Optimizar

Jorge Loría

Optimizar

Optimizar

Siempre se quiere optimizar **algo**, ya sea minimizar riesgos, maximizar utilidades, minimizar costos, entre otras...

Para esto existe una función en R que permite encontrar los puntos máximos y mínimos de una **función** que puede recibir como parámetro. Esta función es muy flexible y se aprovecha de lo que vimos en la clase de funciones en que se pueden mandar funciones como parámetros de otras funciones.

Funciones

Defina la siguiente función con el nombre `rast_1`:

$$f(x) = 10 + (x^2 - 10\cos(2\pi x))$$

Nota: en R existen funciones trigonométricas que se llaman solo por su nombre `cos`, `sin`, etc... Además ya está definida la constante π con el nombre `pi`.

`optim`

¿Cuál era la función para obtener los parámetros de una función?

optim

¿Cuál era la función para obtener los parámetros de una función?

```r
str(formals(optim))
```

```
## Dotted pair list of 9
##  $ par    : symbol
##  $ fn     : symbol
##  $ gr     : NULL
##  $ ...    : symbol
##  $ method : language c("Nelder-Mead", "BFGS", "CG", "L-B
##  $ lower  : language -Inf
##  $ upper  : num Inf
##  $ control: language list()
##  $ hessian: logi FALSE
```

Optimizando

Definiendo la función, se hace:

```r
rast_1 <- function(x)10 + (x^2 - 10 *cos(2*pi*x))
```

Optimizando

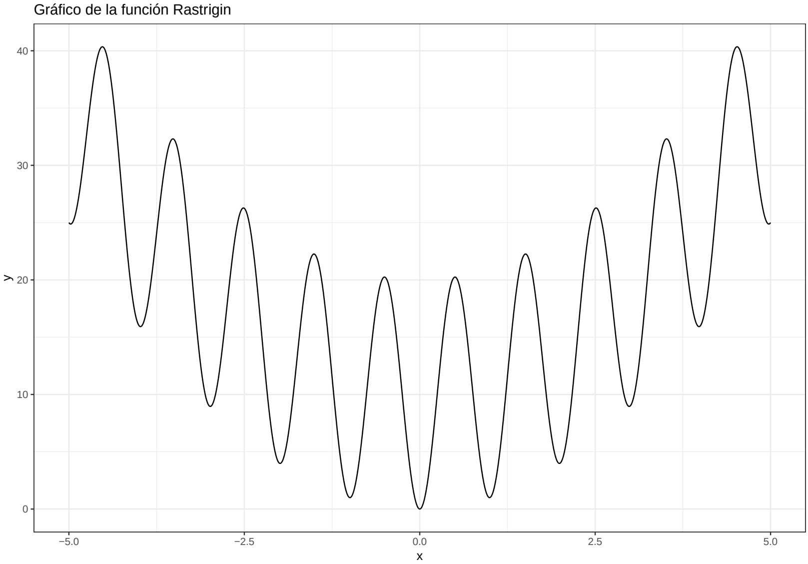
Definiendo la función, se hace:

```
rast_1 <- function(x)10 + (x^2 - 10 *cos(2*pi*x))
```

Ok, grafiquémosla en el intervalo $[-5, 5]$, a ver cómo se ve:

```
df1 <- data.frame(x = seq(-5,5,length.out = 1001)) %>%
  mutate(y = rast_1(x))
```

Gráfico:



Optimizando:

Se obtiene una lista:

```
l_1 <- optim(par = 0.5,fn = rast_1)
```

```
## Warning in optim(par = 0.5, fn = rast_1): one-dimensiona
## use "Brent" or optimize() directly
```

```
str(l_1)
```

```
## List of 5
##  $ par        : num 0.995
##  $ value      : num 0.995
##  $ counts     : Named int [1:2] 32 NA
##   ..- attr(*, "names")= chr [1:2] "function" "gradient"
##  $ convergence: int 0
##  $ message    : NULL
```

Y una advertencia…

Restricciones

Figure 1: Literalmente…

Visualizar:

