Capítulo 3. Gráficos en R

Una de las ventajas mayores del programa R es su capacidad de producir gráficos de alta calidad. Posee un gran número de funciones que son fáciles de usar y de modificar para ajustarlas a nuestras necesidades.

3.1 ¿Cómo Imprimir Gráficos?

Si queremos imprimir un gráfico generalmente procedemos en dos pasos:

1) Creamos un archivo con el gráfico

2) Posterior a nuestra sesión R imprime el archivo con el gráfico.

Para crear el archivo con el gráfico, el R posee la función postscript(), esta función crea un archivo para ser impreso posteriormente en una impresora con lenguaje postscript.

Trabajando bajo Windows la impresión puede realizarse directamente de la pantalla, escogiendo la opción de impresión.

3.2 Para Graficar una variable

Los siguientes son algunas funciones que permiten graficar una variable: boxplot(), hist(), stem(), qqnorm().

stem(): presenta un gráfico de tallos y hojas (stem-and-leaf) para unos datos dados. Por ejemplo,

```
> stem(pesos.monedas)
```

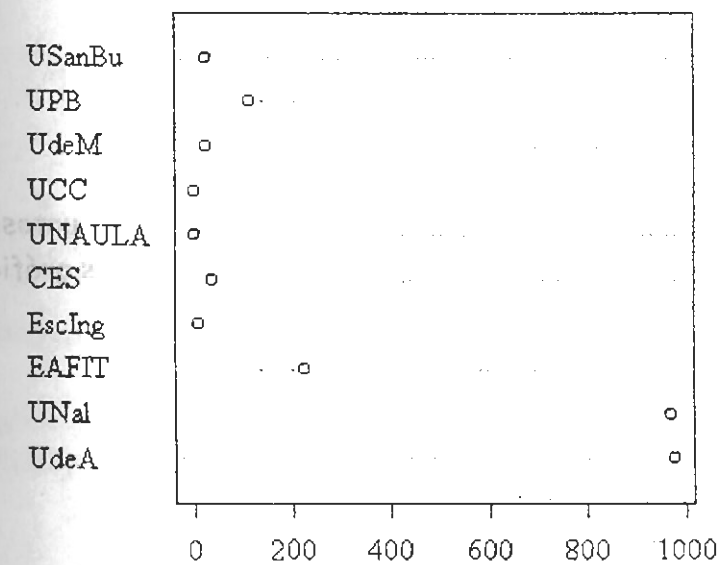
```
The decimal point is 1 digit(s) to the left of the |

48 | 4
49 |
50 |
51 | 6889
52 | 000111122223334555555666677777777777788888899999999999
53 | 00000111111111111222233333333345555667789
54 | 4

>
```

dotplot: Esta función nos permite crear un gráfico de punto de Cleveland. En el siguiente ejemplo graficamos el número de profesores de tiempo completo de ciertas universidades del departamento de Antioquia.

```
> dotplot(universidades.dat[,2],labels=c('UdeA','UNal','EAFIT',
+ 'EscIng','CES','UNAULA','UCC','UdeM','UPB','USanBu'))
>
```

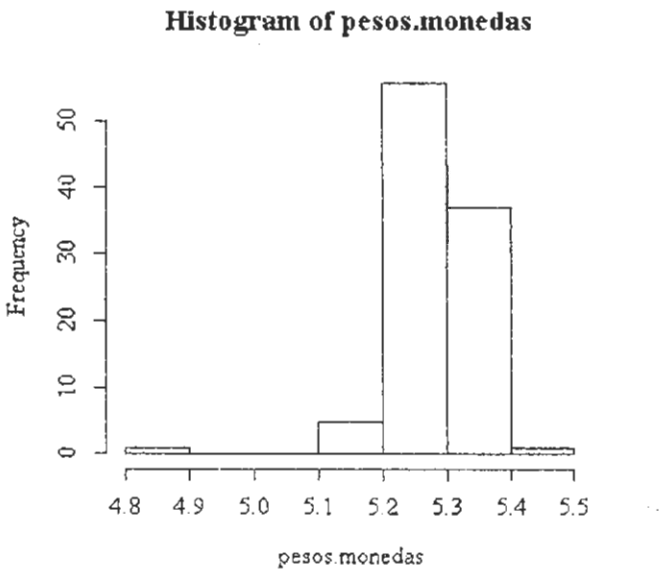


hist(): construye un histograma.Como argumentos esta función recibe

un vector con datos. Además, opcionalmente, puede entrarse como argumento el número de barras a ser graficadas o en su defecto el número de clases se determina con la fórmula de Sturges.

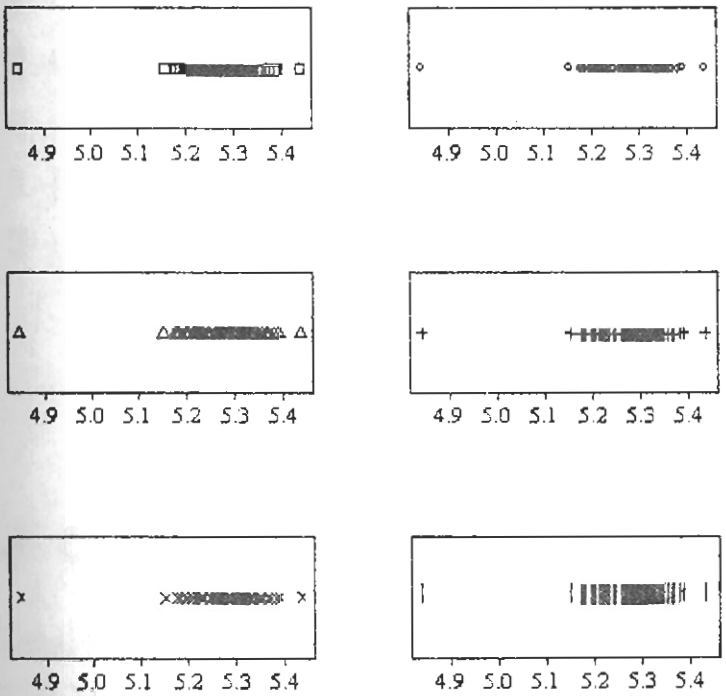
```
> hist(datos)

>hist(datos,nclass=5)
> hist(pesos.monedas)
```



stripplot(): Esta función nos produce un sencillo gráfico de puntos, que puede ser complementario para muchos otros gráficos.

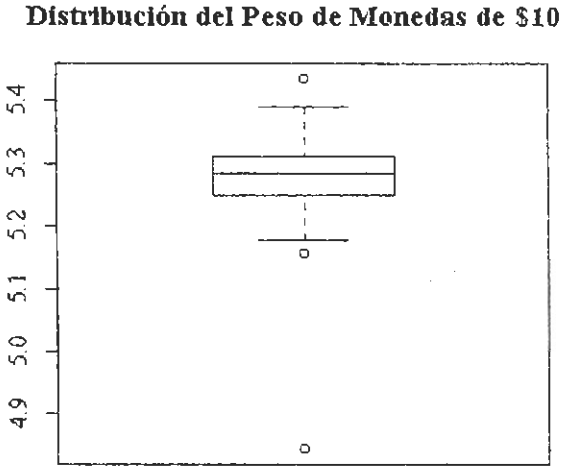
```
> par(mfrow=c(3,2))
> stripplot(pesos.monedas,pch=0)
> stripplot(pesos.monedas,pch=1)
> stripplot(pesos.monedas,pch=2)
> stripplot(pesos.monedas,pch=3)
> stripplot(pesos.monedas,pch=4)
> stripplot(pesos.monedas,pch=124)
>
```

