Práctica 5: Creación de funciones I

1. Estimación por máxima verosimilitud

Contexto teórico

En Inferencia Estadística paramétrica el método de máxima verosimilitud nos permite obtener estimadores de parámetros poblacionales desconocidos con buenas propiedades[1]. El problema inferencial básico y la solución por dicho método de máxima verosimilitud se formula como sigue:

Problema: Sea $X_1, \ldots, X_n$ una muestra aleatoria simple (m.a.s.)[2] de una variable aleatoria X.[3]. Supongamos que la distribución de probabilidad de X pertenece a una familia paramétrica de distribuciones de probabilidad P_θ con parámetro desconocido[4] $\theta \in \Theta \subset \mathbb{R}^k$. El objetivo principal es "determinar" (estimar) θ en base a la muestra.

Solución (Método de Máxima Verosimilitud): Para una realización particular de la muestra, $(x_1, \ldots, x_n)$, se define la función de verosimilitud como:

$$L(\theta) = \prod_{i=1}^{n} f_\theta(x_i)$$

donde f_θ es la función de densidad[5] de X.
El *estimador máximo verosímil* de θ basado en la muestra anterior se define como el valor $\widehat{\theta}$ que maximiza la función de verosimilitud.
Nota: En la práctica es más fácil maximizar la *log-verosimilitud*) $l(\theta) = \log(L(\theta))$, ya que el producto se convierte en una suma.

Procedimiento de cálculo:

1. Construir la función de verosimilitud a partir de la muestra y calcular su logaritmo, $l(\theta)$.

2. Para cada componente θ_j de $\theta = (\theta_1, \ldots, \theta_k)$ formular y resolver:

$$\frac{\partial}{\partial \theta_j} l(\theta) \quad = \quad 0 \quad (j = 1, \ldots, k) \tag{1}$$

   Las ecuaciones anteriores se denominan *ecuaciones normales*.

[1] Invarianza, consistencia, insesgadez asintótica y, bajo ciertas condiciones, normalidad asintótica.
[2] Una m.a.s. es una colección de variables aleatorias independientes y con la misma distribución (i.i.d.).
[3] Por abuso del lenguaje identificamos la población con dicha variable aleatoria.
[4] Al parámetro θ se le denomina *parámetro poblacional* y al conjunto donde toma valores, Θ, espacio paramétrico.
[5] O la función masa de probabilidad si la variable es discreta.

3. Comprobar que las soluciones de las ecuaciones anteriores, $\widehat{\theta} = (\widehat{\theta}_1, \ldots, \widehat{\theta}_k)$ en efecto constituyen un máximo, por ejemplo comprobando que:

$$\frac{\partial}{\partial \theta_j} l(\theta)|_{\theta_j = \widehat{\theta}_j} < 0$$

Aplicación: Estimadores máximo-verosímiles de los paramétros de una distribución Gamma

A continuación se recogen datos correspondientes a cuantías pagadas (en miles de euros) para una muestra de 50 reclamaciones de una aseguradora:

25.03	18.59	47.20	80.20	187.67
95.94	35.07	145.38	9.52	128.14
136.69	180.82	49.67	33.41	4.16
94.87	102.25	11.04	35.14	151.15
17.14	81.94	20.01	125.26	7.11
61.36	55.59	10.80	31.88	16.39
45.95	4.98	23.20	8.78	30.68
22.65	13.19	40.62	2.78	35.41
8.63	17.04	8.02	126.54	2.11
136.93	17.39	37.73	84.53	14.22

Almacena los datos anteriores en un vector de R con nombre `muestra`. Para ello puedes por ejemplo usar la función `scan`, copiando los datos directamente del pdf.

Estudios previos revelan que los datos anteriores se pueden modelizar mediante una distribución de probabilidad Gamma. Supongamos por tanto que dichos datos constituyen una realización de una m.a.s. $X_1, \ldots, X_n$ de una población $X \sim Gamma(a, b)$, para ciertos parámetros $a, b > 0$. Nuestro objetivo a continuación es obtener los estimadores máximo verosímiles de los parámetros a y b a partir de la muestra anterior de tamaño $n = 50$.

