1:3,3,1,byrow=TRUE)
> layout(mat); layout.show(length(1:3))
> plot(dec.baterias$trend); plot(baterias.ttr.ts[,4])
> fitt.baterias.hw<-fitted(baterias.hw)
> plot(fitt.baterias.hw[,3])
```

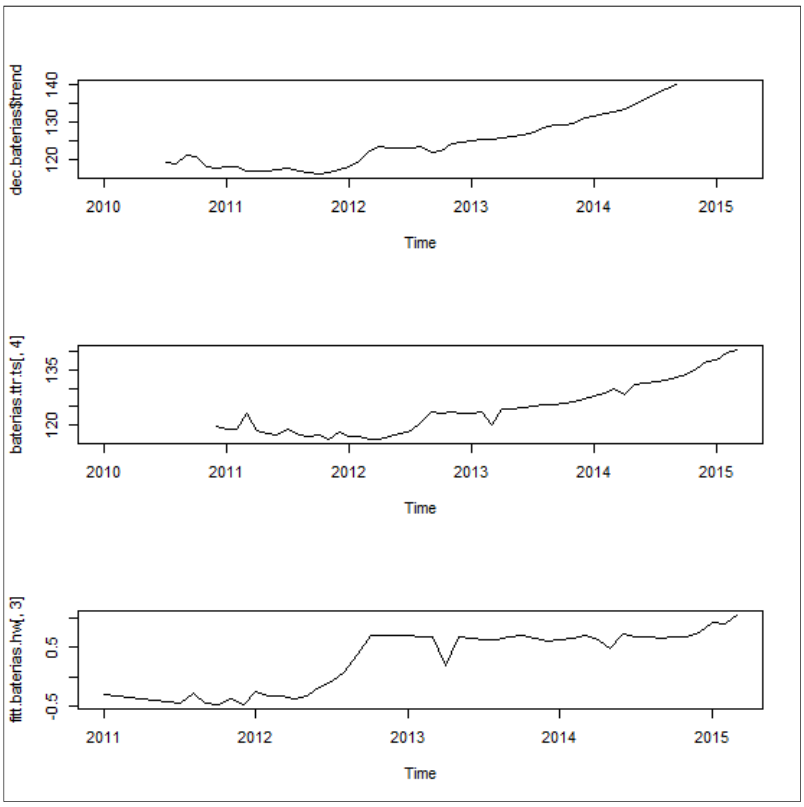
5.2.2.3. Paquete mFilter

Los supuestos iniciales de una serie integrada de segundo orden I(2),la cual es estacionaria en la segunda diferencia y con ruido blanco con una distribución normal, son aquellos que se requieren para aplicar el filtro *Hodrick - Prescott*.

El filtro *Hodrick - Prescott* es una herramienta diseñada para la extracción de la tendencia y del ciclo en una serie de tiempo. Con base en una serie de tiempo en el que t=1,2,3….T, Y_t y en la tendencia τ_t, entonces la media de las fluctuaciones viene dada por $C_t = Y_t - \tau_t$.

El componente de tendencia se calcula resolviendo y minimizando la siguiente ecuación sí y solo sí, se escoge un adecuado valor de λ.

Figura 112. **Comparación tendencia, MA(12) y *Holt – Winters***



Fuente: elaboración propia, con base en la función plot(*ts*).

Se recomienda utilizar un "*lambda*" o λ de 100 para valores anuales, 1600 para valores trimestrales y 14400 para valores mensuales.

$$\min \sum_{t=1}^{T} (Y_t - \tau_t)^2 + \lambda \sum_{t=2}^{T-1} [(\tau_{t+1} - \tau_t) - (\tau_t - \tau_{t+1})]^2$$

O resumiendo la ecuación:

$$\min \sum_{t=1}^{T} C_t^2 + \lambda \sum_{t=3}^{T} (\Delta^2 \tau_t)^2$$

Donde Δ^2es el operador de los rezagos "L" o $\Delta^2 = (1 - L)^2$.

Considerar que $\sum_{t=1}^{T} (Y_t - \tau_t) = 0$; siempre será igual a cero debido a que la tendencia calculada pasa por el centro de la serie original Y_t. Considerar ahora los cuadrados de dichas diferencias; tal y como aparecen en el primer término de la ecuación, constituyen el grado de ajuste.

En el paquete mFilter, la función *hpfilter*( ), tiene como argumento el objeto de las serie de tiempo, así como la frecuencia "freq" para el parámetro "l*ambda*", en el caso que se diseñe un modelo que use un parámetro de suavizamiento; de lo contrario el argumento es "*frequency"* cuando se utiliza una de tipo "corte" y está relacionado con el parámetro "*lambda*" como:

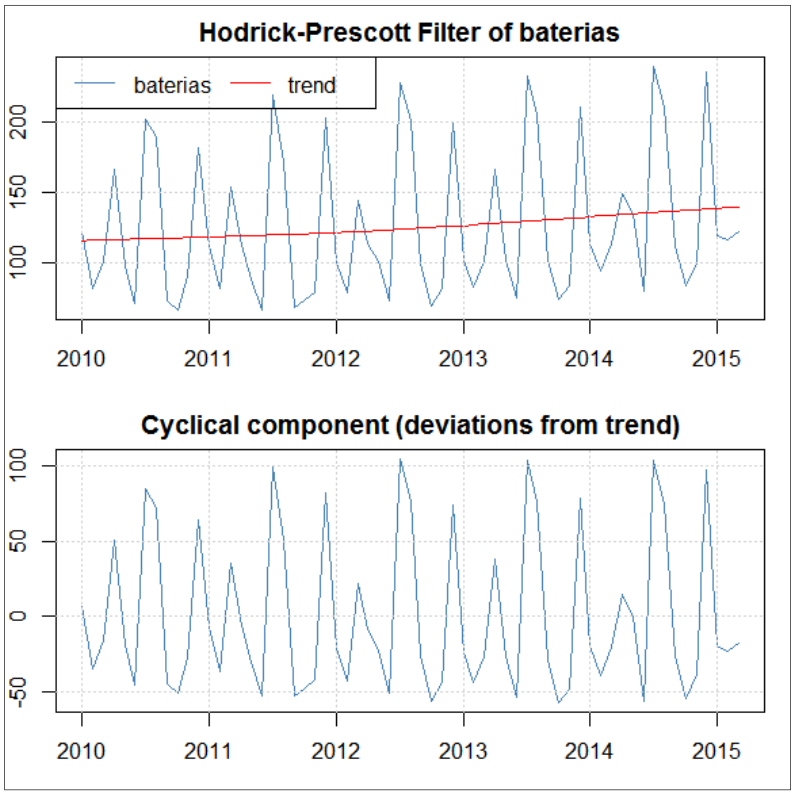
$$\lambda = \left(2 \operatorname{sen}\left(\frac{\pi}{f}\right)\right)^{-4}$$

Siendo f la frecuencia o "frequency".

Siguiendo con el ejemplo de las baterías y del tipo de cambio, se obtiene de cada serie de tiempo un objeto de clase “mFilter” de modo “lista”.

```
> library(mFilter)
> hp.baterias<-hpfilter(baterias,freq=14400,drift=TRUE)
> plot(hp.baterias)
```

Figura 113. **Serie de tiempo, filtro *Hodrick – Prescott***



Fuente: elaboración propia, con paquete mFilter.

Puede apreciarse que los resultados son bastante congruentes con los obtenidos en gráficas recién expuestas, donde se extrajo la tendencia por métodos lineales y utilizando el razonamiento matemático para aislar el

componente estacional o cíclico para el largo plazo como:

$$X(t) - T(t) = E(t) + A(t)$$

En el ejemplo del tipo de cambio, se aplica el filtro Hodrick - Prescott a dicha serie, para observar la tendencia que de acuerdo con el principio del filtro, debe hacerse notar tal y como si se dibujase una línea a mano alzada sobre la serie de tiempo original.

El valor del argumento *freq* es función de *lambda* = 100 x (número de periodos al año)^2; por ello, en el caso de los datos mensuales *lamda* = 100x12^2 = 14 400 y en el caso de ser diarios, la cifra es francamente grande 100 x 365^2 = 13322500, lo que en una serie de tiempo como la del tipo de cambio diario (despreciando la vinculación con la teoría económica) resulta en un proceso de cómputo complejo y que requiere más tiempo y recursos de memoria de la computadora, mientras que el valor de *lambda* puede prestarse a criterio; dicha acción es considerada por algunos expertos como una debilidad del filtro.

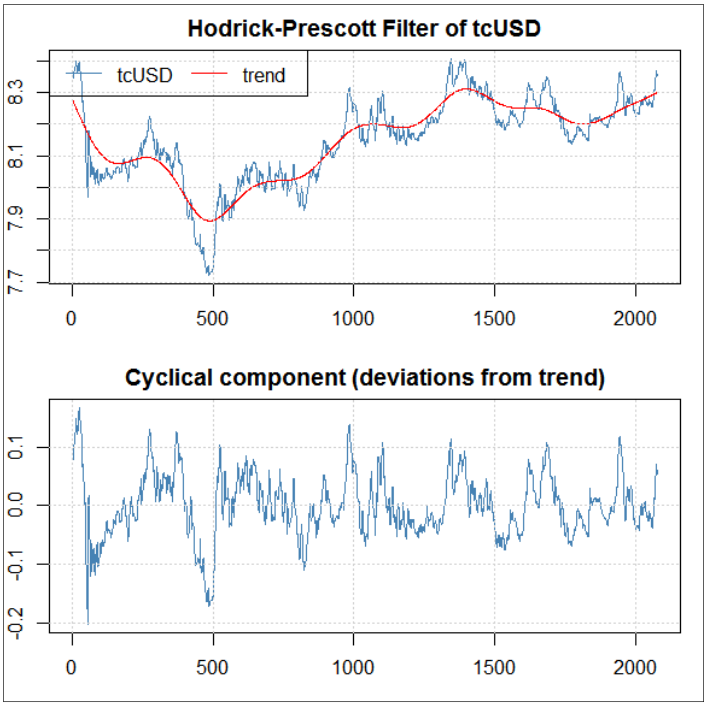
Por el contrario si *lamda* es igual a cero, el suavizamiento es nulo y la serie es igual a la original. Sin embargo los resultados presentan una mejor aproximación como si se tratase de un modelo lineal, ya que la complejidad de la serie llevaría a calcular una tendencia a partir de polinomios de grado mayor.

```
> hp.tcUSD<-hpfilter(tcUSD,freq=13322500,drift=TRUE);  class(hp.tcUSD)
[1] "mFilter"
> hp.tcUSD
Title:
 Hodrick-Prescott Filter
Call:
 hpfilter(x = tcUSD, freq = 13322500, drift = TRUE)
```

Method:
hpfilter
Filter Type:
lambda
Series: tcUSD

Notar en la siguiente figura la separación del componente cíclico, expresado como la desviación de la tendencia respecto de los valores observados. En los resultados del objeto clase “mFilter” se incluyen los valores originales de la serie y de la tendencia y los valores cíclicos o estacionales; la gráfica se obtiene empleando la función *plot*( ).

Figura 114. **Filtro *Hodrick - Prescott*, ejemplo "Tipo de cambio"**



Fuente: elaboración propia, con paquete mFilter.

Además del filtro *Hodrick – Prescott*, el paquete mFilter contiene la función para calcular los filtros *Baxter – King*, *Butterworth*, *Cristiano - Fitzgerald* y la regresión trigonométrica, los cuales son empleados habitualmente en investigación económica.

5.2.3. Prueba de estacionariedad

Según lo expuesto en el inciso 5, una simple explicación de la presencia de raíz unitaria equivale a afirmar que una serie de tiempo no es estacionaria; es que la serie tiene una tendencia o evolución a través del tiempo, que causa dificultades para inferir de forma estadística. No obstante, lo importante es que de hecho la mayoría de las series de tiempo presentan alguna tendencia; sin embargo se trata más bien de cambios en la tendencia para detectar raíz unitaria.

En las cartas de control de calidad, al existir algún corrimiento en la media, esto implica que un proceso está fuera de control, pero cuando no es evidente dicho corrimiento, se puede recurrir a la detección de raíz unitaria.

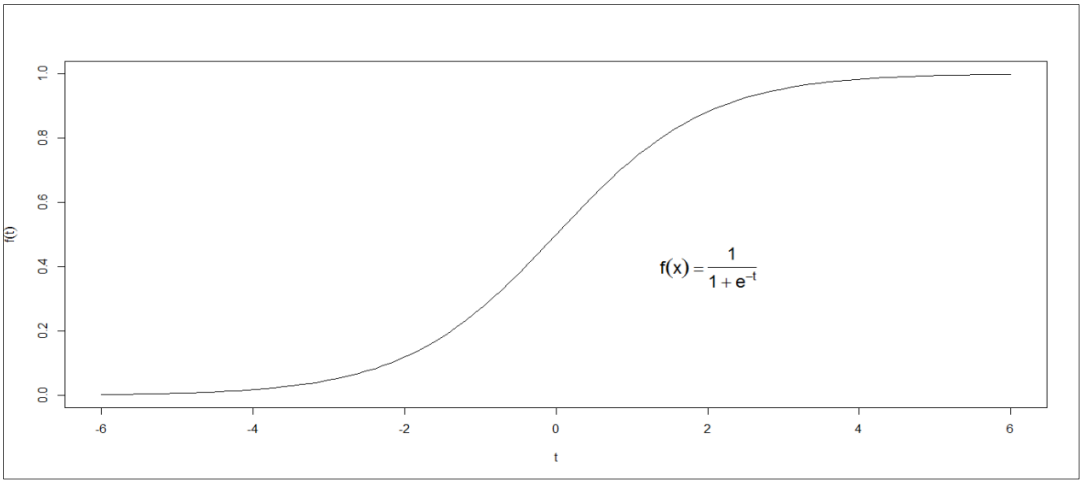
Una serie, cuyos datos fluctúen alrededor de la media se presume estacionaria; por el contrario, una serie de tiempo en la que los datos fluctúen en periodos diferentes, sobre dos o más líneas que representan una media con pendiente diferente o incluso diferente nivel, aunque con igual tendencia, se presume *a priori* como “no estacionaria”.

No obstante, no siempre es evidente la presencia de ambos escenarios; una forma funcional vinculada a la presencia de raíz unitaria es cuando la tendencia tiene un cambio similar a la función logística como la figura 115.

```
> # Código para la curva de la función logística
> curve(1/(1+exp(-x)),-6,6, xlab="t", ylab=expression(f(t)))
> text(c(3,0.4),expression(f(x)==frac(1,1+e^{-t})), cex=1.5)
```

Se puede observar que después de la “transición” la tendencia es la misma; sin embargo la media ha variado; esto rompe con la condición $E(X_t) = \mu$, por lo tanto gráficamente es una serie de tiempo “no estacionaria”.

Figura 115. **Función logística**



Fuente: elaboración propia, con base en función curve( ).

Otro ejemplo, es cuando se combina ese proceso de transición junto con una tendencia, similar a la función $X(t) = f(t) = \alpha t + \frac{1}{1+e^{-t}}$, donde α es el valor de la pendiente, que implica que el nivel es el único que ha cambiado en periodo de transición donde t = 0. En efecto, tal situación ocasiona problemas de inferencia estadística, sin embargo es habitual que se presente este tipo de transiciones, por ejemplo, en economía suceden los fenómenos de ajuste estructural o políticas económicas agresivas que provocan “quiebres” de cierta

variable respecto del tiempo; también se manifiesta en fluctuaciones abruptas de la inflación o del tipo de cambio. Un escenario permisible es cuando la transición queda dispuesta de tal forma que exista una transición “suave”.

En los fenómenos medio ambientales se presentan cambios drásticos en el promedio histórico de nutrientes en el agua por una súbita presencia de agentes contaminantes, los cuales tal y como sucede en economía, representan en ese periodo de transición un factor endógeno que provoca un cambio permanente en la media o *shock* permanente y en la tendencia registrada hasta el periodo “t”.

En control estadístico de calidad, considerando un cambio planificado en el límite de especificación central, un cambio de maquinaria y de procedimiento, y los cambios permanentes en la calidad de las materias primas, ocasiona que la media se corra a otro nivel, tal y como se ha expuesto sobre el factor *k* de corrimiento de la media de μ_0 a μ_1 y un riesgo β de no detectar el corrimiento entre cada iteración. En los pronósticos industriales el síntoma de cambios en la media representa un problema serio de confiabilidad de los pronósticos, ya que la estimación sobre modelos de mínimos cuadrados resulta en parámetros estadísticamente erróneos. Por lo anterior expuesto, la detección de raíz unitaria, debe practicarse previo a la formulación de un modelo.

5.2.3.1. Detección de raíz unitaria y prueba Dickey Fuller

Se ha mencionado de una “caminata aleatoria”[29], por medio de un modelo AR(1):

[29] COCHRANE, John. *Time series for macroeconomics and finance*. p. 106.

La ecuación 1 es: $Y_t = \rho Y_{t-1} + u_t$

Donde: $E_{t-1}(u_t) = 0$

Y se espera que: $(-1 \leq \rho \leq 1)$

En tanto que: $\rho = 1$

Se dice que hay raíz unitaria o que el proceso es una caminata aleatoria sin deriva y en forma de prueba de hipótesis; entonces, $\mathrm{H}_0: \rho = 1$ y alternativamente $\mathrm{H}_1: \rho > 1$.

Sin embargo, al practicar una regresión por mínimos cuadrados ordinarios o por su siglas MCO, el modelo de la ecuación 1 posee sesgo, ya que el parámetro "ρ" no es asintótico inclusive en muestras grandes. Por ello se plantea un modelo en primera diferencia, restando de ambos lados de la ecuación el término rezagado.

Aplicando primero la ecuación 1 $Y_t - Y_{t-1} = \rho Y_{t-1} - Y_{t-1} + u_t$

Se obtiene la ecuación 2 $\Delta Y_t = Y_{t-1}(\rho - 1) + u_t$

Y simplificando se obtiene la ecuación 3 $\Delta Y_t = \delta Y_{t-1} + u_t$

Donde el símbolo "Δ " significa que es la primera diferencia, es decir la diferencia entre el valor del periodo "t " y el valor del periodo anterior " t - 1"; el símbolo "δ " es el parámetro a calcular y es igual a $(\rho - 1)$; por ello la hipótesis nula dice $\mathrm{H}_0: \delta = 0$ y la alternativa $\mathrm{H}_1: \delta < 0$.

La prueba *Dickey – Fuller* considera que "δ " no sigue una distribución normal, por ello *Dickey* y *Fuller* desarrollaron un conjunto de tablas para contrastar dicho parámetro. Los valores para el contraste se obtuvieron por

medio de simulaciones de “Monte Carlo” de valores críticos del estadístico “DF”[30] a diferentes valores de significancia estadística y distintos tamaños de muestra. El valor calculado en contraste se conoce como "τ" o “*tau*”, el cual corresponde al valor del parámetro obtenido por mínimos cuadrados ordinarios, dividido entre su error estándar:

$$\tau = \frac{\delta}{e.e.}$$

Los valores críticos de la tabla, a la vez que dependen del tamaño de la muestra y del valor de significancia, dependen del tipo de modelo que se emplee para calcular el parámetro "δ". El primer modelo, es el de “caminata aleatoria” (ecuación 1), cuyo identificador en los argumentos empleados en las funciones de paquetes en R es “nc”; esto se detalla a continuación:

Tabla XXXVIII. **Modelos empleados en prueba *Dickey – Fuller***

Modelo	**Ecuación**	**Argumento**
Caminata aleatoria	$\Delta Y_t = \delta Y_{t-1} + u_t$	“nc”
Caminata aleatoria con deriva	$\Delta Y_t = \beta_1 + \delta Y_{t-1} + u_t$	“c”
Caminata aleatoria con deriva y alrededor de una tendencia determinística.	$\Delta Y_t = \beta_1 + \beta_2 t + \delta Y_{t-1} + u_t$	“ct”

Fuente: elaboración propia.

Los argumentos “nc”, “c” y “ct” son el identificador de “no término constante”; de “hay término constante” y “hay término constante y tendencia”, respectivamente.

[30]GUJARATI, Damodar. *Econometría*. p. 893.

El procedimiento para la prueba se practica con el paquete “fUnitRoots”, limitados únicamente a la prueba “Aumentada Dickey Fuller” o “prueba ADF”; la cual adiciona a los modelos los valores rezagados de la variable dependiente ΔY_t, que da como resultado la ecuación 4.

$$\Delta Y_t = \beta_1 + \beta_2 t + \delta Y_{t-1} + \sum_{i=1}^{m} \alpha_i Y_{t-i} + \varepsilon_t$$

De esa forma se evita el problema de autocorrelación entre los errores; por ello figura ε_t como un ruido blanco “puro”. La función para la prueba ADF es *adfTest*( ) la cual tiene como argumentos lo siguiente:

Tabla XXXIX. **Argumentos función adfTest**

Argumento	**Descripción**
X	Vector u objeto de serie de tiempo.
Lags	Número de rezagos para corrección del término de error.
type =c(“nc”,”c”,”ct”)	Tipo de modelo de regresión a emplear, el valor por defecto es “c”.

Fuente: elaboración propia.

Simplificando el procedimiento, para la obtención de la prueba *ADF* con *fUnitRoots*, existen otros criterios como el de *Schwartz Bayesian Information Criterion* o SBIC, el cual está relacionado con el criterio AIC, que sugiere que se incremente el número de rezagos hasta 14 sobre la ecuación 4,

introduciendo un término de penalización por la adición de parámetros; de esa forma se logra ajustar a un modelo dentro de un número finitos de modelos. Se emplean tres rezagos en el modelo con término constante y tendencia o “ct” y el criterio de SBIC se emplea en una segunda corrida.

El objetivo del procedimiento es obtener el estadístico “*tau*” del parámetro “δ”; el número de rezagos como se indicó es para evitar el problema de autocorrelación. Empleando el ejemplo de la serie de tiempo “baterías”, se da a conocer si la serie es “no estacionaria” (hipótesis nula).

```
> adfTest(baterias, lags=3, type="ct")
Title:
 Augmented Dickey-Fuller Test
Test Results:
  PARAMETER:
    Lag Order: 3
  STATISTIC:
    Dickey-Fuller: -5.1702
  P VALUE:
    0.01
Warning message:
In adfTest(baterias, lags = 3, type = "ct") :
  p-value smaller than printed p-value
> adfTable(trend="ct", statistic="t")
$x
[1]  25  50 100 250 500 Inf
$y
[1] 0.010 0.025 0.050 0.100 0.900 0.950 0.975 0.990
$z
  0.010  0.025  0.050  0.100  0.900  0.950  0.975  0.990
25       -4.38  -3.95  -3.60  -3.24  -1.14  -0.80  -0.50  -0.15
```

```
 50      -4.15  -3.80  -3.50  -3.18  -1.19  -0.87  -0.58  -0.24
100      -4.04  -3.73  -3.45  -3.15  -1.22  -0.90  -0.62  -0.28
250      -3.99  -3.69  -3.43  -3.13  -1.23  -0.92  -0.64  -0.31
500      -3.98  -3.68  -3.42  -3.13  -1.24  -0.93  -0.65  -0.32
Inf             -3.96  -3.66  -3.41  -3.12  -1.25  -0.94  -0.66  -0.33

attr(,"class")
[1] "gridData"
attr(,"control")
   table    trend statistic
"adf"     "ct"      "t"
```

Observar el estadístico *Dickey Fuller* o *"tau*" en el segmento *Test Results* que es igual a - 5,17, siendo más negativo que el valor de tabla, suponiendo al 1% con el mismo modelo "ct" igual a - 4,04; se rechaza la hipótesis nula de que la serie es "no estacionaria". Notar que se ha usado la función *adfTable*( ) para calcular los valores de tabla o críticos.

Otro criterio para rechazar la hipótesis nula, es observar el "p-value" igual a 0,01, que siendo igual o menor al valor de significancia (1 %) es suficiente argumento para rechazar que el valor obtenido en el estadístico *"tau*" es correcto, por lo tanto se rechaza la hipótesis nula. Ahora bajo el criterio *SBIC* los resultados difieren sustancialmente como se puede apreciar a continuación:

```
> adfTest(baterias, lags=14, type="ct")
Title:
 Augmented Dickey-Fuller Test

Test Results:
  PARAMETER:
    Lag Order: 14
```

STATISTIC:
Dickey-Fuller: -2.2516
P VALUE:
0.473

El estadístico crítico o de tabla (- 4,04) es mayor en forma absoluta al valor absoluto de "*tau*" (- 2,2516) y el *p-value* igual a 0,473; estadísticamente es mayor que el valor de significancia estadísticamente, en tal sentido, hay argumentos para aceptar la hipótesis nula H_0: de que la serie de tiempo "baterías" es "no estacionaria", empleando el criterio de *Schwartz* o *SBIC*.

Otro paquete que calcula la prueba *ADF* es "urca", en el que la función *CADFtest*( ) retorna más información relacionada con el modelo de la ecuación 4; generando un resultado con el criterio *SBIC*, se obtienen los siguientes resultados:

```
> ADFbaterias.2<-CADFtest(baterias, max.lag.y=14)
> summary(ADFbaterias.2)
Augmented DF test
                                    ADF test
t-test statistic:                    -2.2516129
p-value:                              0.4512477
Max lag of the diff. dependent variable:   14.0000000

Call:
dynlm(formula = formula(model), start = obs.1, end = obs.T)

Residuals:
   Min     1Q  Median     3Q    Max
-36.872  -5.232  -2.246   3.434  30.732
Coefficients:
```

```
  Estimate        Std. Error       t value          Pr(>|t|)
(Intercept) 230.266360          110.275039    2.088          0.04510 *
trnd          1.510226        0.436946      3.456          0.00161 **
L(y, 1)      -2.285186
  1.014911        -2.252           0.45125
L(d(y), 1)    0.897364   0.998086       0.899          0.37554
L(d(y), 2)    0.632505   0.975715       0.648          0.52160
L(d(y), 3)    0.431321   0.940284       0.459          0.64964
L(d(y), 4)    0.245592   0.867158       0.283          0.77890
L(d(y), 5)    0.149540   0.789957       0.189          0.85109
L(d(y), 6)   -0.127007   0.720439       -0.176         0.86121
L(d(y), 7)   -0.243325   0.649555       -0.375         0.71051
L(d(y), 8)   -0.520569   0.581985       -0.894         0.37796
L(d(y), 9)   -0.703660    0.513717      -1.370         0.18061
L(d(y), 10)  -0.938091   0.453798       -2.067         0.04715 *
L(d(y), 11)  -1.068587   0.396890       -2.692         0.01134 *
L(d(y), 12)  -0.348685   0.346220       -1.007         0.32167
L(d(y), 13)  -0.069934   0.250616       -0.279         0.78206
L(d(y), 14)  -0.009023   0.145976       -0.062         0.95111
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 15.17 on 31 degrees of freedom
Multiple R-squared:  0.9744,    Adjusted R-squared:  0.9611
F-statistic: 40.62 on 14 and 31 DF,  p-value: 5.418e-16
```

Observando el resultado del parámetro L(y, 1) (igual a - 2,25) o el *t-test statistic* coincide con los resultados de la función *adfTest*( ) de *fUnitRoots,* así como el *p-value* obtenido, que comprueba la necesidad irrefutable de aceptar la hipótesis nula de que la serie de tiempo observada es “no estacionaria”.

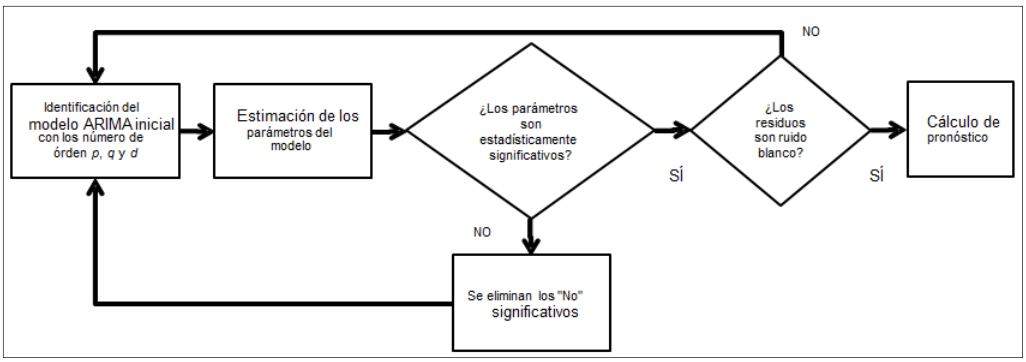
5.2.4. Estimación correcta de modelos

La estimación correcta de modelos, comprende el empleo de una metodología con una robusta capacidad predictiva. Para ello se utiliza el modelo ARIMA, y con un orden de ideas basado en el método Box – Jenkins, el cual es de carácter iterativo, ya que conduce en forma secuencial a deducir el modelo adecuado para calcular el pronóstico.

5.2.4.1. Modelos ARIMA con el método *Box - Jenkins*

El objetivo principal del modelo ARIMA es obtener los valores enteros de cada orden de “p”, “d” y “q”, según lo expuesto en la sección 5.1.4.4, para “que los datos hablen por sí mismos”, es decir, que se omite incluir otras variables explicativas; únicamente se incluyen los valores de la variable explicada rezagadas en el tiempo. El método Box – Jenkins se esquematiza de la siguiente forma:

Figura 116. **Método *Box– Jenkins***



Fuente: elaboración propia, empleando Word.

El anterior esquema implica que los modelos ARIMA están definidos de antemano para las series de tiempo estacionarias[31], quiere decir que si se inicia con una serie "no estacionaria" es necesario calcular la primera diferencia o las "d" diferencias que sean necesarias para obtener una serie estacionaria; seguido de ello, los valores de los procesos AR(*p*) o MA(*q*) para un proceso ARMA(*p, q*).

Como ya se había expuesto, la función *diff*( ) permite calcular de acuerdo con determinados argumentos explícitos, una serie de tiempo o un vector diferenciado con orden "d", por ejemplo, la serie de tiempo **baterias,** ya diferenciada y asignada a la serie de tiempo **bateria.1.d.**

```
> bateria.1.d<-diff(baterias,differences=1)
```

Al diferenciar una serie se elimina el componente tendencia, como lo que sucede en la figura 102, la que aporta fuertes indicios de que es estacionaria respecto de la media y la varianza; en otras palabras el modelo ARIMA a construir empieza con el orden (*p*,1,*q*), ya que está integrado en un orden "d" igual a 1.

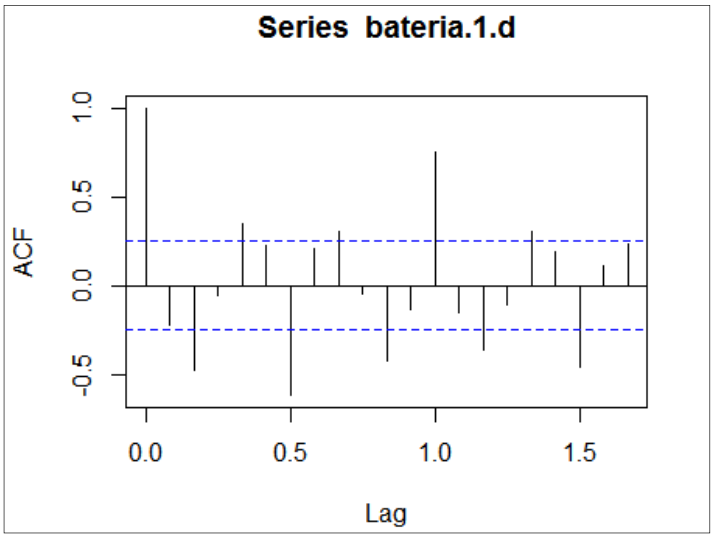
En tal sentido, en el proceso de identificación del modelo se procede a graficar el correlograma y correlograma parcial, empleando las funciones *acf*( ) y *pacf*( ), respectivamente, ambas contenidas en el núcleo central de R.

Por ejemplo, tomando la serie diferenciada **bateria.1.d** se tienen los siguientes valores y gráfica de la función de autocorrelación con un máximo de 20 rezagos. Para mostrar únicamente los valores, se especifica el valor "FALSE" en el argumento plot.

[31] COHLAN, Avril. *Little book of R for time series*. p. 45.