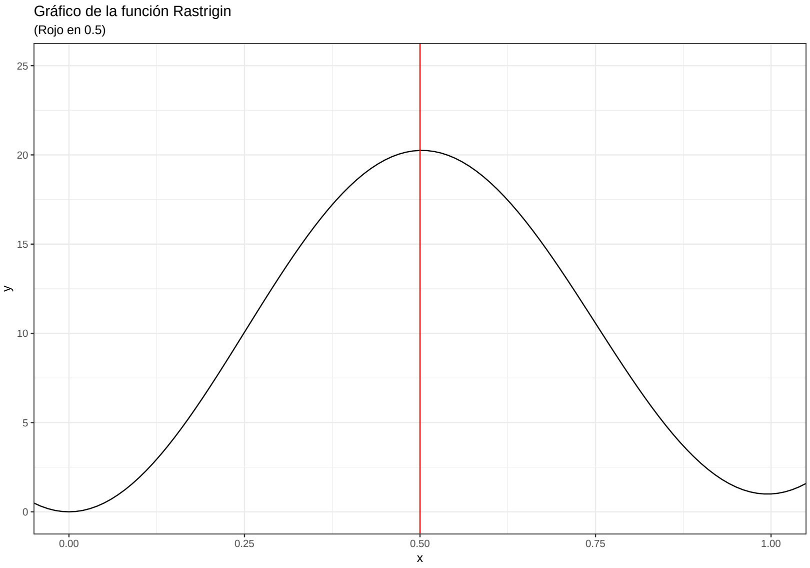


Gráfico de la función Rastrigin
(Rojo en 0.5)
y
25
20
15
10
5
0
0.00
0.25
0.50
0.75
1.00
x

Optimizando con restricciones:

Para optimizar con restricciones usamos el método de Brent. Pero, para usarlo indica que hay que definir bien el intervalo en que se va a realizar la optimización:

```
l_brent <- optim(par = 0.5,fn = rast_1,
                 method = 'Brent',
                 lower = -5,upper = 5)
str(l_brent)
```

```
## List of 5
##  $ par        : num 5.33e-15
##  $ value      : num 0
##  $ counts     : Named logi [1:2] NA NA
##   ..- attr(*, "names")= chr [1:2] "function" "gradient"
##  $ convergence: int 0
##  $ message    : NULL
```

Ejercicio restricciones:

Programe la siguiente función, y defínale el nombre c_2:

$$f(x) = \sqrt{\pi - x}\cos(\pi - x) + \sqrt{x + \pi}\sin(x + \pi)$$

Grafíquela en el intervalo $[-\pi, \pi]$ y optimice en el mismo intervalo.

Solución:

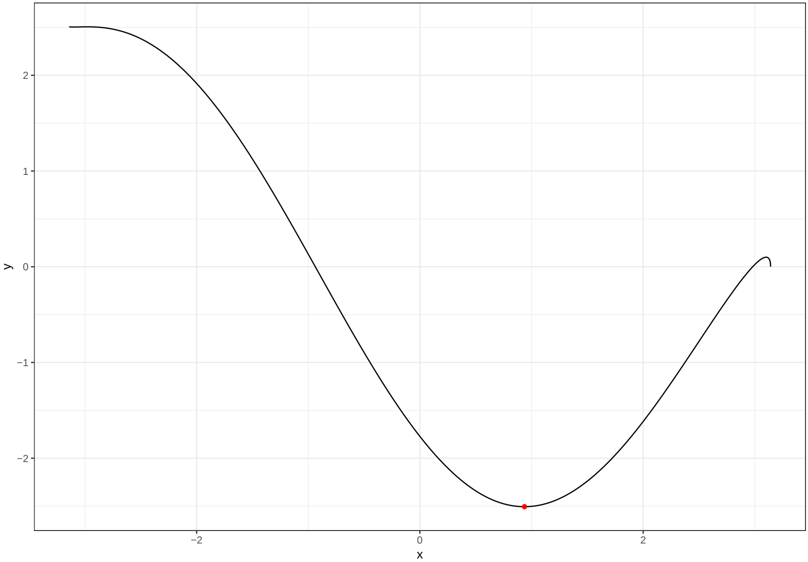
```
c_2 <- function(x){
  sqrt(pi - x)*cos(pi - x) + sqrt(x + pi) * sin(x + pi)
}
df2 <- data.frame(x = seq(-pi,pi,length.out = 10000)) %>%
  mutate(y = c_2(x))

pl_2 <- ggplot(df2) +
  geom_line(aes(x = x,y = y)) +
  theme_bw()

opt_2 <- optim(par = pi/2,c_2,method = 'Brent',
               lower = -pi,upper = pi)

df_2.1 <- data.frame(x = opt_2$par,y = opt_2$value)
pl_2.1 <- pl_2 +
  geom_point(data = df_2.1,aes(x,y),color = 'red')
```

Solución gráfica



Dos parámetros, uno fijo

Programe la siguiente función, con el nombre `c_3`:

$$f(x,y) = 20 + x^2 + y^2 - 10(\cos(2\pi x) + \cos(2\pi y))$$

Dos parámetros, uno fijo

Programe la siguiente función, con el nombre c_3:

$$f(x, y) = 20 + x^2 + y^2 - 10(\cos(2\pi x) + \cos(2\pi y))$$

¿Cómo la podría graficar?