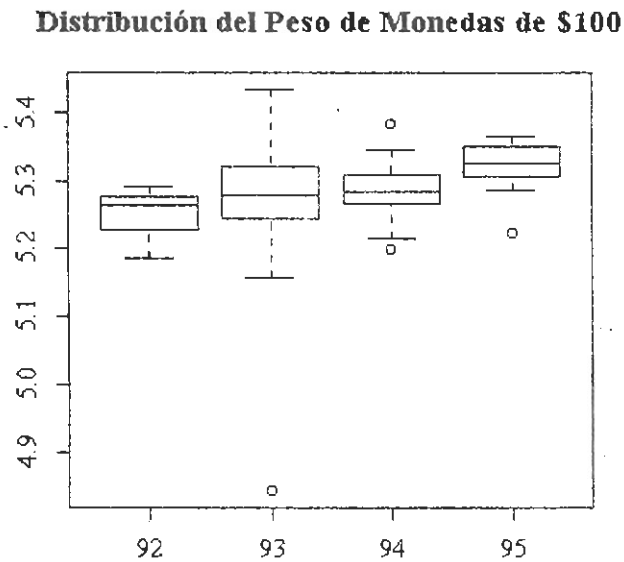


boxplot(): Produce gráficos de caja. Si se dan como argumentos varios vectores, el gráfico será igual número de cajas colocadas una al lado de las otras. Ejemplo:

```
> boxplot(hombres, mujeres)

> boxplot(pesos.monedas)
> title(main='Distribución del Peso de Monedas de $100')
```

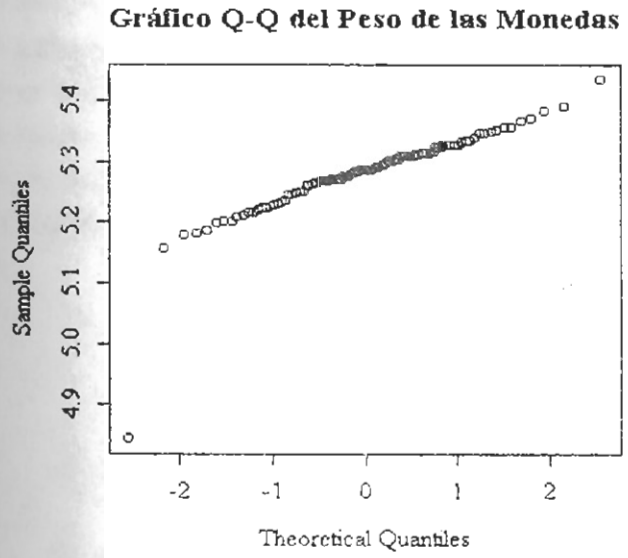




El gráfico anterior se logra con los siguientes comandos:

```
> boxplot(split(pesos.monedas,anos.monedas))
> title(main='Distribución del Peso de Monedas de $100')
>

> qqnorm(pesos.monedas,main='Gráfico Q-Q del Peso de las Monedas')
```



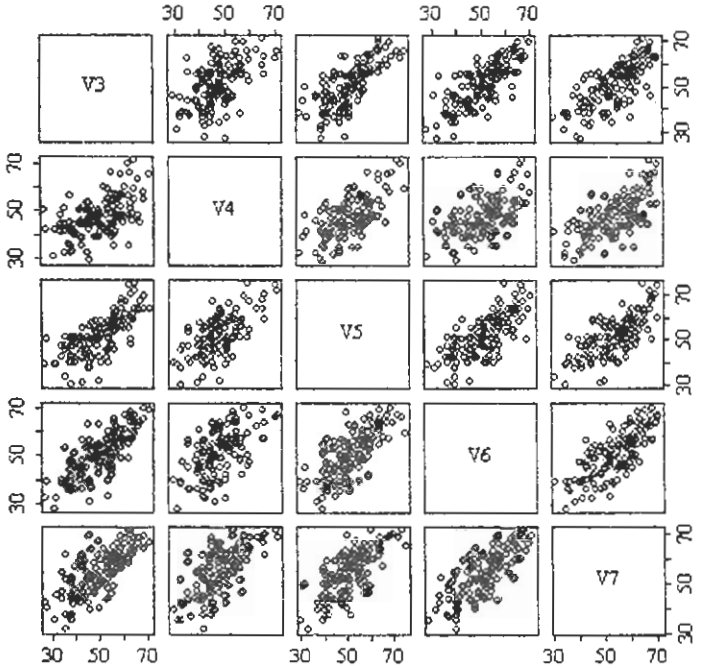
3.3 ¿ Cómo graficar más de una variable?

R provee muchas funciones para el caso multivariado. plot(), pairs(), matplot() persp() son algunas de ellas.

plot(): permite obtener un gráfico de dispersión entre un par de vectores. Realmente esta función es mucho más poderosa, ya que depende del objeto con que se trabaje.

pairs(): es la versión múltiple de plot(). Genera una matriz de gráficos de dispersión. El argumento es una matriz de datos.

```
> pairs(icfes.dat[,3:7])
>
```

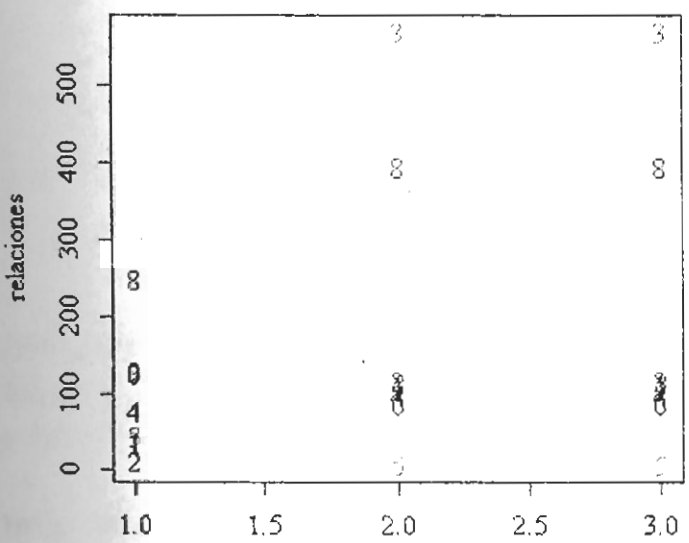


matplot(): Grafica las columnas de una matriz contra las columnas de otra. El siguiente ejemplo presenta información sacada del Anuario Estadística de Antioquia 1994 sobre la composición profesoral de distintas universidades en Antioquia y el número de estudiantes:

```
> universidades.dat
```

	Universidad	ProfTC	ProfMT	ProfC	Magister	PhD	Estudiantes
1	UdeA	977	308	235	454	67	34406
2	UNal	968	82	262	373	79	9495
3	EAFIT	224	17	651	NA	NA	9684
4	EscIng	9	7	182	28	0	688
5	CES	36	227	27	6	1	1582
6	UNAULA	0	0	193	19	1	2006
7	UCC	0	0	1041	26	0	8295
8	UdeM	24	15	786	99	10	5907
9	UPB	111	136	956	98	21	14083
10	USanBu	26	38	182	45	6	3239

```
> estudiantes.profesorTC<-universidades.dat[,7]/universidades.dat[,2]
> estudiantes.profesorMT<-universidades.dat[,7]/universidades.dat[,3]
> estudiantes.profesorC<-universidades.dat[,7]/universidades.dat[,3]
> relaciones<-rbind( estudiantes.profesorTC, estudiantes.profesorMT,
+  estudiantes.profesorC)
> matplot(relaciones)
>
```