```
> acf(bateria.1.d, lag.max=20)
```

Figura 117. **Correlograma de serie "baterias" en primera diferencia**

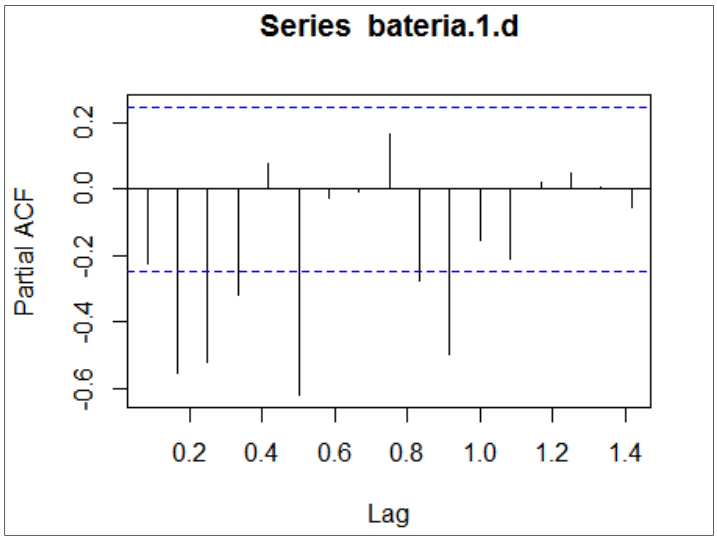


Fuente: elaboración propia, con base en la función acf(*ts*).

Observar en la gráfica anterior los parámetros que son estadísticamente significativos y que sobrepasan los límites de la significancia (línea punteada), estos son (2, 4, 6, 8, 10, 12, 14, 16 y 18). Sin embargo, por sentido de parsimonia, es necesario fijar la atención únicamente hasta el rezago 2. Mientras que la función de autocorrelación parcial *pacf*( ), permite bajo los mismos argumentos que la función anterior, obtener la autocorrelación parcial de la siguiente forma: al igual que la función de autocorrelación, se observa que los parámetros de los rezagos 2 al 4 parecen ser estadísticamente significativos. De hecho, hay más rezagos estadísticamente significativos, por el mismo sentido de parsimonia; se debe tomar hasta el último significativo a partir del primero; los demás, se toman como asintóticos (tienden a cero).

> pacf(bateria.1.d)

Figura 118. **Correlograma parcial de serie "baterias", primera diferencia**



Fuente: elaboración propia, con base en función pacf(*ts*).

Por las razones expuestas se contemplan los siguientes posibles modelos de un proceso ARMA, debido a que ya se inició con una serie integrada.

- Autocorrelación ARMA(0,2)
- Autocorrelación parcial ARMA(5,0)

A este nivel de análisis es recomendable basarse una vez más en el principio de parsimonia, es decir, “el modelo más adecuado”, el que contenga el menor número de parámetros (el más simple); en tal sentido el modelo ARMA(0,2) cumple con dicho principio. En forma preliminar y de acuerdo con el método *Box – Jenkins*, el modelo obtenido es definido por ARIMA(0,1,2), partiendo de la serie original **baterias**.

Con esta información se utilizan las funciones de R para ajustar el modelo en otro objeto y a partir del mismo, probar que los residuos son ruido blanco y estimar pronósticos.

El modelo obtenido por el métodos *Box – Jenkins*, puede contrastarse, por medio de la función *auto.arima*( ) contenida en el paquete *"forecast*". De esta forma, la determinación del modelo es francamente simple, ya que no hay que recurrir a iteraciones para seleccionar el mejor. En el siguiente ejemplo se emplea entre sus argumentos el criterio de información Bayesiano o "BIC" en la función:

```
> arima.baterias<-auto.arima(baterias, ic="bic")
> arima.baterias
Series: baterias
ARIMA(0,1,2)(0,1,0)[12]
Coefficients:
         ma1     ma2
      -1.2576  0.3420
s.e.   0.1315  0.1275
sigma^2 estimated as 303.2:  log likelihood=-214.98
AIC=435.97   AICc=436.49   BIC=441.7
```

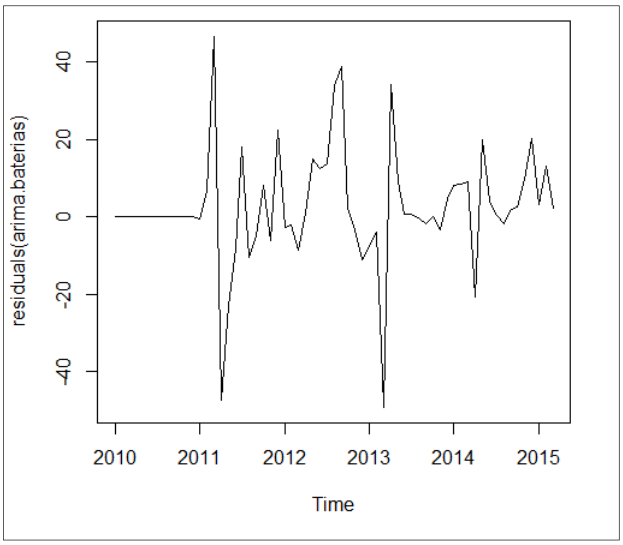
Se puede observar que el modelo ARIMA que determina la función es (0,1, 2); además muestra el modelo que ajusta la parte estacional con (0,1,0) con m = 12 periodos; en notación de los modelos ARIMA es (*p, d, q*), $(P, D, Q)_m$ que son los valores enteros para la parte no estacional y la estacional, respectivamente. El paso siguiente consiste determinar si los parámetros son estadísticamente significativos y considerando que el cociente de los coeficientes (*coefficients*) dividido entre su respectivo error estándar (*s.e.*) da como resultado valores iguales o mayores que |2|, esto implica que son

significativos.

Seguido de ello se determina si los residuos son “ruido blanco”, lo cual se puede lograr primero observando el gráfico de los residuos y luego elaborando un autocorrelograma de los mismos, para observar que ningún coeficiente sea estadísticamente significativo.

```
> plot(residuals(arima.baterias))
```

Figura 119. **Residuos del modelo ARIMA de la serie "baterias"**



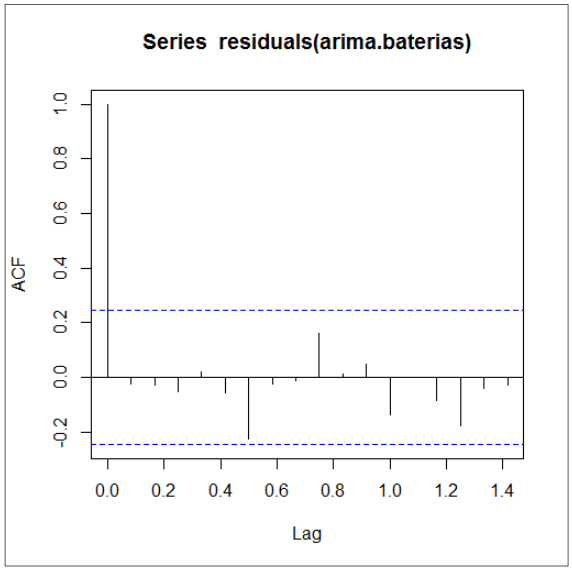
Fuente: elaboración propia, con base en la función plot(*ts*).

La figura 119 no parece mostrar patrones y conduce a afirmar que los residuos son ruido blanco. Además, notar la función *residuals*( ) que calcula la diferencia entre los valores observados y los calculados. En la figura 120 se puede observar que no hay parámetros de autocorrelación significativos después del primero (son significativos aquellos que sobrepasan las líneas

punteadas) o numéricamente se expresan de la siguiente forma:

```
> acf(residuals(arima.baterias), plot=FALSE)
Autocorrelations of series 'residuals(arima.baterias)', by lag
1.000 -0.023 -0.029 -0.052  0.021 -0.058 -0.228 -0.024 -0.011  0.160  0.012
0.047 -0.139  0.000 -0.083 -0.176 -0.040 -0.027
> acf(residuals(arima.baterias))
```

Figura 120**.** **Correlograma de los residuos del modelo ARIMA**



Fuente: elaboración propia, con base en la función acf(*ts*).

El último paso, según la metodología de Box – Jenkins, es utilizar el modelo para el pronóstico, lo que se logra mediante la función *forecast*( ):

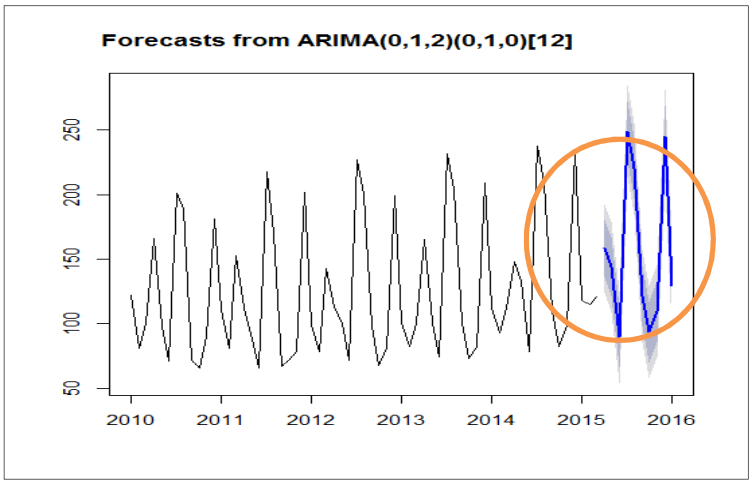
```
> forecast(arima.baterias, h=10)
Point      Forecast    Lo 80       Hi 80       Lo 95       Hi 95
Apr 2015          158.5006  136.18538 180.8158  124.37242 192.6288
```

May 2015	144.3098	121.26633	167.3533	109.06785	179.5517
:	:	:	:	:	:
Dec 2015	245.3098	221.73341	268.8862	209.25282	281.3668
Jan 2016	129.3098	105.65825	152.9613	93.13788	165.4817

El argumento de la función es el objeto que contiene el modelo ARIMA. Puede apreciarse además del pronóstico, los valores máximos y mínimos de los límites de confianza al 80 y 95 por ciento, respectivamente. En efecto, tanto el pronóstico como la gráfica se ajustan mejor en el corto plazo, ya que a largo plazo pierde capacidad predictiva el modelo; por ello los márgenes de confianza se ensanchan. Observar el círculo en la figura 121, donde se remarcan los intervalos de confianza para el pronóstico, vistos como una degradación del tono de la línea; estos son parte del resultado de aplicar la función *forecast*( ) al modelo.

```
> plot(forecast(arima.baterias,h=10))
```

Figura 121**.** **Pronóstico de modelo ARIMA, serie "baterias"**



Fuente: elaboración propia, con base en la función plot(*ts*).

5.2.4.2. Vectores autorregresivos VAR

Los vectores autorregresivos son como tal, ya que entre las variables explicativas de un modelo se emplean vectores que contienen a la misma variable explicada rezagada en el tiempo y otras variables que se asumen endógenas, también rezagadas en el tiempo; por ello tiene un carácter autorregresivo y muy similar a modelos de ecuaciones simultáneas. La idea central es que hay una causación entre todas las variables, siendo endógenas; no hay lugar para variables exógenas que conduzcan a una correlación espuria.

Lo anterior implica que se debe calcular un sistema de ecuaciones simultáneas donde la causación entre variables endógenas es similar a lo que se conoce como “Causación grangeana”, donde se estima que las variables incluidas en el modelo son causa de las demás.

La utilidad de los vectores autorregresivos radica en que se puede establecer una relación de los valores de una variable con la misma variable rezagada en el tiempo, efecto conjunto con otras variables estrechamente relacionadas, también rezagadas en el tiempo. Para ello, se detallan varios ejemplos de tales relaciones de causalidad en la tabla XL.

La ecuación matricial que define el sistema de ecuaciones está dada como:

$$\mathbf{y_t} = \alpha + \sum_{i=1}^{p} \alpha_i \mathbf{y_{t-i}} + \varepsilon_t$$

Y el sistema de ecuaciones (con p=1 rezago) está identificado por:

$$y_{1t} = \alpha_{10} + \alpha_{11}y_{2t} + \alpha_{12}y_{1t-1} + \alpha_{13}y_{2t-1} + \varepsilon_{1t}$$
$$y_{2t} = \alpha_{20} + \alpha_{21}y_{1t} + \alpha_{22}y_{1t-1} + \alpha_{23}y_{2t-1} + \varepsilon_{2t}$$

De esa forma, los resultados deben de mostrar los resultados de tantas ecuaciones con variables endógenas incluidas.

Tabla XL. **Ejemplos de vectores autorregresivos**

Variable dependiente	**Variables explicativas**
Ventas en unidades de autos nuevos: es evidente que los autos usados son sustitutos de los autos nuevos, y cómo el precio del combustible puede afectar la demanda por vehículos nuevos y esta a su vez al combustible.	Las ventas en unidades de autos nuevos de “k” periodos anteriores; la importación en unidades de autos usados rezagados, el precio histórico del combustible.
Las importaciones de bienes: la demanda por las divisas utilizadas para pagar los bienes y servicios adquiridos en el exterior es explicada a su vez por variables como la tasa de interés y el mismo volumen de importaciones, con un efecto diferido.	El tipo de cambio de la divisa de referencia; el índice de precios del país de origen y el del destino, la tasa de interés, el nivel de las reservas monetarias.
Los niveles de producción de bienes de alta tecnología: la cadena de relaciones entre productividad y uso intensivo de capital humano sobre mano de obra no calificada tiene una relación directa y con doble causación, así como el componente de capital fijo.	La inversión en capital humano, la inversión en tecnología y capital fijo

Fuente: elaboración propia.

Siguiendo el orden de ideas del método *Box – Jenkins*, se deben eliminar los parámetros no significativos para obtener el modelo más simple. Otro aspecto consiste en que se espera hacer varias iteraciones hasta encontrar el modelo con el menor criterio de información, entre AIC o BIC. Para formular un modelo que posea la capacidad explicativa y predictiva en R, se emplea por ejemplo el paquete "vars" el cual contiene, entre otras, la función *VAR*( ), que devuelve bajo los siguientes argumentos un objeto de clase "vars" y modo "lista".

Tabla XLI. **Argumentos función VAR**

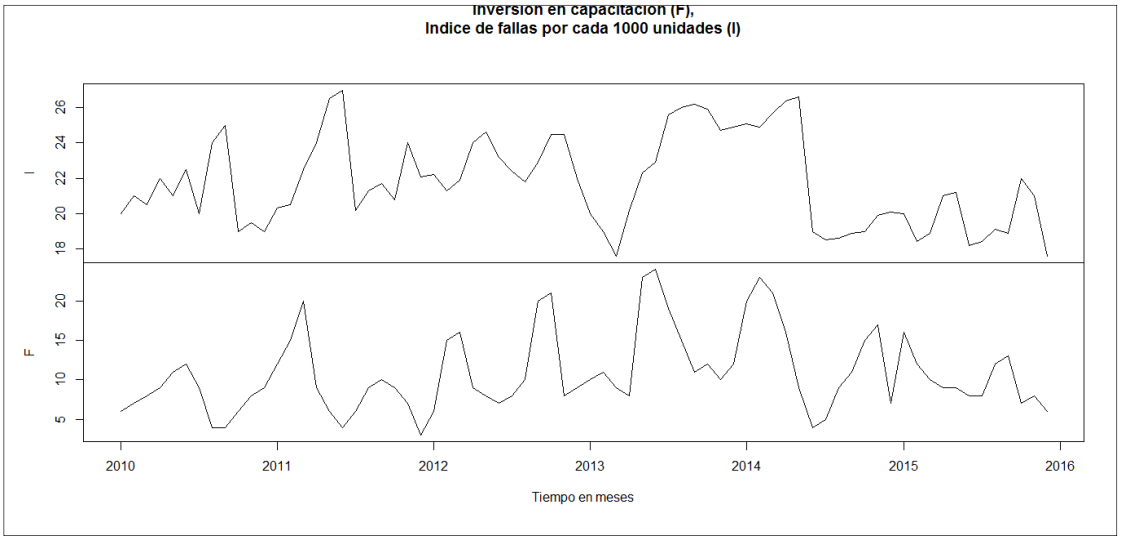
Argumento	**Descripción**
y	Objeto conteniendo las variables endógenas.
p	Número entero para especificar la cantidad de rezagos.
type=c("const", "trend", "both", "none")	Tipo de modelo de regresión a emplear (constante en el origen, tendencia determinística, ambos o ninguno).
ic=c("AIC", "SC",….)	Criterio de información a emplear si el argumento 'lag.max' no es nulo.

Fuente: elaboración propia.

Por ejemplo, se tiene una serie de tiempo donde figuran dos variables endógenas a saber: Inversión mensual en capacitación de obreros (en miles de quetzales Q.) (I), contra el índice de número de fallas calificadas por cada 1000 piezas fabricadas de la producción de un mes (F). Se espera que el monto de inversión en capacitación aumente como paliativo del incremento del indicador de fallas; no obstante la reacción de la primera (F) no es inmediata, por lo que hay un rezago en el tiempo de ejecución de la inversión (I).

Esta última reacciona con cierto rezago en el tiempo, disminuyendo la cuantía de la inversión al presentarse mejoras notorias en el índice (F).

Figura 122**.** **Vectores autorregresivos**



Fuente: elaboración propia, con base en la función plot(*ts*).

La anterior gráfica muestra cierto grado de asociación del comportamiento entre ambas variables en el tiempo con un rezago aproximado de dos meses, lo que puede dar una pista empírica para la formulación del modelo con la especificación de los argumentos en la función *VAR*( ) con los datos previamente almacenados en el objeto de clase ts de nombre **falla**, de la siguiente forma:

```
> var.falla<-VAR(falla, p=1, type="trend");  summary(var.falla)
VAR Estimation Results:
=========================
Endogenous variables: I, F
Deterministic variables: trend
```

Sample size: 71

Log Likelihood: -343.97

Roots of the characteristic polynomial:

0.9938 0.5329

Call:

VAR(y = falla, p = 1, type = "trend")

Estimation results for equation I:

==========

I = I.l1 + F.l1 + trend

	Estimate	Std. Error	t value	Pr(>|t|)
I.l1	0.92525	0.02727	33.924	< 2e-16 ***
F.l1	0.17381	0.04710	3.690	0.000448 ***
Trend	-0.01028	0.01073	-0.958	0.341485

Signif. codes: 0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 1.878 on 68 degrees of freedom

Multiple R-Squared: 0.993, Adjusted R-squared: 0.9927

F-statistic: 3229 on 3 and 68 DF, p-value: < 2.2e-16

Estimation results for equation F:

==========

F = I.l1 + F.l1 + trend

	Estimate	Std. Error	t value	Pr(>|t|)
I.l1	0.15465	0.06111	2.531	0.0137 *
F.l1	0.60141	0.10554	5.698	2.84e-07 ***
trend	0.02282	0.02404	0.949	0.3459

Signif. codes: 0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 4.208 on 68 degrees of freedom

Multiple R-Squared: 0.882, Adjusted R-squared: 0.8768
F-statistic: 169.4 on 3 and 68 DF, p-value: < 2.2e-16

Covariance matrix of residuals:

	I	F
I	3.519	1.455
F	1.455	17.699

Correlation matrix of residuals:

	I	F
I	1.0000	0.1843
F	0.1843	1.0000

Los resultados pertenecen a un solo modelo de tendencia (argumento "trend" o tendencia determinística) con un rezago, en el que el modelo de (I) como variable dependiente; los primeros parámetros son estadísticamente significativos, no así el último, pero, con un F estadístico alto que hace rechazar la suposición de que el último parámetro es "no significativo", mientras que el criterio de información Akaike es de 699, esto se calcula como:

```
> AIC(var.falla)
[1] 699.9399
```

En la tabla XLIIse resumen los resultados de todos los modelos, con p=1,2,3 y 4 rezagos, con los modelos "*const*", "*tend*", "*both*" y "*none*".

Tabla XLII. **Resumen resultados modelos VAR**

Modelo	**p =1**	**p =2**	**p =3**	**p =4**
const	AIC = 687,1 F= (44,1; 19,6)	AIC= 674,6 F= (23,6; 11.8)	AIC= 672,5 F= (15,7; 7,6)	AIC= 665,8 F= (13,1; 5,2)
trend	AIC = 699,9 F = (3229 169,4)	AIC=683,3 F=(1935;114.5)	AIC=675,7 F=(1444; 80,5)	AIC=667,8 F=(1189, 60.3)
both	AIC = 684,8 F= (33,5; 12,9)	AIC = 666,1 F= (23,4; 9,6)	AIC=664,6 F= (16,3; 6.7)	AIC=662,4 F= (12,7; 5.1)
none	AIC = 698,2 F=(4849, 254)	AIC=684,3 F=(2401, 140.3)	AIC=676,3 F=(1678, 91.92)	AIC=666,9 F= (1 354, 66,3)

Fuente: elaboración propia, con base en la función VAR( ).

En primer lugar, se analiza cuál es el modelo con menor estadístico AIC, por lo que el modelo con p= 2 rezagos y con “none” (ningún regresor implícito), es el que cumple con estos criterios rápido; los modelos son los siguientes:

Estimation results for equation I:

==========

I = I.l1 + F.l1 + I.l2 + F.l2

	Estimate	Std. Error	t value	Pr(>|t|)
I.l1	0.75341	0.12394	6.079	6.77e-08 ***
F.l1	0.14290	0.05452	2.621	0.0109 *
I.l2	0.13818	0.11533	1.198	0.2352
F.l2	0.06295	0.05825	1.081	0.2837

Estimation results for equation F:

==========

F = I.l1 + F.l1 + I.l2 + F.l2

	Estimate	Std. Error	t value	Pr(>|t|)
I.l1	0.20752	0.26603	0.780	0.43815
F.l1	0.82358	0.11703	7.037	1.39e-09 ***
I.l2	0.06265	0.24755	0.253	0.80100
F.l2	-0.37366	0.12502	-2.989	0.00393 **

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

5.3. Estadístico Durbin - Watson para la detección de patrones no aleatorios en muestras de C.E.C y autocorrelación

La breve introducción a las series de tiempo, permite en las siguientes líneas asimilar los supuestos matemáticos y estadísticos que hacen ciertamente similares las series de tiempo como tal y las muestras de C.E.C., ya que estas últimas constituyen una secuencia ordenada de datos.

Es común que los registros de las cartas de control reflejen cierto patrón no sistemático, tal y como las reglas Western Electric proponen analizar de forma previa. No obstante, no siempre resulta evidente, dada la proximidad de los valores observados al límite central. Dentro de los límites de control se presentan patrones no aleatorios que deben de interpretarse como un proceso fuera de control.

De antemano, el método gráfico es la primera herramienta para detectar esos patrones no aleatorios o sistemáticos, como lo muestra la función *we_rules*( ) del paquete XRSCC, que se utilizó para verificar la existencia de algún patrón dentro de las zonas de alerta.

5.3.1. Prueba de rachas

El anterior escenario se puede analizar mediante la “prueba de rachas”, que consiste en una prueba no paramétrica donde se analiza el número de rachas en función del tamaño de la muestra, lo que lleva a identificar el síntoma de autocorrelación negativa o positiva (sin son muchas o muy pocas, respectivamente).

En contexto, la autocorrelación se entiende como aquellas perturbaciones que afectan en forma sistemática a los valores observados en forma consecutiva.

La causa de las rachas desde el enfoque de C.E.C. provienen de causas asignables, por lo que una vez detectados esos patrones no aleatorios, se deben de analizar y eliminar.

Como un ejemplo se toman los datos del apartado 3.2 sobre las cartas de control y se trata dicha información con la función *runs.test*( ) del paquete “tseries” la cual calcula si el número de rachas se encuentra dentro de los límites de confianza al 95 %, lo que constituye la hipótesis nula; de lo contrario, se rechaza.

```
> library(tseries) # cargar el paquete
> x.muestra<-read.table("clipboard", header = TRUE); head(x.muestra) # leer los datos
  n1  n2  n3  n4  n5
1 501 505 510 496 499
2 497 503 499 503 509
3 504 509 510 505 504
4 502 501 498 501 501
```

```
> X.prom<-numeric(length(x.muestra)) # crear el objeto que contiene las medias de fila
> X.prom<-apply(x.muestra,1,mean) # aplicar la funcion mean a cada fila
> # convertir todas la medias en desviacion respecto a la media general
> X.prom<-apply(x.muestra,1,mean); > head(X.desv)
[1] -1.174 -1.174  3.026 -2.774 -2.974 -0.174
```

La prueba de rachas trabaja sobre factores como objetos, por lo que hasta el objeto **X.desv** aún no se cuenta con el factor. En tal caso se emplea la función *sign*( ) para tomar lectura del signo de cada desviación respecto de la media.

```
> x.sign<-sign(X.desv)
> head(x.sign)
[1] -1 -1  1 -1 -1 -1
> head(as.factor(x.sign))
[1] -1 -1 1  -1 -1 -1
Levels: -1 1
```

La función *runs.test*( ) necesita un objeto de clase factor, con niveles “-1” y “+1”. Además para probar o rechazar la hipótesis nula se debe conocer la cantidad de rachas contenidas en el objeto **x.sign**. Aunque no es necesario, se recurre a la función interna de R que es *rle*( ), que almacena un objeto de clase “rle” con modo lista, que contiene largos y valores de las rachas o corridas (*lenghts* y *values*, respectivamente)

```
> x.run<-rle(x.sign)
> x.run
Run Length Encoding
  lengths: int [1:45] 2 1 3 1 3 3 2 6 1 1 ...
  values : num [1:45] -1 1 -1 1 -1 1 -1 1 -1 1 ...
```

```
> sum(x.sign==1)
[1] 51
> sum(x.sign==-1)
[1] 49
```

Son un total de 45 rachas, 51 valores individuales con signo positivo (n_1) y 49 con signo negativo (n_2). Esto sirve para calcular el valor esperado del número de rachas $E(k)$ y la respectiva varianza σ_k^2.

$$E(k) = \frac{2n_1n_2}{n_1 + n_2} + 1$$

$$\sigma_k^2 = \frac{2n_1n_2(2n_1n_2 - n_1 - n_2)}{(n_1 + n_2)^2(n_1 + n_2 - 1)}$$

El número individual de diferencias positivas y negativas es respectivamente mayor que 10, por lo que se asume que las rachas están normalmente distribuidas[32], por ello se espera que la media esté entre el intervalo $[E(k) \pm 1.96\sigma_k]$ al 95 % de confianza; por ello se asume el argumento *two.sided* (por defecto) de la función *runs.test*( ). Dicha función únicamente necesita la forma de factor del objeto **x.sign** para probar o rechazar la hipótesis.