Recuerda que la distribución Gamma es un modelo de probabilidad para variables continuas cuya función de densidad tiene la forma:

$$f(x) = \frac{b^a}{\Gamma(a)} x^{a-1} e^{-x/b}, \quad x > 0 \tag{2}$$

con parámetros $a, b > 0$.
A partir de la densidad anterior la función de verosimilitud y su logaritmo se escribirían

como sigue:

$$\begin{aligned} L(a,b) &= \prod_{i=1}^{n} \frac{b^a}{\Gamma(a)} x_i^{a-1} e^{-x_i/b} \\ l(a,b) &= log(L(a,b)) \qquad (3) \\ &= -na\log(b) - n\log(\Gamma(a)) + (a-1)\sum_{i=1}^{n}\log(x_i) - \sum_{i=1}^{n} x_i/b. \qquad (4) \end{aligned}$$

Y nuestra tarea es obtener los valores $\widehat{a}$ y $\widehat{b}$ que maximizan $l(a,b)$. Para ello vamos a implementar dos posibles soluciones: la optimización directa y la resolución de las ecuaciones normales.

Método 1: Optimización directa

En R tenemos varias funciones que permiten resolver problemas de optimización[6] como el que hemos planteado. Entre ellas está la función de propósito general `optim` en el paquete *stats*, la cual implementa los métodos BFGS (Broyden-Fletcher-Goldfarb-Shanno) y CB (gradiente conjugado), entre otros. Otra función interesante es `solnp` en el paquete *Rsolnp*, para resolver problemas de optimización no lineal usando el método de los multiplicadores de Lagrange aumentados, incluyendo restricciones de igualdad o desigualdad. Y finalmente destacamos la función `maxLik` del paquete *maxLik* que permite resolver el problema de maximización de la verosimilitud. A continuación vamos a probar estas tres funciones.

El primer paso es definir una función que calcule la log-verosimilitud. Para facilitar la tarea vamos a utilizar la función `dgamma` que implementa la función de densidad de la distribución Gamma con parámetros a (`shape`) y b (`scale`). Con ella nuestra función se define como:

```
logl<-function(theta)
{
  a<-theta[1]
  b<-theta[2]
  l<-sum(log(dgamma(x=muestra,shape=a,scale=b)))
  return(-l)
}
```

Observa que hemos creado la función con un sólo argumento (`theta`) que corresponde a al vector de parámetros (a,b). En el cuerpo de la función está el cálculo directo de la log-verosimilitud (3), donde utilizamos el vector de datos `muestra`. Este vector constituye una variable local en la función. Finalmente el valor devuelto es la log-verosimilitud cambiada de signo. Esto último se hace ya que los optimizadores de propósito general calculan

[6] Puedes ver una descripción en `https://cran.r-project.org/web/views/Optimization.html`

mínimos en lugar de máximos. Una vez escrita la función puedes hacer una pequeña comprobación de que funciona correctamente pasándole valores al argumento[7].
Una vez que tenemos la función a optimizar pasamos a realizar la optimización. De las tres posibilidades que comentábamos antes comenzamos usando la función `optim`. Esta función requiere que le pasamos la función a optimizar a través del argumento `fn`, así como un vector de valores iniciales en el argumento `par`:

```
> # Establecer valores iniciales para los parámetros
> b0<-a0<-1
> res<-optim(par=c(a0,b0),fn=logl)
> res$par

[1]  1.060301 50.605110
```

En este caso hemos considerado como valores iniciales $a = b = 1$. Aunque es una elección un tanto arbitraria vemos que el algoritmo converge rápido y la función encuentra el óptimo (almacenado en la componente `res$par`).

Inspecciona qué otros resultados contiene la lista (`res`) que devuelve la función. Luego prueba a cambiar los valores iniciales y observa los cambios en el resultado.

Repetimos el cálculo ahora usando la función `solnp` del paquete *Rsolnp*. Para ello primero debes instalarlo y luego cargarlo. Si inspeccionas la ayuda de la función verás que su uso es muy similar al de `optim`. También verás que esta función permite introducir de una forma sencilla restricciones en la optimización. En nuestro caso los parámetros a y b que buscamos deben ser positivos por lo que vamos a usar esta información en la llamada a la función usando el argumento `LB`. Para nuestro problema sería:

```
> library(Rsolnp)

Warning: package 'Rsolnp' was built under R version 4.1.3

> b0<-a0<-1
> res<-solnp(pars=c(a0,b0),fun=logl,LB=c(0,0))

Warning in .safefunx(tmpv, .solnp_fun, .env, ...):
solnp->warning: Inf detected in function call...check your function
Warning in .safefunx(tmpv, .solnp_fun, .env, ...):
solnp->warning: Inf detected in function call...check your function


Iter: 1 fn: 249.0377  Pars:   1.06079 50.54313
Iter: 2 fn: 249.0377  Pars:   1.06078 50.54378
solnp--> Completed in 2 iterations
```

[7] Prueba por ejemplo $(a, b) = (1, 1)$, si has implementado correctamente la función debes obtener $l(1, 1) = 2680.8$.

```
> res$pars

[1]  1.060784 50.543776
```

Observa del resultado anterior que la función encuentra el óptimo, aunque nos da una advertencia relacionada con la evaluación de la log-verosimilitud. Podemos resolver el problema proporcionado unos valores iniciales menos arbitrarios y más cercanos a la solución.

Un método alternativo a la estimación máximo-verosímil es el método de los momentos. Para la distribución Gamma este método ofrece estimadores $\widehat{a} = s^2/\overline{x}$ y $\widehat{b} = \overline{x}/\widehat{a}$. Repite la optimización que has hecho antes con la función `solnp` pero ahora usando estos valores iniciales. Observa la diferencia con respecto a los valores iniciales arbitrario que usábamos antes.

```
> a0<-var(muestra)/mean(muestra)
> b0<-mean(muestra)/a0
> res<-solnp(pars=c(a0,b0),fun=logl,LB=c(0,0))


Iter: 1 fn: 249.0377  Pars:   1.06079 50.54410
Iter: 2 fn: 249.0377  Pars:   1.06078 50.54368
solnp--> Completed in 2 iterations

> res$pars

[1]  1.060784 50.543684
```

Finalmente usamos la función `maxLik` del paquete *maxLik*. Como comentábamos antes está diseñado para los problemas de optimización relativos a la máxima verosimilitud, esto conlleva una primera diferencia y es que buscará un máximo (en lugar de un mínimo como hacían las otras funciones que hemos probado antes). Por tanto tendremos que modificar el signo del valor devuelto por la función de verosimilitud `logl` que construíamos antes. Con este cambio utilizaríamos la función `maxLik` como se muestra a continuación (recuerda que debes instalar primero el paquete):

```
> library(maxLik)

Warning: package 'maxLik' was built under R version 4.1.3
Loading required package: miscTools
Warning: package 'miscTools' was built under R version 4.1.3

Please cite the 'maxLik' package as:
```

```
Henningsen, Arne and Toomet, Ott (2011). maxLik: A package for maximum likelihood
estimation in R. Computational Statistics 26(3), 443-458. DOI 10.1007/s00180-010-0217-1
If you have questions, suggestions, or comments regarding the 'maxLik' package,
please use a forum or 'tracker' at maxLik's R-Forge site:
https://r-forge.r-project.org/projects/maxlik/

> logl2<-function(theta) -logl(theta)
> maxLik(logl2,start=c(1,1))

Warning in dgamma(x = muestra, shape = a, scale = b): NaNs produced
Warning in dgamma(x = muestra, shape = a, scale = b): NaNs produced
Warning in dgamma(x = muestra, shape = a, scale = b): NaNs produced
Warning in dgamma(x = muestra, shape = a, scale = b): NaNs produced
Warning in dgamma(x = muestra, shape = a, scale = b): NaNs produced
Warning in dgamma(x = muestra, shape = a, scale = b): NaNs produced
Warning in dgamma(x = muestra, shape = a, scale = b): NaNs produced
Warning in dgamma(x = muestra, shape = a, scale = b): NaNs produced
Warning in dgamma(x = muestra, shape = a, scale = b): NaNs produced
Warning in dgamma(x = muestra, shape = a, scale = b): NaNs produced



                                                                        (reltol)
```

Observa que el máximo se almacena en la componente `estimate` de la lista devuelta. El algoritmo converge pero nos da varios mensajes de advertencia relacionados con la evaluación de la log-verosimilitud.

> En la llamada de la función hemos usado como valores iniciales $a = b = 1$, proporcionados en el argumento `start`. Una mejor elección podrían ser los estimadores del método de los momentos que comentábamos antes. Prueba a poner estos valores iniciales y observa si se produce algún cambio.

```
> a0<-var(muestra)/mean(muestra)
> b0<-mean(muestra)/a0
> maxLik(logl2,start=c(a0,b0))

Warning in dgamma(x = muestra, shape = a, scale = b): NaNs produced
Warning in dgamma(x = muestra, shape = a, scale = b): NaNs produced
Warning in dgamma(x = muestra, shape = a, scale = b): NaNs produced
Warning in dgamma(x = muestra, shape = a, scale = b): NaNs produced
```

```
Warning in dgamma(x = muestra, shape = a, scale = b): NaNs produced

Maximum Likelihood estimation
Newton-Raphson maximisation, 17 iterations
Return code 8: successive function values within relative tolerance limit (reltol)
Log-Likelihood: -249.0377 (2 free parameter(s))
Estimate(s): 1.060607 50.55734
```

Un último comentario es que los parámetros deben ser positivos. La función `maxLik` permite pasar restricciones para la optimización usando el argumento `constraints`. A continuación vamos a repetir el resultado anterior incluyendo dos restricciones $a > 0$ y $b > 0$. Observa que dichas restricciones se pueden escribir en formato matricial como $A\theta + B > 0$, donde $\theta = (a, b)$, A es la matriz identidad de orden 2, y B un vector de ceros de dimensión 2. Con esta representación introducimos las restricciones de la forma siguiente:

```
> A<-matrix(c(1,0,0,1),2)
> B<-c(0,0)
> maxLik(logl2,start=c(1,1),constraints=list(ineqA=A,ineqB=B))

Maximum Likelihood estimation
Nelder-Mead maximization, 87 iterations
Return code 0: successful convergence
Log-Likelihood: -249.0377 (2 free parameter(s))
Estimate(s): 1.060301 50.60511
```

Observa que el resultado es similar al que obteníamos sin restricciones pero ahora no aparecen mensajes de advertencia.

Método 2: Resolución de las ecuaciones normales

Partiendo de la ecuación (4) podemos deducir las ecuaciones normales calculando las derivadas parciales:

$$\begin{aligned} \frac{\partial}{\partial a} l(a,b) &= -n\log(b) - n\frac{\Gamma'(a)}{\Gamma(a)} + \sum \log(x_i) = 0 \\ \frac{\partial}{\partial b} l(a,b) &= -na/b + \sum x_i/b^2 = 0 \end{aligned}$$

De la segunda ecuación resolvemos $b = \overline{x}/a$, y sustituyendo en la primera llegamos a:

$$\log(a) - \frac{\Gamma'(a)}{\Gamma(a)} - \log(\overline{x}) + \overline{\log(x)} = 0, \tag{5}$$

donde $\overline{x} = \sum x_i/n$ y $\overline{\log(x)} = \sum \log(x_i)/n$.
Observa que la ecuación (5) no se puede resolver analíticamente y por tanto tenemos que recurrir a métodos numéricos. Para ello podemos usar por ejemplo la función `uniroot`, que implementa el método de la bisección. Lo hacemos escribiendo primero una función implementando la expresión (5)[8] y luego la usamos como argumento en la función `uniroot` como sigue:

```
> f<-function(a) log(a)-digamma(a)- log(mean(muestra))+mean(log(muestra))
> res<-uniroot(f,c(0.1,100))
> res

$root
[1] 1.060808

$f.root
[1] -1.330094e-05

$iter
[1] 13

$init.it
[1] NA

$estim.prec
[1] 6.103516e-05
```

Puedes consultar en la ayuda los detalles de la función `uniroot`, en este caso hemos usado dos argumentos, el primero es la función que define la ecuación a resolver, y el segundo un intervalo donde encontrar la solución. Observa que la función te devuelve la solución de la ecuación en la componente `root` de la lista obtenida al ejecutarla.

Con la solución anterior para a puedes ahora obtener el de b como $b = \overline{x}/a$. Compara con las soluciones que nos dieron los métodos anteriores.

```
> b<-mean(muestra)/res$root
> b

[1] 50.54261
```

[8] La función $\Gamma'(a)/\Gamma(a)$ se denomina función digamma y está implementada en la función de R `digamma`.

2. Creación de funciones estadísticas elementales

1. Construye una función nombre `medias` que devuelva una lista con tres componentes: la media aritmética, la media armónica y la media geométrica. La función debe tener un único argumento `x` (el vector de datos). Además debe incluir comprobaciones básicas (como que argumento proporcionado es numérico y si tiene datos perdidos, en cuyo caso deberá ignorarlos para el cálculo). Si alguna de las medias no se puede calcular para los datos proporcionados se devolverá el valor `NA` para dicha media y un mensaje de advertencia al usuario.

```
> medias <- function(x) {
+   if (!is.numeric(x)) stop("El vector debe ser numérico")
+   x<-x[!is.na(x)]
+   m.geo<- function (x) exp(mean(log(x)))
+   m.arm<- function (x) 1/mean(1/x)
+   mx.geo<-if(min(x)>0) m.geo(x) else NA
+   mx.arm<-if(any(x==0)) NA else m.arm(x)
+   if (is.na(mx.geo) || is.na(mx.arm)) warning("El vector
                                  tiene valores negativos o cero")
+   return(list(media.aritm=mean(x), media.geom=mx.geo,
+               media.arm=mx.arm))
+ }
```

Comprueba el resultado de tu función con los siguientes resultados:

```
> medias(1:10)

$media.aritm
[1] 5.5

$media.geom
[1] 4.528729

$media.arm
[1] 3.414172

> medias(c(1:10,NA))

$media.aritm
[1] 5.5

$media.geom
```

```
[1] 4.528729

$media.arm
[1] 3.414172

> medias(0:10)

Warning in medias(0:10): El vector
tiene valores negativos o cero

$media.aritm
[1] 5

$media.geom
[1] NA

$media.arm
[1] NA

> medias(-1:10)

Warning in medias(-1:10): El vector
tiene valores negativos o cero

$media.aritm
[1] 4.5

$media.geom
[1] NA

$media.arm
[1] NA
```

2. La mediana es una medida de posición central. Para un vector de datos numéricos x la mediana es el valor que divide los datos ordenados en dos partes iguales. Observa que cuando el número de datos es par no hay ningún valor que cumpla este papel, en ese caso se define la mediana como la media de los dos valores centrales.

   Con esta definición construye una función que calcule la mediana de un vector de datos. La función tendrá por nombre `mediana` y tan sólo un argumento `x` (el vector de datos). El valor que debe devolver es la mediana calculada. Una vez que tengas la función, crea una versión que incluya comprobaciones básicas como que el argumento

proporcionado es numérico, y si tiene datos perdidos, en cuyo caso deberá ignorarlos para el cálculo.

```
> mediana <- function(x) {
+   if (missing(x)|||!is.numeric(x)) stop("Debe proporcionar
                                  un argumento 'x' numérico")
+   x<-x[!is.na(x)]
+   xord<-sort(x)
+   n<-length(x)
+   if (n%%2 == 0) (xord[n/2]+xord[1+ n/2])/2  else xord[ceiling(n/2)]
+ }
```

Finalmente comprueba el resultado de tu función con los siguientes resultados:

```
> mediana(1:5)

[1] 3

> mediana(1:6)

[1] 3.5

> mediana(c(1:6,NA))

[1] 3.5

> mediana("hola")

Error in mediana("hola"): Debe proporcionar
un argumento 'x' numérico

> set.seed(1)
> mediana(runif(20))

[1] 0.6009837
```

3. Los cuantiles (cuartiles, deciles, percentiles, etc.) constituyen medidas de posición en general. Para un vector de datos numéricos x los cuartiles son tres valores (Q_1, Q_2 y Q_3) que dividen los datos en 4 grupos iguales, siendo por tanto Q_2 igual a la mediana.

a) Construye una función con nombre `cuartiles` y tan sólo un argumento `x` (el vector de datos), que calcule los cuartiles. El valor que debe devolver la función es una lista con los tres cuartiles calculados. La función debe incluir comprobaciones básicas como que el argumento proporcionado es numérico, y si tiene datos perdidos, en cuyo caso deberá ignorarlos para el cálculo.

b) En el paquete *stats* tenemos la función `quantile` que permite calcular el cuantil de orden $0 < \alpha < 1$, definido como el valor que deja por debajo el $\alpha * 100\,\%$ de los datos. Inspecciona la ayuda de la función y compara para unos datos el resultado que te da la función que has calculado con el que te daría esta función.

```
> cuartiles <- function(x) {
+   if (missing(x)||!is.numeric(x)) stop("Debe proporcionar
                                  un argumento 'x' numérico")
+   x<-x[!is.na(x)]
+   xord<-sort(x)
+   n<-length(x)

+   ## Q1 ocupa la posición (n+1)/4 (interpolando si es decimal)
+   pos.Q1<-(n+1)/4
+   i<-trunc(pos.Q1)
+   Q1<-x[i]+ (pos.Q1-i)*(x[i+1]-x[i])
+  ## Q2 es la mediana
+   if (n%%2 == 0) Q2<-(xord[n/2]+xord[1+ n/2])/2  else Q2<-xord[ceiling(n/2)]
+   ## Q3 ocupa la posición 3*(n+1)/4 (interpolando si es decimal)
+   pos.Q3<-3*(n+1)/4
+   i<-trunc(pos.Q3)
+   Q3<-x[i]+ (pos.Q3-i)*(x[i+1]-x[i])
+   return(list(Q1=Q1,Q2=Q2,Q3=Q3))
+ }
> cuartiles(1:9)

$Q1
[1] 2.5

$Q2
[1] 5

$Q3
[1] 7.5

> cuartiles(1:10)
```

```
$Q1
[1] 2.75

$Q2
[1] 5.5

$Q3
[1] 8.25

> ## comparando con la función quantile():
> quantile(1:9,c(.25,.5,.75))

25% 50% 75%
  3   5   7

> quantile(1:10,c(.25,.5,.75))

 25%  50%  75%
3.25 5.50 7.75

> ## quantile usa otro método de cálculo de Q1 y Q3
```