Solución

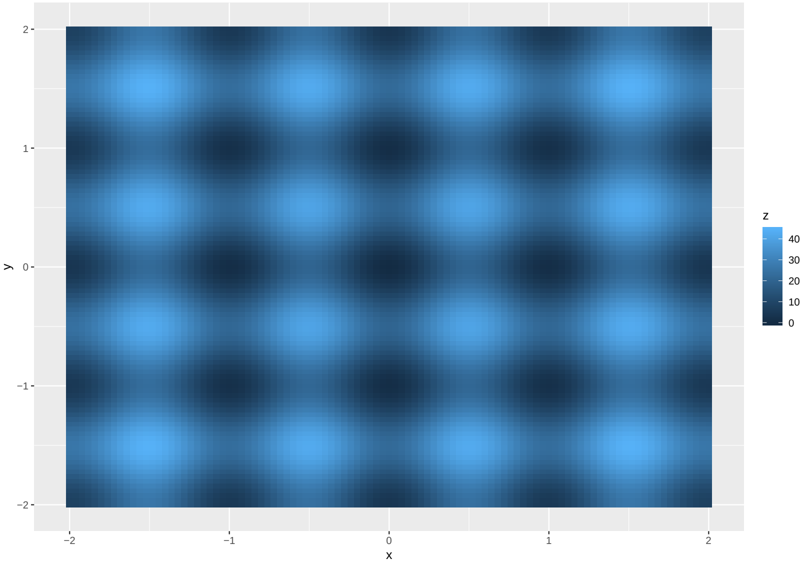
```r
c_3 <- function(x,y){
  20 + x^2 + y^2 - 10*(cos(2*pi*x) + cos(2*pi*y))
}

s3 <- seq(-2,2,length.out = 101)

df3 <- expand.grid(x = s3,y = s3) %>%
  rowwise() %>%
  mutate(z = c_3(x,y)) %>%
  ungroup()

pl_3 <- ggplot(df3,aes(x = x,y = y,fill = z)) +
  geom_raster() # O puede usar geom_point
```

Gráfico



Optimizar fijando una variable ...

Para optimizar fijando una variable, se usa el parámetro ... (son tres puntos. No suspensivos.)

. . .

...

Uno de los parámetros más útiles (que hemos estado usando implícitamente) son los *tres puntitos* o *ellipsis*, que lo que hacen es que pasan variables que no se han nombrado explícitamente, de una función a otra que es llamada por esta y que sí espera recibirlos. Por ejemplo:

```r
fun_1 <- function(...){
  sum(...,na.rm = TRUE)
}
fun_1(1:5,NA,6:15)
```

```
## [1] 120
```

...

Los argumentos, que se pasan con los tres puntitos también pueden ir con nombres:

```
fun_2 <- function(a,b)a - b
fun_3 <- function(...){
  fun_2(...)
}
fun_3(2,3)
fun_3(b = 2,a = 3)
```

...

Los argumentos, que se pasan con los tres puntitos también pueden ir con nombres:

```
fun_2 <- function(a,b)a - b
fun_3 <- function(...){
  fun_2(...)
}
fun_3(2,3)
fun_3(b = 2,a = 3)
```

Y toman los nombres de la función que recibe los 3 puntitos.