```
> runs.test(as.factor(x.sign))
        Runs Test
data:  as.factor(x.sign)
Standard Normal = -1.2026, p-value = 0.2291
alternative hypothesis: two.sided
```

El valor del *p-value* igual a 0,2291 indica que a un valor de significancia de

[32] GUJARATI, Damodar.*Econometría*. p. 433.

(1 - 0,95) igual a 0,05 hay que aceptar la hipótesis nula de que la secuencia varía en forma aleatoria.

Sin el ánimo de hacer cálculos manuales, pero con el afán de contrastar los resultados con las ecuaciones anteriores, se emplea la simbología que permite R para calcular el intervalo de confianza al 95%, en el que el valor esperado del número de rachas debe figurar en función del número de valores positivos y valores negativos.

```
> k <- 1 + 2 * n1 * n2/(n1 + n2)
> n1<-sum(x.sign==1)
> n2<-sum(x.sign==-1)
> n1;n2
[1] 51
[1] 49
> k
[1] 50.98
> s <- sqrt(2 * n1 * n2 * (2 * n1 * n2 - n1 - n2)/((n1 + n2)^2 *
+         (n1 + n2 - 1)))
> s
[1] 4.972673
> Li.infe<-k-1.96*s
> Li.supe<-k+1.96*s
> Intervalo<-cbind(Li.infe, Li.supe)
> Intervalo
      Li.infe  Li.supe
[1,] 41.23356 60.72644
```

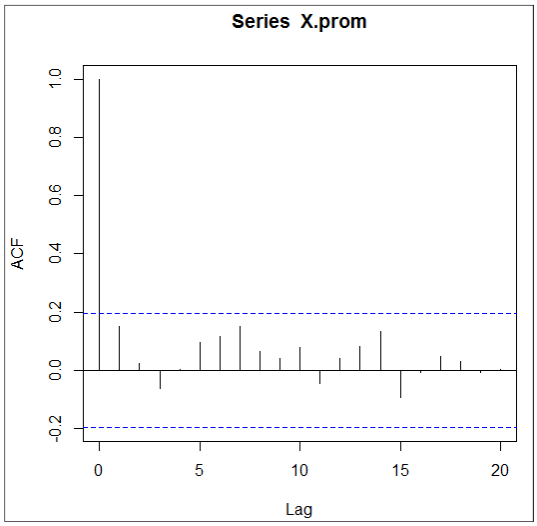
Se puede observar claramente que el valor observado del número de rachas (45) está dentro del intervalo de confianza al 95 %; que es otra forma de expresar los resultados según la función *runs.test*( ).

5.3.2. Estadístico Durbin - Watson

La figura 123 muestra la función del autocorrelación del objeto **X.prom**, donde se observa que ningún rezago es estadísticamente significativo; ello da una referencia que en sí, los propios registros de los promedios de las muestras se presentan como ruido blanco. En tal sentido, se espera que la prueba de presencia de patrones no aleatorios por medio del estadístico Durbin – Watson confirme la presencia de autocorrelación.

La prueba de autocorrelación por el estadístico Durbin – Watson tiene por objetivo detectar los residuos de predicción en un modelo de regresión. No obstante, los datos del objeto **X.prom**, no corresponden ni aun modelo de regresión lineal ni a una serie de tiempo en forma estricta.

Figura 123. **Correlograma de la carta $\overline{X}$**



Fuente: elaboración propia, con base en función ACF(*ts*).

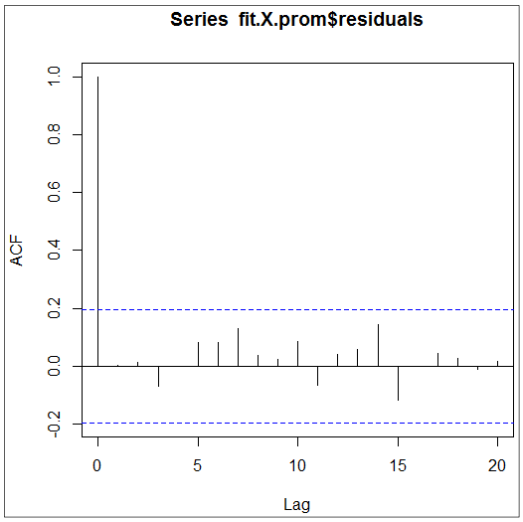
De hecho se prevé que los datos como los de una carta de control se asemejen al componente aleatorio de una serie de tiempo; no obstante, en determinadas muestras de algunos procesos existen corrimientos en la media debido a causas asignables, cambios en las especificaciones o correcciones mismas al proceso; entonces, como es lógico suponer, habrá una tendencia desde el estado original del proceso al estado final. Otra forma de la presencia de dichos corrimientos en la media en el proceso, es en presencia de los saltos, cuya media se ajustará a la función logística, previamente expuesta.

Suponiendo que se ajusten los datos del objeto a un modelo ARIMA de la siguiente forma:

```
> library(forecast)
> fit.X.prom<-auto.arima(X.prom)
> fit.X.prom
Series: X.prom
ARIMA(1,0,0) with non-zero mean
Coefficients:
    ar1     intercept
  0.1511   503.3709
s.e.       0.0985       0.2361
sigma^2 estimated as 4.032:
log likelihood=-211.62
AIC=429.24  AICc=429.49  BIC=437.05
```

Los resultados demuestran en forma gráfica la similitud de los residuos tanto de los datos originales como de los residuos del modelo ARIMA, lo que apoya la robustez del mismo. Sin embargo, en función de un modelo dinámico como el anterior, que resultó en uno AR(1), la prueba deja de ser eficiente, ya que por motivos de especificación, la autocorrelación está presente.

Figura 124. **Ajuste de modelo ARIMA de carta $\overline{X}$**



Fuente: elaboración propia, con base en la función ACF(*ts*).

Por lo tanto, se opta por ajustar el modelo a uno lineal (sin rezagos) de la siguiente forma:

```
> y<-X.prom
> x<-1:length(X.prom)
> X.prom.lm<-lm(y~x)    # Modelo
> summary(X.prom.lm)
Call:
lm(formula = y ~ x)
Coefficients:
Estimate          Std. Error      t value                 Pr(>|t|)
(Intercept) 504.008485          0.406917        1238.604        <2e-16 ***
x           -0.012564       0.006996        -1.796          0.0756 .
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
Residual standard error: 2.019 on 98 degrees of freedom
```

Multiple R-squared: 0.03187, Adjusted R-squared: 0.02199
F-statistic: 3.226 on 1 and 98 DF, p-value: 0.07558
> AIC(X.prom.lm)
[1] 428.3219

El anterior ajuste de modelo carece de buena bondad de ajuste, sin embargo, no existe propósito de pronóstico, y los parámetros del modelo son estadísticamente significativos, considerando que la pendiente de una serie de datos que se sitúan aproximadamente en promedio a la misma distancia de una media fija (límite central), que implica que debe de tender a cero, que contrasta con el valor - 0,0126 que es muy cercano a cero. El valor de AIC, incluso es menor respecto del modelo ARIMA, por lo que se cuenta con elementos para aceptar dicho modelo lineal. El estadístico *Durbin – Watson*, depende un modelo para calcular los valores de tabla (dL y dU) y contrastarlos con el valor calculado "d", el cual se determina de la siguiente forma.

$$d = \frac{\sum_{t=2}^{t=n}(\hat{u}_t - \hat{u}_{t-1})^2}{\sum_{t=1}^{t=n}\hat{u}_t^2}$$

De forma práctica, se espera que el valor del estadístico d sea igual a 2, puesto que:

$$d = 2(1 - \rho)$$

Dado que el parámetro "rho" (ρ) proviene del modelo AR(1), que corresponde al proceso autorregresivo de primer orden de los residuos $\hat{u}_t$ de la muestra, contra los residuos de la muestra rezagados en un periodo o $\hat{u}_{t-1}$. Dicho parámetro se espera que sea igual a cero.

Empleando el paquete “car” mediante la función *durbinWatsonTest* ( ) se calcula automáticamente el valor “d”, utilizando como argumento el modelo **X.prom.lm** ajustado en el objeto de clase “lm”.

```
> library(car)
> durbinWatsonTest(X.prom.lm)
 lag Autocorrelation D-W Statistic p-value
1       0.1221411      1.747641   0.184
 Alternative hypothesis: rho != 0
```

La función plantea una hipótesis alternativa como H_1: Rho no es igual cero; en contraparte, la hipótesis nula plantea H_0: Rho es igual cero. Los resultados confirman la H_1, que sugiere la presencia de cierto grado de autocorrelación; sin embargo, el valor “d” ha de contrastarse con los valores de tabla a un valor de significancia del 5 % y k regresores igual a 1.

Tabla XLIII. **Análisis resultados prueba *Durbin - Watson***

Límite	**O**	**dL**	**dU**	**4-dU**	**4-dL**	**4**
Valor	0	1,654	1,694	2,306	2,346	n/a

Fuente: elaboración propia.

Lo que indica es que el “d” calculado está ubicado entre dU y 4-dU, por lo que no debe de rechazar la hipótesis nula, apoyado por el *p-value* igual a 0,184, el cual es mayor que el valor de significancia del 5%. Por lo tanto, la evidencia muestra que no hay presencia de autocorrelación, por lo que no existen patrones no aleatorios. Por lo tanto, al utilizar el instrumento del estadístico Durbin – Watson se puede confirmar que el proceso está bajo control.

CONCLUSIONES

1. Siendo R un lenguaje de programación, resulta en cierto grado de dificultad para su aprendizaje, no obstante, la cantidad de funciones internas de R seleccionadas y revisadas a través de todo el texto, conforman el instrumental de nivel básico e intermedio, para su comprensión. Respecto del orden y detalle expuesto de las funciones, para desarrollar secuencialmente los conocimientos necesarios, destacan las que permiten la vectorización y manipulación de los datos contenidos en objetos para el posterior análisis ya sea en estadística, matemática o finanzas; sin embargo es muy útil auxiliarse de la plataforma de *R Studio*, que contiene un entorno más visual, un editor de códigos ASCII y las funciones para construir paquetes.

2. Conocer los atributos de los objetos permite en primer lugar hacer uso correcto de las funciones internas y de las contenidas en los paquetes. Por otro lado, es el fundamento para asignar cierto comportamiento a los mismos, mediante la programación de nuevas funciones que se unen en un ambiente general o específico, que constituye a su vez, la base de la creación de paquetes. Todo se logra mediante la estructuración de expresiones, declaraciones, ciclos, argumentos y la depuración del código de las funciones.

3. Se programaron dos paquetes: “XRSCC” que contiene las funciones que permiten el control estadístico procesos de la calidad por variables y atributos y “Planesmuestra”, cuyo eje central es la determinación del plan de muestreo de aceptación para variables y atributos. Ambos paquetes,

aunque separados, deben de considerarse complementarios; tomando en cuenta los aspectos probabilísticos relacionados con los errores tipo I y II, aplicado a procesos y a muestreos de aceptación se dice “aceptar lotes defectuosos” o “rechazar lotes aceptables” respectivamente. Ambos paquetes fueron subidos al *CRAN* junto con la documentación que exige el estándar de los paquetes para la plataforma de R.

4. El paquete XRSCC se auxilia de las tablas de factores para calcular las cartas de control, utilizando datos adjuntos al paquete para efectos demostrativos y requisitos importantes en la documentación del mismo. Entre las funciones principales del paquete, resaltan aquellas como *xrs_gr*( ) que simula la primera iteración y gráficas para las cartas $\bar{X}$, R y S. Seguido de ello, las iteraciones *X_it*( ), *R_it*( ) las cuales son utilizadas en el mismo orden (si fuera necesario), para calcular los verdaderos límites de control, considerando que el proceso simulado en la carta $\bar{X}$ no está “bajo control”, si la carta *R* no está “bajo control”, porque implica la presencia de un grado de variabilidad en el proceso debido a causas asignables.

5. El paquete XRSS incluye también funciones para simular las cartas de control para los procesos por atributos y la respectiva función de iteración, por ejemplo, la función *u_gr*( ) (número de inconformidades por unidad), es seguida de la función para iésima iteración *U_it*( ), únicamente si en la primera iteración el proceso está fuera de control. De tal manera que se han programado funciones de inicio y de iteración para las cartas descritas, y para la carta “c” del número de no conformidades por unidad; la carta “np” para el número de no conformes; la carta “p”, para la proporción de los “no conformes”.

6. De los resultados de la carta $\bar{\bar{X}}$, mediante las funciones *xrs_gr*( ), *X_it*( ) y *R_it*( ) se obtiene el cálculo del índice de la capacidad del proceso con la función *Cp_X*( ). La determinación de la capacidad del proceso se complementa con la inclusión de los límites de especificación, que en conjunto se pueden apreciar en la estimación puntual bajo la curva normal de probabilidades. No obstante, los límites de especificación pueden mostrarse "descentrados" respecto de los límites naturales del proceso; por ello, uno de los resultados calcula el índice C_{pk}. Otro resultado útil de dicha función, es el índice P_{cp}, que es el porcentaje del rango de los límites de especificación que los límites de control utilizan; dicho resultado constituye un elemento técnico adicional para examinar la variabilidad del proceso.

7. En la función *we_rules( )* se logró detectar la presencia o la existencia de rachas o corridas basadas en las reglas Western Electric, demostrando que un determinado proceso está bajo o fuera de control, ampliando las conclusiones de las funciones *xrs_gr*( ) y las iteraciones *X_it*( ) y *R_it*( ), que calculan los límites de control y depuran aquellos registros que producen que un proceso sea calificado como "fuera de control".

8. El paquete "Planesmuestra" contiene funciones que determinan el plan de muestreo aceptación en función de algunos argumentos que sirven de criterio para la búsqueda en las tablas adjuntas al paquete, que únicamente son útiles en la secuencia de ejecución de una función en particular. Entre las funciones del paquete "Planesmuestra", se encuentra *f_CO.plan*( ), la cual se utiliza para calcular la curva característica de operación, como una herramienta gráfica para demostrar la probabilidad de aceptación o rechazo de un plan previamente especificado en las

funciones como *f_milstd414*( ) para muestreo de aceptación por variables y para el muestreo de aceptación por atributos, se programaron las funciones *f_milstd105e*( ) y *f_dodge.romig.simple*( ).

9. Para programar planes de muestreo de aceptación, basados en los sistemas MIL STD o Dodge Romig, se necesita la tabla de los valores de cada plan, almacenados en objetos para su lectura y en función de algunos argumentos específicos, tales como cambiar de nivel de restricción del plan de muestreo de aceptación, entre otros. De allí que los resultados de los planes de muestreos de aceptación establecidos sirven de información para levantar las gráficas de la curva característica de operación. Los valores de las tablas, aunque constituyen un tabulado, en efecto provienen de distribuciones de probabilidad discreta.

10. Respecto del método MILSTD414, de muestreo de aceptación por variables, la función *f_milstd414.test*( ) contenida en “Planesmuestra”, acepta o rechaza un plan de muestreo de aceptación por variables, considerando un factor *k* de corrimiento de la media, una desviación estándar *S* conocida y un límite inferior o superior conocido; el resultado de dicha función, es categórico: “Rechazar” o “Aceptar” el lote.

11. En apoyo a la función método MILSTD414, en el paquete “XRSCC”, la función *Beta.X*( ), calcula y presenta la gráfica de la relación probabilística binomial de no detectar el corrimiento *k* de la media, dada una desviación estándar y un tamaño de muestra definido. Bajo la misma función, se calcula el ancho medio de la corrida o ARL (por sus siglas en inglés) para detectar un corrimiento; es decir se calcula el número de corridas necesarias antes de detectar un corrimiento *k* en la media.

12. Otros paquetes utilizados tales como “AcceptanceSampling”, permiten por medio de la función *find.plan*( ), calcular el plan de muestreo (cantidad de muestra y número de aceptación) que satisface ambas posiciones, la del productor (*Producer Risk Point*) o el par (NCA,$(1-\alpha)$) y la del consumidor (*Consumer Risk Point*), cuyo par es (NCL, β). En conjunto, dicha asociación es conocida como la “NCA – NCL”, cuyo resultado es la determinación de la muestra y el número de aceptación, asumiendo un tamaño del lote mucho mayor al tamaño de la muestra. Para tal propósito la función emplea las distribuciones binomial, la de Poisson o la normal, como un argumento explícito, atendiendo al contexto del análisis.

13. Los paquetes como “LM”, *“forecast”* y *“car*” permiten formular, validar y calcular pronósticos en función de series de tiempo. Los filtros como el *Holt Winters* y ARIMA, capturan la dinámica de las series temporales. A su vez, el núcleo de R cuenta con varias funciones para este propósito y otras como *AIC*( ) y *BIC*( ), proporcionan información y criterio con suficiente significancia estadística para seleccionar el modelo que mejor se ajusta a los datos.

14. Se espera que el arreglo cronológico de las muestras de la carta de control manifieste una característica de aleatoriedad, no obstante, en algunos procesos, se dan patrones no aleatorios, (ampliando los resultados de la función *we_rules*( ) contenido en el paquete “XRSCC”) como producto de causas asignables que generan comportamiento particular en patrones sistemáticos, que equivalen a calificar al proceso como fuera de control. Una de las pruebas utilizadas para identificar la presencia de patrones sistemáticos, es la de rachas, la cual es “no paramétrica”, y se basa en la hipótesis nula de que el número de rachas

de una muestra se encuentra dentro de los límites de confianza; en otras palabras, la secuencia de datos varía en forma aleatoria, de lo contrario, surgen rachas o secuencias no aleatorias producto de causas asignables, por lo tanto el proceso estaría fuera de control.

15. En términos estadísticos y econométricos, la presencia de patrones sistemáticos de los datos respecto de la media, se conoce como autocorrelación o correlación serial. Una de las pruebas utilizadas para este fin, es el estadístico de Durbin – Watson, el cual que la serie de muestras se ajusta a un modelo lineal, donde se espera que el intercepto de la curva del modelo sea aproximadamente igual al límite central de la carta $\bar{X}$ y la pendiente, aproximadamente igual a cero; esto significa que no hay una tendencia de los datos respecto de la media, al menos en el corto plazo, ya que es habitual que haya tendencias producidas por presencia de causas asignables y mejoras en los procesos. Se concluye que el parámetro de autocorrelación es igual a cero, es decir, no hay autocorrelación, sin embargo se debe contrastar el resultado con los valores críticos de la tabla, con un porcentaje de significancia.

RECOMENDACIONES

1. Implementar el aprendizaje de R en los cursos de la Escuela de Ingeniería Mecánica Industrial de la Facultad de Ingeniería, como Investigación de Operaciones I y II, Controles Industriales y Control de la Producción; para la Escuela de Ciencias, en los cursos de Estadística I, II y III y Análisis Probabilístico. En general, donde se necesite formular y calcular procedimientos extensos. Dicha implementación se recomienda como parte de la práctica de laboratorio.

2. Utilizar las herramientas compiladas en paquetes para apoyar el aprendizaje específico de la asignatura en curso, seguido de los aspectos básicos del lenguaje y programación, ya que es una realidad que la programación como oficio, no constituye una destreza común para el estudiante promedio de la Escuela de Ingeniería Mecánica Industrial.

3. Incentivar la investigación y mejora continua por medio de la plataforma de R, en instancias como las prácticas iniciales, intermedias y finales y trabajos de graduación, ya que se aprovecha la creatividad y proyección de los estudiantes para la programación de más contenido útil, tanto en la academia como en la vida productiva. Como consecuencia, el resultado de invertir recursos en investigación dará como resultado un aporte útil, tanto en la academia como la sociedad misma.

4. Es necesario el apoyo de los instrumentos gráficos en la interpretación de los resultados del estudio de la variabilidad de un proceso, por lo

tanto, permite dedicar más tiempo y esfuerzo al análisis de las causas asignables. Considerar además la opción de simular otros tipos de escenarios, tanto para fines académicos como prácticos, para evitar elevar el grado dinamismo del aprendizaje, la cantidad y calidad de aportes.

5. Debido a que los problemas de la realidad en el control estadístico de la calidad son en efecto complejos, se necesita mejorar y ampliar la base para propiciar el salto a los siguientes niveles de complejidad en la simulación, por ejemplo, programar funciones que validen procesos por variables, en donde el número de las mismas sea mayor que uno. Además, se necesita ampliar la investigación en el escenario de muestras con tamaño variable y el tamaño mismo de los grupos para detectar corrimientos en la media, ya que en la práctica dichos eventos son poco estudiados.

6. Ampliar el análisis de la capacidad del proceso a cartas de control para el control de atributos. Se considera que la complejidad para ello no es mayor que la encontrada en las cartas de control por variables, sin embargo, se ha diseñado la simulación de la capacidad del proceso para la carta $\bar{X}$, atendiendo únicamente al objetivo de demostración.

7. El paquete “Planesmuestra”, es una aproximación sencilla en términos de programación, dado que existen funciones separadas que podrían integrarse en desarrollos posteriores de nuevas versiones que integren los resultados gráficos al plan determinado y la asignación de ciertas características a los resultados numéricos para ser exportados.

8. Ampliar la base práctica de conocimiento de los métodos de muestreo de aceptación, a los métodos con mayor contenido probabilístico, es decir,

aquellos que no dependen de tablas para determinar el plan. Por otro lado, se necesita ampliar el desarrollo de funciones para simular el cálculo de planes de muestreo de aceptación secuenciales.

9. Considerando la existencia de un corrimiento en la media en muestras por variables y utilizando la vectorización, pueden ampliarse los cálculos numéricos y gráficos a varios escenarios o configuraciones de muestras, dando como resultado un análisis comparativo de los distintos valores de riesgos al variar de forma unilateral el tamaño de la muestra o el factor de corrimiento. Dicha técnica de vectorización es útil sí y solo sí la función lo permite, ya que por orden de ideas, se ha utilizado en los paquetes el argumento en forma escalar o vectorial de ancho “uno”.

10. Es necesario el apoyo de funciones que utilicen las capacidades matemáticas y estadísticas del lenguaje hacia el cálculo de índices de calidad; aunque es una realidad que ya existen paquetes que simulan dicha recomendación o bien puede representar metas en el sentido académico.

11. Las pruebas “no paramétricas” para la detección de patrones no aleatorios o rachas, utilizadas deben considerarse como una aproximación, dependiendo de la longitud total de la muestra. Además se adapta mejor a la carta $\bar{X}$, ya que los valores reales se agrupan en dos grandes grupos (positivos y negativos) respecto de la mediana, sin embargo no considera la magnitud de la variabilidad; por eso, para que los resultados de dicha prueba se fundamenten únicamente en la posición del promedio del grupo respecto de la mediana, es necesario que el proceso esté calificado previamente como “bajo control”, utilizando la secuencia prevista para la carta $\bar{X}$ y carta R.

12. Una alternativa para la prueba Durbin – Watson para detectar la presencia de autocorrelación es la prueba Breusch – Godfrey, ya que en términos técnicos la primera presenta una limitante al utilizar zonas de incertidumbre y la inclusión de variables exógenas deterministas en la regresión, por lo que la sola inclusión implica la invalidación del funcionamiento de la prueba, incluso si se tratase de muestras grandes. En tal sentido, para la detección de autocorrelación se ha utilizado dicha prueba para efectos demostrativos, sin embargo, ya que existen alternativas con mayor robustez, sin duda constituyen líneas de investigación y desarrollo.

BIBLIOGRAFÍA

1. ALAIN, Zuur*. A beginner´s guide to R (Use R!).* Estados Unidos: Springer, 2009. 220 p. ISBN 978-0387938363.

2. ARRIAZA, A.J. *Estadística Básica con R y R-Commander*. [en línea]. <http://knuth.uca.es/moodle/mod/url/view.php?id=1126> [Consulta: agosto de 2015].

3. BLANCO, Edgar*. Cartas R y S con límites de control estimados.* [en línea]. <http://www.bdigital.unal.edu.co/8712/1/edgareliecer blancoguerrero.2012.pdf> [Consulta: agosto de 2015]

4. CHANDRA, M. Jeya. *Statistical quality control*. [en línea]. <www.crcpress.com> [Consulta: agosto de 2015] ISBN 0-8493-2347-9.

5. CHANG, Winston. *R Graphics cookbook*. Estados Unidos: O´Reilly Media, 2013. 416 p. ISBN 978-1449316952.

6. COCHRANE, John H. *Time series for macroeconomics and finance.* [en línea].<http://faculty.chicagobooth.edu/john.cochrane/research/ papers/time_series_book.pdf> [Consulta: enero de 2016].

7. COHLAN, Avril. *Little book of R for time series.* [en línea]. <https://media.readthedocs.org/pdf/a-little-book-of-r-for-time-series/ latest/a-little-book-of-r-for-time-series.pdf> [Consulta: enero de

2016].

8. CONTRERAS, José; ARTEAGA, Pedro. *Introducción a la programación estadística con R para profesores. Formación de profesores*. [en línea]. <http://www.ugr.es/~batanero/ARTICULOS/libroR.pdf> [Consulta: enero de 2015] ISBN: 978-84-693-4859-8.

9. CORREA, Juan Carlos. Gráficos *Estadísticos con R.* 2002 [en línea]. <http://cran.r-project.org/doc/contrib/ grafi3.pdf> [Consulta: enero de 2015].

10. CRAWLEY, Michael. *The R book*. Estados Unidos: Wiley & Sons, 2012. 1076 p. ISBN: 978-04709733929.

11. DELGAARD, Peter. *Introductory statistics with R (Statistics and computing)*. Estados Unidos: Springer, 2013. 267 p. ISBN: 978-0387954752.

12. DEVORE, Jay L. *Probabilidad y estadística para ingeniería y ciencias*. 6a ed. México: Thomson, 2005. 752 p. ISBN 970-686-457-1.

13. DOMíNGUEZ, José. *Dirección de operaciones, Aspectos tácticos y operativos en la producción y servicios*. Madrid: McGraw Hill, 1995. 503 p. ISBN 84-481-1803-0.

14. GALILI, Tal. *R- Statistics blog*. [en línea]. <http://www.r-statistics.com/2013/03/updating-r-from-r-on-windows-using-the-installr-package/> [Consulta: enero de 2016].

15. GENTLEMAN, Robert. *R Programming for bioinformatics*. Boca Raton, Florida: Chapman & Hall / CRC Press computer science and data analysis series, 2009 [en línea]. <http://www.crcpress.com> [Consulta: enero de 2016] ISBN 978-1-4200-6367-7.

16. GUJARATI, Damodar. *Econometría.* 5a ed. México: McGraw Hill, 2010. 946 p. ISBN 978-607-15-0294-0.

17. GUTIÉRREZ, Humberto. *Control estadístico de la calidad y seis sigma.* 3a ed. México D.F.: Mc Graw Hill Education, 2013. 468 p. ISBN 978-607-15-0929-1.

18. Instituto Nacional de Estadística. *Encuesta nacional de empleo e ingresos, 2013.* [en línea]. <http://www.ine.gob.gt> [Consulta: mayo de 2015].

19. JURAN, Joseph. *Análisis y planeación de la calidad*. 3a ed. México: McGraw Hill, 1995. 633 p. ISBN 970-10-0612-7.

20. KABACOFF, Robert. *R in action*. Estados Unidos: Manning Publications, 2011. 472 p. ISBN 978-1935182399.

21. KIERMEIER, Andreas*. Visualising and assessing acceptance sampling.* [en línea]. <http://cran.r-project.org/web/packages/AcceptanceSampling/vignettes/acceptance_sampling_manual.pdf> [Consulta: enero de 2016].

22. KRAJEWSKI, Lee J. *Administración de operaciones: estrategia y análisis*. 5a ed. Pearson Educación, 2000. 892 p. ISBN

9684444117.

23. LAM, Longhow. *An introduction to R.* [en línea]. <http://www.r-project.org> [Consulta: mayo de 2015].

24. LUMLEY, Thomas. *R Fundamentals and programming techniques.* [en línea]. <http://faculty.washington.edu/tlumley/Rcourse/R-fundamentals.pdf> [Consulta: mayo de 2015].

25. MAIN YAQUE, Paloma. *Análisis exploratorio de datos con R y Minitab*. 2012 [en línea]. <http://www.mat.ucm.es/~palomam/ aed.pdf> [Consulta: mayo de 2015]. ISBN 978-84-692-6255-9.

26. MATLOFF, Norman. *The art of R programming*. [en línea]. <http://www.atmos.albany.edu/facstaff/timm/ATM315spring14/R/The%20Art%20of%20R%20Programming.pdf> [Consulta: enero de 2015] ISBN:978-1-59327-384-2.

27. MONTGOMERY, Douglas. *Applied statistics and probability for engineers*. 4a ed. Estados Unidos: John Wiley & Sons, 2006. 900 p. ISBN: 0471745898.

28. _____. *Introduction to statistical quality control.* 5a ed. Estados Unidos: John Wiley & Sons, 2005. 761 p. ISBN: 0-471-65631-3.

29. OWEN, J.W. *The R Guide.* 2010 [en línea]. <http://www.r-project.org> [Consulta: mayo de 2015].

30. PARADIS, Emmanuel. *R para Principiantes*. 2003 [en línea].

<http://cran.r-project.org/doc/contrib/rdebuts_es.pdf> [Consulta: mayo de 2015].

31. R Core Team. *Frequently asked questions*. 2014 [en línea]. <http://CRAN.R-project.org/doc/FAQ/R-FAQ.html> [Consulta: abril de 2015].

32. _____. *Introducción a R (Notas sobre R: Un entorno de programación para Análisis de Datos y Gráficos).* 2012 [en línea]. <http://www.r-project.org> [Consulta: abril de 2015].

33. _____. *R Data import / export.* 2010 [en línea]. <http://www.r-project.org> [Consulta: abril de 2015] ISBN 3-900051-10-0.

34. _____. *R Internals*. 2012 [en línea]. <http://www.r-project.org> [Consulta: mayo de 2015] ISBN 3-900051-11-9.

35. _____. *R Languaje Definition*. 2012 [en línea]. <http://cran.r-project.org/doc/manuals/r-release/R-lang.html>. [Consulta: mayo de 2015].

36. _____. *Writing R extensions*. 2012 [en línea]. <http://www.r-project.org> [Consulta: agosto de 2015] ISBN 3-900051-11-9.

37. SÁEZ CASTILLO, Antonio. *Métodos estadísticos con R y R Commander.* [en línea]. <http://cran.r-project.org/doc/contrib/Saez-Castillo-RRCmdrv21.pdf> [Consulta: mayo de 2015].

38. SMALL, Bonnie B. *Statistical quality control handbook*. 2a ed. Easton: Western Electric Co, Inc., 1956. 328 p.

39. VENABLES, William. *An Introduction to R*. *R Project.* 2012. [en línea]. <http//www.r-project.org> [Consulta: mayo de 2015] ISBN 3-900051-12-7.

40. WHALPOLE, Myers*. Probabilidad y estadística para ingenieros*. 6a ed. México: Prentice - Hall Hispanoamérica, 1999. 752 p.

41. ZUCCHINI, Walter. *Professuren für statistik und ökonometrie. Time series analysis with R - Part I*. [en línea]. <http://www.statoek.wiso.uni-goettingen.de/veranstaltungen/zeitreihen/sommer03/ts_r_intro.pdf> [Consulta: enero de 2016].

APÉNDICES

Apéndice 1. **Formulario de estadística aplicada con R**

Operación	Función y argumentos
Sintaxis general de funciones Dónde: X Args '...'	FUN(**x, argumentos,...**) Objeto Conjunto de argumentos que condicionan que el resultado sea uno en particular Argumentos pasados hacia o de otra función
Ajustar un modelo ARIMA a una serie de una tiempo de una sola variable	arima(x, order = c(0L, 0L, 0L), seasonal = list(order = c(0L, 0L, 0L), period = NA), xreg = NULL, include.mean = TRUE, transform.pars = TRUE, fixed = NULL, init = NULL, method = c("CSS-ML", "ML", "CSS"), n.cond, SSinit = c("Gardner1980", "Rossignol2011"), optim.method = "BFGS", optim.control = list(), kappa = 1e6)
Ajuste de modelos lineales a una descripción simbólica del predictor lineal y la distribución del error	glm(formula, family = gaussian, data, weights, subset, na.action, start = NULL, etastart, mustart, offset, control = list(...), model = TRUE, method = "glm.fit", x = FALSE y = TRUE, contrasts = NULL, ...)

Continuación del apéndice 1.

Ajuste de un modelo logarítmico – linear	loglin(table, margin, start = rep(1, length(table)), fit = FALSE, eps = 0.1, iter = 20, param = FALSE, print = TRUE)
Ajuste o estimación de modelos lineales	lm(formula, data, subset, weights, na.action, method = "qr", model = TRUE, x = FALSE y = FALSE, qr = TRUE, singular.ok = TRUE, contrasts = NULL, offset, ...)
Análisis de Varianza (Tabla Anova)	anova(object, ...)
Aplicación de una función en columnas o filas	apply(X, MARGIN, FUN, ...)
Aplicar diferencias a un objeto para modelos lineales	diff(x, lag = 1, differences = 1, ...)
Aplicar rezagos a un modelo lineal	lag(x, k = 1, ...)
Calcular los parámetros *b* de un modelo lineal para un objeto de datos	lsfit(x, y, wt = NULL, intercept = TRUE, tolerance = 1e-07, yname = NULL)
Correlación entre *X* y *Y* (matriz de covarianza)	cor(x y = NULL, use = "everything", method = c("pearson", "kendall", "spearman"))
Covarianza entre *X* y *Y* (matriz de covarianza)	cov(x y = NULL, use = "everything", method = c("pearson", "kendall", "spearman"))
Descomposición de una serie de tiempo en los componentes estacionales, tendencia e irregular	decompose(x, type = c("additive", "multiplicative"), filter = NULL)
Desviación absoluta media	mad(x, center = median(x), constant = 1.4826, na.rm = FALSE, low = FALSE, high = FALSE)

Continuación del apéndice 1.

Desviación estándar	sd(x, na.rm = FALSE)
Distribución Binomial	
Función de densidad	dbinom(x, size, prob, log = FALSE)
Función probabilística	pbinom(q, size, prob, lower.tail = TRUE, log.p = FALSE)
Función acumulada	qbinom(p, size, prob, lower.tail = TRUE, log.p = FALSE)
Función aleatoria	rbinom(n, size, prob)
Distribución Chi Cuadrada	
Función de densidad	dchisq(x, df, ncp = 0, log = FALSE)
Función probabilística	pchisq(q, df, ncp = 0, lower.tail = TRUE, log.p = FALSE)
Función acumulada	qchisq(p, df, ncp = 0, lower.tail = TRUE, log.p = FALSE)
Función aleatoria	rchisq(n, df, ncp = 0)
Distribución de Poisson	
Función de densidad	dpois(x, lambda, log = FALSE)
Función probabilística	ppois(q, lambda, lower.tail = TRUE, log.p = FALSE)
Función acumulada	qpois(p, lambda, lower.tail = TRUE, log.p = FALSE)
Función aleatoria	rpois(n, lambda)
Distribución F	
Función de densidad	df(x, df1, df2, ncp, log = FALSE)
Función probabilística	pf(q, df1, df2, ncp, lower.tail = TRUE, log.p = FALSE)
Función acumulada	qf(p, df1, df2, ncp, lower.tail = TRUE, log.p = FALSE)
Función aleatoria	rf(n, df1, df2, ncp)

Continuación del apéndice 1.

Distribución Gamma	
Función de densidad	dgamma(x, shape, rate = 1, scale = 1/rate, log = FALSE)
Función probabilística	pgamma(q, shape, rate = 1, scale = 1/rate, lower.tail = TRUE, log.p = FALSE)
Función acumulada	qgamma(p, shape, rate = 1, scale = 1/rate, lower.tail = TRUE, log.p = FALSE)
Función aleatoria	rgamma(n, shape, rate = 1, scale = 1/rate)
Distribución geométrica	
Función de densidad	dgeom(x, prob, log = FALSE)
Función probabilística	pgeom(q, prob, lower.tail = TRUE, log.p = FALSE)
Función acumulada	qgeom(p, prob, lower.tail = TRUE, log.p = FALSE)
Función aleatoria	rgeom(n, prob)
Distribución hipergeométrica	
Función de densidad	dhyper(x, m, n, k, log = FALSE)
Función probabilística	phyper(q, m, n, k, lower.tail = TRUE, log.p = FALSE)
Función acumulada	qhyper(p, m, n, k, lower.tail = TRUE, log.p = FALSE)
Función aleatoria	rhyper(nn, m, n, k)

Continuación del apéndice 1.

Distribución Normal	
Función de densidad	dnorm(x, mean = 0, sd = 1, log = FALSE)
Función probabilística	pnorm(q, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)
Función acumulada	qnorm(p, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)
Función aleatoria	rnorm(n, mean = 0, sd = 1)
Distribución T de Student	
Función de densidad	dt(x, df, ncp, log = FALSE)
Función probabilística	pt(q, df, ncp, lower.tail = TRUE, log.p = FALSE)
Función acumulada	qt(p, df, ncp, lower.tail = TRUE, log.p = FALSE)
Función aleatoria	rt(n, df, ncp)
Distribución Uniforme	
Función de densidad	dunif(x, min = 0, max = 1, log = FALSE)
Función probabilística	punif(q, min = 0, max = 1, lower.tail = TRUE, log.p = FALSE)
Función acumulada	qunif(p, min = 0, max = 1, lower.tail = TRUE, log.p = FALSE)
Función aleatoria	runif(n, min = 0, max = 1)
Extraer coeficientes de un modelo residente en un objeto	coef(object, ...)
Extraer los pesos o ponderaciones de un objeto	weights(object, ...)
Extraer máxima verisimilitud de un objeto	logLik(object, ...)

Continuación del apéndice 1.

Extraer residuales de un modelo residente en un objeto	residuals(object, ...)
Filtro lineal a una serie de tiempo de una variable	filter(x, filter, method = c("convolution", "recursive"), sides = 2, circular = FALSE, init)
Generar una muestral de cuantiles dadas las probabilidades	quantile(x, probs = seq(0, 1, 0.25), na.rm = FALSE, names = TRUE, type = 7, ...)
Graficar una serie de tiempo	plot(x y = NULL, plot.type = c("multiple", "single"), xy.labels, xy.lines, panel = lines, nc, yax.flip = FALSE, mar.multi = c(0, 5.1, 0, if(yax.flip) 5.1 else 2.1), oma.multi = c(6, 0, 5, 0), axes = TRUE, ...)
Graficar varias series de tiempo	ts.plot(..., gpars = list())
Intervalos de confianza para los parámetros de un modelo ajustado	confint(object, parm, level = 0.95, ...)
Mediana de una muestra	median(x, na.rm = FALSE)
Muestra con o sin reemplazo de un objeto	sample(x, size, replace = FALSE, prob = NULL)
Objeto de serie de tiempo	ts(data = NA, start = 1, end = numeric(), frequency = 1, deltat = 1, ts.eps = getOption("ts.eps"), class = , names =)
Coerciona un objeto al tipo ts	as.ts(x, ...)
Comprueba si es un objeto tipo ts	is.ts(x)

Continuación del apéndice 1.

Obtener un vector aplicando una función	lapply(X, FUN, ...)
Promedio de un objeto	mean(x, ...)
Promedio ponderado	weighted.mean(x, w, ...)
Prueba T	t.test(x, ...)
Rango de datos (vector)	range(..., na.rm = FALSE)
Rango intercuartílico	IQR(x, na.rm = FALSE, type = 7)
Resumen de medidas de tendencia central y de posición o resumen de un objeto como una lista conteniendo el resultado de un modelo de regresión lineal	summary(objeto, ...)
Resumen por grupos y variables	numSummary(x, statistics, groups)
Suma de un vector	sum(..., na.rm = FALSE)
Test de correlación	cor.test(x, ...)
Test de normalidad Kolmogorov-Smirnov Tests (prueba no paramétrica)	ks.test(x, y, ..., alternative = c("two.sided", "less", "greater"), exact = NULL)
Test de Poisson	poisson.test(x, T = 1, r = 1, alternative = c("two.sided", "less", "greater"), conf.level = 0.95)
Test de Varianza	var.test(x, ...)
Test de raíz unitaria de Phillips Perron	PP.test(x, lshort = TRUE)
Varianza	var(x y = NULL, na.rm = FALSE, use)

Fuente: elaboración propia, con ayuda de R <help(base)>.

Apéndice 2. **Datos para práctica control estadístico por variables**

Datos para gráfica X R y S: Contenido en ml

n1	n2	n3	n4	n5	n1	n2	n3	n4	n5	n1	n2	n3	n4	n5	n1	n2	n3	n4	n5
501	505	510	496	499	510	504	503	509	510	506	498	501	508	500	498	505	506	501	503
497	503	499	503	509	498	503	501	510	507	509	506	506	502	499	501	499	507	499	507
504	509	510	505	504	503	509	508	500	510	508	504	507	510	500	497	498	508	509	510
502	501	498	501	501	502	497	496	500	509	501	506	509	508	509	505	501	498	497	509
503	496	498	503	502	508	502	503	509	507	509	509	510	508	499	508	509	503	502	496
498	496	504	510	508	507	510	501	503	507	500	502	509	499	501	498	503	505	505	501
499	507	503	504	510	505	496	509	496	502	505	504	498	502	510	505	505	498	498	506
504	497	496	498	506	510	502	505	509	505	507	510	504	506	498	498	507	499	497	507
502	497	496	504	505	508	505	498	502	504	502	496	503	501	506	509	496	508	505	503
500	496	510	507	496	508	504	500	508	507	502	509	505	503	506	501	498	504	505	498
509	503	501	504	503	496	510	504	497	510	504	497	502	506	506	510	497	499	509	500
509	506	500	501	509	505	503	504	503	502	496	505	501	505	497	496	509	504	500	506
498	504	506	508	509	505	506	501	500	500	502	503	496	496	503	507	505	508	510	501
501	509	503	496	498	497	507	507	508	509	507	498	505	500	503	510	501	503	501	498
501	501	507	499	497	498	509	510	502	498	503	496	500	504	501	499	499	500	508	502
502	507	503	508	501	497	502	510	500	510	509	505	506	504	509	503	498	500	500	504
510	506	499	509	510	507	508	497	508	499	504	510	508	509	500	500	496	496	498	498
508	503	500	503	510	510	506	505	510	498	509	500	500	510	506	509	507	498	507	509
509	502	509	504	504	500	510	497	497	507	496	510	499	496	504	503	501	510	505	509
510	503	504	502	499	499	508	503	503	499	500	497	503	501	503	508	506	509	500	502
503	509	509	497	509	504	497	497	496	510	503	506	499	510	500	502	507	503	503	499
501	498	507	498	510	507	501	510	508	498	506	500	503	500	510	496	501	498	507	510
510	509	504	498	509	509	503	508	499	506	496	496	504	502	497	498	497	496	508	510
504	509	497	496	508	502	501	510	505	497	501	498	503	507	502	502	499	498	496	498
504	500	510	509	508	505	509	505	497	506	506	504	501	507	502	507	498	504	503	502

Fuente: elaboración propia.

Apéndice 3. **Códigos y documentación del paquete XRSCC**

Los siguientes cuadros son la transcripción del contenido de la documentación en inglés de los paquetes XRSS y Planesmuestra (apéndice 4). Están detallados cada uno de los *script* de todas las funciones, omitiendo la documentación de los datos adjuntos al paquete. Al principio de cada uno figura el nombre del paquete o de la función.

Las descripciones de los paquetes contienen texto tanto en español como en inglés; la ortografía y formato de la documentación puede diferir de la fuente original ya que se adaptó al texto.

En la documentación de las funciones, notar el orden de las viñetas y significado de cada una.

Estructura de la documentación de las funciones

Inglés	**Español**
Title	Título de la función
Descript ion	Descripción breve de que hace la función
Usage	Uso de la función o sintaxis
Arguments	Especificación de cada argumento de la función
Details	Detalles necesarios
Value	Valores o resultados al ejecutar la función
Note	Notas necesarias
Author(s)	Autor o autores de la función
References	Referencias documentales para la programación de la función
See Also	Ver también: vínculos a otras funciones o paquetes
Examples	Ejemplos funcionales de la función

Continuación del apéndice 3.

- **Paquete XRSCC**

XRSCC-package {XRSCC} R *Documentation*

Calcula y grafica las cartas de control por variables (X, R y S) y atributos (p, np, c, u). ***Calculates and plots variable and attributes control charts*****.**

Description

Calcula los límites de control de las graficas de control por variables y atributos y utiliza funciones de iteracion específicas para calcular los verdaderos límites de control. Tambien calcula la capacidad del proceso de la grafica X y analiza la existencia de patrones no aleatorios en la misma utilizando las primeras 4 reglas de Western Electric. *Calculates the control limits for each type of variable or attribute control chart, then using an iteration to get the true control limits. Also calculates the process capability and gives a conclusion for non random patterns bassed in first four Western Electric rules.*

Details

Package: XRSCC

Type: Package

Version: 0.1

Date: 2016-05-04

License: GPL

Author(s) Erick Marroquin

Maintainer: Erick Marroquin <ericksuhel@gmail.com>

[Package XRSCC version 0.1]

Continuación del apéndice 3.

- ***Script*** **función xrs_gr( )**

