Tarea 2. Funciones de densidad y MLE

Análisis Estadístico Multivariado

Emmanuel Alcalá

Nota:

Recuerden que en el repositorio de Github se encuentran varios tutoriales de R

1. r_intro para una vista eficiente al lenguaje de R, como llamadas de función, operaciones, etc.
2. r_flujo_funciones para una introducción rápida a control de flujo y creación de funciones en R.
3. r_data_wrangling_dplyr para una introducción a la paquetería de manipulación de tablas y data.frames, `dplyr`.
4. data_viz_ggplot para una introducción rápida al uso de la paquetería de visualización de `ggplot2`.

Para guardar gráficos en RStudio, sigue el siguiente link.

Resolver

1. (20 puntos). Sea $X_1, X_2, \dots, X_n$ una muestra aleatoria de la función de densidad de probabilidad dada por

$$f(x \mid \theta) = \frac{2x}{\theta} e^{-x^2/\theta}, \ \theta > 0, x > 0$$

   a) Encuentre el MLE para θ.
   b) Comprobar que es un máximo.
2. (20 puntos). Sea $X_1, X_2, \dots, X_n$ una muestra aleatoria de la función de la distribución

$$f(x \mid \theta) = \theta x^{\theta-1}, \ 0 < x1$$

   a) Encuentra el MLE para θ.
   b) Demuestra que es un máximo.
3. (20 puntos) Hacer una investigación sobre el MLE de una distribución uniforme y explicar el resultado. Poner un ejemplo.

Ejemplo ilustrativo (uso de R para obtener los MLE)

> Leer cuidadosamente el siguiente ejemplo en R para resolver dos problemas parecidos.

Simularemos un conjunto de datos para una distribución normal usando la función `rnorm()` de R, con media y desviación estándar que definiremos. Posteriormente, estimaremos los parámetros del vector aleatorio creado. Si el método funciona bien, debemos obtener estimadores cercanos a los parámetros reales que usamos.

El siguiente chunk de código se puede reproducir en un script de R

```
par(las=1, mar = c(2.5, 2, 0, 0)) # opciones de gráficos
# asignar parámetros reales
mu_real <- 50
sd_real <- 5
# simular variable aleatoria normal con media mu_real y desviación estándar sd_real
rx_sim <- rnorm(100, mu_real, sd_real)
# histograma, freq = FALSE para probabilidad en vez de frecuencia
hist(rx_sim, freq = FALSE, ylim=c(0, 0.1), main = '')
# la función curve crea una curva continua basada en la función que coloquemos
# como primer argumento, dnorm en este caso; dejamos x sin especificar, pero
# agregamos from y to para definir el rango del cómputo de la densidad en x;
# notar también que aunque no especificamos x, sí especificamos mean y sd
# finalmente, add=TRUE indica que sobreponga la curva al histograma; sin este
# argumento, no se sobrepondría y crearía un gráfico aparte
curve(dnorm(x, mean = mu_real, sd = sd_real), from = 0, to = 200, add = TRUE)
```

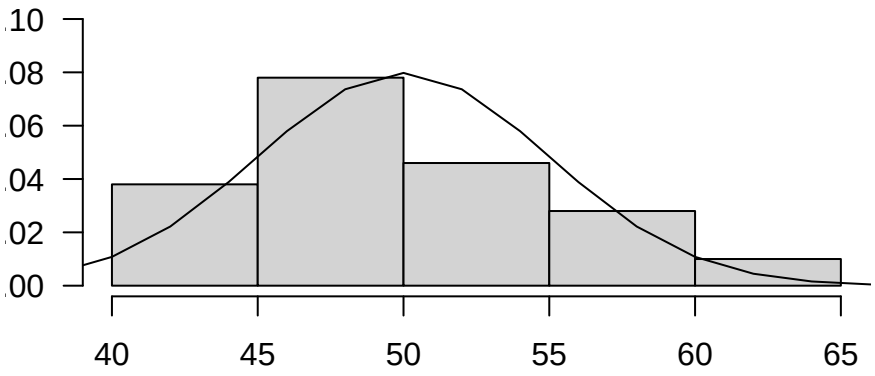


Figure 1: Gráfico de datos simulados.