. . . Optimizar fijando una variable

```
opt_3 <- optim(par = 0.1,c_3,y = 0.76)
```

```
## Warning in optim(par = 0.1, c_3, y = 0.76): one-dimensio
## use "Brent" or optimize() directly
```

```
str(opt_3)
```

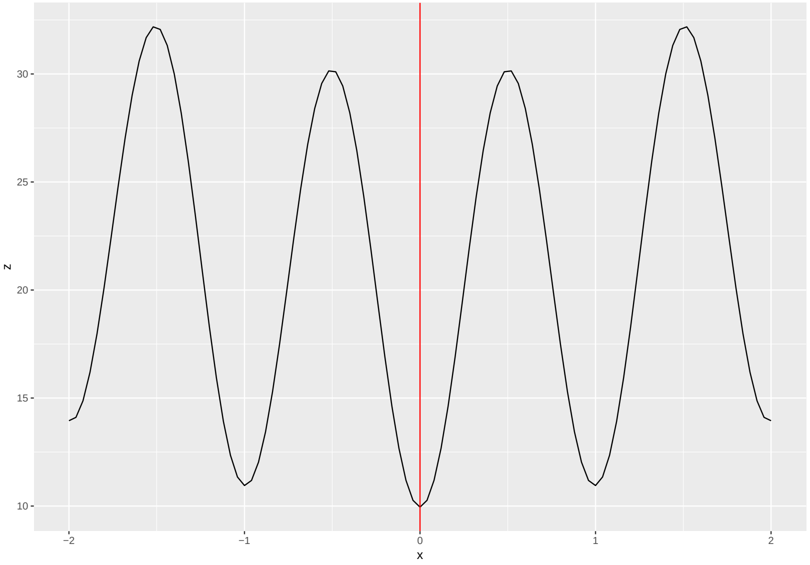
```
## List of 5
##  $ par        : num -8.33e-17
##  $ value      : num 9.95
##  $ counts     : Named int [1:2] 30 NA
##   ..- attr(*, "names")= chr [1:2] "function" "gradient"
##  $ convergence: int 0
##  $ message    : NULL
```

Grafique la función con y fijo, y su respectiva solución.

Solución

```
pl_3.1 <- df3 %>%
  filter(abs(y - 0.76)<0.01) %>%
  ggplot(aes(x,y=z)) +
  geom_line() +
  geom_vline(xintercept = opt_3$par,color = 'red')
```

Gráfico



Función con dos parámetros

Ahora vamos a optimizar la misma función, pero sobre los dos parámetros a la vez:

Función con dos parámetros

Ahora vamos a optimizar la misma función, pero sobre los dos parámetros a la vez:

```
c_4 <- function(xy){c_3(xy[1],xy[2])}
opt_4 <- optim(par = c(0.2,0.2),c_4)
```

Gradiente

También se puede incluir el gradiente para realizar la optimización, este debe ser una función que retorne un vector de `n` variables, con `n`= la cantidad de entradas que recibe la función a optimizar

Recordando c_3, sabemos que la función es simétrica en sus entradas, y que cada una de sus derivadas va a ser igual, pero cambiando la variable.

Gradiente calculado

Es decir:

```
derivada_parcial <- function(xoy){
  2*xoy + 20*pi*sin(2*pi*xoy)
}

grad_4 <- function(xy){
  c(derivada_parcial(xy[1]),
    derivada_parcial(xy[2]))
}
```

Optimizar con gradiente

Para optimizar con gradiente se utiliza el método `'CG'`, o el `'BFGS'`

```
opt_5 <- optim(par =c( 0.2,0.2),fn = c_4,
                gr = grad_4,method = 'CG')
str(opt_5)
```

```
## List of 5
##  $ par        : num [1:2] 1.4e-09 1.4e-09
##  $ value      : num 0
##  $ counts     : Named int [1:2] 65 17
##   ..- attr(*, "names")= chr [1:2] "function" "gradient"
##  $ convergence: int 0
##  $ message    : NULL
```

Optimizar con gradiente

Para optimizar con gradiente se utiliza el método `'CG'`, o el `'BFGS'`

```
opt_5 <- optim(par =c( 0.2,0.2),fn = c_4,
                gr = grad_4,method = 'CG')
str(opt_5)
```

```
## List of 5
##  $ par        : num [1:2] 1.4e-09 1.4e-09
##  $ value      : num 0
##  $ counts     : Named int [1:2] 65 17
##   ..- attr(*, "names")= chr [1:2] "function" "gradient"
##  $ convergence: int 0
##  $ message    : NULL
```

Repita el ejercicio anterior, pero usando el método `'BFGS'`.

Optimizar con gradiente

Para optimizar con gradiente se utiliza el método `'CG'`, o el `'BFGS'`

```
opt_5 <- optim(par =c( 0.2,0.2),fn = c_4,
               gr = grad_4,method = 'CG')
str(opt_5)
```

```
## List of 5
##  $ par        : num [1:2] 1.4e-09 1.4e-09
##  $ value      : num 0
##  $ counts     : Named int [1:2] 65 17
##   ..- attr(*, "names")= chr [1:2] "function" "gradient"
##  $ convergence: int 0
##  $ message    : NULL
```

Repita el ejercicio anterior, pero usando el método `'BFGS'`.¿Qué pasa si pone `par = c(1,1)`?