```
xrs_gr<-function(X){
  # Validar la existencia del objeto con las muestras
if (missing(X)){
    stop("No hay muestras para leer, No sample to read")
  } else {
    x<-X
    m<-nrow(x)
    n<-ncol(x)
    X.prom<-apply(x,1,mean)
    f.rango<-function(x){
      f.rango.p<-range(x)
      return(f.rango.p[2]-f.rango.p[1])
    }
    X.range<-apply(x,1,f.rango)
    X.S<-apply(x,1,sd)
# Cargar las tablas para calcular los limites de control
    data(factor.a)
    # Calcular las expresiones de los limites de control
    # para ser evaluadas en las graficas
#   Limites de control grafica X
    LCS.X<-expression(mean(X.prom) + factor.a$A2[n-1] * mean(X.range))
    LCI.X<-expression(mean(X.prom) - factor.a$A2[n-1] * mean(X.range))
    LC.X<-expression(mean(X.prom))
    #   Limites de control grafica R
    LCS.R<-expression(mean(X.range)*factor.a$D4[n-1])
    LCI.R<-expression(mean(X.range)*factor.a$D3[n-1])
    LC.R<-expression(mean(X.range))
    #   Limites de control grafica S
    LCS.S<-expression(mean(X.S)*factor.a$B4[n-1])
    LCI.S<-expression(mean(X.S)*factor.a$B3[n-1])
    LC.S<-expression(mean(X.S))
#
    # Marco de Graficas
    mat<-matrix(1:4,2,2,byrow=TRUE)
layout(mat)
    layout.show(length(1:4))
    # Mostrar un histograma
    hist.X<-function(x=Xprom,breaks="Sturges"){
      hist(X.prom, breaks = breaks,
xlab="Valores", ylab="Frecuencia",
           main="Histograma de los promedios")
}
    # Script  para Gráfica X
    plot.X<-function(x=X.prom,type="b",col="blue",pch =19){
plot(x=x, xlab= "Numero de muestra", ylab="Valores de cada muestra",
```

Continuación del apéndice 3.

```
        main="Grafica X, Control Estadistico de la Calidad",type=type, col=col,
ylim=c(min(eval(LCI.X), min(X.prom)), max(eval(LCS.X), max(X.prom))),
        ylim=c(min(eval(LCI.X), min(X.prom)), max(eval(LCS.X), max(X.prom))),
        xlim=c(-0.05*m, 1.05*m), pch = pch)
    abline(h= c(eval(LCS.X), eval(LCI.X), eval(LC.X)),col="lightgray")
    text(c(rep(1,3),rep(7,3)), rep(c(eval(LCS.X),eval(LC.X),eval(LCI.X)),2),
        c(c("LCS = ","LC = ","LCI = "), c(round(eval(LCS.X),2),
                                round(eval(LC.X),2),
                                round(eval(LCI.X),2))), col="red")
    }
  # Script  para Grafica R
  plot.R<-function(x=X.range,type="b",col="black",pch =15){
plot(x=x, xlab= "Numero de muestra", ylab="Rangos de cada muestra",
        main="Grafica R, Control Estadistico de la Calidad",type=type, col=col,
ylim=c(min(eval(LCI.R)-min(X.range)*0.05, min(X.range)*.95),
                max(eval(LCS.R)+max(X.range)*0.05, max(X.range)*1.05)),
        xlim=c(-0.05*m, 1.05*m), pch = pch)
    abline(h= c(eval(LCS.R), eval(LCI.R), eval(LC.R)), col="lightgray")
    text(c(rep(1,3),rep(7,3)), rep(c(eval(LCS.R),eval(LC.R),eval(LCI.R)),2),
        c(c("LCS = ","LC = ","LCI = "),
          c(round(eval(LCS.R),2), round(eval(LC.R),2),
            round(eval(LCI.R),2))), col="red")
    }
  # Script  para Grafica S
  plot.S<-function(x=X.S,type="b",col="gray",pch =15){
plot(x=x, xlab= "Numero de muestra", ylab="Desviacion estandar de cada muestra",
        main="Grafica S, Control Estadistico de la Calidad",type=type, col=col,
        ylim=c(min(eval(LCI.S)-min(X.S)*0.05, min(X.S)*0.95),
             max(eval(LCS.S)+max(X.S)*0.05, max(X.S)*1.05)),
xlim=c(-0.05*m, 1.05*m), pch = pch)
    abline(h= c(eval(LCS.S), eval(LCI.S), eval(LC.S)), col="lightgray")
    text(c(rep(1,3),rep(7,3)), rep(c(eval(LCS.S),eval(LC.S),eval(LCI.S)),2),
        c(c("LCS = ","LC = ","LCI = "),
          c(round(eval(LCS.S),2), round(eval(LC.S),2),
            round(eval(LCI.S),2))), col="red")
    }
  X.pos <- which(X.prom > eval(LCI.X) & X.prom < eval(LCS.X))
  x.1<-x[X.pos,]
  X.fs<- which(X.prom >= eval(LCS.X))
  X.fi<- which(X.prom <= eval(LCI.X))
  X.f<-c(X.fs,X.fi)
  R.pos <- which(X.range > eval(LCI.R) & X.range < eval(LCS.R))
  X.range.1<-X.range[R.pos]
  S.pos <- which(X.S > eval(LCI.S) & X.S < eval(LCS.S))
  bin.X<-if(length(X.pos)< m){
    bin.X<-1
  } else {
    bin.X<-0
  }
```

Continuación del apéndice 3.

```
    bin.R<-if(length(R.pos)< m){
      bin.R<-1
    } else {
      bin.R<-0
    }
    bin.S<-if(length(S.pos)< m){
      bin.S<-1
    } else {
      bin.S<-0
    }
    hist.X()
    plot.X()
    plot.R()
    plot.S()
  }
  structure(list("in.control" = X.pos,
"R.in.control" = R.pos,
                 "out.control" = X.f,
                 "Iteraciones" = 1,
"data.0"= x,
                 "data.1"= x.1,
                 "data.r.1" = X.range.1,
                 "bin" = c(bin.X, bin.R, bin.S),
                 "LX"= c("LCI"=eval(LCI.X),  "LC"=eval(LC.X),"LCS"=eval(LCS.X)),
"LR"= c("LCI"=eval(LCI.R), "LC"=eval(LC.R), "LCS"=eval(LCS.R)),
                 "LS"= c("LCI"=eval(LCI.S), "LC"=eval(LC.S), "LCS"=eval(LCS.S)),
                 "Limites Grafica X" = c("LCI.X"=eval(LCI.X),
"LC.X"=eval(LC.X),"LCS.X"=eval(LCS.X)),
                 "Limites Grafica R" = c("LCI.R"=eval(LCI.R), "LC.R"=eval(LC.R),
"LCS.R"=eval(LCS.R)),
                 "Limites Grafica S" = c("LCI.S"=eval(LCI.S), "LC.S"=eval(LC.S),
"LCS.S"=eval(LCS.S)),
                 "Conclusion del proceso"= c(if(length(X.pos)< m){
                   print("Proceso fuera de Control en Grafica X")
                 } else {
                   print("El proceso esta bajo control en Grafica X")
}, if(length(R.pos)< m){
print("Proceso fuera de control en Grafica R")
                 } else {
                   print("El proceso esta bajo control en Grafica R")
}, if(length(S.pos)< m){
print("Proceso fuera de control en Grafica S")
                 } else {
                   print("El proceso esta bajo control en Grafica S")
                   }))
}
```

Continuación del apéndice 3.

- **Documentación función xrs_gr( )**

xrs_gr {XRSCC} R *Documentation*

Calculate and plot the X, R and S Charts for variable charts

Description

Calculates the control limits for X, R and S charts, using a data frame with a fixed subgroup size. Plots the corresponding graph, the function estimates if any value is out of the control limits, returns a list with calculations.

Usage

xrs_gr(X)

Arguments

X *A sample in a dataframe object, with m rows like subgroups, and n columns like sample size.*

Value

in.control	*The under control row list for the X chart*
R.in.control	*The under control row list for the R chart*
out.control	*The out of control row list for the X chart*
Iteraciones	*The iterations number, the firts and the last one on this function*
data.0	*The original data frame*

Continuación del apéndice 3.

data.1	*The under control subset after iteration*
data.r.1	The calculated ranges of data.0
bin	The binary values for out of control equal to one and under controlequal to zero, for X, R and S charts
LX	The X chart control limits vector
LR	The R chart control limits vector
LS	*The S chart control limits vector*
Limites Grafixa X	*The X chart control limits vector*
Limites Grafixa R	*The R chart control limits vector*
Limites Grafixa S	*The S chart control limits vector*
Conclusion del proceso	*The same results in a phrase as the bin values*
Author(s)	Erick Marroquin

References

Montgomery, D.C. (2005) *Introduction to statistical quality control*, 5th ed. New York: John Wiley & Sons, ISBN 0-471-65631-3

See Also

X_it, we_rules, R_it, Cp_X, Beta.X

Continuación del apéndice 3.

Examples

data(vol_sample)

results1<-xrs_gr(vol_sample)

[Package XRSCC version 0.1]

- **Script función X_it( )**

```
X_it<-function(prev.results){
# Validar la existencia del objeto con los resultados previos
if (missing(prev.results)){
   stop("No elementos para iteracion, No elements for iteration")
   } else {
     if(prev.results$bin[1]==0){
       stop("El proceso ya esta bajo control, The process is already under control")
       } else {
         x.1<-prev.results$data.1
         m.1<-nrow(x.1)
         n.1<-ncol(x.1)
         X.prom.1<-apply(x.1,1,mean)
         f.rango.1<-function(x.1){
           f.rango.p.1<-range(x.1)
           return(f.rango.p.1[2]-f.rango.p.1[1])
           }
         X.range.1<-apply(x.1,1,f.rango.1)
# Cargar las tablas para calcular los limites de control
         data(factor.a)
         # Calcular las expresiones de los límites de control
         # para ser evaluadas en las graficas
#   Limites de control grafica X
         LCS.X.1<-expression(mean(X.prom.1) + factor.a$A2[n.1-1] * mean(X.range.1))
         LCI.X.1<-expression(mean(X.prom.1) - factor.a$A2[n.1-1] * mean(X.range.1))
         LC.X.1<-expression(mean(X.prom.1))
         X.pos.1 <- which(X.prom.1 > eval(LCI.X.1) & X.prom.1 < eval(LCS.X.1))
         x.2<-x.1[X.pos.1,]
         X.fs.1<-which(X.prom.1 >= eval(LCS.X.1))
         X.fi.1<-which(X.prom.1 <= eval(LCI.X.1))
         X.f<-c(X.fs.1, X.fi.1)
         X.pos<-as.numeric(X.pos.1)
         #   Limites de control grafica R
         LCS.R<-expression(mean(X.range.1)*factor.a$D4[n.1-1])
```

Continuación del apéndice 3.

```
        LCI.R<-expression(mean(X.range.1)*factor.a$D3[n.1-1])
        LC.R<-expression(mean(X.range.1))
        #
        # Script  para Gráfica X Final
        plot.X.1<-function(x=X.prom.1,type="b",col="black",pch =19){
plot(x=x, xlab= "Numero de muestra", ylab="Valores de cada muestra",
main="Grafica X, Muestra Final",type=type, col=col,
             ylim=c(min(eval(LCI.X.1), min(X.prom.1)), max(eval(LCS.X.1), max(X.prom.1))),
             xlim=c(-0.05*m.1, 1.05*m.1), pch = pch)
          abline(h= c(eval(LCS.X.1), eval(LCI.X.1), eval(LC.X.1)),col="lightgray")
          text(c(rep(1,3),rep(7,3)), rep(c(eval(LCS.X.1),eval(LC.X.1),eval(LCI.X.1)),2),
             c(c("LCS = ","LC = ","LCI = "), c(round(eval(LCS.X.1),2),
                                    round(eval(LC.X.1),2),
                                    round(eval(LCI.X.1),2))), col="red", cex=1.25)
          }
        dev.off()
        R.pos <- which(X.range.1 > eval(LCI.R) & X.range.1 < eval(LCS.R))
        R.pos<- as.numeric(R.pos)
        X.range.2<-X.range.1[R.pos]
        bin.X<-if(length(X.pos)< m.1){
          bin.X<-1
        } else {
          bin.X<-0
        }
        bin.R<-if(length(R.pos)< m.1){
          bin.R<-1
        } else {
          bin.R<-0
        }
        plot.X.1()
        structure(list("in.control" = X.pos,
"R.in.control" = R.pos,
"out.control" = X.f,
                       "data.0" = prev.results$data.0,
                       "data.1" = x.2,
                       "data.r.1" = X.range.2,
                       "bin" = c(bin.X,bin.R,0),
"Iteraciones"= prev.results$Iteraciones + 1,
                       "LX"= c("LCI"=eval(LCI.X.1),  "LC"=eval(LC.X.1),"LCS"=eval(LCS.X.1)),
                       "LR"= c("LCI"=eval(LCI.R), "LC"=eval(LC.R), "LCS"=eval(LCS.R)),
                       "Limites Grafica X" =
c("LCI.X"=eval(LCI.X.1),"LC.X"=eval(LC.X.1),"LCS.X"=eval(LCS.X.1)),
                       "Limites Grafica R" = c("LCI.R"=eval(LCI.R), "LC.R"=eval(LC.R),
"LCS.R"=eval(LCS.R)),
```

Continuación del apéndice 3.

