1. (1 punto) Obtén el máximo de la función `f <- function (x) 50*sin(0.3*x)*sin(1.3*x^2) - 0.000002*x^4 - 0.6*x+80` entre `-100` y `100`.

```
> f <- function (x) 50*sin(0.3*x)*sin(1.3*x^2) - 0.000002*x^4 - 0.6*x+80
> plot(f,-100,100) # aspecto general
> plot(f,-100,100,n=1e4) # más puntos porque la función es irregular
> plot(f,-50,-30,n=1e4) # ampliar cerca del máximo
> plot(f,-50,-30,n=1e4,ylim=c(140,150)) # intentar discernir entre dos cumbres
> plot(f,-50,-30,n=1e4,ylim=c(148,149))
> plot(f,-50,-30,n=1e5,ylim=c(148,149)) # más puntos, por si acaso
> plot(f,-50,-30,n=1e6,ylim=c(148,149)) # ídem
> ## el máximo está entre -48 y -46
> plot(f,-48,-46,n=1e6,ylim=c(148.4,148.5))
> plot(f,-47.1,-47.2,n=1e7,ylim=c(148.4,148.5))
> ## máximo en -47.14 más o menos, donde f = 148.41
> plot(f,-47.1,-47.2,n=1e5) # hay dos picos muy parecidos
> optim(-47.14,function(x)-f(x))
$par
[1] -47.13887

$value
[1] -148.4076

$counts
function gradient
      50       NA

$convergence
[1] 0

$message
NULL

Warning message:
In optim(-47.14, function(x) -f(x)) :
  one-dimensional optimization by Nelder-Mead is unreliable:
use "Brent" or optimize() directly
> optim(-47.19,function(x)-f(x))
$par
[1] -47.1901

$value
[1] -148.386

$counts
function gradient
      52       NA

$convergence
[1] 0
```

$message
NULL

Warning message:
In optim(-47.19, function(x) -f(x)) :
  one-dimensional optimization by Nelder-Mead is unreliable:
use "Brent" or optimize() directly
> ## El máximo está en -47.13887 donde f vale 148.4076

2. (1 punto) Dados los datos:

```
x1 <- mtcars$wt
x2 <- mtcars$hp
y <- mtcars$qsec
```

encuentra los valores de los escalares `a`, `b` y `c` que minimizan la expresión

```
sum((y-(a+b*x1+c*x2))^2)
```

```
> f <- function (x) {a=x[1];b=x[2];c=x[3];sum((y-(a+b*x1+c*x2))^2)}
> optim (rep(0,3), f)
$par
[1] 18.81857042  0.94393348 -0.02731861

$value
[1] 34.44513

$counts
function gradient
     274       NA

$convergence
[1] 0

$message
NULL

> ## La minimización de ese error cuadrático es equivalente
> ## a una regresión lineal:
> summary (lm (y ~ x1 + x2))

Call:
lm(formula = y ~ x1 + x2)

Residuals:
    Min      1Q  Median      3Q     Max
-1.8283 -0.4055 -0.1464  0.3519  3.7030

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 18.825585   0.671867  28.020  < 2e-16 ***
x1           0.941532   0.265897   3.541  0.00137 **
x2          -0.027310   0.003795  -7.197 6.36e-08 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 1.09 on 29 degrees of freedom
Multiple R-squared:  0.652,   Adjusted R-squared:  0.628
F-statistic: 27.17 on 2 and 29 DF,  p-value: 2.251e-07
```

(1 punto) Un fabricante de coches produce dos modelos: uno pequeño (10.000 €) y uno grande (16.000 €). Tiene tanto éxito que vende todo lo que produce, pero su producción requiere dos materias primas actualmente escasas: caucho y acero. Un coche pequeño requiere una unidad de caucho y una de acero y, uno grande, una de caucho y dos de acero. El fabricante dispone de 400 unidades de caucho y 700 de acero. ¿Cuántos coches de cada modelo debe fabricar para maximizar los € de las ventas?

Sea “p” el número de coches pequeños fabricados y sea “g” el número de grandes.
Se trata de maximizar 10.000·p + 16.000·g sujeto a: p + g < 400; p + 2g < 700. En R:

```
> library("lpSolve")
> lp ("max", c(10000,16000), rbind(c(1,1),c(1,2)), "<", c(400,700)) -> sol
> sol$solution
[1] 100 300
> sol
Success: the objective function is 5800000
> ## Debe fabricar 100 coches pequeños y 300 grandes para obtener 5,8 M€.
```

(2 puntos) A partir de **http://bellman.ciencias.uniovi.es/~carleos/master/manadine/curso2/AnalisisDatos2/apuntes/optimiza/recocido-nreinas.odp** comenta:

- La solución obtenida ejecutando el listado tal cual está.

```
$par
[1] 7 2 4 1 8 5 3 6

$value
[1] 0

$counts
function gradient
   1000       NA

$convergence
[1] 0

$message
NULL
```

Lo anterior es la salida producida al ejecutar el “optim” de dicho fichero.
La parte de “convergence 0” no tiene relevancia (y tampoco “message NULL”) porque con el método SANN siempre da lo mismo. Según “?optim”:

> convergence: An integer code. ‘0’ indicates successful completion
> (which is always the case for ‘"SANN"’ and ‘"Brent"’).

La parte “counts” indica que se hicieron 1000 evaluaciones de la función “calculaAdaptacion” y ninguna (NA) de la función gradiente. El “gradiente” en este caso sería la función “mutar”, pero parece que “optim” no la considera un gradiente al uso y no contabiliza sus evaluaciones. Se realizan 1000 por la opción “control=list(maxit=1000)” porque por omisión se harían 10.000 evaluaciones.

La parte “value 0” indica que se ha llegado a una solución óptima del problema de la 8 reinas, es decir, una disposición de las damas en el tablero con cero colisiones.
La parte “par 7 2 4 1 8 5 3 6” indica los valores del “parámetro” en la solución óptima, es decir, que se pueden conseguir 0 colisiones ubicando la primera reina en la primera fila y la séptima columna; la segunda reina, en la segunda fila y la segunda columna; la tercera reina, en la tercera fila y la cuarta columna; etc.

- Qué consecuencias tiene variar `niter`.

“niter” es el número de iteraciones del recocido. Como se ve en la salida siguiente,

- al aumentar niter aumenta el tiempo de ejecución
- al aumentar niter, siguen obteniéndose soluciones óptimas (distintas)
- al disminuir niter, puede no obtenerse una solución óptima (value > 0)

```
> g <- function (nRei, niter)
+    optim (sample (1:nRei),
+        calculaAdaptacion,
+        gr = mutar,
+        method = "SANN",
+        control = list (maxit=niter))
> g(8,1000)
$par
[1] 4 2 5 8 6 1 3 7

$value
[1] 0

$counts
function gradient
    1000       NA

$convergence
[1] 0

$message
NULL

> g(8,10000)
$par
[1] 3 1 7 5 8 2 4 6

$value
[1] 0

$counts
function gradient
   10000       NA

$convergence
[1] 0

$message
NULL
```

```
> g(8,100000)
$par
[1] 4 6 8 2 7 1 3 5

$value
[1] 0

$counts
function gradient
  100000       NA

$convergence
[1] 0

$message
NULL

> g(8,100)
$par
[1] 6 3 7 2 8 5 1 4

$value
[1] 0

$counts
function gradient
     100       NA

$convergence
[1] 0

$message
NULL

> g(8,10)
$par
[1] 7 4 3 1 8 5 2 6

$value
[1] 1

$counts
function gradient
      10       NA

$convergence
[1] 0

$message
NULL
```

```
> system.time(g(8,100000))
   user  system elapsed
  1.419   0.000   1.418
> system.time(g(8,1000000))
   user  system elapsed
 14.597   0.000  14.599
```

- Qué consecuencias tiene variar `nRei`.

"nRei" es el tamaño (número de filas o de columnas) del tablero, es decir, la magnitud del problema que se quiere resolver. Con niter=1000, al disminuir nRei se obtiene una solución óptima siempre (siempre que exista: https://es.wikipedia.org/wiki/Problema_de_las_ocho_reinas#N%C3%BAmero_de_soluciones):

> g(8,1000)$value
[1] 0
> g(7,1000)$value
[1] 0
> g(6,1000)$value
[1] 0
> g(5,1000)$value
[1] 0
> g(4,1000)$value
[1] 0
> g(3,1000)$value
[1] 1
> g(2,1000)$value
[1] 1

Al aumentar nRei, puede ser necesario incrementar también niter para obtener solución óptima:

> g(8,1000)$value
[1] 0
> g(10,1000)$value
[1] 0
> g(20,1000)$value
[1] 2
> g(20,10000)$value
[1] 2
> g(20,100000)$value
[1] 0

- Cómo se podría usar `parallel::mclapply` para obtener dos y tres soluciones en el mismo tiempo que emplea el listado original para obtener una.

> ejecutable <- function () g(20,100000)[c("par","value")]
> ejecutable() # devuelve una solución óptima al problema de 20 reinas
$par
 [1] 3 10 13 20 8 12 4 17 5 9 18 16 14 19 15 1 6 2 7 11

$value

```
[1] 0

> system.time(ejecutable()) # tarda unos cinco segundos en mi ordenata
   user  system elapsed
  4.807   0.000   4.807
> ## tarda unos diez segundos si lo repito 2 veces con lapply
> system.time(resultado<-lapply(1:2,function(x)ejecutable()))
   user  system elapsed
  9.664   0.000   9.665
> library(parallel)
> ## tarda unos cinco segundos si uso mclapply, o sea,
> ## que ejecuta ambos procesos en paralelo
> system.time(resultado<-mclapply(1:2,function(x)ejecutable()))
   user  system elapsed
  0.007   0.007   5.056
> ## si pido 3 en paralelo, tarda diez segundos, porque hace dos en paralelo
> ## y el tercero va después ya que, por omisión, mc.cores=2
> system.time(resultado<-mclapply(1:3,function(x)ejecutable()))
   user  system elapsed
  5.026   0.049   9.875
> ## indicando mc.cores=3 se vuelve a tardar sobre cinco segundos
> system.time(resultado<-mclapply(1:3,function(x)ejecutable(),mc.cores=3))
   user  system elapsed
  5.159   0.044   5.200
> ## para que funcione lo anterior necesito un ordenata con al menos 3 cores
> detectCores()
[1] 20
> ## el resultado es una lista con tres soluciones óptimas (value 0)
> resultado
[[1]]
[[1]]$par
 [1] 19  2 15  8  4  9 18  3  1  7 12 17 20 16 13  6 10 14  5 11

[[1]]$value
[1] 0


[[2]]
[[2]]$par
 [1] 16  8 13  2  4 14  1  3 10 19 11 15 20 18  6 12  9  7  5 17

[[2]]$value
[1] 0


[[3]]
[[3]]$par
 [1] 13 17  9  7  5  3  6 12 18 11  4 19 15 10 20  1  8 16 14  2

[[3]]$value
[1] 0
```