Lo siguiente que haremos es crear una función de verosimilitud para estos datos. Recordar que la función de verosimilitud es una distribución conjunta para cada $X_1, X_2, \ldots, X_n$. En nuestra

simulación, $n = 100$, y cada X_i corresponde a un valor de `rx_sim`. Por lo tanto, nuestra distribución conjunta es una probabilidad para cada valor de `rx_sim`.

Para obtener la probabilidad de un X_i en `rx_sim` usaremos `dnorm()`. Por ejemplo, la probabilidad del primer dato de `rx_sim`, que lo obtenemos como `rx_sim[1]`, asumiendo que tiene media de 10 y desviación estándar de 3, se obtendría como

```
dnorm(rx_sim[1], 10, 3)
```

```
[1] 4.083934e-37
```

Ahora crearemos una función llamada `neg_log_lik_gaussian`. El `neg_` es porque obtendremos la función de verosimilitud negativa. Notar que si una función es cóncava si es positiva, al multiplicarla por -1 se convierte en convexa. Si en una cóncava buscamos el máximo, en una convexa buscamos el mínimo, sin pérdida de generalidad. Haremos esto porque el algoritmo que usaremos, implementado en una función llamada `nlm` (non-linear optimization), trabaja mejor buscando mínimos que buscando máximos.

```
# argumentos:
# theta: numeric de longitud 2, su primer valor es la media, el segundo la sd
# x: numeric con los datos de entrada
neg_log_lik_gaussian <- function(theta, x) {
  # creamos la función de verosimilitud; log=TRUE retorna la transformación
  # logarítmica, de otra forma tendríamos que obtener L, luego l <- log(L)
  l <- dnorm(x, theta[1], theta[2], log=TRUE)
  # dado que ya hicimos la transformación logarítmica, sumamos la log-prob
  # notar que multiplicamos por -1 (signo negativo por delante de sum())
  # esto nos va a retornar la log_likelihood negativa
  -sum(l)
}

# Ahora vamos a definir un vector de parámetros iniciales. La mayoría
# de los algoritmos de optimización necesitan parámetros iniciales para
# comenzar con la búsqueda de los mejores parámetros.
# Al parámetro inicial de mu lo asignamos al primer valor de rx_sim;
# al parámetro inicial de sd como la media del rango intercuartílico.
# Esto podría funcionar o no; escoger parámetros iniciales es un arte
theta.start <- c(rx_sim[1], IQR(rx_sim)/2)
# Ahora usaremos el algoritmo de optimización no lineal, nlm.
# Se puede pedir ayuda con ?nlm.
# Los argumentos necesarios son una función, los parámetros iniciales,
# el vector x (que es un argumento de la función neg_log_lik_gaussian)
```

```r
# y agrego el argumento interlim para aumentar la cantidad de iteraciones
# por defecto tiene 100, que pueden ser insuficientes
nlm(neg_log_lik_gaussian, theta.start, x=rx_sim, iterlim = 1e5)
```

```
$minimum
[1] 305.2164

$estimate
[1] 49.943569  5.120364

$gradient
[1] -1.319330e-06 -2.350175e-07

$code
[1] 1

$iterations
[1] 11
```

Cuando se corre este algoritmo puede arrojar `warnings`, pero no necesariamente es algo malo. Lo que a veces sucede es que en la búsqueda de parámetros, puede obtener valores de densidad con `dnorm()` para los cuales el logaritmo no está definido (por ejemplo, `log(-1)`). Los estimadores de la media y la desviación se darán en el output como `$estimate`.

Si decidimos asignar los resultados a un objeto, debemos hacer algo como lo siguiente para obtener los estimadores

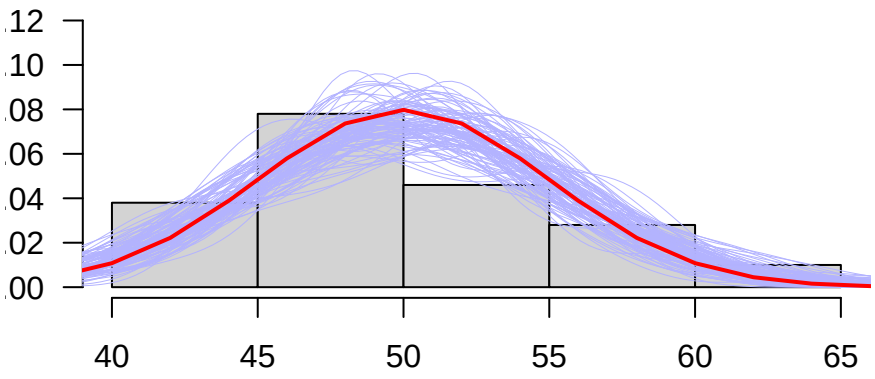
```r
estimadores <- nlm(neg_log_lik_gaussian, theta.start, x=rx_sim, iterlim = 1e5)
estimadores$estimate
```

```
[1] 49.943569  5.120364
```

En mi caso, arroja valores bastantes cercanos (media de 49.07, sd de 4.49) con solo 19 iteraciones.