```
                    "Conclusion del proceso"= c(if(length(X.pos)< m.1){
                      print("Proceso fuera de Control en Grafica X")
                      } else {
                        print("El proceso esta bajo control en Grafica X")
}, if(length(R.pos)< m.1){
print("Proceso fuera de control en Grafica R")
                        } else {
                          print("El proceso esta bajo control en Grafica R")
                        }))
      }
   }
}
```

- **Documentación función X_it( )**

X_it {XRSCC} R Documentation

Calculates the iteration i'th X Chart

Description

With the results of xrs_gr followed by previous X_it iterations, the function calculates the X control limits charts, using a data frame with a fixed subgroup size n. In the graph plotting, the function estimates if any value (row or subgroup average) is out of control limits, and returns a list with calculations. Also, gives the R chart and control limits, which will be used in R_it function.

Usage

X_it(prev.results)

Continuación del apéndice 3.

Arguments

prev.results *Is a list of previous results obtained by the* xrs_gr *function in the first iteration, or a list of results obtained in further iterations by the* X_it *function.*

Details

The function stops if the X chart is under control already, and also stops if there is not any active graphic device.

Value

in.control	*The under control row list for the X chart*
R.in.control	*The under control row list for the R chart*
out.control	*The out of control row list for the X chart*
Iteraciones	*The iterations number, It is assumed to be the second or later*
data.0	*The original data frame*
data.1	*The under control subset after iteration*
data.r.1	*The calculated ranges of data.0*
bin	*The binary values for out of control equal to one and under controlequal to zero, for X and R charts*
LX	*The X chart control limits vector*
LR	*The R chart control limits vector*

Continuación del apéndice 3.

Limites Grafixa X	*The X chart control limits vector*
Limites Grafixa R	*The R chart control limits vector*
Conclusion del proceso	*The same results in a phrase as the bin values*

Note

For the true Range control limits calculation, use R_it.

Author(s)

Erick Marroquin

References

Montgomery, D.C. (2005) *Introduction to statistical quality control*, 5th ed. New York: John Wiley & Sons, ISBN 0-471-65631-3

See Also

xrs_gr, R_it, Cp_X, we_rules

Examples

```
data(vol_sample)
results1<-xrs_gr(vol_sample)
results2<-X_it(results1)
```

[Package XRSCC version 0.1]

Continuación del apéndice 3.

Script función R_it( )

```
R_it<-function(prev.results){
# Validar la existencia del objeto con los resultados previos
if (missing(prev.results)){
   stop("No elementos para iteracion, No elements for iteration")
  } else {
   if(prev.results$bin[2]==0){
     stop("El proceso ya esta bajo control, The process is already under control")
   } else {
     x.0<-prev.results$data.1
     R.pos.0<-prev.results$R.in.control
x.1<-x.0[R.pos.0,]
     m.1<-nrow(x.1)
n.1<-ncol(x.1)
     X.prom.1<-apply(x.1,1,mean)
     f.rango.1<-function(x.1){
       f.rango.p.1<-range(x.1)
       return(f.rango.p.1[2]-f.rango.p.1[1])
     }
     X.range.1<-apply(x.1,1,f.rango.1)
# Cargar las tablas para calcular los limites de control
     data(factor.a)
     # Calcular las expresiones de los límites de control
#   Limites de control grafica X
     LCS.X.1<-expression(mean(X.prom.1) + factor.a$A2[n.1-1] * mean(X.range.1))
     LCI.X.1<-expression(mean(X.prom.1) - factor.a$A2[n.1-1] * mean(X.range.1))
     LC.X.1<-expression(mean(X.prom.1))
     X.pos.1 <- which(X.prom.1 > eval(LCI.X.1) & X.prom.1 < eval(LCS.X.1))
     x.2<-x.1[X.pos.1,]
     X.fs.1<-which(X.prom.1 >= eval(LCS.X.1))
     X.fi.1<-which(X.prom.1 <= eval(LCI.X.1))
     X.f<-c(X.fs.1, X.fi.1)
     X.pos<-as.numeric(X.pos.1)
     #   Limites de control grafica R
     LCS.R<-expression(mean(X.range.1)*factor.a$D4[n.1-1])
     LCI.R<-expression(mean(X.range.1)*factor.a$D3[n.1-1])
     LC.R<-expression(mean(X.range.1))
     # Script  para Gráfica X Final
     plot.X.1<-function(x=X.prom.1,type="b",col="black",pch =19){
plot(x=x, xlab= "Numero de muestra", ylab="Valores de cada muestra",
main="Grafica X, Muestra Final",type=type, col=col,
          ylim=c(min(eval(LCI.X.1), min(X.prom.1)), max(eval(LCS.X.1), max(X.prom.1))),
          xlim=c(-0.05*m.1, 1.05*m.1), pch = pch)
       abline(h= c(eval(LCS.X.1), eval(LCI.X.1), eval(LC.X.1)),col="lightgray")
       text(c(rep(1,3),rep(7,3)), rep(c(eval(LCS.X.1),eval(LC.X.1),eval(LCI.X.1)),2),
          c(c("LCS = ","LC = ","LCI = "), c(round(eval(LCS.X.1),2),
```

Continuación del apéndice 3.

```
          c(c("LCS = ","LC = ","LCI = "), c(round(eval(LCS.X.1),2),
                                            round(eval(LC.X.1),2),
                                            round(eval(LCI.X.1),2))), col="red", cex=1.25)
    }
    dev.off()
    R.pos <- which(X.range.1 > eval(LCI.R) & X.range.1 < eval(LCS.R))
    R.pos<-as.numeric(R.pos)
    X.range.2<-X.range.1[R.pos]
    bin.X<-if(length(X.pos)< m.1){
      bin.X<-1
    } else {
      bin.X<-0
    }
    bin.R<-if(length(R.pos)< m.1){
      bin.R<-1
    } else {
      bin.R<-0
    }
    plot.X.1()
    structure(list("in.control" = X.pos,
                   "R.in.control" = as.numeric(R.pos),
                   "out.control" = X.f,
                   "data.0" = prev.results$data.0,
                   "data.1" = x.2,
                   "data.r.1" = X.range.2,
                   "bin" = c(bin.X,bin.R,0),
"Iteraciones"= prev.results$Iteraciones + 1,
                   "LX"= c("LCI"=eval(LCI.X.1),  "LC"=eval(LC.X.1),"LCS"=eval(LCS.X.1)),
                   "LR"= c("LCI"=eval(LCI.R), "LC"=eval(LC.R), "LCS"=eval(LCS.R)),
                   "Limites Grafica X" =
c("LCI.X"=eval(LCI.X.1),"LC.X"=eval(LC.X.1),"LCS.X"=eval(LCS.X.1)),
                   "Limites Grafica R" = c("LCI.R"=eval(LCI.R), "LC.R"=eval(LC.R),
"LCS.R"=eval(LCS.R)),
                   "Conclusion del proceso"= c(if(length(X.pos)< m.1){
                     print("Proceso fuera de Control en Grafica X")
                   } else {
                     print("El proceso esta bajo control en Grafica X")
}, if(length(R.pos)< m.1){
print("Proceso fuera de control en Grafica R")
                   } else {
                     print("El proceso esta bajo control en Grafica R")
                   }))
   }
  }
}
```

Continuación del apéndice 3.

Documentación función xrs_gr( )

R_it {XRSCC} R Documentation

Calculates the i'th iteration R Chart

Description
Calculates the iteration i'th for R chart, after the X chart is under control. The function estimates if any value (range) is out of control limits, and returns a values list.

Usage
R_it(prev.results)

Arguments

prev.results	*Is a list of previous results obtained by the xrs_gr, followed by X_itfunction if it is necessary. In other cases, needs more than one iteration to obtain the true control limits for R chart, before take conclusions about the process.*

Details
The function stops if the R chart is under control already, and also stops if there is not any active graphic device.

Value

in.control	*The under control row list for the X chart*
R.in.control	*The under control row list for the R chart*
out.control	*The out of control row list for the X chart*
Iteraciones	*The number of iterations, It is assumed to be the second or later*
data.0	The original data frame
data.1	The filtered data frame

Continuación del apéndice 3.

data.r.1	*The calculated ranges of data.0*
bin	*The binary values for out of control equal to one and under controlequal to zero, for X and R charts*
LX	*The X chart control limits vector*
LR	*The R chart control limits vector*
Limites Grafixa X	*The X chart control limits vector*
Limites Grafixa R	*The R chart control limits vector*
Conclusion del proceso	*The same results in a phrase as the bin values*

Author(s)

Erick Marroquin

References

Montgomery, D.C. (2005) *Introduction to statistical quality control*, 5th ed. New York: John Wiley & Sons, ISBN 0-471-65631-3

See Also

xrs_gr X_it we_rules Cp_X

Examples

```
data(dato2)
results1<-xrs_gr(dato2)
results2<-X_it(results1)
results3<-R_it(results2)
```

[Package XRSCC version 0.1]

Continuación del apéndice 3.

Script función we_rules( )

```
we_rules<-function(prev.results){
  if (missing(prev.results)){
    stop("No elementos para evaluar")
  } else {
    data(factor.a)
    n.1<-ncol(prev.results$data.1)
    LCR<-as.numeric(prev.results$LR[2])
    X.sigma<-expression(eval(LCR)/((factor.a$d2[n.1-1])*sqrt(n.1)))
    x.1<-prev.results$data.1
    m.1<-nrow(x.1)
    X.prom.1<-apply(x.1,1,mean)
    f.rango.1<-function(x.1){
      f.rango.p.1<-range(x.1)
      return(f.rango.p.1[2]-f.rango.p.1[1])
    }
    X.range.1<-apply(x.1,1,f.rango.1)
LCI.X<-prev.results$LX[1]
    LC.X<-prev.results$LX[2]
    LCS.X<-prev.results$LX[3]
    #
    # Script  para Grafica X con zonas
plot.Xzones<-function(x=X.prom.1,type="b",col="blue",pch =19){
plot(x=x, xlab= "Numero de muestra",
          ylab="Valores de cada muestra",
          main="Grafica X, Control Estadistico de la Calidad",
          sub="Zonas para el analisis de las reglas Western Electric",
type=type, col=col,
          ylim=c(min(eval(LCI.X)-mean(X.range.1)*0.05, LC.X -3*eval(X.sigma)*1.05),
                max(eval(LCS.X)+mean(X.range.1)*0.05, LC.X +3*eval(X.sigma)*1.05)),
          xlim=c(-0.05*m.1, 1.05*m.1), pch = pch)
# Agregar opciones de gráficos de bajo nivel
abline(h= c(eval(LCS.X), eval(LCI.X), eval(LC.X)),col="lightgray")
      text(c(rep(0,3),rep(7,3)), rep(c(eval(LCS.X),eval(LC.X),eval(LCI.X)),2),
          c(c("LCS = ","LC = ","LCI = "), c(round(eval(LCS.X),3),
                                    round(eval(LC.X),3), round(eval(LCI.X),3))),
col="red",cex=1)
      # Agregar zonas de + / - 3 sigmas
abline(h= c(c(-3:-1,1:3)*eval(X.sigma)+eval(LC.X)), col="green")
      text(rep(m.1,6), c(c(-3:-1,1:3)*eval(X.sigma)+eval(LC.X)),
          c(expression(-3*hat(sigma)),expression(-2*hat(sigma)),
            expression(-1*hat(sigma)),expression(+1*hat(sigma)),
            expression(+2*hat(sigma)), expression(+3*hat(sigma))),
col="black",cex=1)
```

Continuación del apéndice 3.

```
    # Agregar nombre a las zonas
text(rep(m.1,6),c(c(-2.5,-1.5,-0.5,0.5,1.5,2.5)*eval(X.sigma)+eval(LC.X)),
c("A","B","C","C","B","A"), cex=2,col="gray")
  }
  # Funcion para encontrar zonas
  limites.zona <- seq(-3, 3)
  # Crea el conjunto de zonas para cada punto de la carta X
zonas <- sapply(limites.zona,
                function(i) {
                  i * rep(eval(X.sigma), m.1)
})
  zonas<-zonas + eval(LC.X)
  colnames(zonas) <- paste("zona", -3:3, sep="_")
  # Evaluar las zonas
  x.zonas <- rowSums(X.prom.1 > zonas)
  # Pasar por cada valor de 1 hasta m.1
  for(i in 1:m.1){
    # Cualquier valor mas alla +/- 3 sigmas
v1 <- x.zonas[i]
    regla1 <- ifelse(any(v1 == 7), 1,
ifelse(any(v1 == 0),1, 0))
    #
    # Dos de cada tres puntos en la zona A
v2 <- x.zonas[max(i-3, 1):i]
    regla2 <- ifelse(sum(v2 == 6) >= 2, 1,
                     ifelse(sum(v2 == 1) >= 2, 1, 0))
# Cuatro de cada cinco puntos mas alla de la zona C
v3 <- x.zonas[max(i-5, 1):i]
    regla3 <- ifelse(sum(v3 >= 5) >= 4, 1,
                     ifelse(sum(v3 <= 2) >= 4, 1, 0))
# Ocho puntos consecutivos de un lado de la
    # linea de control central
    v4 <- x.zonas[max(i-8, 1):i]
regla4 <- ifelse(all(v4 >= 4), 1,
                     ifelse(all(v4 <= 3), 1, 0))
}
  bin.w<- if(sum(c(regla1,regla2,regla3,regla4))> 0){
bin.w<-1
  } else {
    bin.w<-0
}
  # Ejecutar grafico
  dev.off()
  plot.Xzones()
  # vector de resultados
  structure(list("Resultados del analisis" = if(bin.w>0){
    print("El proceso esta fuera de control por las Reglas Western Electric")
    print("Las siguientes reglas tienen al menos un grupo que viola la regla")
```

Continuación del apéndice 3.

```
      c("Regla 1"= regla1,
        "Regla 2"= regla2,
        "Regla 3"= regla3,
        "Regla 4"= regla4)
    } else {
      print("El proceso esta bajo control por las reglas Western Electric")
    }))
  }
}
```

Documentación función we_rules( )

we_rules {XRSCC} R Documentation

Estimates the first four Western Electric Rules for detecting patterns

Description

Estimates the first four Western Electric Rules for detecting patterns, starting with under control X chart obtained in the sequence xrs_gr, X_it, R_it functions. At the same time, plots the X chart including the zones above and below the central limit. For last, a binary value for each rule is presented if at least one rule is violated, '1' for 'yes', 0 for 'no'.

Usage

we_rules(prev.results)

Arguments

prev.results *Its a list of previous results obtained by the xrs_gr function in the first iteration, or a list of results obtained in further iterations by the X_it, and if necessary by the R_it function.*

Continuación del apéndice 3.

Details

The previous results may say that the process is under control, but, it's a conclusion concerning the first Western Electric rule only.

Value

Resultados de analisis	*A phrarse saying the process is or not under control*
Las siguientes reglas tienen al menos un grupo que viola la regla	*The conclussion about the Western Electric rules from 1 to 4, showing a binary response, '1' for 'yes', 0 for 'no'.*

Author(s)

Erick Marroquin

References

Montgomery, D.C. (2005) *Introduction to statistical quality control*, 5th ed. New York: John Wiley & Sons, ISBN 0-471-65631-3

SMALL, Bonnie B. (1956) *Statistical Quality Control Handbook*, 2th ed. Easton : Western Electric Co, Inc.

yhat The Yhat Blog. Machine Learning, Data Science, Engineering, [On line] http://blog.yhathq.com/posts/quality-control-in-r.html

See Also

xrs_gr, X_it, R_it, Cp_X

Continuación del apéndice 3.

Examples

```
data(qqsugar)
results1<-xrs_gr(qqsugar)
results2<-R_it(results1)
we_rules(results2)
```

[Package XRSCC version 0.1]

Script función Beta_X( )

```
Beta.X<-function(k,n){
  beta.x1<-function(k,n,L=3){
    phi1 <- pnorm(c(L-k*sqrt(n)), mean=0, sd=1, lower.tail=TRUE)
    phi2 <- pnorm(c(-L-k*sqrt(n)), mean=0, sd=1, lower.tail=TRUE)
    beta.1<-phi1-phi2
    return(beta.1)
  }
  beta.2<-beta.x1(k,n)
  k.1<-seq(0,k*1.5,length.out=100)
  beta.x2<-beta.x1(k.1,n)
  ARL.x2<-1/(1-beta.x2)
  #CO
  plot(k.1, beta.x2, type = "l", lty = 1, lwd = 2,
       lend = par("lend"),
       col = "green", cex = 2, bg = NA,
       xlab = expression(kappa), ylab = expression(beta),
       xlim = c(0, k*1.5),
       ylim = c(0,1),
pch = 19,
       main ="Curva Caracteristica de Operacion
       Corrimiento de la Media")
  # Agrega opciones de graficas de bajo nivel
grid(10, 10, lwd = 0)
  segments(x0=0, y0=0, x1=k*1.5, y1=0, col="black",lwd=1)
segments(x0=0, y0=0, x1=0, y1=1, col="black",lwd=1)
  segments(x0=k, y0=0, x1=k, y1=eval(beta.2), col=2,lwd=2)
  segments(x0=0, y0=eval(beta.2), x1=k, y1=eval(beta.2), col=2,lwd=2)
text(k*1.1,eval(beta.2),expression(beta), cex = 1)
```

Continuación del apéndice 3.

```
  text(k*1.15,eval(beta.2),paste(" = "), cex = 1)
  text(k*1.25,eval(beta.2),paste(round(beta.2,3)), cex = 1)
#ARL
  plot(k.1, ARL.x2, type = "l", lty = 1, lwd = 2,
       lend = par("lend"),
       pch = NULL,
       col = "red", cex = 2, bg = NA,
       xlab = expression(kappa), ylab = paste("ARL"),
       xlim = c(0, k*1.5), ylim = c(0,eval(1/(1-beta.2))*1.2),
main ="Ancho medio de la corrida
       para detectar corrimiento de la Media")
  # Agrega opciones de graficas de bajo nivel
grid(10, 10, lwd = 0)
  segments(x0=0, y0=0, x1=k*1.5, y1=0, col="black",lwd=1)
  segments(x0=0, y0=0, x1=0, y1=eval(1/(1-beta.2))*1.2, col="black",lwd=1)
  segments(x0=k, y0=0, x1=k, y1=eval(1/(1-beta.2)), col="blue",lwd=2)
  segments(x0=0, y0=eval(1/(1-beta.2)), x1=k, y1=eval(1/(1-beta.2)), col="blue",lwd=2)
  text(k*1.05,eval(1/(1-beta.2)),paste("ARL"), cex = 1)
  text(k*1.10,eval(1/(1-beta.2)),paste(" = "), cex = 1)
  text(k*1.25,eval(1/(1-beta.2)),paste(round(eval(1/(1-beta.2)),3)), cex = 1)
  structure(list("beta"= beta.2,
                 "ARL"=1/(1-beta.2)))
}
```

Documentación función we_rules( )

Beta.X {XRSCC} R Documentation

X chart OC Curve

Description

Calculates and plots the risk of not detecting shifts and the Average Run Length

Usage

Beta.X(k,n)

Continuación del apéndice 3.

Arguments

k *A numeric vector, of length one, is the k standard deviations factor since the known mean*

n *An integer, equal the sample size*

Value

beta *risk of not detecting shifts*

ARL *Average Run Lengh*

Author(s)

Erick Marroquin

References

Montgomery, D.C. (2005) *Introduction to statistical quality control*, 5th ed. New York: John Wiley & Sons, ISBN 0-471-65631-3

See Also

xrs_gr

Examples

```
Beta.X(k=1,n=5)
Beta.X(k=0.5,n=5)
Beta.X(k=1,n=3)
```

[Package XRSCC version 0.1]

Continuación del apéndice 3.

Script función Cp_X( )

```
Cp_X<-function(prev.results,LES,LEI,mu){
  if (missing(prev.results)){
    stop("No elementos para evaluar")
} else {
    if(missing(LES) | missing(LEI)){
stop("Al menos uno de los limites de especificacion no esta definido")
} else {
      data(factor.a)
      n<-ncol(prev.results$data.1)
      LCR<-as.numeric(prev.results$LR[2])
      if (missing(mu)){
        mu<-prev.results$LX[2]
        X.sigma<-expression(eval(LCR)/(factor.a$d2[n-1]*sqrt(n)))
      } else {
X.sigma<-expression(eval(LCR)/(factor.a$d2[n-1]))
      }
      #Sequencia
      sd<-eval(X.sigma)
.x<-seq(mu -4*sd,mu +4*sd, length = 1000)
      fx0<-expression((1/(sd*sqrt(2*pi)))*exp(-0.5*(((.x-mu)/sd)^2)))
      plot(eval(fx0)~.x,type="l", xlab="X", ylab="Densidad",
main=paste("Distribucion Normal: Normalizacion Grafica X"),
xlim=c(min(mu -4*sd, LEI), max(mu +4*sd,LES)))
x0<-c(mu -3*eval(X.sigma), mu,  mu +3*eval(X.sigma))
      y0<-rep(0,3)
      # Definir la funcion de densidad
fx0<-expression((1/(sd*sqrt(2*pi)))*exp(-0.5*(((x0-mu)/sd)^2)))
segments(x0, y0, x1=x0, y1=eval(fx0), col=2,lwd=2)
      abline(a=0,b=0)
      # Se agregan las siguientes lineas que los limites de especificacion
      # bajo la misma funcion de densidad
      x0<-c(eval(LEI), eval(LES))
      xt.1<-eval(LES)
      y0<-rep(0,2)
#Densidades
      fx1<-expression((1/(sd*sqrt(2*pi)))*exp(-0.5*(((x0-mu)/sd)^2)))
      fx2<-expression((1/(sd*sqrt(2*pi)))*exp(-0.5*(((xt.1-mu)/sd)^2)))
      fx3<-expression((1/(sd*sqrt(2*pi)))*exp(-0.5*(((0)/sd)^2)))
segments(x0, y0, x1=x0,y1=eval(fx1), col=4,lwd=2)
      # Definir el texto de los limites para mostrar
text(c(mu -3*sd, mu,  mu +3*sd,
eval(LEI), eval(LES),mu -3*sd, mu +3*sd, mu), c(rep(eval(fx3)*0.3,3),
```

Continuación del apéndice 3.

```
                                        rep(eval(fx3)*0.15,2),
                                        rep(eval(fx3)*0.25,3)),
        c(expression(-3*hat(sigma)),expression(mu), expression(+3*hat(sigma)),
          paste(c("LEI =","LES ="), c(round(eval(LEI),1), round(eval(LES),1))),
          paste(c(round(mu -3*sd,3), round(mu +3*sd,3), round(mu,3)))), cex = 1.25, col="blue")
# Define la ecuacion simbolica de la funcion de densidad
text(mu -3*sd,eval(fx3)*0.9,
expression(f(x) == paste(frac(1, sigma * sqrt(2 * pi)), " ", e^{frac(-(x - mu)^2, 2 * sigma^2)})),
        cex = 1.25, col="gray")
    # Muestra la funcion simbolica de la capacidad del proceso
text(mu + 3*sd,eval(fx3)*0.9,
        expression(Cp == paste(frac("LES - LEI", 6 * hat(sigma)))),cex = 1, col="gray")
# Define el indicador de la Capacidad del proceso Cp
    Cp <- (LES - LEI)/(6 * sd)
    Cpk<- min(LES - mu, mu - LEI)/(3 * sd)
    # Muestra El resultado de la capacidad del proceso
text(mu + 3*sd,eval(fx3)*0.7, paste("Cp =", round(eval(Cp),2)))
# Define el indicador del porcentaje del uso de los limites de especificacion
    P.cp <- (1/eval(Cp))*100
    # Muestra El resultado del porcentaje del uso de los limites de especificacion
text(mu + 3*sd,eval(fx3)*0.6, paste("P =", round(eval(P.cp),2), "%"))
#
    # Crear la lista de los resultados Cp y P
    structure(list("Cp"=eval(Cp), "Cpk" = eval(Cpk), "P"=eval(P.cp), "X.sigma"= eval(X.sigma)),
            "Concluision del Proceso" = if(eval(Cp)>1){
              print("Los limites naturales estan dentro de los limites de especificacion")
            } else {
              if(eval(Cp)<1){
                print("Los limites naturales sobrepasan  los limites de especificacion")
              } else {
                print("Los limites naturales son iguales a los limites de especificacion")
                }
            })
  }
 }
}
```

Continuación del apéndice 3.

Documentación función Cp_X( )

Cp_X {XRSCC} R Documentation

Calculates the process capability

Description

Given a variable sample, the function calculates the process capability and, assuming a normal distribution of the X chart, after the true control limits were found.

Usage

Cp_X(prev.results, LES, LEI, mu)

Arguments

prev.results	*Is a list of previous results obtained by the xrs_gr function in the first iteration, or the results obtained in further iterations by the X_it function.*
LES	*A numeric vector of length one, containing the upper specification limit.*
LEI	*A numeric vector of length one, containing the lower specification limit.*
mu	*A numeric vector of length one, containing the average specification, if not exists, function takes the Control Limit of previous results.*

Continuación del apéndice 3.

Details

The function stops for the lack of any arguments.

Value

Cp	*The process capability index*
Cpk	*The process capability index in case is not centered*
P.cp	*The specification range percentage used by the control limits*
X.sigma	*The process standard deviation*
Conclusion del proceso	*A phrase to take conclusion about the process capability*

Author(s)

Erick Marroquin

References

Montgomery, D.C. (2005*) Introduction to statistical quality control*, 5th ed. New York: John Wiley & Sons, ISBN 0-471-65631-3

See Also

xrs_gr X_it R_it we_rules

Examples

```
data(vol_sample)
results1<-xrs_gr(vol_sample)
```

Continuación del apéndice 3.

```
results2<-X_it(results1)
# Type dev.off() function before use Cp_X
Cp_X(results2, LES=510, LEI=490, mu=500)
```

[Package XRSCC version 0.1]

Script función p_gr( )

```
p_gr<-function(D,n){
  if (missing(D)){
    stop("No hay muestras para leer, No sample to read")
  } else {
if (missing(n)){
      stop("Debe fijar un ancho de la muestra para calcular la proporcion")
} else {
        p.0<-D[,1]/n
        if(max(p.0) >= 1){
stop("No puede existir una proporcion mayor a la unidad")
          } else {
            m <-nrow(D)
            # Calculo de limites de control para la grafica P
LCS.p.0<-expression(mean(p.0)+3*sqrt((mean(p.0)*(1-mean(p.0)))/n))
            LCI.p.0<-expression(mean(p.0)-3*sqrt((mean(p.0)*(1-mean(p.0)))/n))
            LC.p.0<-expression(mean(p.0))
            if (eval(LCI.p.0)>0){
              LCI.p.0<-eval(LCI.p.0)
              } else {
                LCI.p.0 <- 0
                }
            p.pos<-which(p.0 >= eval(LCI.p.0)  & p.0 < eval(LCS.p.0))
            p.1<-p.0[p.pos]
            p.fi.0<-which(p.0 < eval(LCI.p.0))
p.fs.0<-which(p.0 >= eval(LCS.p.0)) # Solo valores positivos
bin.p<-if(length(p.pos)< m){
              bin.p<-1
              } else {
                bin.p<-0
                }
```

Continuación del apéndice 3.