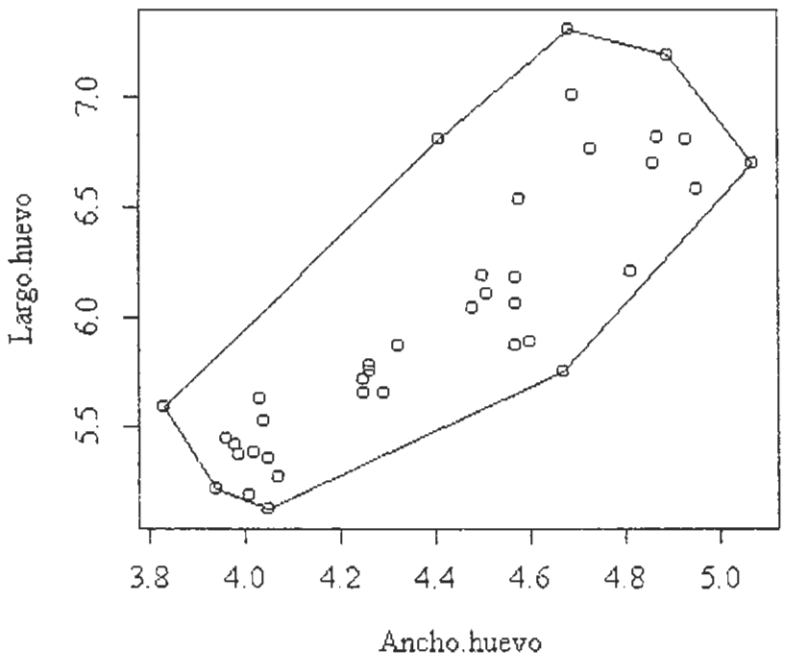
El anterior gráfico presenta las variables estudiantes/profesorTC, estudiantes/profesorMT, estudiantes/profesorC, en tres columnas respectivamente. En cada columna aparece la universidad codificada de acuerdo a su posición en el archivo.

chull Calcula el casco convexo de un conjunto de puntos

```
> plot(Ancho.huevo,Largo.huevo)
> hpts<-chull(Ancho.huevo,Largo.huevo)
> temp<-rbind(Ancho.huevo,Largo.huevo)
> dim(temp)
[1]  2 39
> temp<-t(temp)
```

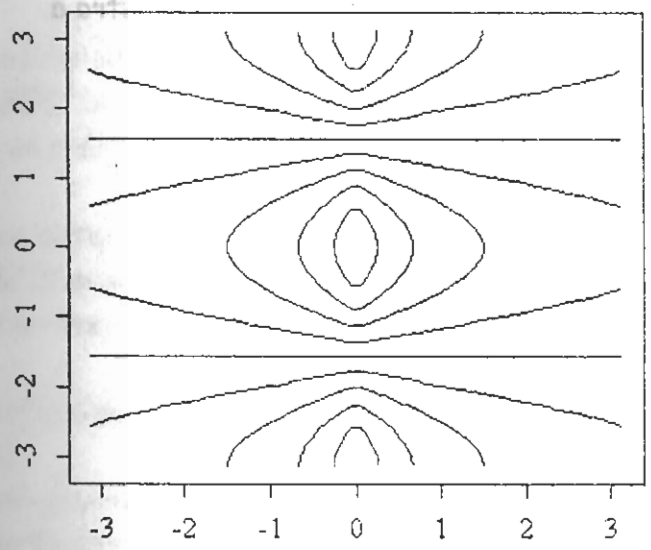
```
> hpts<-c(hpts,hpts[1])

> lines(temp[hpts,])
>
```



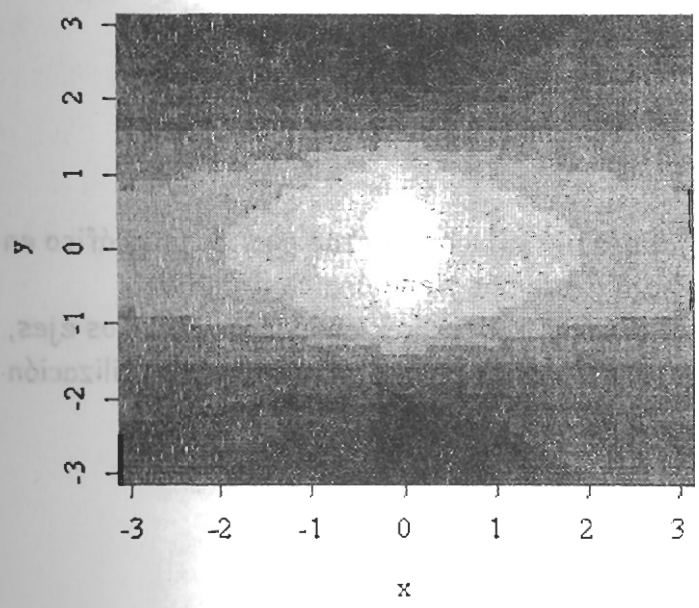
Contornos: Un gráfico importante es el de contornos cuando tenemos una función de dos variables.

```
> x<-seq(-pi,pi,len=50)
> y<-x
> f<-outer(x,y,function(x,y)cos(y)/(1+abs(x)))
> contour(x,y,f)
>
```



Observe el uso de la función outer(). Esta función tiene dos vectores y una función como argumentos, outer(x,y,function) y produce una matriz donde se presenta la evaluación.

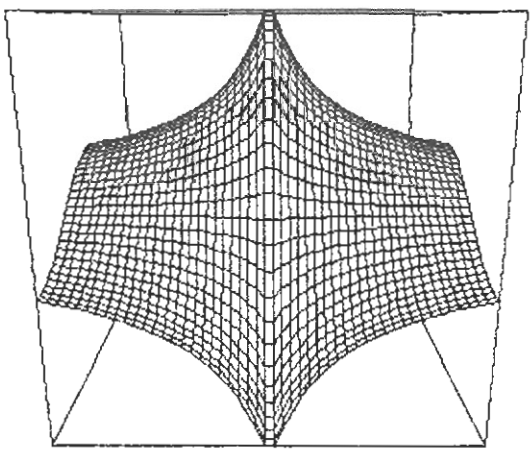
Un gráfico relacionado con el anterior es el siguiente



que se obtiene con la función image()

La función persp() permite graficar superficies como la que se ilustra a continuación

```
> persp(x,y,f)
>
```



3.4 Manejo de Parámetros Gráficos

Una función de utilidad es par(). Esta permite presentar más de un gráfico en la misma pantalla o en la misma hoja y además tiene una cantidad de parámetros que permiten colocar títulos, subtítulos, coordenadas en los ejes, controlar tamaño de las fuentes, controlar colores entre otras. Su utilización es simple:

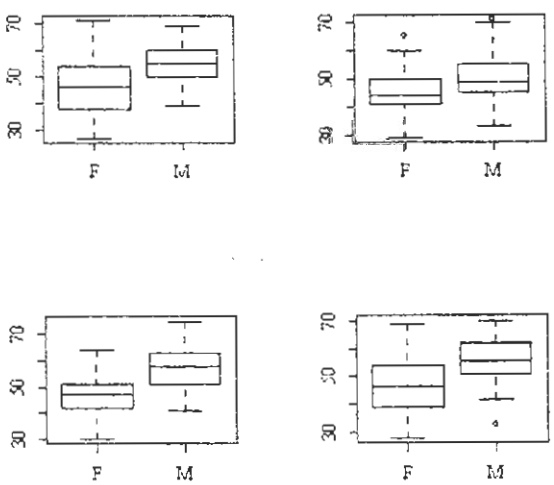
```
> par(mfrow=c(2,2))
```

Con el anterior comando le estamos indicando que hasta cuatro gráficas van a aparecer en la pantalla o en la hoja. Ejemplo:

Consideremos el conjunto de datos que hace referencia a las pruebas del ICFES. Si estamos interesados en una comparación por sexos podemos pensar en un análisis gráfico como el siguiente

Para obtenerlo asumamos que tenemos nuestro archivo de datos en formato ASCII guardado en el subdirectorio datos del disco duro. Una vez estamos en R corremos los siguientes comandos:

```
> icfes<-scan(''c:/datos/icfes.dat'',type=list('' '',1, 1, 1, 1, 1, 1, 1, 1, 1, 1,''
'')))
> win.graph()
> par(mfrow=c(2,2))
> boxplot(split(icfes[[3]],split[[1]]))
> boxplot(split(icfes[[4]],split[[1]]))
> boxplot(split(icfes[[5]],split[[1]]))
> boxplot(split(icfes[[6]],split[[1]]))
> par(mfrow=c(2,2))
> boxplot(split(icfes.dat[,3],icfes.dat[,1]))
> boxplot(split(icfes.dat[,4],icfes.dat[,1]))
> boxplot(split(icfes.dat[,5],icfes.dat[,1]))
> boxplot(split(icfes.dat[,6],icfes.dat[,1]))
>
```