Para saber qué tan buenos resultados se obtienen con estos estimadores con respecto a los datos reales, podemos simular datos y graficarlos encima de los datos reales para visualizar gráficamente.

```r
# simulación de diferentes valores aleatorios
par(las=1, mar = c(2.5, 2, 0, 0))
# graficamos histograma original
hist(rx_sim, freq = FALSE, ylim = c(0,0.12), main = '')
# guardamos los estimadores en un vector que llamaremos coefs
coefs <- estimadores$estimate
# simularemos 100 veces la variable aleatoria
nsims <- 100
# cada experimento aleatorio será de 100 valores (n=100), y tendremos 100
# de estos experimentos (nsims)
n <- length(rx_sim)
# usaremos un ciclo for para crear líneas
for (i in 1:nsims) {
  # creamos una variable aleatoria temporal que se renueva en cada ciclo;
  # usaremos los coeficientes coefs obtenidos con nlm
  rx <- rnorm(n, mean = coefs[1], sd = coefs[2])
  # usaremos la función density() de R, que obtiene densidades
  # usando un algoritmo no paramétrico llamado kernel density estimate;
  # por el momento, lo importante es que permite obtener densidad de probabilidad
  # y la usaremos para crear líneas que sobrelapen al histograma antes creado
  # obtendremos la densidad entre 0 y 150
  den_rx <- density(rx, from = 0, to = 150)
  # la función lines permite añadir elementos al gráfico original, en este
  # caso, el creado con hist(). lwd es el grosor de la línea; haremos
  # líneas muy delgadas
  lines(den_rx$x, den_rx$y, lwd = 0.1, col = '#B3B3FF')
}
# repetir la curva original con los datos reales, pero en color rojo y gruesa
curve(dnorm(x, mean = mu_real, sd = sd_real), col = 'red',
  lwd = 2,from = 0, to = 200, add = TRUE)
```



Podemos ver que con los parámetros obtenidos, obtenemos densidades (líneas negras) bastante próximas a los parámetros reales (línea roja).

Resolver

4. (20 puntos). Dado el siguiente vector de datos aleatorios `x` (que representa el peso en libras de 10 personas)

```
# vector de datos aleatorios
x <- c(115, 122, 130, 127, 149, 160, 152, 138, 149, 180)
# coloca correctamente entre las comas los argumentos de la función para
# obtener los estimadores
# estimadores <- nlm(, , , iterlim = 1e5) # quita el primer '#'
```

a) obtén los MLE (de forma analítica) de la media (θ_1) y la desviación estándar ($\sqrt{\theta_2}$) asumiendo que están normalmente distribuidos.

b) obtén los MLE usando R. Luego, realiza un gráfico de histograma y añade 100 simulaciones con los parámetros estimados en el paso anterior. Con base en ello, concluye qué tan bien se ajusta la simulación a los datos en `x`.

5. (20 puntos)

a) Obtén el MLE de la función de densidad exponencial, que tiene distribución

$$f(x) = \frac{1}{\lambda} \exp\left\{-\frac{x}{\lambda}\right\}$$

b) Realiza una simulación con datos exponencialmente distribuidos, similar al ejemplo ilustrativo, con `n=100` y una tasa de 60. Realiza 100 simulaciones aleatorias y grafica 1) un histograma de las *medias* de cada simulación (¿qué tan frecuente es observar una media como la obtenida?); 2) un histograma de la simulación original y la densidad de las 100 simulaciones. Guíate con el ejemplo ilustrativo.

Para obtener la variable aleatoria de datos exponenciales, se usa `rexp(100, 1/60)`. Por diseño, en R la tasa en las exponenciales se parametriza como el inverso; en vez de `lambda=60`, es `rate=1/60`.

Hint: en la función de verosimilitud negativa que hagas, en vez de usar `dnorm` ahora usarás `dexp`, y solo se tendrá un parámetro.