```
          #
          # Script  para Grafica p inicial
          plot.p<-function(P=p.0,type="b",col="blue",pch =19){
plot(P, xlab= "Numero de muestra",
                ylab="Proporcion de los no conformes de cada muestra",
                main="Grafica P, Control Estadistico de la Calidad",
type=type, col=col,
                ylim=c(eval(LCI.p.0)-mean(p.0)*0.05,
                     max(eval(LCS.p.0)*1.1, max(p.0)*1.1)),
                xlim=c(-0.05*m, 1.05*m), pch = pch)
            abline(h= c(eval(LCS.p.0), eval(LCI.p.0),
                     eval(LC.p.0)),col="lightgray")
            text(c(rep(1,3),rep(7,3)),
                rep(c(eval(LCS.p.0),eval(LC.p.0),eval(LCI.p.0)),2),
                c(c("LCS = ","LC = ","LCI = "),
                  c(round(eval(LCS.p.0),3),
                    round(eval(LC.p.0),3), round(eval(LCI.p.0),3))),
col="red") }
          plot.p()
          # Crea la lista de los resultados
structure(list("in.control" = p.pos,
                       "out.control"= c(p.fi.0,p.fs.0),
"Iteraciones" = 1,
                       "data.n"=n,
                       "data.0"= D,
"data.1"= p.1,
                       "bin" = bin.p,
"Limites de Control Grafica p" = c("LCI.p"=eval(LCI.p.0),"LCS.p"=eval(LCS.p.0),
                                                "LC.p"=eval(LC.p.0)),
                       "Conclusion del proceso"= c(if(length(p.pos)< m){
                         print("Proceso fuera de Control en Grafica p")
                         } else {
                           print("El proceso esta bajo control en Grafica p")
                           }))
        }
    }
  }
}
```

Continuación del apéndice 3.

Documentación función p_gr( )

p_gr {XRSCC} R Documentation

P control chart for attributes

Description

Calculates the p control chart for attributes, using a sample D of number of defectives or nonconforming items and a constant sample size n. The values plotted in graph are the fractions pof defectives.

Usage

p_gr(D, n)

Arguments

D	*A data frame containing in one column the non conforming items, and must be integer and non negative.*
n	*A vector of length one, integer and nonnegative, to fix the sample size.*

Value

in.control	*The under control row list for the p chart*
out.control	*The out of control row list for the p chart*
Iteraciones	*The number of iterations, in this function always will be the first and the last one*
data.n	*The fixed sample size*
data.0	*The original data frame*

Continuación del apéndice 3.

data.1	*The filtered data frame*
bin	*The binary values for out of control equal to one and under control equal to zero*
Limites de Control Grafica p	*The p chart control limits vector*
Conclusion del proceso	*The same results in a phrase as the bin values*

*Author(*s)
Erick Marroquin

References
Montgomery, D.C. (2005) *Introduction to statistical quality control*, 5th ed. New York: John Wiley & Sons, ISBN 0-471-65631-3

See Also
P_it, c_gr, C_it, np_gr, NP_it, u_gr, U_it

Examples
```
data(bottles)
p_gr(bottles, n=100)
```

[Package XRSCC version 0.1]

Continuación del apéndice 3.

Script función P_it( )

```
P_it<-function(prev.results){
  if (missing(prev.results)){
    stop("No elementos para iteracion, No elements for iteration")
  } else {
    if(prev.results$bin[1]==0){
      stop("El proceso ya esta bajo control, The process is already under control")
    } else {
      p.0<-prev.results$data.1
      m <-length(p.0)
n <-prev.results$data.n
      # Calculo de limites de control para la grafica P
LCS.p.0<-expression(mean(p.0)+3*sqrt((mean(p.0)*(1-mean(p.0)))/n))
      LCI.p.0<-expression(mean(p.0)-3*sqrt((mean(p.0)*(1-mean(p.0)))/n))
      LC.p.0<-expression(mean(p.0))
      if (eval(LCI.p.0)>0){
        LCI.p.0<-eval(LCI.p.0)
        } else {
          LCI.p.0 <- 0
          }
      p.pos<-which(p.0 > eval(LCI.p.0)  & p.0 < eval(LCS.p.0))
      p.1<-p.0[p.pos]
      p.fi.0<-which(p.0 < eval(LCI.p.0))
      p.fs.0<-which(p.0 >= eval(LCS.p.0))
      bin.p<-if(length(p.pos)< m){
        bin.p<-1
        } else {
          bin.p<-0
          }
      #
      # Script  para Grafica p inicial
      plot.p<-function(P=p.0,type="b",col="blue",pch =19){
plot(P, xlab= "Numero de muestra", ylab="Proporcion de los no conformes de cada muestra",
            main="Grafica P, Control Estadistico de la Calidad",type=type, col=col,
ylim=c(eval(LCI.p.0)-mean(p.0)*0.05, max(eval(LCS.p.0)*1.1, max(p.0)*1.1)),
            xlim=c(-0.05*m, 1.05*m), pch = pch)
        abline(h= c(eval(LCS.p.0), eval(LCI.p.0), eval(LC.p.0)),col="lightgray")
        text(c(rep(1,3),rep(7,3)), rep(c(eval(LCS.p.0),eval(LC.p.0),eval(LCI.p.0)),2),
            c(c("LCS = ","LC = ","LCI = "), c(round(eval(LCS.p.0),3), round(eval(LC.p.0),3),
                                    round(eval(LCI.p.0),3))),
            col="red") }
```

Continuación del apéndice 3.

```
    plot.p()
    # Crea la lista de los resultados
structure(list("in.control" = p.pos,
                "out.control"= c(p.fi.0, p.fs.0),
"Iteraciones" = prev.results$Iteraciones + 1,
                "data.n"= prev.results$data.n,
"data.0"= prev.results$data.0,
                "data.1"= p.1,
"bin" = bin.p,
              "Limites de Control Grafica p" = c("LCI.p"=eval(LCI.p.0),"LCS.p"=eval(LCS.p.0),
                                                 "LC.p"=eval(LC.p.0)),
              "Conclusion del proceso"= c(if(length(p.pos)< m){
                print("Proceso fuera de Control en Grafica p")
                } else {
                  print("El proceso esta bajo control en Grafica p")
                  }))
  }
 }
}
```

Documentación función P_it( )

P_it {XRSCC} R Documentation

Iteration of p control chart for attributes

De*scription*

Calculates the iteration i'th for the control limits of p chart using the results obtained in p_gror further P_it iterations.

Usage

P_it(prev.results)

Continuación del apéndice 3.

Arguments

prev.results	*Is a list of previous results obtained by the p_gr function. In other cases, needs more than one iteration, to obtain the true control limits for p chart before take conclusions about the process.*
in.control	*The under control row list for the p chart in this iteration*
out.control	*The out of control row list for the p chart*
Iteraciones	*The number of iterations, It is assumed to be the second or later*
data.n	*The fixed sample size*
data.0	*The original data frame*
data.1	*The under control subset after iteration*
bin	*The binary values for out of control equal to one and under control equal to zero*
Limites de Control Grafica p	*The p chart control limits vector*
Conclusion del proceso	*The same results in a phrase as the bin values*

Author(s)

Erick Marroquin

References

Montgomery, D.C. (2005) *Introduction to statistical quality control*, 5th ed. New York: John Wiley & Sons, ISBN 0-471-65631-3

Continuación del apéndice 3.

See Also

p_gr, c_gr, C_it, np_gr, NP_it, u_gr, U_it

Examples

```
data(bottles)
r1<-p_gr(bottles, n=100)
r2<-P_it(r1)
r3<-P_it(r2)
```

[Package XRSCC version 0.1]

Script función np_gr( )

```
np_gr<-function(D,n){
  if (missing(D)){
    stop("No hay muestras para leer, No sample to read")
  } else {
if (missing(n)){
      stop("Debe fijar un ancho de la muestra")
} else {
      p.0<-D[,1]/n
      np.0<-D[,1]
if(max(p.0) >= 1){
        stop("No puede existir una proporcion mayor a la unidad")
      } else {
        m <-nrow(D)
        # Calculo de limites de control para la grafica P
LCS.np.0<-expression(n*mean(p.0)+3*sqrt(n*mean(p.0)*(1-mean(p.0))))
        LCI.np.0<-expression(n*mean(p.0)-3*sqrt(n*mean(p.0)*(1-mean(p.0))))
        LC.np.0<-expression(n*mean(p.0))
        if (eval(LCI.np.0)>0){
          LCI.p.0<-eval(LCI.np.0)
        } else {
          LCI.np.0 <- 0
        }
```

Continuación del apéndice 3.

```
      np.pos<-which(np.0 >= eval(LCI.np.0)  & np.0 < eval(LCS.np.0))
      np.1<-np.0[np.pos]
      np.fi.0<-which(np.0 < eval(LCI.np.0))
np.fs.0<-which(np.0 >= eval(LCS.np.0)) # Solo valores positivos
bin.np<-if(length(np.pos)< m){
        bin.np<-1
      } else {
        bin.np<-0
      }
      #
      # Script  para Grafica p inicial
      plot.np<-function(NP=np.0,type="b",col="blue",pch =19){
plot(NP, xlab= "Numero de muestra", ylab="Numero de No conformes",
           main="Grafica NP, Control Estadistico de la Calidad",type=type, col=col,
ylim=c(eval(LCI.np.0)-mean(np.0)*0.05, max(eval(LCS.np.0)*1.1, max(np.0)*1.1)),
           xlim=c(-0.05*m, 1.05*m), pch = pch)
        abline(h= c(eval(LCS.np.0), eval(LCI.np.0), eval(LC.np.0)),col="lightgray")
        text(c(rep(1,3),rep(7,3)), rep(c(eval(LCS.np.0),eval(LC.np.0),eval(LCI.np.0)),2),
           c(c("LCS = ","LC = ","LCI = "), c(round(eval(LCS.np.0),3), round(eval(LC.np.0),3),
                                    round(eval(LCI.np.0),3))),
           col="red") }
plot.np()
      # Crea la lista de los resultados
structure(list("in.control" = np.pos,
                  "out.control"= c(np.fi.0,np.fs.0),
"Iteraciones" = 1,
                  "data.n"=n,
                  "data.0"= D,
"data.1"= np.1,
                  "bin" = bin.np,
"Limites de Control Grafica np" = c("LCI.np"=eval(LCI.np.0),"LCS.np"=eval(LCS.np.0),
                                        "LC.np"=eval(LC.np.0)),
                  "Conclusion del proceso"= c(if(length(np.pos)< m){
                    print("Proceso fuera de Control en Grafica np")
                  } else {
                    print("El proceso esta bajo control en Grafica np")
                  }))
    }
  }
 }
}
```

Continuación del apéndice 3.

Documentación función np_gr( )

np_gr {XRSCC} R Documentation

The np chart control for attributes

Description

Calculates the np control chart for attributes, using a sample D of number of defectives or nonconforming items and a constant sample size n. The values plotted in graph are the defectives number.

Usage

np_gr(D, n)

Arguments

D	*A data frame containing the non conforming items, and must be integer and non negative.*
n	*A vector of length one, integer and nonnegative, to fix the sample size.*

Value

in.control	*The under control row list for the np chart*
out.control	*The out of control row list for the np chart*
Iteraciones	*The number of iterations, in this function always will be the first and the last one*
data.n	*The fixed sample size*
data.0	*The original data frame*

Continuación del apéndice 3.

data.1	*The filtered data frame*
bin	*The binary values for out of control equal to one and under control equal to zero*
Limites de Control Grafica np	*The np chart control limits vector*
Conclusion del proceso	*The same results in a phrase as the bin values*

Author(s)
Erick Marroquin

References
Montgomery, D.C. (2005*) Introduction to statistical quality control*, 5th ed. New York: John Wiley & Sons, ISBN 0-471-65631-3

See Also
p_gr, u_gr, c_gr, P_it, NP_it, C_it, U_it

Examples
data(bottles)
np_gr(bottles, n=100)

[Package XRSCC version 0.1]

Continuación del apéndice 3.

- **Script función NP_it( )**

```
NP_it<-function(prev.results){
  if (missing(prev.results)){
    stop("No elementos para iteracion, No elements for iteration")
  } else {
    if(prev.results$bin[1]==0){
      stop("El proceso ya esta bajo control, The process is already under control")
    } else {
      np.0<-prev.results$data.1
      p.0<-prev.results$data.1/prev.results$data.n
      m <-length(np.0)
      n <-prev.results$data.n
# Calculo de limites de control para la grafica P
LCS.np.0<-expression(n*mean(p.0)+3*sqrt(n*mean(p.0)*(1-mean(p.0))))
      LCI.np.0<-expression(n*mean(p.0)-3*sqrt(n*mean(p.0)*(1-mean(p.0))))
      LC.np.0<-expression(n*mean(p.0))
      if (eval(LCI.np.0)>0){
        LCI.p.0<-eval(LCI.np.0)
      } else {
        LCI.np.0 <- 0
      }
      np.pos<-which(np.0 >= eval(LCI.np.0)  & np.0 < eval(LCS.np.0))
      np.1<-np.0[np.pos]
      np.fi.0<-which(np.0 < eval(LCI.np.0))
      np.fs.0<-which(np.0 >= eval(LCS.np.0))
      bin.np<-if(length(np.pos)< m){
        bin.np<-1
      } else {
        bin.np<-0
}
      #
      # Script  para Grafica NP iestima iteracion
plot.np<-function(NP=np.0,type="b",col="blue",pch =19){
plot(NP, xlab= "Numero de muestra", ylab="Numero de No conformes",
            main="Grafica NP, Control Estadistico de la Calidad",type=type, col=col,
ylim=c(eval(LCI.np.0)-mean(np.0)*0.05, max(eval(LCS.np.0)*1.1, max(np.0)*1.1)),
            xlim=c(-0.05*m, 1.05*m), pch = pch)
        abline(h= c(eval(LCS.np.0), eval(LCI.np.0), eval(LC.np.0)),col="lightgray")
        text(c(rep(1,3),rep(7,3)), rep(c(eval(LCS.np.0),eval(LC.np.0),eval(LCI.np.0)),2),
            c(c("LCS = ","LC = ","LCI = "), c(round(eval(LCS.np.0),3), round(eval(LC.np.0),3),
                                    round(eval(LCI.np.0),3))),
            col="red") }
```

Continuación del apéndice 3.

```
      plot.np()
      # Crea la lista de los resultados
structure(list("in.control" = np.pos,
                    "out.control"= c(np.fi.0,np.fs.0),
"Iteraciones" = prev.results$Iteraciones + 1,
                    "data.n"= prev.results$data.n,
"data.0"= prev.results$data.0,
                    "data.1"= np.1,
"bin" = bin.np,
                    "Limites de Control Grafica np" =
c("LCI.np"=eval(LCI.np.0),"LCS.np"=eval(LCS.np.0),
                                                "LC.np"=eval(LC.np.0)),
                    "Conclusion del proceso"= c(if(length(np.pos)< m){
                      print("Proceso fuera de Control en Grafica np")
                    } else {
                      print("El proceso esta bajo control en Grafica np")
                    }))
    }
  }
}
```

Documentación función NP_it( )

NP_it {XRSCC} R Documentation

Iteration of np control chart for attributes

Description

Calculates the iteration i'th for the control limits of p chart using the results obtained innp_gr or further NP_it iterations.

Usage

NP_it(prev.results)

Continuación del apéndice 3.

Arguments

prev.results *Is a list of previous results obtained by the np_gr function. In other cases, needs more than one iteration, to obtain the true control limits fornp chart before take conclusions about the process.*

Value

in.control	*The under control row list for the np chart in this iteration*
out.control	*The out of control row list for the np chart*
Iteraciones	*The number of iterations, It is assumed to be the second or later*
data.n	*The fixed sample size*
data.0	*The original data frame*
data.1	*The under control subset after iteration*
bin	*The binary values for out of control equal to one and under control equal to zero*
Limites de Control Grafica np	*The np chart control limits vector*
Conclusion del proceso	*The same results in a phrase as the bin values*
Author(s)	Erick Marroquin

References

Montgomery, D.C. (2005) *Introduction to statistical quality control*, 5th ed. New York: John Wiley & Sons, ISBN 0-471-65631-3

Continuación del apéndice 3.

See Also

p_gr, np_gr, c_gr, u_gr, P_it, C_it, U_it

Examples

```
data(bottles)
r1<-np_gr(bottles, n=100)
r2<-NP_it(r1)
r3<-NP_it(r2)
```

[Package XRSCC version 0.1]

Script función u_gr( )

```
u_gr<-function(U){
  if (missing(U)){
    stop("No hay muestras para leer, No sample to read")
  } else {
    if (dim(U)[2]!=2){
      stop("La cantidad de variables no es la correcta, The number of variables is not correct")
} else {
     if(((any(names(U)=="n")) & (any(names(U)=="d")))==FALSE){
stop("Las variables no estan especificadas con el nombre correcto")
} else {
         if(any(U$n==0)){
stop("No puede existir alguna muestra igual a cero")
} else {
            u.0 <- U
            m <- nrow(u.0)
ui.0 <- u.0$d/u.0$n
            # Calculo de limites de control para la grafica u
LCS.u.0<-expression(mean(ui.0)+3*sqrt(mean(ui.0)/mean(u.0$n)))
            LCI.u.0<-expression(mean(ui.0)-3*sqrt(mean(ui.0)/mean(u.0$n)))
            LC.u.0<-expression(mean(ui.0))
            if (eval(LCI.u.0)>0){
              LCI.u.0<-eval(LCI.u.0)
              } else {
```

Continuación del apéndice 3.

```
                LCI.u.0 <- 0
                }
            u.pos<-which(ui.0 >= eval(LCI.u.0)  & ui.0 < eval(LCS.u.0))
            ui.1<-ui.0[u.pos]
            u.fi.0<-which(ui.0 < eval(LCI.u.0))
u.fs.0<-which(ui.0 >= eval(LCS.u.0))
            bin.u<-if(length(u.pos)< m){
              bin.u<-1
              } else {
                bin.u<-0
}
            # Script  para Grafica U inicial
            plot.u<-function(U=ui.0,type="b",col="blue",pch =19){
              plot(U, xlab= "Numero de muestra", ylab="Numero de inconformidades por unidad",
                   main="Grafica U, Control Estadistico de la Calidad",type=type, col=col,
                   ylim=c(eval(LCI.u.0)-mean(ui.0)*0.05, max(eval(LCS.u.0)*1.1, max(ui.0)*1.1)),
                   xlim=c(-0.05*m, 1.05*m), pch = pch)
abline(h= c(eval(LCS.u.0), eval(LCI.u.0), eval(LC.u.0)),col="lightgray")
              text(c(rep(1,3),rep(7,3)), rep(c(eval(LCS.u.0),eval(LC.u.0),eval(LCI.u.0)),2),
                   c(c("LCS = ","LC = ","LCI = "), c(round(eval(LCS.u.0),3), round(eval(LC.u.0),3),
round(eval(LCI.u.0),3))),
                   col="red") }
            plot.u()
            # Crea la lista de los resultados
structure(list("in.control" = u.pos,
                        "out.control"= c(u.fi.0,u.fs.0),
"Iteraciones" = 1,
                        "data.0"= U,
                        "data.1"= ui.1,
                        "bin" = bin.u,
                        "Limites de Control Grafica U" =
c("LCI.u"=eval(LCI.u.0),"LCS.u"=eval(LCS.u.0),
                                                "LC.u"=eval(LC.u.0)),
                        "Conclusion del proceso"= c(if(length(u.pos)< m){
                          print("Proceso fuera de Control en Grafica U")
                          } else {
                            print("El proceso esta bajo control en Grafica U")
                            }))
          }
        }
    }
  }
}
```

Continuación del apéndice 3.

Documentación función u_gr( )

u_gr {XRSCC} R Documentation

The u chart control for attributes

Description

Calculates the u control chart for attributes, given a variable sample n and a number of nonconformities u per sample. The plotted values in graph are the average number of nonconformities per unit.

Usage

u_gr(U)

Arguments

U	A data frame containing the number d of nonconformities per sample, the sample n can be variable. Note that the variable names must be lowercase letter, say **d** and **n**.

Value

in.control	The under control row list for the u chart
out.control	The out of control row list for the u chart
Iteraciones	The number of iterations, in this function always will be the first and the last one
data.0	The original data frame
data.1	Subsetting the data frame with under control rows
bin	The binary values for out of control equal to one and under control equal to zero

Continuación del apéndice 3.

Limites de Control Grafica u The u chart control limits vector
Conclusion del proceso The same results in a phrase as the bin values
Author(s) Erick Marroquin
References
Montgomery, D.C. (2005) *Introduction to statistical quality control*, 5th ed. New York: John Wiley & Sons, ISBN 0-471-65631-3
See Also
p_gr, np_gr, c_gr, P_it, NP_it, C_it, U_it
Examples
data(udata2)
u_gr(udata2)
[Package XRSCC version 0.1]

- **Script función u_it( )**

```
U_it<-function(prev.results){
  if (missing(prev.results)){
    stop("No elementos para iteracion, No elements for iteration")
  } else {
    if(prev.results$bin[1]==0){
      stop("El proceso ya esta bajo control, The process is already under control")
    } else {
          u.0 <- prev.results$data.0
          u.pos<-prev.results$in.control
u.1<-u.0[u.pos,]
          m <- nrow(u.1)
          ui.1 <- u.1$d/u.1$n
```

Continuación del apéndice 3.

```
        # Calculo de limites de control para la grafica u
LCS.u.1<-expression(mean(ui.1)+3*sqrt(mean(ui.1)/mean(u.1$n)))
        LCI.u.1<-expression(mean(ui.1)-3*sqrt(mean(ui.1)/mean(u.1$n)))
        LC.u.1<-expression(mean(ui.1))
        if (eval(LCI.u.1)>0){
          LCI.u.1<-eval(LCI.u.1)
        } else {
          LCI.u.1 <- 0
        }
        u.pos<-which(ui.1 >= eval(LCI.u.1)  & ui.1 < eval(LCS.u.1))
        ui.2<-ui.1[u.pos]
        u.fi.1<-which(ui.1 < eval(LCI.u.1))
        u.fs.1<-which(ui.1 >= eval(LCS.u.1))
        bin.u<-if(length(u.pos)< m){
          bin.u<-1
        } else {
          bin.u<-0
}
        #
        # Script  para Grafica U inicial
plot.u<-function(U=ui.1,type="b",col="blue",pch =19){
plot(U, xlab= "Numero de muestra", ylab="Numero de inconformidades por unidad",
              main="Grafica U, Control Estadistico de la Calidad",type=type, col=col,
              ylim=c(eval(LCI.u.1)-mean(ui.1)*0.05, max(eval(LCS.u.1)*1.1, max(ui.1)*1.1)),
              xlim=c(-0.05*m, 1.05*m), pch = pch)
abline(h= c(eval(LCS.u.1), eval(LCI.u.1), eval(LC.u.1)),col="lightgray")
          text(c(rep(1,3),rep(7,3)), rep(c(eval(LCS.u.1),eval(LC.u.1),eval(LCI.u.1)),2),
              c(c("LCS = ","LC = ","LCI = "), c(round(eval(LCS.u.1),3), round(eval(LC.u.1),3),
round(eval(LCI.u.1),3))),
              col="red") }
        plot.u()
        # Crea la lista de los resultados
structure(list("in.control" = u.pos,
                    "out.control"= c(u.fi.1,u.fs.1),
"Iteraciones" = prev.results$Iteraciones + 1,
                    "data.0"= prev.results$data.0,
"data.1"= ui.2,
                    "bin" = bin.u,
"Limites de Control Grafica U" = c("LCI.u"=eval(LCI.u.1),"LCS.u"=eval(LCS.u.1),
                                                "LC.p"=eval(LC.u.1)),
                    "Conclusion del proceso"= c(if(length(u.pos)< m){
                      print("Proceso fuera de Control en Grafica U")
                    } else {
                      print("El proceso esta bajo control en Grafica U")
                    }))
   }
  }
}
```

Continuación del apéndice 3.

Documentación función U_it( )

U_it {XRSCC} R Documentation

Iteration of u control chart for attributes

Description

Calculates the iteration i'th for the control limits of c chart using the results obtained in c_grand previous U_it iteration.

Usage

U_it(prev.results)

Arguments

prev.results	*Is a list of previous results obtained by the u_gr function. In other cases, needs more than one iteration, to obtain the true control limits for u chart before take conclusions about the process.*

Value

in.control	*The under control row list for the u chart*
out.control	*The out of control row list for the u chart*
Iteraciones	*The number of iterations, in this function always will be the first and the last one*
data.0	*The original data frame*
data.1	*Subsetting the data frame with under control rows*
bin	*The binary values for out of control equal to one and under control equal to zero*
Limites de Control Grafica u	*The u chart control limits vector*

Continuación del apéndice 3.

Conclusion del proceso *The same results in a phrase as the bin values*

Author(s) Erick Marroquin

References

Montgomery, D.C. (2005) *Introduction to statistical quality control*, 5th ed. New York: John Wiley & Sons, ISBN 0-471-65631-3

See Also

p_gr, np_gr, c_gr, u_gr, P_it, NP_it, C_it

Examples

```
data(udata2)
r1<-u_gr(udata2)
r2<-U_it(r1)
```

[Package XRSCC version 0.1]

Script función c_gr( )

```
c_gr<-function(C){
  if (missing(C)){
    stop("No hay muestras para leer, No sample to read")
} else {
    c <- C
    c.0<-C[,1]
    m <- length(c.0)
# Calculo de limites de control para la grafica c
LCS.c.0<-expression(mean(c.0)+3*sqrt(mean(c.0)))
        LCI.c.0<-expression(mean(c.0)-3*sqrt(mean(c.0)))
        LC.c.0<-expression(mean(c.0))
        if (eval(LCI.c.0)>0){
          LCI.c.0<-eval(LCI.c.0)
```

Continuación del apéndice 3.

```
        } else {
          LCI.c.0 <- 0
        }
        c.pos<-which(c.0 >= eval(LCI.c.0)  & c.0 < eval(LCS.c.0))
c.1<-c.0[c.pos]
        c.fi.0<-which(c.0 < eval(LCI.c.0))
        c.fs.0<-which(c.0 >= eval(LCS.c.0))
        bin.c<-if(length(c.pos)< m){
          bin.c<-1
        } else {
          bin.c<-0
}
        #
        # Script  para Grafica c inicial
        plot.c<-function(C=c.0,type="b",col="blue",pch =19){
          plot(C, xlab= "Numero de muestra", ylab="Numero de inconformidades",
              main="Grafica c, Control Estadistico de la Calidad",type=type, col=col,
ylim=c(eval(LCI.c.0)-mean(c.0)*0.05, max(eval(LCS.c.0)*1.1, max(c.0)*1.1)),
              xlim=c(-0.05*m, 1.05*m), pch = pch)
          abline(h= c(eval(LCS.c.0), eval(LCI.c.0), eval(LC.c.0)),col="lightgray")
          text(c(rep(1,3),rep(7,3)), rep(c(eval(LCS.c.0),eval(LC.c.0),eval(LCI.c.0)),2),
              c(c("LCS = ","LC = ","LCI = "), c(round(eval(LCS.c.0),3), round(eval(LC.c.0),3),
                                    round(eval(LCI.c.0),3))),
              col="red") }
plot.c()
        # Crea la lista de los resultados
structure(list("in.control" = c.pos,
                    "out.control"= c(c.fi.0,c.fs.0),
"Iteraciones" = 1,
                    "data.0"= C,
                    "data.1"= c.1,
                    "bin" = bin.c,
                    "Limites de Control Grafica c" = c("LCI.c"=eval(LCI.c.0),"LCS.c"=eval(LCS.c.0),
                                          "LC.p"=eval(LC.c.0)),
                    "Conclusion del proceso"= c(if(length(c.pos)< m){
                      print("Proceso fuera de Control en Grafica c")
                    } else {
                      print("El proceso esta bajo control en Grafica c")
                    }))
  }
}
```

Continuación del apéndice 3.

Documentación función c_gr( )

c_gr {XRSCC} R Documentation

The c chart control for attributes

Description

Calculates the c control chart for attributes, using a sample C of number of nonconformities. The plotted values in graph are the nonconformities number on each sample at a regular time interval when there is not a standard given.

Usage

c_gr(C)

Arguments

C	*A data frame or a vector containing the number of nonconformities per sample. Note that the variable name must be the uppercase letter, like **D**.*

Value

in.control	*The under control row list for the c chart*
out.control	*The out of control row list for the c chart*
Iteraciones	*The number of iterations, in this function always will be the first and the last one*
data.0	*The original data frame*
data.1	*Subsetting the data frame with under control rows*

Continuación del apéndice 3.

bin	*The binary values for out of control equal to one, and resultsunder control equal to zero*
Limites de Control Grafica c	*The c chart control limits vector*
Conclusion del proceso	*The same results in a phrase as the bin values*

Author(s) Erick Marroquin

References

Montgomery, D.C. (2005) *Introduction to statistical quality control*, 5th ed. New York: John Wiley & Sons, ISBN 0-471-65631-3

See Also

p_gr, np_gr, u_gr, P_it, NP_it, C_it, U_it

Examples

```
data(clothes)
c_gr(clothes)
```

[Package XRSCC version 0.1]

Continuación del apéndice 3.

Script función C_it( )

```
C_it<-function(prev.results){
  if (missing(prev.results)){
    stop("No hay muestras para leer, No sample to read")
  } else {
    if(prev.results$bin[1]==0){
      stop("El proceso ya esta bajo control, The process is already under control")
      } else {
        c.1 <- prev.results$data.1
        m <- length(c.1)
# Calculo de limites de control para la grafica c
LCS.c.1<-expression(mean(c.1)+3*sqrt(mean(c.1)))
        LCI.c.1<-expression(mean(c.1)-3*sqrt(mean(c.1)))
        LC.c.1<-expression(mean(c.1))
        if (eval(LCI.c.1)>0){
          LCI.c.1<-eval(LCI.c.1)
          } else {
            LCI.c.1 <- 0
            }
        c.pos<-which(c.1 >= eval(LCI.c.1)  & c.1 < eval(LCS.c.1))
        c.2<-c.1[c.pos]
        c.fi.1<-which(c.1 < eval(LCI.c.1))
        c.fs.1<-which(c.1 >= eval(LCS.c.1))
        bin.c<-if(length(c.pos)< m){
          bin.c<-1
          } else {
            bin.c<-0
}
        #
        # Script  para Grafica c inicial
    plot.c<-function(C=c.1,type="b",col="blue",pch =19){
      plot(C, xlab= "Numero de muestra", ylab="Numero de inconformidades",
           main="Grafica c, Control Estadistico de la Calidad",type=type, col=col,
ylim=c(eval(LCI.c.1)-mean(c.1)*0.05, max(eval(LCS.c.1)*1.1, max(c.1)*1.1)),
           xlim=c(-0.05*m, 1.05*m), pch = pch)
      abline(h= c(eval(LCS.c.1), eval(LCI.c.1), eval(LC.c.1)),col="lightgray")
      text(c(rep(1,3),rep(7,3)), rep(c(eval(LCS.c.1),eval(LC.c.1),eval(LCI.c.1)),2),
           c(c("LCS = ","LC = ","LCI = "), c(round(eval(LCS.c.1),3), round(eval(LC.c.1),3),
col="red") }
    plot.c()
    # Crea la lista de los resultados
structure(list("in.control" = c.pos,
                  "out.control"= c(c.fi.1,c.fs.1),
"Iteraciones" = prev.results$Iteraciones + 1,
                  "data.0"= C,
```

Continuación del apéndice 3.

```
                "data.1"= c.2,
                "bin" = bin.c,
"Limites de Control Grafica c" = c("LCI.c"=eval(LCI.c.1),"LCS.c"=eval(LCS.c.1),
                                   "LC.c"=eval(LC.c.1)),
                "Conclusion del proceso"= c(if(length(c.pos)< m){
                  print("Proceso fuera de Control en Grafica c")
                } else {
                  print("El proceso esta bajo control en Grafica c")
                }))
    }
  }
}
```

- **Documentación función C_it( )**

C_it {XRSCC} R Documentation

Iteration of c control chart for attributes

Description

Calculates the iteration i'th, for the control limits of c chart using the results obtained inc_gr and previous C_it iteration.

Usage

C_it(prev.results)

Arguments

prev.results	*Its a list of previous results obtained by the c_gr function. In other cases, needs more than one iteration, to obtain the true control limits, before take conclusions about the process.*

Continuación del apéndice 3.

Value

in.control	*The under control row list for the c chart*
out.control	*The out of control row list for the c chart*
Iteraciones	*The number of iterations, It is assumed to be the second or later*
data.0	*The original data frame or vector*
data.1	*The under control subset after iteration*
bin	*The binary values for out of control equal to one and under control equal to zero*
Limites de Control Grafica c	*The c chart control limits vector*
Conclusion del proceso	*The same results in a phrase as the bin values*

Author(s) Erick Marroquin

References

Montgomery, D.C. (2005) *Introduction to statistical quality control*, 5th ed. New York: John Wiley & Sons, ISBN 0-471-65631-3

See Also

p_gr, np_gr, u_gr, c_gr, P_it, NP_it, U_it

Examples

```
data(clothes)
```

Continuación del apéndice 3.

```
r1<-c_gr(clothes)
r2<-C_it(r1)
r3<-C_it(r2)
```

[Package XRSCC version 0.1]

Fuente: elaboración propia, con documentación de paquete XRSCC 0.1.

Apéndice 4. **Códigos y documentación del paquete Planesmuestra**

Paquete Planesmuestra

Planesmuestra-package {Planesmuestra} R Documentation

Acceptance sampling plan according the Dodge Romig, MIL STD105E and MIL STD414 plans.

Description

Use a funcion for each plan and a special one for graphic an OC curve. The plan functions are bassed in the Dodge Romig, MIL STD 105E and MIL STD 414. However, the OC curve is calculated using the binomial trials, after calculating acceptance sampling plan.

Details

Package: Planesmuestra

Type: Package

Version: 1.0

Date: 2015-02-17

License: GPL

Author(s) Erick Marroquin

Maintainer: Erick Marroquin <ericksuhel@gmail.com>

[Package Planesmuestra version 0.1]

Continuación del apéndice 4.

Script función f_dodge.romig.simple( )

```
f_dodge.romig.simple<-function(N,plan,p){
# Encontrar el numero de linea de intervalo dellote
  # Atendiendo al tipo de plan AOQL 0.03 O LPTD 0.01
  if (missing(N)){
    # si no encuentra el lote, para la funcion
    stop("El lote debe ser igual o mayor que 2")
  } else {
    # Filtra las series de lotes de acuerdo al plan declarado
data(lot_size_DR)
    plan_lot<-lot_size_DR[lot_size_DR$plan==plan,]
# Si el plan es AOQL
    if (plan == "AOQL"){
      # Compara si el lote existe dentro de los limites superiores de
      # las series filtrada de lotes
      if (any(plan_lot$N == N)){
        # De ser cierto encuentra el intervalo especifico
lot_interval<-findInterval(N,as.integer(levels(as.factor(plan_lot$N))))
} else {
        # De lo contrario compara si es mayor a cualquier valor
        # para asignarle el mayor de los limites superiores
        # de lo contrario lo busca y lo ubica en el lugar i + 1
if (N > max(as.integer(levels(as.factor(plan_lot$N))))){
          lot_interval<-findInterval(N,as.integer(levels(as.factor(plan_lot$N))))
        } else {
          lot_interval<-findInterval(N,as.integer(levels(as.factor(plan_lot$N))))+1
}
      }
      plan2<-"LPTD"
    } else {
      # Repite el proceso anterior con la variante del tipo de plan
      # Para el proceso si no esta definido cualquiera de los dos planes
if (plan == "LPTD"){
        if(any(plan_lot$N == N)){
          lot_interval<-findInterval(N,as.integer(levels(as.factor(plan_lot$N))))
        } else {
          if (N > max(as.integer(levels(as.factor(plan_lot$N))))){
            lot_interval<-findInterval(N,as.integer(levels(as.factor(plan_lot$N))))
          } else {
            lot_interval<-findInterval(N,as.integer(levels(as.factor(plan_lot$N))))+1
}
        }
        plan2<-"AOQL"
      } else {
```

Continuación del apéndice 4.

```
      stop("Debe de definir plan como AOQL o LPTD")
    }
  }
  # Fija el lote exacto segun las series
lot_fix<-plan_lot[lot_interval,1]
#
  # Busca y fija la proporcion de los no conformes de acuerdo a la tabla
  # Al igual que con lote, compara y fija el intervalo correcto
  if (missing(p)){
    stop("Debe definir una proporcion promedio de no conformes")
} else {
    data(ap_DR)
    if(any(ap_DR[plan] == p)){
      p_interval<-findInterval(p,ap_DR[,plan])
    } else {
      if (p > max(ap_DR[,plan])){
        p_interval<-findInterval(p,ap_DR[,plan])
      } else {
        p_interval<-findInterval(p,ap_DR[,plan])+1
}
    }
    # con el intervalo se fija el valor p de la tabla
    # encuentra la serie de valores con p de tabla
    # y el lote de la interpolacion
p_fix<-ap_DR[p_interval,plan]
    plan_lot_p_fix<-plan_lot[plan_lot$p==p_fix,]
    plan_lot_n<-plan_lot_p_fix[plan_lot_p_fix$N==lot_fix,]
  }
  # Presentar resultados
  structure(list("plan" = c("n"=eval(as.vector(plan_lot_n$n)),
                            "c"= eval(as.vector(plan_lot_n$c)),
"p"= eval(p_fix)),
                 "Argumentos" = c("Lote" = as.integer(N),"Tipo de plan" = plan,
                                  "Porcentaje Promedio de no conformes"= p_fix*100),
                 "Resultados" = c("Muestra" =as.vector(plan_lot_n$n),
                                  "Numero de Aceptacion" = as.vector(plan_lot_n$c),
                                  "Numero Rechazo" = as.vector(plan_lot_n$c)+1,
"plan2" = c(plan2,as.vector(plan_lot_n$LPTD._AOQL)))))
}
}
```

Continuación del apéndice 4.

Documentación f_dodge.romig.simple( )

f_dodge.romig.simple {Planesmuestra} R Documentation

Calculate the acceptance sampling for Dodge Romig method

Description

Starting with a known lot N, and a specific AOQL or LPTD plan, and an average of proportion of defectives or nonconforming items, the plan is calculated, giving the sample size, the acceptance number and the rejection number. The function is for simple acceptance sample plans only.

Usage

f_dodge.romig.simple(N,plan,p)

Arguments

N	Is the lot size, an integer number, must be grater than 2
plan	A character string for specify the AOQL or LPTD plan
p	Fraction average of the nonconforming items

Author(s) Erick Marroquin

References

Montgomery, D.C. (2005) *Introduction to statistical quality control*, 5th ed. New York: John Wiley & Sons, ISBN 0-471-65631-3

Continuación del apéndice 4.

See Also

f_DR.CO f_milstd414 f_milstd105e

Examples

f_dodge.romig.simple(N=5000,plan="AOQL",p=0.017)

[Package Planesmuestra version 0.1]

Script función f_milstd105e( )

```
f_milstd105e<-function(N,L,NCA,type){
# Encontrar el numero de linea de intervalo del lote
  if (missing(N)){
    stop("El lote debe ser igual o mayor que 2")
} else {
              data(lot_size)
          lot_interval<-findInterval(N,lot_size[,1])+1
# Asigna  el nivel de inspeccion si no es declarado explicitamente
          if (missing(L)){
            stop("Es necesario el argumento nivel o L")
            } else {
# Encuentra la letra correspondiente al tama?o del lote y nivel de inspeccion
data(code_letter)
                      code_l<-code_letter[lot_interval,L]
}
# Asigna el valor por defecto de NCA si no es declarado
# Intepolando el valor dado de NCA a los valores de tabla
if (missing(NCA)){
            stop("Es necesario el argumento NCA")
            } else {
                  data(NCA_values)
                  if (any(NCA==NCA_values)){
NCA2<-NCA
                    } else {
                          NCA_interval<-findInterval(NCA,NCA_values)
                          NCA2<-NCA_values[NCA_interval]
                         }
                  }
```

Continuación del apéndice 4.

```
# Asigna el tipo de inpeccion si no es declarado
        if (missing(type)){
           stop("Es necesario el argumento type del tipo de inspeccion")
}
        if(type=="n"){
          T_ins<-"Normal"
          } else {
                if(type=="r"){
                  T_ins<-"Reducida"
} else {
                        T_ins<-"Rigurosa"
                        }
              }
# Leer los planes de inspeccion
# La informacion queda
        data(milstd105eplans)
# Busca con los argumetos code_l, T, NCA2 los valores n y c para el
# Plan de inspecci?n
        code1<-milstd105eplans[milstd105eplans$code_letter==as.vector(code_l),]
        T1<-code1[code1$T==type,]
NCA3<-T1[T1$NCA==NCA2,]
        c<-NCA3$c
        muestra<-NCA3$n
# Objetos con los nombres de los argumentos y los argumentos
        argumentos_nombres<-c("Lote","Tipo de Inspeccion",
        "Nivel de Inspeccion", "Nivel de Calidad Aceptable")
        argumentos_plan<-c(N,T_ins,L,NCA)
# Objeto con los parametros del plan
        resultados_nombres<-c("Codigo Letra","Nivel de Calidad Aceptable", "Muestra",
        "Numero de Aceptacion", "Numero de Rechazo")
        resultados_plan<-c(as.vector(code_l),NCA2,muestra,c,c+1)
# Integrar los resultados en un data frame
        names(argumentos_plan)<-argumentos_nombres
        names(resultados_plan)<-resultados_nombres
# Presentar resultados
        print(argumentos_plan)
        print(resultados_plan)
         }
}
```

Continuación del apéndice 4.

Documentación f_milstd105e( )

f_milstd105e {Planesmuestra} R Documentation

Calculate the acceptance sampling for MIL STD 105E / ANSI ASQ C Z 1.4 / ISO 2589 plan

Description

Given lot size, a type of inspection (Normal, Reduced, Tightened), type of sampling (Simple, double or multiple), and the AQL, show the calculated acceptance plan based in the MIL STD 105e tables. The function is for simple acceptance sample plans only.

Usage

f_milstd105e(N,L,NCA,type)

Arguments

N	*Is the lot size, an integer number, must be grater than 2*
L	*A character string for inspection level (S-1,S-2,S-3,S-4,I, II, III)*
NCA	*A numeric value for the AQL*
type	*A character string whith the type of inspection, - n - normal, - r - reduced, in other case is tightened*

Author(s) Erick Marroquin

References

Montgomery, D.C. (2005) *Introduction to statistical quality control*, 5th ed. New York: John Wiley & Sons, ISBN 0-471-65631-3

Continuación del apéndice 4.

See Also

f_DR.CO f_dodge.romig.simple f_milstd414

Examples

For a 1200 lot size, an AQL (NCA) of 1, level III, and tightened inspection, locate the MIL STD 414 acceptance plan

f_milstd105e(N=11000,L="II",type="n",NCA=15)

[Package Planesmuestra version 0.1]

Script función f_milstd414( )

```
f_milstd414<-function(N,L,NCA,type){
# Encontrar el intervalo del lote si este existe
  if (missing(N)){
    # si no encuentra el lote, para la funcion
    stop("Debe definir un Lote para continuar")
  } else {
    if (N<3){
      stop("El lote debe ser igual o mayor que 3")
    } else {
    # De ser cierto encuentra el intervalo especifico
data(lot_size.milstd414)
        lot_interval<-findInterval(N,lot_size.milstd414$N)
        lot_fix<-lot_size.milstd414[lot_interval,1]
if (missing(L)){
          stop("Debe definir un nivel de inspeccion")
} else {
            data(code_letter.milstd414)
# Asigna la letra segun el intervalo del lote
code_letter<-as.vector(code_letter.milstd414[lot_interval,L])
          }
if (missing(type)){
          stop("Debe definir un tipo de inspeccion")
          } else {
            data(code_letter.milstd414)
```

Continuación del apéndice 4.

```
   # Asigna la letra segun el intervalo del lote
code_letter<-as.vector(code_letter.milstd414[lot_interval,L])
        }
if (missing(type)){
        stop("Debe definir un tipo de inspeccion")
        } else {
          data(k_plans.milstd414)
# Indexa todos los planes y filtra de acuerdo al plan
NCA_T<-k_plans.milstd414[k_plans.milstd414$T==type,]
# Indexa el objeto anterior y lo filtra de acuerdo a la letra
NCA_T_c.l<-NCA_T[NCA_T$code_letter==code_letter,]
# Busca el intervalo del NCA
          NCA_interval<-findInterval(NCA,NCA_T_c.l$NCA)
   # Decide si el intervalo es menor al menor del NCA de tabla
   # para aproximarlo al primero, de lo contrario la fijacion del
   # NCA de tabla es normal y los valores mayores se quedan en el
   # valor maximo de tabla
          if (NCA < min(NCA_T_c.l$NCA)){
            NCA_fix<-NCA_T_c.l$NCA[1]
          } else {
            NCA_fix<-NCA_T_c.l$NCA[NCA_interval]
          }
   # Indexa y filtra la unica fila de acuerdo al NCA de tabla
          milstd414_plan<-NCA_T_c.l[NCA_T_c.l$NCA==NCA_fix,]
if(type=="n"){
            T_plan<-"Normal"
} else {
            T_plan<-"Riguroso"
          }
        }
   #
   # Consolida los resultados en vectores con nombres
   #
   argumentos_nombres<-c("Lote","Plan",
                         "Nivel de Inspeccion","NCA Inicial")
   argumentos_plan<-c(as.integer(N),T_plan,L,NCA)
   resultados_nombres<-c("Codigo Letra","NCA Tabla","Muestra","k")
   resultados_plan<-c(code_letter,NCA_fix,milstd414_plan$sample,
                              milstd414_plan$k)
   names(argumentos_plan)<-argumentos_nombres
   names(resultados_plan)<-resultados_nombres
   # Presentar resultados
   print(argumentos_plan)
   print(resultados_plan)
   }
  }
}
```

Continuación del apéndice 4.

Documentación f_milstd414( )

f_milstd414 {Planesmuestra} R Documentation

Calculate the acceptance sampling for MIL STD 414 / ANSI ASQ C Z 1.9 / ISO 3951 plan

Description

Given lot size, an inspection level, a type of inspection and the NCA, show the calculated acceptance plan based in the MIL STD 414 tables.

Usage

f_milstd414(N,L,NCA,type)

Arguments

N	Is the lot size, an integer number, must be grater than 2
L	A character string for inspection level (I,II,III,IV,V)
NCA	A numeric value for the NCA
type	Type of inspection, - n - normal, - t - tightened

Details

The master table of MIL STD 414 for plans based in variables, contains the values for both type of inspection.

Author(s) Erick Marroquín

Continuación del apéndice 4.

References

Montgomery, D.C. (2005) *Introduction to statistical quality control*, 5th ed. New York: John Wiley & Sons, ISBN 0-471-65631-3

See Also

f_DR.CO, f_dodge.romig.simple, f_milstd105e , f_milstd414.test

Examples

For a 1200 lot size, an NCA of 1, level III, and tightened inspection, locate the MIL STD 414 acceptance plan

##

f_milstd414(N=1200,NCA=1,L="III",type="t")

[Package Planesmuestra version 0.1]

Script función f_milstd414.test( )

```
f_milstd414.test<-function(x,k,S,Limite,L){
if(missing(x)){
   stop("Debe de haber una referencia valida de los datos de muestra")
} else {
   .x<-x[,1]
   med_x<-mean(.x)
  }
  if(missing(S)){
   S<-sd(.x)
   } else {
     S<-S
   }
  if(missing(k)){
stop("Falta el argumento de corrimiento k")
} else {
```

Continuación del apéndice 4.

```
    } else {
      if(missing(Limite)){
        stop("Falta especificar si es limite superior o inferior, S o I")
      } else {
        if(missing(L)){
stop("Falta especificar el valor del limite de especificacion")
} else {
           k_critico<-(L - med_x)/S
           if(Limite=="S"){
             if(k_critico >= k){
print("Aceptar el lote")
               } else {
                 print("Rechazar el lote")
}
           } else {
             if(k_critico <= k){
print("Aceptar el lote")
               } else {
                 print("Rechazar el lote")
               }
           }
         }
      }
    }
}
```

Documentación f_milstd414.test( )

f_milstd414.test {Planesmuestra} R Documentation

Accept or reject a variable sample considering a shift factor

Description

Accept or reject a variable sample considering a shift factor, the data comes from an especific variable plan.

Continuación del apéndice 4.

Usage

f_milstd414.test(x,k,S,Limite,L)

Arguments

x	Vector or data frame containing the taken sample values, the function evaluates only the first column or variable
k	A vector of length one, equal shift factor
S	Know standard deviation, if value not exists, function gives the sample standard deviation
Limite	A character vector of length one, "S" for upper control limit and "I" for lower control limit
L	A vector of length one, equal to a specific Control Limit value

Author(s) Erick Marroquín

References

Montgomery, D.C. (2005) *Introduction to statistical quality control*, 5th ed. New York: John Wiley & Sons, ISBN 0-471-65631-3

See Also

f_milstd414

Examples

```
x<-c(4.7,5.1,4.9,4.9,4.8,4.9,4.9,4.8,4.8,4.7,4.7,4.9,4.8,4.9,
4.6,4.8,4.9,5.1,4.8,5,5,4.7,5,5,4.8)
f_milstd414.test(as.data.frame(x),k=1.98,Limite="S", L=5.1)
f_milstd414.test(as.data.frame(x),k=1.98,Limite="I", L=4.7)
```

Continuación del apéndice 4.

[Package Planesmuestra version 0.1]

Script función f_DR.CO( )

```
f_DR.CO<-function(c,n,p){
if (missing(p)){
    stop("El promedio de los aceptables debe de definirse,
        para calcular la probabilidad de aceptacion")
  }
  if (missing(n)){
    stop("La muestra debe de ser un numero entero y positivo")
  }
  if (missing(c)){
    stop("El numero de aceptacion debes ser entero e igual o mayor que cero")
}
  beta.x<-pbinom(c,n,p)
  prob.x<-seq(0,0.1,by=0.001)
  n.1<-pbinom(c,n,prob.x)
  plot(prob.x, n.1, type = "l", lwd = 2,
       col = 2, cex = 2, bg = NA,
       xlab = "p", ylab = expression(1- alpha),
xlim = c(0,0.1), ylim = c(0,1),
       main = "CURVA DE OPERACION
       OC Curve")
  # Agrega opciones de graficas de bajo nivel
segments(x0=0.0,y0=beta.x,x1=p,y1=beta.x,col="blue",lwd=2)
  segments(x0=p,y0=0.0,x1=p,y1=beta.x,col="blue",lwd=2)
  segments(x0=-0.0,y0=0,x1=0.1,y1=0,col="black",lwd=1)
  segments(x0=-0.0,y0=0,x1=0,y1=1,col="black",lwd=1)
  text(0.07,0.9, expression(paste(beta)),cex = 1, col="black")
  text(0.075,0.9, expression(" = "),cex = 1, col="black")
  text(0.09,0.9, round(beta.x,3),cex = 1, col="black")
  grid(10, 10, lwd = 1)
  structure(cbind("c"=c, "n"=n, "p"=p, "beta"=beta.x))
}
```

Continuación del apéndice 4.

Documentación f_milstd414( )

f_DR.CO {Planesmuestra} R Documentation

Plot the OC Curve for a specific acceptance sampling plan

Description

Plot the OC Curve for a specific acceptance plan. Needs the acceptance number c , the sample size n, and the fraction of the non conforming items p. The calculation uses the binomial trials. Applies for attribute plans.

Usage

f_DR.CO(c,n,p)

Arguments

c	*An integer number grater than zero, for the acceptance number.*
n	*An integer number grater than the acceptance number for the sample size.*
p	*Fraction average of the nonconforming items.*

Value

c	*An integer number grater than zero, for the acceptance number.*
n	*An integer number grater than the acceptance number for the sample size.*
p	*Fraction average of the nonconforming items.*
beta	*Acceptance probability.*

Continuación del apéndice 4.

Author(s) Erick Marroquín

References

Montgomery, D.C. (2005) *Introduction to statistical quality control*, 5th ed. New York: John Wiley & Sons, ISBN 0-471-65631-3

See Also

f_dodge.romig.simple, f_milstd414, f_milstd105e, f_CO.plan

Examples

 # A sample of 125 items, with an acceptance number of two and a average fraction of nonconforming items of 0.01

f_DR.CO(2,125,0.1)

[Package Planesmuestra version 0.1]

Script función f_CO.plan( )

```
f_CO.plan<-function(plan){
if (missing(plan)){
    stop("No existe el vector que especifica el plan")
}
  c<-as.numeric(plan["c"])
  n<-as.numeric(plan["n"])
  p<-as.numeric(plan["p"])
  beta.x<-pbinom(c,n,p)
  prob.x<-seq(0,0.1,by=0.001)
  n.1<-pbinom(c,n,prob.x)
  plot(prob.x, n.1, type = "l", lwd = 2,
       col = 2, cex = 2, bg = NA,
       xlab = "p", ylab = expression(1- alpha),
xlim = c(0,0.1), ylim = c(0,1),
```

Continuación del apéndice 4.

```
      main = "CURVA DE OPERACION
      OC Curve")
  # Agrega opciones de graficas de bajo nivel
segments(x0=0.0,y0=beta.x,x1=p,y1=beta.x,col="blue",lwd=2)
  segments(x0=p,y0=0.0,x1=p,y1=beta.x,col="blue",lwd=2)
  segments(x0=-0.0,y0=0,x1=0.1,y1=0,col="black",lwd=1)
  segments(x0=-0.0,y0=0,x1=0,y1=1,col="black",lwd=1)
  text(0.07,0.9, expression(paste(beta)),cex = 1, col="black")
  text(0.075,0.9, expression(" = "),cex = 1, col="black")
  text(0.09,0.9, round(beta.x,3),cex = 1, col="black")
  grid(10, 10, lwd = 1)
  structure(cbind("c"=c, "n"=n, "p"=p, "beta"=beta.x))
}
```

Documentación f_CO.plan( )

f_CO.plan {Planesmuestra} R Documentation

Plot the OC Curve for a specific Dodge Romig acceptance sampling plan results

Description

Plot the OC Curve for a specific acceptance plan. The function need the plan defined in a previous function. The calculation uses the binomial trials. Applies for attribute plans.

Usage

f_CO.plan(plan)

Arguments

plan	A vector with acceptance number c , the sample size n, and the fraction of the non conforming items p.

Continuación del apéndice 4.

Value

c	An integer number grater than zero, for the acceptance number.
n	An integer number grater than the acceptance number for the sample size.
p	Fraction average of the nonconforming items.
beta	Acceptance probability.

Author(s) Erick Marroquin

References

Montgomery, D.C. (2005) *Introduction to statistical quality control*, 5th ed. New York: John Wiley & Sons, ISBN 0-471-65631-3

See Also

f_dodge.romig.simple, f_milstd414, f_milstd105e, f_DR.CO

Examples

```
r1<-f_dodge.romig.simple(N=2500,"AOQL", p=0.01)
f_CO.plan(r1$plan)
```

[Package Planesmuestra version 0.1]

Script función f_CO.NCA.NCL( )

```
f_CO.NCA.NCL<-function(NCA,NCL,n,c){
if (missing(NCL)){
   stop("No existe el nivel de Calidad Limite")
 } else {
```

Continuación del apéndice 4.

```
if (missing(NCA)){
     stop("No existe el nivel de Calidad Aceptable")
} else {
     if (missing(n)){
       stop("Tiene que definir la muestra")
     } else {
       if (missing(c)){
stop("Tiene que defir el numero de aceptacion")
} else {
         beta.NCL<-pbinom(c,n,NCL)
alpha.NCA<-pbinom(c,n,NCA)
prob.x<-seq(0,0.125,by=0.001)
         n.1<-pbinom(c,n,prob.x)
         plot(prob.x, n.1, type = "l", lwd = 2,
              col = 2, cex = 2, bg = "grey",
              xlab = "NCA - NCL", ylab = "Pa",
xlim = c(0,0.13), ylim = c(0,1),
              main = "CURVA DE OPERACION, Relacion NCA-NCL
                    OC Curve AOQL - LPTD")
         # Agrega opciones de graficas de bajo nivel
segments(x0=0.0,y0=beta.NCL,x1=NCL,y1=beta.NCL,col="blue",lwd=2)
         segments(x0=NCL,y0=0,x1=NCL,y1=beta.NCL,col="blue",lwd=2)
segments(x0=0.0,y0=alpha.NCA,x1=NCA,y1=alpha.NCA,col="blue",lwd=2)
         segments(x0=NCA,y0=0,x1=NCA,y1=alpha.NCA,col="blue",lwd=2)
         segments(x0=-0.0,y0=0,x1=0.125,y1=0,col="black",lwd=1)
         segments(x0=-0.0,y0=0,x1=0,y1=1,col="black",lwd=1)
         #Texto beta
         text(NCL*0.95,beta.NCL*1.6, expression(paste(beta)),cex = 1, col="black")
text(NCL,beta.NCL*1.6, expression(" = "),cex = 1, col="black")
         text(NCL*1.05,beta.NCL*1.6, round(beta.NCL,3),cex = 1, col="black")
         text(NCA*1.3,alpha.NCA, expression(paste(alpha)),cex = 1, col="black")
         text(NCA*1.4,alpha.NCA, expression(" = "),cex = 1, col="black")
         text(NCA*1.55,alpha.NCA, round(1-alpha.NCA,3),cex = 1, col="black")
         grid(10, 10, lwd = 1)
         structure(cbind("c"=c, "n"=n, "NCA"=NCA, "NCL"=NCL,
                    "beta"=beta.NCL, "alpha"= 1-alpha.NCA))
       }
}
   }
  }
}
```

Continuación del apéndice 4.

Documentación f_CO.NCA.NCL( )

f_CO.NCA.NCL {Planesmuestra} R Documentation

OC Curve for AOQL and LPTD relation

Description

Given an AOQL, LPTD, sample size and acceptance number, return the plot the OC curve and producer and consumer risk.The calculation uses the binomial trials. Applies for attribute plans.

Usage

f_CO.NCA.NCL(NCA,NCL,n,c)

Arguments

NCA	Specific AOQL value
NCL	Specific LPTD value
n	sample size
c	acceptance number

Details

Functionn stops if any value is missing

Value

NCA	Specific AOQL value
NCL	Specific LPTD value
n	sample size

Continuación del apéndice 4.

c	acceptance number
beta	consumer risk
alpha	producer risk

Author(s) Erick Marroquin

References

Montgomery, D.C. (2005) *Introduction to statistical quality control*, 5th ed. New York: John Wiley & Sons, ISBN 0-471-65631-3

See Also

f_dodge.romig.simple, f_milstd414, f_milstd105e, f_CO.plan, f_DR.CO

Examples

f_CO.NCA.NCL(NCA=0.02,NCL=0.1,n=69,c=3)

[Package Planesmuestra version 0.1]

Fuente: elaboración propia, con documentación de paquete Planesmuestra 0.1.

Apéndice 5. **Manuales de referencia, actualización de R**

Para el desarrollo de este texto, se recurrió a la continua actualización de la versión de R, ya que los desarrolladores de paquetes, van actualizando los mismos de acuerdo al núcleo de la última versión de R.

Para actualizar R, se necesita la instalación del paquete “installr” desde el *CRAN* de preferencia.

El siguiente paso para actualizar R a la última versión es digitar en la línea de comando o *prompt* para llamar al paquete con la función *require* (paquete).