3.5 Adicionando Elementos a un Gráfico

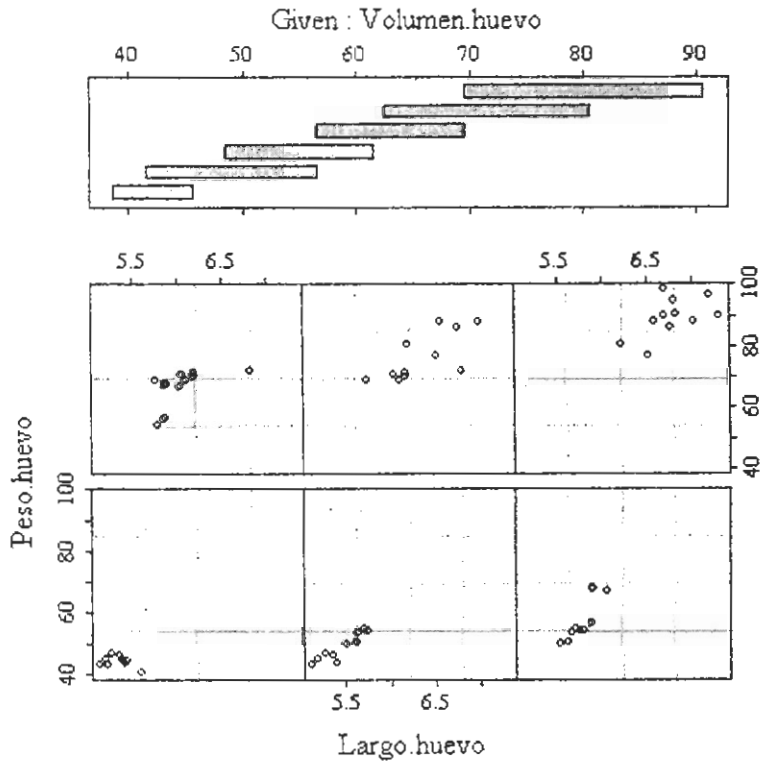
Las siguientes funciones nos permiten adicionarle elementos a un gráfico:

abline: Adiciona una línea recta.
arrows: Adiciona flechas a un gráfico
axis: Adiciona un eje a un gráfico
box: Adiciona una caja alrededor de un gráfico
coplot Gráficos Condicionales

Ejemplo: La base de datos se refiere a unas mediciones hechas a unos huevos

```
>huevos.dat<-read.table('a:huevos.txt')
> Ancho.huevo<-huevos.dat[,2]
> Largo.huevo<-huevos.dat[,1]
> Volumen.huevo<-huevos.dat[,4]
> Peso.huevo<-huevos.dat[,3]
```

- coplot(Peso.huevo~Largo.huevo|Volumen.huevo)

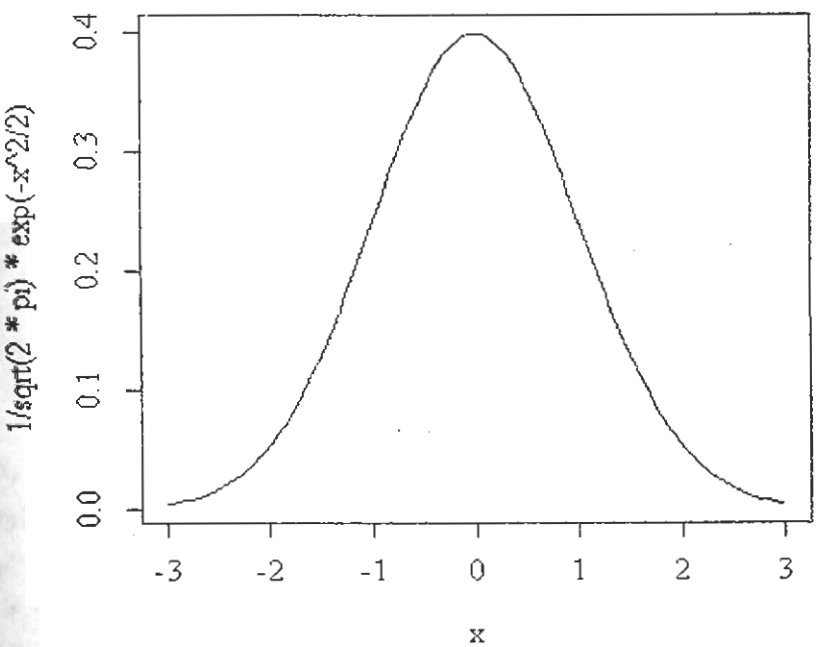


grid Adiciona una rejilla a un gráfico
legend Coloca leyendas a un gráfico
lines Adiciona segmentos de línea conectadas en un gráfico
mtext Nos permite escribir texto en las márgenes de un gráfico
points adiciona puntos a un gráfico
polygon: Dibuja polígonos
rect: Dibuja un rectángulo
text Adiciona texto a un gráfico

3.6 Gráficas de Funciones

Una función que es útil es curve(), ya que nos permite graficar funciones de una forma sencilla. El siguiente ejemplo nos permite graficar la normal estándar.

- curve(1/sqrt(2*pi)*exp(-x^2/2),-3,3)



3.7 Elementos de simulación

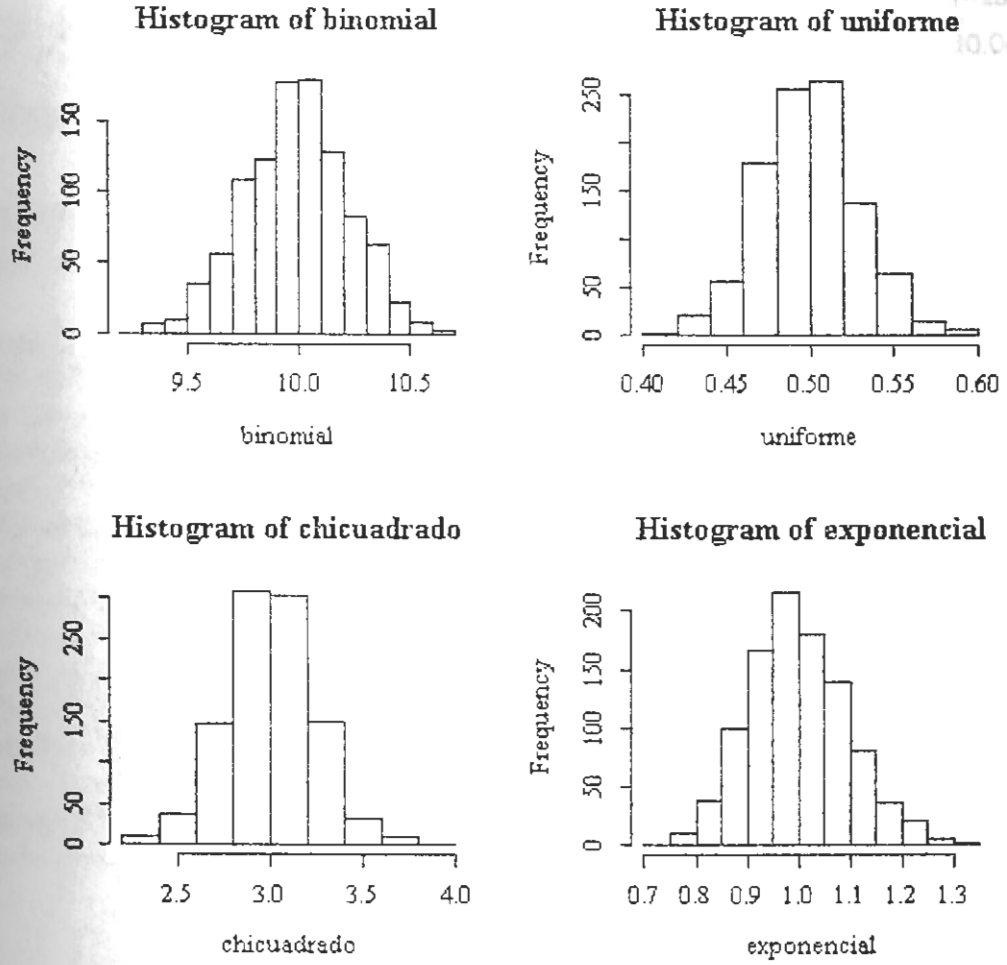
El R permite simular situaciones que permiten la visualización de resultados así como la generación de situaciones sin necesidad de realizar trabajo de campo, como por ejemplo, las llegadas de vehículos a una intersección o las respuestas dadas por n individuos a una encuesta.

3.7.1.Simulación del Teorema de límite central.

Este programa sirve programa para simular el TLC con las siguientes distribuciones: binomial(20,0.5),unif,chi-cuadrada y una exponencial

```
#programa para verificar el TLC con las siguientes
#distribuciones: binomial(20,0.5),unif,chi-cuadrada
#y una exponencial
>win.graph()
>par(mfrow=c(2,2))
>aux<-matrix(0,100,1000)
>muestras<-matrix(rbinom(aux,20,0.5),100,1000)
>binomial<-apply(muestras,2,mean)
>hist(binomial)
>muestras<-matrix(runif(aux),100,1000)
>uniforme<-apply(muestras,2,mean)
>hist(uniforme)
>muestras<-matrix(rchisq(aux,3),100,1000)
>chicuadrado<-apply(muestras,2,mean)
>hist(chicuadrado)
>muestras<-matrix(rexp(aux),100,1000)
>exponencial<-apply(muestras,2,mean)
>hist(exponencial)
>par(oma=c(1,1,1,1),new=T,font=2,cex=1)
>mtext(outer=T,"T.L.C con unas distribuciones",side=3)
```

T.L.C con unas distribuciones

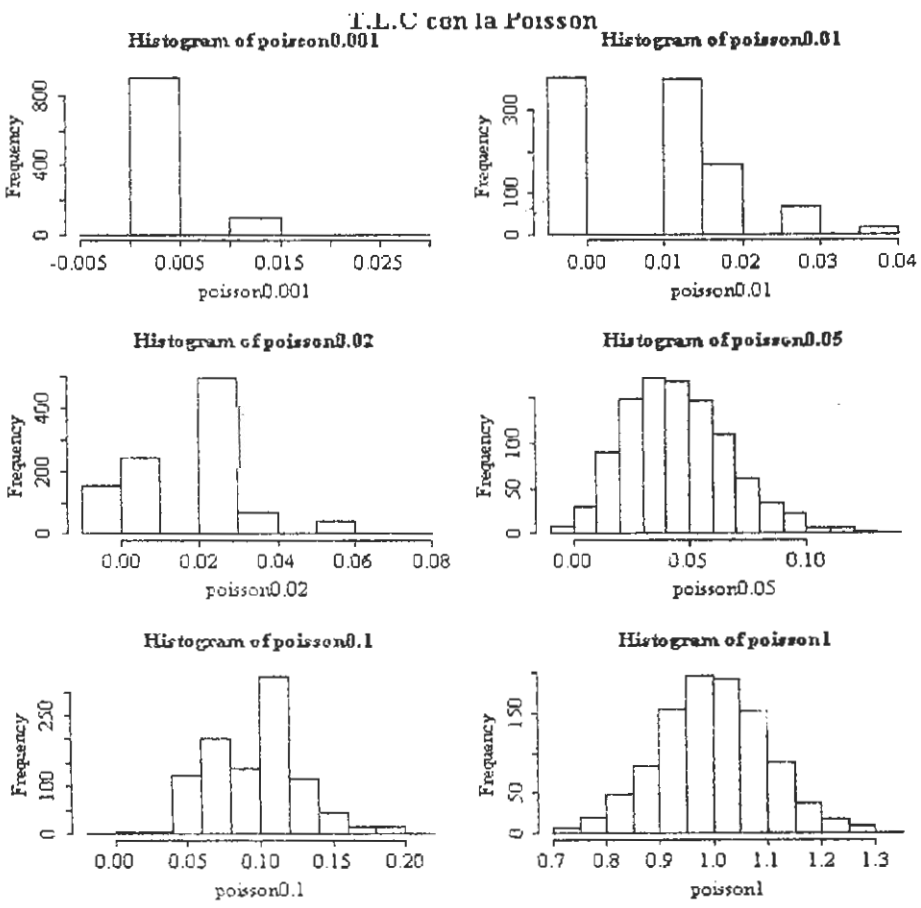


El siguiente programa ilustra un caso especial donde falla el Teorema de límite central

```
#programa para verificar que el TLC puede no funcionar
#en el caso de la Poisson cuando el lambda es pequeno
#Se generan 1000 muestra de tamano 100 con parametros
#0.001,0.01,0.02,0.05,0.1,1

>win.graph()
>par(mfrow=c(3,2))
>aux<-matrix(0,100,1000)
>muestras<-matrix(rpois(aux,0.001),100,1000)
poisson0.001<-apply(muestras,2,mean)
>hist(poisson0.001)
```

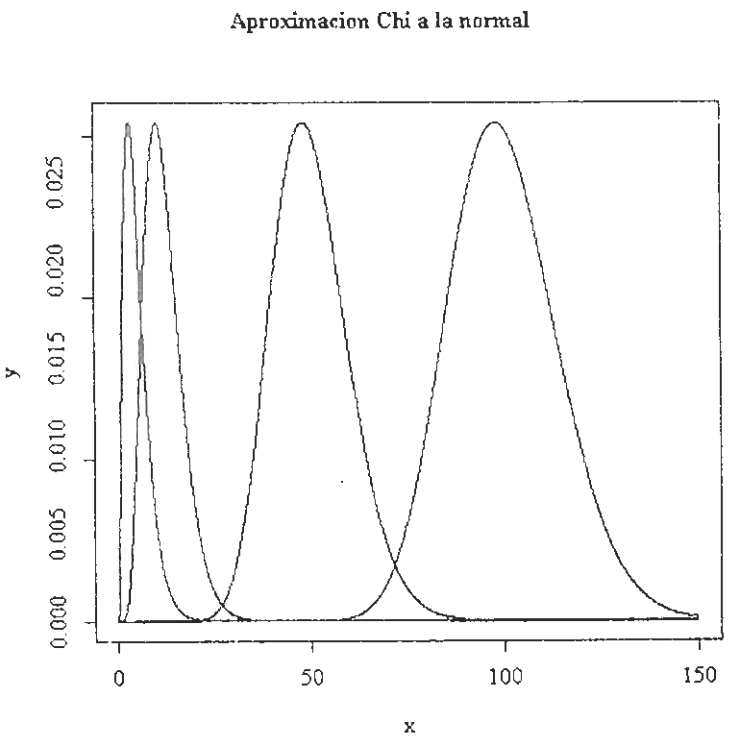
```
>muestras<-matrix(rpois(aux,0.01),100,1000)
>poisson0.01<-apply(muestras,2,mean)
hist(poisson0.01)
>muestras<-matrix(rpois(aux,0.02),100,1000)
>poisson0.02<-apply(muestras,2,mean)
>hist(poisson0.02)
muestras<-matrix(rpois(aux,0.05),100,1000)
>poisson0.05<-apply(muestras,2,mean)
>hist(poisson0.05)
>muestras<-matrix(rpois(aux,0.1),100,1000)
poisson0.1<-apply(muestras,2,mean)
>hist(poisson0.1)
>muestras<-matrix(rpois(aux,1),100,1000)
>poisson1<-apply(muestras,2,mean)
>hist(poisson1)
>par(oma=c(1,1,1,1),new=T,font=2,cex=1)
>mtext(outer=T,"T.L.C con la Poisson",side=3)
```



3.7.2. Simulación de una Chi-cuadrado con diferentes grados de libertad

El siguiente programa ilustra el comportamiento de la distribución Chi-cuadrado cuando se incrementan los grados de libertad

```
x<-seq(0,150,0.1)
win.graph()
y<-dchisq(x,5)
plot(x,y,type="l",col.lab="white",col.axis="white")
par(new="T",tcl=0)
y<-dchisq(x,12)
plot(x,y,type="l",col.lab="white",col.axis="white")
par(new="T",tcl=0)
y<-dchisq(x,50)
plot(x,y,type="l",col.lab="white",col.axis="white")
par(new="T",tcl=0)
y<-dchisq(x,100)
plot(x,y,type="l")
par(oma=c(1,1,1,1),cex=1.5,font=2)
mtext(outer=T,"Aproximacion Chi a la normal",side=3)
```



3.7.3 Simulación del lanzamiento de dos dados

El siguiente programa permite simular el lanzamiento de dos dados no cargados.

```
#Este programa simula el lanzamiento de dos
#dados no cargados n veces. El argumento es el
#numero de veces que se desea lanzar los dados

dados<-function(x){
n<-x
aux<-matrix(0,n,2)
result<-matrix(trunc(6*runif(aux)+1),n,2)
result
}
```

Para ejecutarlo digite

```
>dados(3)
```

El resultado puede ser el siguiente

```
> dados(3)
     [,1] [,2]
[1,]    2    4
[2,]    6    3
[3,]    1    2
>
```

Cada fila representa un lanzamiento.

3.7.8 Simulación del concepto de nivel de significación

La siguiente función permite ver el concepto de nivel de significación de la inferencia estadística.

```
#Se van a simular
#m muestras de tamaño n de una
#distribución normal con mu=u y sigma=s. Cada
#vez que se ejecute el proceso, se va a determinar
#el porcentaje de intervalos que contienen a mu=u.
#Los argumentos son:
#          m:número deseado de muestras
#          n:tamaño muestral
#          u:media poblacional de la normal
#          s:desviacion estandar pob. normal
#       alpha:nivel de significación


signifi<-function(m,n,u,s,alpha)
{
alpha<-alpha
aux<-matrix(0,n,m)
muestras<-matrix(rnorm(aux,u,s),n,m)
media<-apply(muestras,2,mean)
stdes<-sqrt(apply(muestras,2,var))
zalpha2<-qnorm(1-alpha/2,0,1)
liminf<-media-zalpha2*stdes/sqrt(n)
limsup<-media+zalpha2*stdes/sqrt(n)
interv<-as.matrix(rbind(t(liminf),t(limsup)))
porcent<-as.numeric(liminf<=u & u<=limsup)
confianza<-sum(porcent)*n/m
cat("Porcentaje de intervalos que contienen a mu de los",m,"construidos:
",confianza,"%","\n")
}
```

Para ejecutar esta función se pueden digitar los siguientes argumentos (el usuario puede colocar los que desee)

```
> signifi(1000,100,3,2,0.05)
Porcentaje de intervalos que contienen a mu de los 1000 construidos:  95.6 %
>
```

Si se escoge alpha=0.01 el resultado puede ser:

```
> signifi(1000,100,3,2,0.01)
Porcentaje de intervalos que contienen a mu de los 1000 construidos:  98.7 %
```

Capítulo 4. Creación de Nuevas Funciones en R

Una de las mayores ventajas del programa R es la facilidad que le deja al usuario de crear nuevas funciones que se integran casi automáticamente (usted necesita salvar el workspace) al sistema y que pueden seguir siendo usadas posteriormente. La sintaxis para escribir funciones es la siguiente:

```
> mi.funcion<-function(argumento1,
argumento2,...) {

...Instrucciones en R...

}
```

Si solo tenemos una línea de instrucciones las llaves no son necesarias. Por ejemplo si queremos crear una función que nos entregue el coeficiente de kurtosis, podemos entrar el siguiente comando

```
> kurtosis<-function(x) mean(((x-mean(x))/sqrt(var(x)))^4)-3
```

Asi mismo si queremos definir una función que calcule el coeficiente de asimetría podemos entrar el siguiente comando

```
> asimetria<-function(x) mean(((x-mean(x))/sqrt(var(x)))^3)
```

La siguiente función nos presenta el lanzamiento de un dado:

```
> lanzar.dado<-function()trunc(runif(1)*6+1)
> lanzar.dado()
[1] 2
> lanzar.dado()
[1] 6
```

El siguiente ejemplo nos permite calcular una distribución muestral vía simulación. Suponga que lanzamos simultáneamente dos dados 130 veces.

Queremos determinar los percentiles 2.5% y 97.5% para el resultado 7, o sea la suma de los dados sea 7

```
> lanzar.dos.dados<-function(x)lanzar.dado()+lanzar.dado()

> res<-matrix(rep(NA,1000),ncol=1)
> for(i in 1:1000){
+ tem<-matrix(rep(0,130),ncol=1)
+ tem<-lapply(tem,lanzar.dos.dados)
+ res[i]<-sum(ifelse(tem==7,1,0))
+ }
> win.graph()
> hist(res)
> quantile(res,c(0.025,0.975))
 2.5% 97.5%
   13    30
>
```

Las anteriores funciones quedan integradas al R si salvamos la hoja de trabajo al salir de la sesión y pueden seguir siendo utilizadas sin necesidad de definirlas. Si queremos ver cómo está definida una función damos su nombre simplemente sin paréntesis.

En caso de tener más de una línea las llaves son necesarias para distinguir la función de los otros comandos. Una vez la función es definida pasa a ser parte integral del R y puede llamarse en otras ocasiones.

Por ejemplo, para calcular la media geométrica de un vector x podemos definir la siguiente función:

```
> media.geometrica<-function(x) prod(x)^(1/length(x))
```

Si queremos usarla posteriormente para calcular la media geométrica de un vector y simplemente escribimos el comando

```
> media.geometrica(y)
```

Otro ejemplo es la siguiente función que permite hacer un rápido análisis exploratorio gráfico de una variable:

```
> forma.aed<-function(x){
par(mfrow=c(2,2))
hist(x)
boxplot(x)
dic<-summary(x)[5]-summary(x)[2]
plot(density(x,width=2*dic), xlab=''x'', ylab='' '', type=''l'')
qqnorm(x)
qqline(x)
}
```

Esta es una función que produce cuatro gráficas: un histograma, una caja de Tukey, una estimación de la densidad y un gráfico q-q (cuantil vs. cuantil), y le adiciona una línea que pasa por el primer y tercer cuartil

4.1. Ejemplo: Función para calcular estadísticas básicas

```
calcule.todo <-function(x)
{
        cat("\n", "Estadisticas Basicas", "\n", "\n")
        mini <- min(x)
        maxi <- max(x)
        media <- mean(x)
        mediana <- median(x)
        varianza <- var(x)
        desviacion.tipica <- sqrt(varianza)
        DMA <- mean(abs(x - media))
        DMA.robusta <- mean(abs(x - mediana))
        cuantiles <- quantile(x, c(0.05, 0.1, 0.25, 0.75, 0.9, 0.95))
        Q1 <- cuantiles[3]
        Q3 <- cuantiles[4]
        per05 <- cuantiles[1]
        per10 <- cuantiles[2]
        per90 <- cuantiles[5]
        per95 <- cuantiles[6]
        IQR <- Q3 - Q1
```

```
    Gini <- mean(abs(outer(x, x, FUN = "-")))
    sesgo <- mean(((x - media)/desviacion.tipica)^3)
    kurtosis <- mean(((x - media)/desviacion.tipica)^4) - 3
    cat("Medidas de Localizacion", "\n","\n")
    cat("Media=", media, "\n")
    cat("Mediana=", mediana, "\n")
    cat("Minimo=", mini, "\n")
    cat("Percentil 5%=", per05, "\n")
    cat("Percentil 10%=", per10, "\n")
    cat("Primer Cuartil=", Q1, "\n")
    cat("Tercer Cuartil=", Q3, "\n")
    cat("Percentil 90%=", per90, "\n")
    cat("Percentil 95%=", per95, "\n")
    cat("Maximo=", maxi, "\n")
    cat("\n", "Medidas de Dispersion", "\n", "\n")
    cat("rango Intercuartil=", IQR, "\n")
    cat("Varianza=", varianza, "\n")
    cat("Desviacion Tipica=", desviacion.tipica, "\n")
    cat("Desviacion Media Absoluta=", DMA, " \n")
    cat("Desv.  Media Absoluta Robusta=",DMA.robusta, "\n")
    cat("Diferencia Media de Gini=", Gini, "\n")
    cat("\n", "Medidas de Forma", "\n", "\n")
    cat("Sesgo=", sesgo, "\n")
    cat("Kurtosis=", kurtosis, "\n")
}

> calcule.todo(pesos.monedas)


Estadisticas Basicas

Medidas de Localizacion

Media= 5.2796
Mediana= 5.286
Minimo= 4.843
Percentil 5%= 5.19745
Percentil 10%= 5.2145
```

Primer Cuartil= 5.251
Tercer Cuartil= 5.31225
Percentil 90%= 5.346
Percentil 95%= 5.3565
Maximo= 5.436

Medidas de Dispersion

rango Intercuartil= 0.06125
Varianza= 0.004443091
Desviacion Tipica= 0.06665651
Desviacion Media Absoluta= 0.04332
Desv. Media Absoluta Robusta= 0.04284
Diferencia Media de Gini= 0.0634616

Medidas de Forma

Sesgo= -2.708467
Kurtosis= 16.42673

4.2. Ejemplo: Prueba para varianzas de Levene

Las siguientes líneas presentan la creación de las funciones necesarias para aplicar la *Prueba de Levene* para la igualdad de varianzas.

```
desviacion1.fun<-function(x){
z<-sum((x-mean(x))^2)
z
}
fun<-function(x){
z<-abs(x-median(x))
z
}
<-function(Lista){
N1<-lapply(Lista,length)
N<-sum(unlist(N1))
I<-length(Lista)
Des<-lapply(Lista,desviacion.fun)
Medias<-lapply(Des,mean)
```

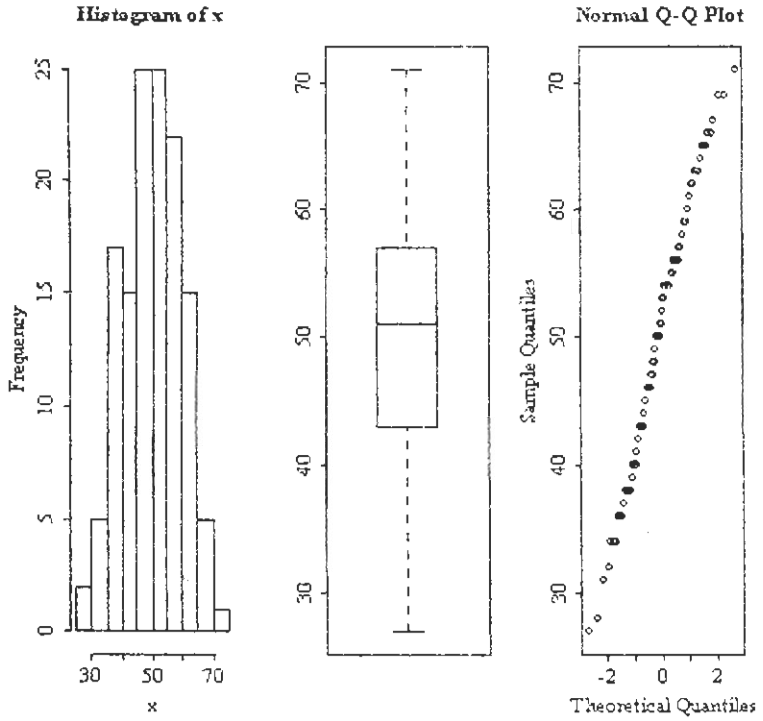
```
media<-mean(unlist(Des))
Des2<-lapply(Des,desviacion1.fun)
S2<-sum(unlist(Des2))/(N-I)
<-as.vector(as.numeric(unlist(Medias)))
S1<-sum(as.vector(N)*((Medias-media)^2))
L<-(S1*(I-1))/S2
gl1<-I-1
gl2<-N-I
valor.p<-1-pf(L,gl1,gl2)
list(L=L,gl1=gl1,gl2=gl2,valor.p=valor.p)
}
```

4.3. Ejemplo: Gráficos simultáneos

Una función para producir tres gráficos univariables simultáneamente es la siguiente

```
graficos <-function(x)
{
        win.graph()
        par(mfrow = c(1, 3))
        hist(x)
        boxplot(x)
        qqnorm(x)
}


>graficos(icfes.dat[,3])
```



4.4. Ejemplo: Gráfico de contornos

La siguiente función nos permite crear un gráfico de contornos, bajo el supuesto de una normal bivariable, para un par de vectores de datos.

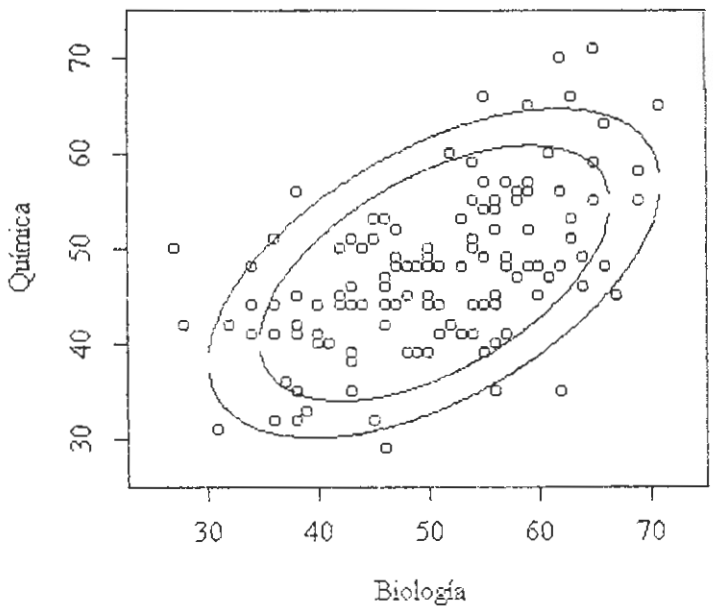
```
 elipse<-function(d1, d2, xtexto = "Variable 1", ytexto = "Variable2", titulo = "")
{
        options(warn = -1)
        sigma <- chol(var(cbind(d1, d2)))
        x <- (-100:100 * 2.14)/100
        y <- sqrt(4.605 - x^2)
        win.graph()
        xx <- rbind(x, y)
        y2 <-  - sqrt(4.605 - x^2)
        xx2 <- rbind(x, y2)
        yy <- t(xx) %*% sigma
        yy[, 1] <- yy[, 1] + mean(d1)
        yy[, 2] <- yy[, 2] + mean(d2)
        ymax <- max(yy[, 2])
        xmax <- max(yy[, 1])
        yy2 <- t(xx2) %*% sigma
```

```
yy2[, 1] <- yy2[, 1] + mean(d1)
yy2[, 2] <- yy2[, 2] + mean(d2)
ymin <- min(yy2[, 2])
xmin <- min(yy2[, 1])
rango1<-0.05*(max(d1)-min(d1))
lx1<-min(d1)-rango1
lx2<-max(d1)+rango1
rango2<-0.05*(max(d2)-min(d2))
ly1<-min(d2)-rango2
ly2<-max(d2)+rango2
plot(d1, d2, ylim=c(ly1,ly2),xlim=c(lx1,lx2),xlab = xtexto, ylab =   ytexto,
main = titulo)
par(new = T)
plot(yy[, 1], yy[, 2], type = "l", xlab = "", ylab = "",
ylim=c(ly1,ly2),xlim=c(lx1,lx2)    )
par(new = T)
plot(yy2[, 1], yy2[, 2], type = "l", xlab = "", ylab = "",
ylim=c(ly1,ly2),xlim=c(lx1,lx2))
x <- (-100:100 * 1.66)/100
y <- sqrt(2.773 - x^2)
xx <- rbind(x, y)
y2 <-  - sqrt(2.773 - x^2)
xx2 <- rbind(x, y2)
yy <- t(xx) %*% sigma
yy[, 1] <- yy[, 1] + mean(d1)
yy[, 2] <- yy[, 2] + mean(d2)
ymax <- max(yy[, 2])
xmax <- max(yy[, 1])
yy2 <- t(xx2) %*% sigma
yy2[, 1] <- yy2[, 1] + mean(d1)
yy2[, 2] <- yy2[, 2] + mean(d2)
ymin <- min(yy2[, 2])
xmin <- min(yy2[, 1])
par(new = T)
plot(yy[, 1], yy[, 2], type = "l", xlab = "", ylab = "",
ylim=c(ly1,ly2),xlim=c(lx1,lx2)    )
par(new = T)
plot(yy2[, 1], yy2[, 2], type = "l", xlab = "", ylab = "",
```

```
        ylim=c(ly1,ly2),xlim=c(lx1,lx2))
}
```