Problema

Intente optimizar usando el método Brent, con restricciones para encontrar un mínimo global.

Problema

Intente optimizar usando el método Brent, con restricciones para encontrar un mínimo global.

Da un error, pues este método es para una sola dimensión.

Método 'L-BFGS-B'

Al usar este método, se realiza la optimización en una “caja” y se obtiene el resultado deseado:

```
opt_6 <- optim(par =c( 1,1),fn = c_4,
      gr = grad_4,
      method = 'L-BFGS-B',
      lower = c(-5,-5),upper = c(5,5))
str(opt_6)
```

```
## List of 5
##  $ par        : num [1:2] -9.91e-10 -9.91e-10
##  $ value      : num 0
##  $ counts     : Named int [1:2] 46 46
##   ..- attr(*, "names")= chr [1:2] "function" "gradient"
##  $ convergence: int 52
##  $ message    : chr "ERROR: ABNORMAL_TERMINATION_IN_LNSE
```

Ejercicio

Realice el llamado anterior pero sin llamar al gradiente.

En “general”

Se puede tener una función de tantas variables como se quiera:

```
flb <- function(x){
  p <- length(x)
  sum(c(1, rep(4, p-1)) * (x - c(1, x[-p])^2)^2)
}
```

Y realizar la optimización para la cantidad de variables que se requiera:

```
opt_gen <- optim(rep(3, 25), fn = flb,
                 gr = NULL, method = "L-BFGS-B",
                 lower = rep(1, 25),
                 upper = rep(4, 25))
```

Resultado

```
## $par
##  [1] 1.000000 1.000000 1.000000 1.000000 1.000000 1.0000
##  [8] 1.000000 1.000000 1.000000 1.000000 1.000000 1.0000
## [15] 1.000850 1.002308 1.004936 1.010055 1.020297 1.0410
## [22] 1.174671 1.379857 1.904017 3.625275
##
## $value
## [1] 4.90405e-06
##
## $counts
## function gradient
##       64       64
##
## $convergence
## [1] 52
##
## $message
## [1] "ERROR: ABNORMAL TERMINATION IN LNSRCH"
```

Ejercicio

Repita el ejercicio anterior, pero usando 3, 4 y 5 variables.

Ejercicio

Repita el ejercicio anterior, pero usando 3, 4 y 5 variables.

Programe una función que reciba un entero positivo, y que realice la minimización anterior para esa cantidad de variables.

Restricciones lineales

Hasta el momento nos hemos limitado a usar restricciones donde las variables sobre las que se maximizan pueden no "depender" entre sí. Sin embargo, para el caso general, se quiere resolver el siguiente problema:

$$\min_{x \in \mathbb{R}^n} f(x)$$

$$\text{tal que:}$$

$$g_i(x) \geq c_i$$

Donde $g_i(x)$ son funciones lineales.

Restricciones lineales

Hasta el momento nos hemos limitado a usar restricciones donde las variables sobre las que se maximizan pueden no "depender" entre sí. Sin embargo, para el caso general, se quiere resolver el siguiente problema:

$$\min_{x \in \mathbb{R}^n} f(x)$$

$$\text{tal que:}$$

$$g_i(x) \geq c_i$$

Donde $g_i(x)$ son funciones lineales. Las restricciones anteriores se pueden expresar de la manera:

$$Ax \geq c$$

Donde A es una matriz de $m \times n$, $c \in \mathbb{R}^m$, y la desigualdad es entrada por entrada.

Ejemplo restricciones

Programe la función:

$$f(x) = \frac{1}{2}\sum_{i=1}^{n}(x_i^4 - 16x_i^2 + 5x_i)$$

Ejemplo restricciones

Programe la función:

$$f(x) = \frac{1}{2}\sum_{i=1}^{n}(x_i^4 - 16x_i^2 + 5x_i)$$

```
restr_lin1 <- function(x){
  1/2*sum(x^4 - 16*x^2+5*x)
}
```

Optimice

Si la función anterior está definida para $x_i \in [-5, 5]$, optimícela para $n = 5$, comience en el origen.