```
if(!require(installr)) {
install.packages("installr"); require(installr)}
updateR()
```

Seguido, se utiliza la función *updateR*( ), la cual se conectará con el *CRAN* previamente definido, para bajar y comenzar de forma automática la instalación de la última versión, esto implica que el proceso se hace totalmente en línea. Además del proceso normal de instalación, el proceso pregunta sí desea copiar los paquetes de la versión antigua (Old) a la nueva, para lo cual se responde que sí, de lo contrario, será necesario instalar de nuevo todos los paquetes.

El proceso de actualización no borra o desinstala la versión anterior, pero sí es posible eliminar al menos los accesos directos en el escritorio o barra de tareas, no obstante, la versión antigua funciona normalmente.

Fuente: GALILI, Tal. R- Statistics blog. [en línea]. <http://www.r-statistics.com/2013/03/updating-r-from-r-on-windows-using-the-installr-package/>

ANEXOS

Anexo 1. **Tablas para control estadístico de la calidad -CEC-**

Fórmulas para límites de control por variables

Carta de control $\overline{X}$:

Promedio de las observaciones	$\bar{x} = \frac{x_1 + x_2 + \cdots + x}{n}$
Gran promedio de las muestras	$\bar{\bar{x}} = \frac{\bar{x}_1 + \bar{x}_2 + \cdots + \bar{x}_m}{m}$
Número de observaciones por muestra	n
Tamaño de la muestra	m
Rango de la muestra	$R = x_{max} - x_{min}$
Rango promedio	$\bar{R} = \frac{R_1 + R_2 + \cdots + R_m}{m}$
Límite de control superior	$LCS = \bar{\bar{x}} + A_2\bar{R}$
Límite de control inferior	$LCI = \bar{\bar{x}} - A_2\bar{R}$
Límite central	$LC = \bar{\bar{x}}$

Estimador insesgado:

Desviación estándar: σ	$\hat{\sigma} = \frac{\bar{R}}{d_2}$
Media de la población: μ	$\bar{\bar{x}}$

Continuación del anexo 1.

Límite de control superior		$LCS = \bar{\bar{x}} + \frac{3}{d_2\sqrt{n}}\bar{R}$
Sii		$LCS = \bar{\bar{x}} + A_2\bar{R}$
	$\therefore$	$A_2 = \frac{3}{d_2\sqrt{n}}$

Carta de control**R:**

Límite de control superior	$LCS = \bar{R}D_4$
Límite de control inferior	$LCI = \bar{R}D_3$
Límite central	$LC = \bar{R}$

Índice de capacidad del proceso:

Límite de especificación superior	LES
Límite de especificación inferior	LEI
Sigma	$\hat{\sigma} = \frac{\bar{R}}{d_2}$
Capacidad del proceso	$C_p = \frac{LES-LEI}{6\hat{\sigma}}$

Carta de control S:

Está dado σ

Límite de Control Superior	$LCS = B_6\sigma$
Límite de Control Inferior	$LCI = B_5\bar{\sigma}$
Límite central	$LC = c_4\sigma$

No está dado σ

Continuación del anexo 1.

Límite de control superior	$LCS = B_4\bar{\bar{S}}$
Límite de control Inferior	$LCI = B_3\bar{\bar{S}}$
Límite central	$LC = \bar{S}$
Varianza de la muestra	$S^2 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n-1}$
Promedio de la desviación estándar	$\bar{S} = \frac{\sum_{i=1}^{m} S_i}{m}$

Tabla de factores para las gráficas $\overline{X}$ y R:

n	A2	D3	D4	d2
2	1.880	0.000	3.267	1.128
3	1.023	0.000	2.574	1.693
4	0.729	0.000	2.282	2.059
5	0.577	0.000	2.114	2.326
6	0.483	0.000	2.004	2.534
7	0.419	0.076	1.924	2.704
8	0.373	0.136	1.864	2.847
9	0.337	0.184	1.816	2.970
10	0.308	0.223	1.777	3.780

Continuación del anexo1.

Tabla de fórmulas para control de atributos:

Límites de control	$\boldsymbol{p(fracción)}$	$\boldsymbol{np(número\ de\ no\ conformidades)}$	$\boldsymbol{c(cuenta\ de\ no\ conformidades)}$	$\boldsymbol{u(cuenta\ de\ no\ conformidades\ /\ unidad)}$
Central = *LC*	$\bar{p}$	$n\bar{p}$	$\bar{c}$	$\bar{u}$
Superior = *LCS*	$\bar{p} + 3\sqrt{\frac{\bar{p}(1-\bar{p})}{n}}$	$n\bar{p} + 3\sqrt{n\bar{p}(1-\bar{p})}$	$\bar{c} + 3\sqrt{\bar{c}}$	$\bar{u} + 3\sqrt{\frac{\bar{u}}{n}}$
Inferior = *LCI*	$\bar{p} - 3\sqrt{\frac{\bar{p}(1-\bar{p})}{n}}$	$n\bar{p} - 3\sqrt{n\bar{p}(1-\bar{p})}$	$\bar{c} - 3\sqrt{\bar{c}}$	$\bar{u} - 3\sqrt{\frac{\bar{u}}{n}}$
Nota:	Si n varia, usar $\bar{n}$ o cada n_i	n debe de ser constante	n debe de ser constante	Si n varia, usar $\bar{n}$ o cada n_i

Tabla de factores para las gráficas X, R y S para n > 25:

$$A = \frac{3}{\sqrt{n}} \qquad A_3 = \frac{3}{c_4\sqrt{n}} \qquad c_4 = \frac{4(n-1)}{4n-3}$$

$$B_3 = 1 - \frac{3}{c_4\sqrt{2(n-1)}} \qquad B_4 = 1 + \frac{3}{c_4\sqrt{2(n-1)}}$$

$$B_5 = c_4 - \frac{3}{\sqrt{2(n-1)}} \qquad B_6 = c_4 + \frac{3}{\sqrt{2(n-1)}}$$

Continuación del anexo 1.

Tabla de factores para las gráficas X, R y S tabulados:

Factores para calcular las tablas contol por variables

Observaciones en la muestra,	Tabla para promedios			Tabla para desviaciones estándar						Tabla para Rangos					
	Factores para Límites de Control			Factores para Línea Central		Factores para Límites de Control				Factores para Línea Central			Factores para Límites de Control		
n	A	A_2	A_3	c_4	$1/c_4$	B_3	B_4	B_5	B_6	d_2	$1/d_2$	d_3	D_1	D_2	D_3	D_4
2	2.121	1.880	2.659	0.7979	1.2533	0	3.267	0	2.606	1.128	0.8865	0.853	0	3.866	0	3.267
3	1.732	1.023	1.954	0.8862	1.1284	0	2.568	0	2.276	1.693	0.5907	0.888	0	4.358	0	2.574
4	1.500	0.729	1.628	0.9213	1.0854	0	2.266	0	2.088	2.059	0.4857	0.880	0	4.698	0	2.282
5	1.342	0.577	1.427	0.9400	1.0638	0	2.089	0	1.964	2.326	0.4299	0.864	0	4.918	0	2.114
6	1.225	0.483	1.287	0.9515	1.0510	0.030	1.970	0.029	1.874	2.534	0.3946	0.848	0	5.078	0	2.004
7	1.134	0.419	1.182	0.9594	1.0423	0.118	1.882	0.113	1.806	2.704	0.3698	0.833	0.204	5.204	0.076	1.924
8	1.061	0.373	1.099	0.9650	1.0363	0.185	1.815	0.179	1.751	2.847	0.3512	0.820	0.388	5.306	0.136	1.864
9	1.000	0.337	1.032	0.9693	1.0317	0.239	1.761	0.232	1.707	2.970	0.3367	0.808	0.547	5.393	0.184	1.816
10	0.949	0.308	0.975	0.9727	1.0281	0.284	1.716	0.276	1.669	3.078	0.3249	0.797	0.687	5.469	0.223	1.777
11	0.905	0.285	0.927	0.9754	1.0252	0.321	1.679	0.313	1.637	3.173	0.3152	0.787	0.811	5.535	0.256	1.744
12	0.866	0.266	0.886	0.9776	1.0229	0.354	1.646	0.346	1.610	3.258	0.3069	0.778	0.922	5.594	0.283	1.717
13	0.832	0.249	0.850	0.9794	1.0210	0.382	1.618	0.374	1.585	3.336	0.2998	0.770	1.025	5.647	0.307	1.693
14	0.802	0.235	0.817	0.9810	1.0194	0.406	1.594	0.399	1.563	3.407	0.2935	0.763	1.118	5.696	0.328	1.672
15	0.775	0.223	0.789	0.9823	1.0180	0.428	1.572	0.421	1.544	3.472	0.2880	0.756	1.203	5.741	0.347	1.653
16	0.750	0.212	0.763	0.9835	1.0168	0.448	1.552	0.440	1.526	3.532	0.2831	0.750	1.282	5.782	0.363	1.637
17	0.728	0.203	0.739	0.9845	1.0157	0.466	1.534	0.458	1.511	3.588	0.2787	0.744	1.356	5.820	0.378	1.622
18	0.707	0.194	0.718	0.9854	1.0148	0.482	1.518	0.475	1.496	3.640	0.2747	0.739	1.424	5.856	0.391	1.608
19	0.688	0.187	0.698	0.9862	1.0140	0.497	1.503	0.490	1.483	3.689	0.2711	0.734	1.487	5.891	0.403	1.597
20	0.671	0.180	0.680	0.9869	1.0133	0.510	1.490	0.504	1.470	3.735	0.2677	0.729	1.549	5.921	0.415	1.585
21	0.655	0.173	0.663	0.9876	1.0126	0.523	1.477	0.516	1.459	3.778	0.2647	0.724	1.605	5.951	0.425	1.575
22	0.640	0.167	0.647	0.9882	1.0119	0.534	1.466	0.528	1.448	3.819	0.2618	0.720	1.659	5.979	0.434	1.566
23	0.626	0.162	0.633	0.9887	1.0114	0.545	1.455	0.539	1.438	3.858	0.2592	0.716	1.710	6.006	0.443	1.557
24	0.612	0.157	0.619	0.9892	1.0109	0.555	1.445	0.549	1.429	3.895	0.2567	0.712	1.759	6.031	0.451	1.548
25	0.600	0.153	0.606	0.9896	1.0105	0.565	1.435	0.559	1.420	3.931	0.2544	0.708	1.806	6.056	0.459	1.541

Fuente: MONTOGOMERY, Douglas. *Introduction to statistical quality control*. p. 725.

Anexo 2. **Tablas para muestreos de aceptación**

Dodge Romig, tabla de inspección, para planes de muestreo simple para un AOQL = 3%

	Proceso Promedio																					
	0 - 0.06%				0.07 - 0.60%				0.61 - 1.2%				1.2 - 1.8%				1.81 - 2.4%				2.41 - 3.00%			
				LTPD				LTPD				LTPD				LTPD				LTPD				LTPD
Tamaño del Lote	n	c		%	n	c		%	n	c		%	n	c		%	n	c		%	n	c		%
1 a 10	All	0		-	All	0		-	All	0		-	All	0		-	All	0		-	All	0		-
11 a 50	10	0		19.0	10	0		19.0	10	0		19.0	10	0		19.0	10	0		19.0	10	0		19.0
51 a 100	11	0		18.0	11	0		18.0	11	0		18.0	11	0		18.0	11	0		18.0	22	1		16.4
101 a 200	12	0		17.0	12	0		17.0	26	0		17.0	25	1		15.1	25	1		15.1	25	1		15.1
201 a 300	12	0		17.0	12	0		17.0	26	1		14.6	26	1		14.6	26	1		14.6	40	2		12.8
301 a 400	12	0		17.1	12	0		17.1	27	1		14.7	26	1		14.7	41	2		12.7	41	2		12.7
401 a 500	12	0		17.2	27	1		14.1	27	1		14.1	42	2		12.4	42	2		12.4	42	2		12.4
501 a 600	12	0		17.3	27	1		14.2	27	1		14.2	42	2		12.4	42	2		12.4	60	3		10.8
601 a 800	12	0		17.3	27	1		14.2	44	1		14.2	43	2		12.1	60	3		10.9	60	3		10.9
801 a 1,000	12	0		17.4	27	1		14.2	45	2		11.8	44	2		11.8	60	3		11.0	80	4		9.8
1,001 a 2,000	12	0		17.5	28	1		13.8	45	2		11.7	65	3		10.2	80	4		9.8	100	5		9.1
2,001 a 3,000	12	0		17.5	28	1		13.8	65	2		11.7	65	3		10.2	100	5		9.1	140	7		8.2
3,001 a 4,000	12	0		17.5	28	1		13.8	65	3		10.3	85	4		9.5	125	6		8.4	165	8		7.8
4,001 a 5,000	28	1		13.8	28	1		13.8	65	3		10.3	85	4		9.5	125	6		8.4	210	10		7.4
5,001 a 7,000	28	1		13.8	45	2		11.8	65	3		10.3	105	5		8.8	145	7		8.1	235	11		7.1
7,001 a 10,000	28	1		13.9	46	2		11.6	65	3		10.3	105	5		8.8	170	8		7.6	280	13		6.8
10,001 a 20,000	28	1		13.9	46	2		11.7	85	4		9.5	125	6		8.4	215	10		7.2	380	17		6.2
20,001 a 50,000	28	1		13.9	65	3		10.3	105	5		8.8	170	8		7.6	310	14		6.5	560	24		5.7
50,001 a 100,000	28	1		13.9	65	3		10.3	125	6		8.4	215	10		7.2	385	17		6.2	690	29		5.4

Fuente: MONTOGOMERY, Douglas.*Introduction to statistical quality control*. p. 683.

Continuación del anexo 2.

Dodge Romig, tabla de inspección, para planes de muestreo simple para un LPTD = 1%

Proceso Promedio																							
	0 - 0.01%				0.011- 0.10%				0.11 - 0.2%				0.21 - 0.3%				0.31 - 0.40%				0.41 - 0.50%			
				AOQL				AOQL				AOQL				AOQL				AOQL				AOQL
Tamaño del Lote	n	c		%	n	c		%	n	c		%	n	c		%	n	c		%	n	c		%
1 a 120	All	0	-	-	All	0	-	-	All	0	-	-	All	0	-	-	All	0	-	-	All	0	-	-
121 a 150	120	0	0.06	0.06	120	0	0.06	0.06	120	0	0.06	0.06	120	0	0.06	0.06	120	0	0.06	0.06	120	0	0.06	0.06
151 a 200	140	0	0.08	0.08	140	0	0.08	0.08	140	0	0.08	0.08	140	0	0.08	0.08	140	0	0.08	0.08	140	0	0.08	0.08
201 a 300	165	0	0.10	0.10	165	0	0.10	0.10	165	0	0.10	0.10	165	0	0.10	0.10	165	0	0.10	0.10	165	0	0.10	0.10
301 a 400	175	0	0.12	0.12	175	0	0.12	0.12	175	0	0.12	0.12	175	0	0.12	0.12	175	0	0.12	0.12	175	0	0.12	0.12
401 a 500	180	0	0.13	0.13	180	0	0.13	0.13	180	0	0.13	0.13	180	0	0.13	0.13	180	0	0.13	0.13	180	0	0.13	0.13
501 a 600	190	0	0.13	0.13	190	0	0.13	0.13	190	0	0.13	0.13	190	0	0.13	0.13	190	0	0.13	0.13	305	1	0.14	0.14
601 a 800	200	0	0.14	0.14	200	0	0.14	0.14	200	0	0.14	0.14	330	1	0.15	0.15	330	1	0.15	0.15	330	1	0.15	0.15
801 a 1,000	205	0	0.14	0.14	205	0	0.14	0.14	205	0	0.14	0.14	335	1	0.17	0.17	335	1	0.17	0.17	335	1	0.17	0.17
1,001 a 2,000	220	0	0.15	0.15	220	0	0.15	0.15	360	1	0.19	0.19	490	2	0.21	0.21	490	2	0.21	0.21	610	3	0.22	0.22
2,001 a 3,000	220	0	0.15	0.15	375	1	0.20	0.20	505	2	0.23	0.23	630	3	0.24	0.24	745	4	0.26	0.26	870	5	0.26	0.26
3,001 a 4,000	225	0	0.15	0.15	380	1	0.20	0.20	510	2	0.23	0.23	645	3	0.25	0.25	880	5	0.28	0.28	1,000	6	0.29	0.29
4,001 a 5,000	225	0	0.16	0.16	380	1	0.20	0.20	520	2	0.24	0.24	770	4	0.28	0.28	895	5	0.29	0.29	1,120	7	0.31	0.31
5,001 a 7,000	230	0	0.16	0.16	385	1	0.21	0.21	655	3	0.27	0.27	780	4	0.29	0.29	1,020	6	0.32	0.32	1,260	8	0.34	0.34
7,001 a 10,000	230	0	0.16	0.16	520	2	0.25	0.25	660	3	0.28	0.28	910	5	0.32	0.32	1,150	7	0.34	0.34	1,500	10	0.37	0.37
10,001 a 20,000	390	1	0.21	0.21	525	2	0.26	0.26	785	4	0.31	0.31	1,040	6	0.35	0.35	1,400	9	0.39	0.39	1,980	14	0.43	0.43
20,001 a 50,000	390	1	0.21	0.21	530	2	0.26	0.26	920	5	0.34	0.34	1,300	8	0.39	0.39	1,890	13	0.44	0.44	2,570	19	0.48	0.48
50,001 a 100,000	390	1	0.21	0.21	670	3	0.29	0.29	1,040	6	0.36	0.36	1,420	9	0.41	0.41	2,120	15	0.47	0.47	3,150	23	0.50	0.50

Fuente: MONTOGOMERY, Douglas.*Introduction to statistical quality control*. p. 684.

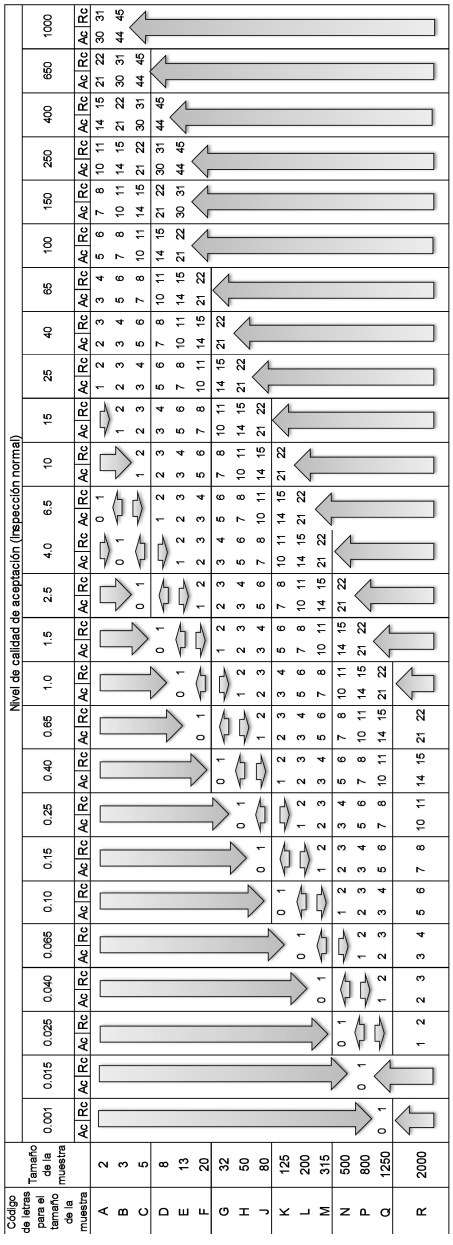
Continuación del anexo 2.

MIL STD 105E, tabla 1, códigos de letras para el tamaño de la muestra:

	Niveles especiales de inspección				Niveles generales de inspección		
Tamaño del Lote	S-1	S-2	S-3	S-4	I	II	III
2 a 8	A	A	A	A	A	A	B
9 a 15	A	A	A	A	A	B	C
16 a 25	A	A	B	B	B	C	D
26 a 50	A	B	B	C	C	D	E
51 a 90	B	B	C	C	C	E	F
91 a 150	B	B	C	D	D	F	G
151 a 280	B	C	D	E	E	G	H
281 a 500	B	C	D	E	F	H	J
501 a 1200	C	C	E	F	G	J	K
1201 a 3200	C	D	E	G	H	K	L
3201 a 10000	C	D	F	G	J	L	M
10001 a 35000	C	D	F	H	K	M	N
35001 a 150000	D	E	G	J	L	N	P
150001 a 500000	D	E	G	J	M	P	Q
500001 y más	D	E	H	K	N	Q	R

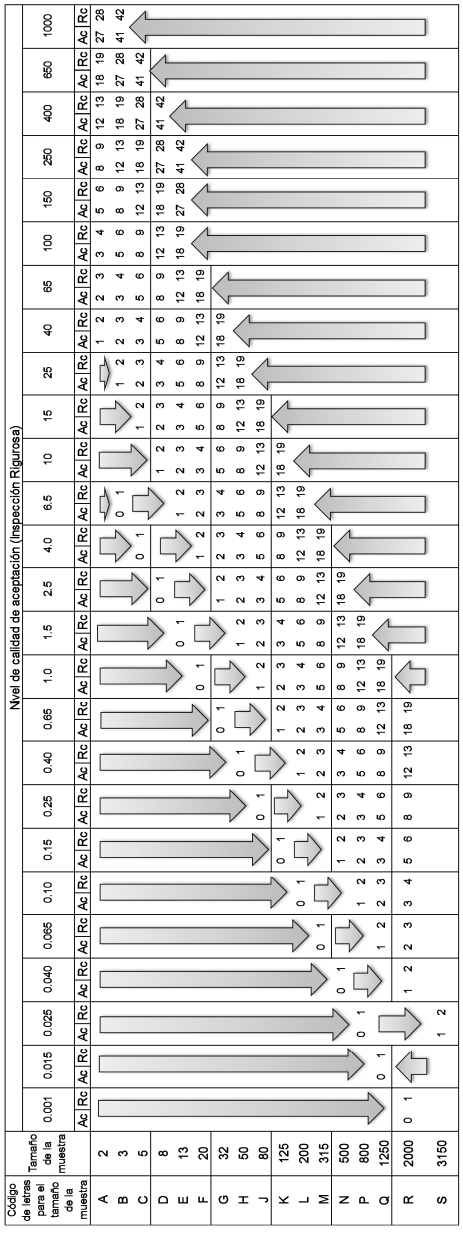
Continuación del anexo 2.

MIL STD 105E, tabla 2-A, tabla maestra para la inspección normal de un muestreo simple:



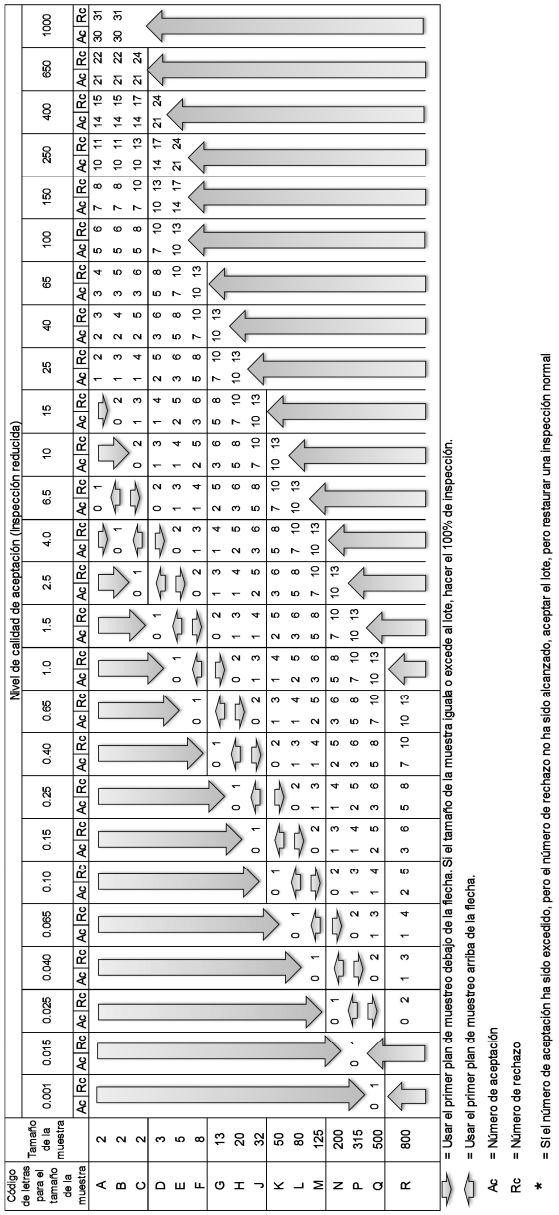
Continuación del anexo 2.

MIL STD 105E, tabla 2-B, tabla maestra para la inspección riguroso de un muestreo simple



Continuación del anexo 2.

MIL STD 105E, tabla 2-C, tabla maestra para la inspección reducida de un muestreo simple



Continuación del anexo 2.

- MIL STD 414, tabla A, código de letras para muestreo simple

	Niveles de inspección				
Tamaño del Lote	I	II	III	IV	V
3 a 8	B	B	B	B	C
9 a 15	B	B	B	B	D
16 a 25	B	B	B	C	E
26 a 40	B	B	B	D	F
41 a 65	B	B	C	E	G
66 a 110	B	B	D	F	H
111 a 180	B	C	E	G	I
181 a 300	B	D	F	H	J
301 a 500	C	E	G	I	K
501 a 800	D	F	H	J	L
801 a 1,300	E	G	I	K	L
1,301 a 3,200	F	H	J	L	M
3,201 a 8,000	G	I	L	M	N
8,001 a 22,000	H	J	M	N	O
22,001 a 110,000	I	K	N	O	P
110,001 a 550,000	I	K	O	P	Q
550,001 y más	I	K	P	Q	Q

Continuación del anexo 2.

MIL STD 414, tabla B, tabla maestra para inspección normal y ajustada para planes basados en variabilidad desconocida

Código de letras para el tamaño de la muestra	Tamaño de la muestra	Nivel de calidad de aceptación (Inspección normal)													
		0.04	0.065	0.1	0.15	0.25	0.4	0.65	1.00	1.50	2.50	4.00	6.50	10.00	15.00
		k	*k*	*k*	*k*	*k*	*k*	*k*	*k*	*k*	*k*	*k*	*k*	*k*	*k*
B	3	↓	↓	↓	↓	↓	↓	↓	▼	▼	1.12	0.958	0.765	0.566	0.341
C	4								1.45	1.34	1.17	1.01	0.814	0.617	0.393
D	5							1.65	1.53	1.40	1.24	1.07	0.874	0.675	0.455
E	7					2.00	1.88	1.75	1.62	1.50	1.33	1.15	0.955	0.755	0.536
F	10				2.24	2.11	1.98	1.84	1.72	1.58	1.41	1.23	1.03	0.828	0.611
G	15	2.64	2.53	2.42	2.32	2.20	2.06	1.91	1.79	1.65	1.47	1.30	1.09	0.886	0.664
H	20	2.69	2.58	2.47	2.36	2.24	2.11	1.96	1.82	1.69	1.51	1.33	1.12	0.917	0.695
I	25	2.72	2.61	2.50	2.40	2.26	2.14	1.98	1.85	1.72	1.53	1.35	1.14	0.936	0.712
J	30	2.73	2.61	2.51	2.41	2.28	2.15	2.00	1.86	1.73	1.55	1.36	1.15	0.946	0.723
K	35	2.77	2.65	2.54	2.45	2.31	2.18	2.03	1.89	1.76	1.57	1.39	1.18	0.969	0.745
L	40	2.77	2.66	2.55	2.44	2.31	2.18	2.03	1.89	1.76	1.58	1.39	1.18	0.971	0.746
M	50	2.83	2.71	2.60	2.50	2.35	2.22	2.08	1.93	1.80	1.61	1.42	1.21	1.00	0.774
N	75	2.90	2.77	2.66	2.55	2.41	2.27	2.12	1.98	1.84	1.65	1.46	1.24	1.03	0.804
O	100	2.92	2.80	2.69	2.58	2.43	2.29	2.14	2.00	1.86	1.67	1.48	1.26	1.05	0.819
P	150	2.96	2.84	2.73	2.61	2.47	2.33	2.18	2.03	1.89	1.70	1.51	1.29	1.07	0.841
Q	200	2.97	2.85	2.73	2.62	2.47	2.33	2.18	2.04	1.89	1.70	1.51	1.29	1.07	0.845
		0.065	0.10	0.15	0.25	0.40	0.65	1.00	1.50	2.50	4.00	6.50	10.00	15.00	
		Nivel de aceptación de calidad (Inspección rigurosa)													

Todos los valores de Nivel de calidad de aceptación (NCA) son porcentajes de defectuosos.

▼ = Usar el primer plan de muestreo debajo de la flecha, es decir, ambos tamaños de muestra como tambien el valor *k*. Cuando el tamaño de la muestra iguala o excede al tamaño del lote, cada artículo del lote debe de inspeccionarse.

Fuente: MONTOGOMERY, Douglas.*Introduction to statistical quality control*. pp.675 - 697.

Anexo 3.**Datos para práctica regresión lineal**

speed	*dist*	*speed*	*dist*	*speed*	*dist*	*speed*	*dist*	*speed*	*dist*
4	2	11	28	14	36	17	50	20	52
4	10	12	14	14	60	18	42	20	56
7	4	12	20	14	80	18	56	20	64
7	22	12	24	15	20	18	76	22	66
8	16	12	28	15	26	18	84	23	54
9	10	13	26	15	54	19	36	24	70
10	18	13	34	16	32	19	46	24	92
10	26	13	34	16	40	19	68	24	93
10	34	13	46	17	32	20	32	24	120
11	17	14	26	17	40	20	48	25	85

Fuente: datos “Cars” de paquete datasets, abril de 2016.