La anterior función la utilizamos para mirar los resultados del ICFES en Biología y Química.

```
> elipse(icfes.dat[,3],icfes.dat[,4],xtexto='Biología',ytexto='Química')
>
```



4.5. Ejemplo: Codificación de variables

La siguiente función nos permite asignar códigos a un vector de datos.

```
#codifica un vector en cuatro categorias
#x es el vector
#x1 el primer limite, x2 el segundo, x3 el tercero
codif<-function(x,x1,x2,x3){
for(i in 1:length(x))
{y<-rep(NA, length(x))
x1<-as.numeric(x1)
x2<-as.numeric(x2)
x3<-as.numeric(x3)
y[x<x1]<-1
```

```
y[x1<=x & x<x2]<-2
y[x2<=x & x<x3]<-3
y[x3<=x]<-4}
y
}
```

Para ilustrar su uso se creará un vector aleatorio que representará las edades de 50 individuos y creara categorías así:

Si 0<=edad<10 entonces nuevaedad=1
Si 10<=edad<30 entonces nuevaedad=2
Si 30<=edad<65 entonces nuevaedad=3

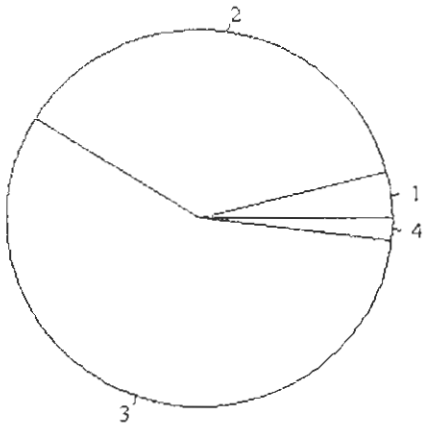
Los comandos pueden ser los siguientes:

```
> a<-seq(0,50,1)
> edad<-trunc(runif(a)*65+1)
> edad
 [1] 11 63 56 24 48 10 20 37  6  3 18 52 59 47 19 50 49 15 28 47 65 26 32 38
37
[26] 62 39 52 63 53 26 40 10 53 62 34 19 61 40 11 45 45 12 26 28 34 52 17
37 12
[51] 26
> nuevaedad<-codif(edad,10,30,65)
> nuevaedad
 [1] 2 3 3 2 3 2 2 3 1 1 2 3 3 3 2 3 3 2 2 3 4 2 3 3 3 3 3 3 3 2 3 2 3 3 3 2 3
[39] 3 2 3 3 2 2 2 3 3 2 3 2 2
```

Si se quiere construir un gráfico de sectores para la variable **nuevaedad**, digite la siguiente secuencia:

```
> piechart(table(nuevaedad))
> par(oma=c(1,1,1,1),new=T,font=2,cex=1)
> mtext(outer=T,"GRAFICO DE SECTORES PARA NUEVAEDAD",side=3)
>
```

GRAFICO DE SECTORES PARA NUEVAEDAD

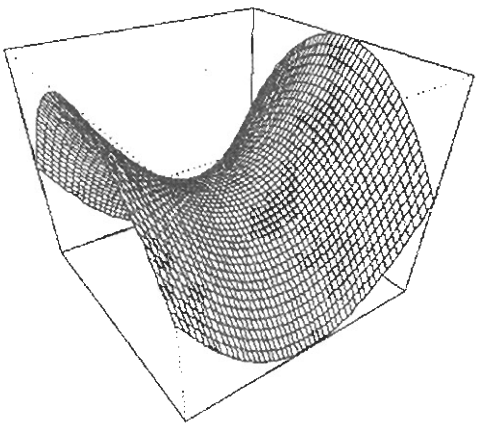


4.6. Ejemplo: Gráfico de una silla de montar

La siguiente función permite graficar un paraboloide hiperbólico o silla de montar

```
> win.graph()
> silla<-function(x,y){x^2-y^2}
> y<-x<-seq(-25, 25, length = 50)
> persp(x,y, outer(x,y, FUN =  silla), theta = -35, phi = 35)
> par(oma=c(1,1,1,1),new=T,font=2,cex=1)
> mtext(outer=T,"PARABOLOIDE HIPERBOLICO",side=3)
```

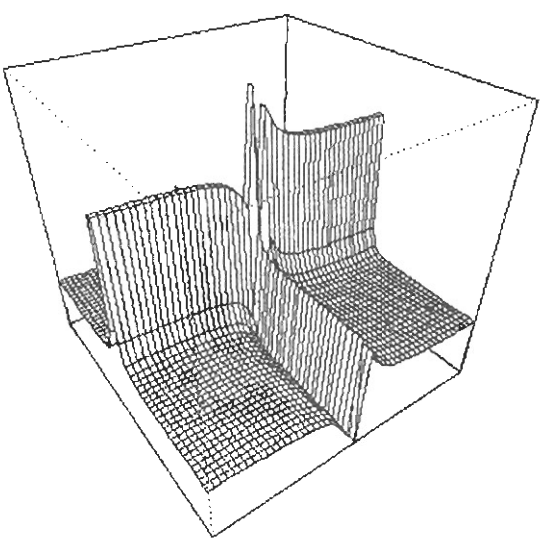
PARABOLOIDE HIPERBOLICO



Usando la siguiente secuencia de comandos se puede graficar una bonita superficie:

```
> super<-function(x,y){ (1/x)*cos(1/x)+(1/y)*sin(1/y)}
> y<-x<-seq(-25, 25, length = 50)
> persp(x,y, outer(x,y, FUN =  super), theta = -35, phi = 35)
> par(oma=c(1,1,1,1),new=T,font=2,cex=1)
>mtext(outer=T,"Grafico de: z=(1/x)cos(1/x)+(1/y)sen(1/y)",side=3)
>
```

Grafico de: z=(1/x)cos(1/x)+(1/y)sen(1/y)



4.7 Algunas reglas para escribir funciones en R

1) Utilice nombres completos tanto para definir funciones como para los argumentos. Aunque el nombre de los argumentos puede ser abreviado, el nombre completo puede dar mayor claridad. Existen muchas funciones en R, por lo tanto la selección de un buen nombre es importante.

2) Escoja "defaults" razonables para los argumentos.

3) Comience de una forma simple, chequeando inmediatamente y agregue capacidades y complejidad gradualmente e interactivamente.

4) Trate de pensar en términos de ''todo el problema''. Qué es lo que se pretende conseguir al final?. No trate de escribir las instrucciones secuencialmente como se hace en un programa de Fortran.

5) Trate con la situación más general, si es posible. Por ejemplo, qué hacer con NA's (datos faltantes), con datos de tipo caracter, listas, argumentos de longitud 0. De otra parte, recuerde que es fácil complicarse demasiado. Un programa corto es preferible si funciona bien el 90% de las veces a un programa extremadamente largo que considera todos los casos posibles.

6) Haga los apropiados chequeos de errores.

7) Sea especialmente cuidadoso en la construcción de un vector elemento a elemento en loops. Es preferible crear un objeto completo y reemplazar sus partes poco a poco que tener que aumentar un objeto cada vez.

8) Utilice líneas de comentario. Escriba buena documentación.

9) Utilice on.exit() para hacer limpieza final: cambios en parámetros, en gráficos, remover archivos temporales, etc.