Optimice

Si la función anterior está definida para $x_i \in [-5, 5]$, optimícela para $n = 5$, comience en el origen.

Para n dimensiones, se minimiza en $x_i = -2.903534$:

```
optim(par = rep(0,5),fn = restr_lin1,method = 'L-BFGS-B',
      lower = rep(-5,5),upper = rep(5,5))
```

```
## $par
## [1] -2.903534 -2.903534 -2.903534 -2.903534 -2.903534
##
## $value
## [1] -195.8308
##
## $counts
## function gradient
##        8        8
##
## $convergence
```

Restricción linear:

Si nos quedamos en dos dimensiones. Y definimos que $x + y >= 2$
la función `optim` ya no nos funciona :(

Restricción linear:

Si nos quedamos en dos dimensiones. Y definimos que $x + y >= 2$ la función `optim` ya no nos funciona :(Entonces, vamos a usar la función: `constrOptim`, que funciona de una forma muy similar, pero hay que agregarle la matriz A que mencionamos anteriormente, y el vector de restricciones.

Restricción linear:

Si nos quedamos en dos dimensiones. Y definimos que $x + y >= 2$ la función `optim` ya no nos funciona :(Entonces, vamos a usar la función: `constrOptim`, que funciona de una forma muy similar, pero hay que agregarle la matriz A que mencionamos anteriormente, y el vector de restricciones.

```
mat_A <- matrix(data = c(1,1),nrow = 1)
const_c <- 2
constrOptim(theta = c(1.5,1.5),f = restr_lin1,grad = NULL,
            ui = mat_A,ci = const_c)
```

```
## $par
## [1] 2.746868 2.746750
##
## $value
## [1] -50.05889
##
## $counts
```

¿Qué hay mal en lo anterior?

¿Qué hay mal en lo anterior?

No consideramos que $x, y \in [-5, 5]$. Entonces tal vez nos hubiéramos podido salir... ¿Cómo podemos hacer para incorporar eso en `constrOptim`?

¿Qué hay mal en lo anterior?

No consideramos que $x, y \in [-5, 5]$. Entonces tal vez nos hubiéramos podido salir... ¿Cómo podemos hacer para incorporar eso en `constrOptim`? Note que podemos incluir las restricciones: $x_i \geq -5$ y $x_i \leq 5$.

¿Qué hay mal en lo anterior?

No consideramos que $x, y \in [-5, 5]$. Entonces tal vez nos hubiéramos podido salir... ¿Cómo podemos hacer para incorporar eso en `constrOptim`? Note que podemos incluir las restricciones: $x_i \geq -5$ y $x_i \leq 5$. Para esto hacemos:

¿Qué hay mal en lo anterior?

No consideramos que $x, y \in [-5, 5]$. Entonces tal vez nos hubiéramos podido salir... ¿Cómo podemos hacer para incorporar eso en constrOptim? Note que podemos incluir las restricciones: $x_i \geq -5$ y $x_i \leq 5$. Para esto hacemos:

```r
a1 <- diag(x = 1,nrow = 2)
a2 <- -a1
mat_A2 <- rbind(a1,a2,mat_A)
const_c2 <- c(-5,-5,-5,-5,const_c)
constrOptim(theta = c(1.5,1.5),f = restr_lin1,grad = NULL,
            ui = mat_A2,ci = const_c2)
```

```
## $par
## [1] 2.746792 2.746864
##
## $value
## [1] -50.05889
##
```

Otra restricción!

Agreguemos que $x - y \geq 1$:

Otra restricción!

Agreguemos que $x - y \geq 1$:

```
mat_A3 <- rbind(mat_A2,c(1,-1))
const_c3 <- c(const_c2,1)
constrOptim(theta = c(2.6,1.5),f = restr_lin1,grad = NULL,
            ui = mat_A3,ci = const_c3)
```

```
## $par
## [1] 3.101839 2.101839
##
## $value
## [1] -43.25999
##
## $counts
## function gradient
##      157       NA
##
## $convergence
```

Ejercicio

Optimice `restr_lin1` con $n = 4$, tal que: $x_i \geq 1 + x_{i+1}$, siempre en la región original donde se definió.

Calcule el gradiente de la función, con n dimensiones, e inclúyalo en la llamada.

Encuentre el máximo en $n = 2$ tal que: $x + 1 \leq y$, $y \geq \frac{-3}{5}x - 3$, $y \geq 15x - 55$: