Laboratorio: Optimización convexa

Jose Ameijeiras Alonso

En esta práctica revisaremos los principales conceptos relacionados con la optimización convexa y estudiaremos el comportamiento de uno de los métodos más habituales en la resolución de dichos problemas.

1 Un problema de optimización convexa

Considera la función $f : \mathbb{R}^2 \to \mathbb{R}$ dada por:

$$f(x, y) = \frac{1}{2}(x^2 + \gamma y^2) \tag{1}$$

siendo $\gamma > 0$. Queremos programar un metodo iterativo para minimizar la función (el óptimo se alcanza en el punto $(0, 0)$). El problema

$$\min_{(x,y)\in\mathbb{R}^2} f(x, y)$$

es un problema de optimización convexa. ¿Por qué?

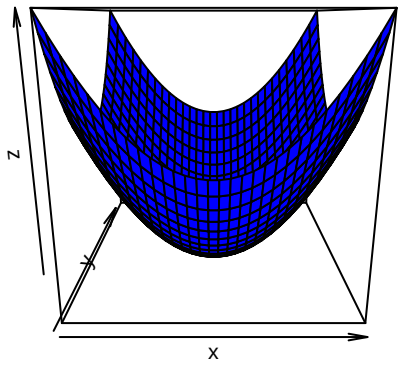
1.1 Representaciones gráficas de superficies en $\mathbb{R}^3$

La función `persp` se utiliza para la representación de superficies en $\mathbb{R}^3$. Podremos por lo tanto utilizarla para representar funciones $f : \mathbb{R}^2 \to \mathbb{R}$ como la que hemos definido anteriormente.

Para su uso, debemos dar como argumentos de entrada dos vectores, `x` e `y`, con los valores que definirán el grid de puntos sobre los que se evaluará la función f que define la superficie a representar.

A continuación, se muestra el código que representa la función dada en (1) para $\gamma = 1$. Necesitaremos definir la función `f` que queremos representar. Puedes consultar al final de este documento algunos aspectos básicos de la programación en R.

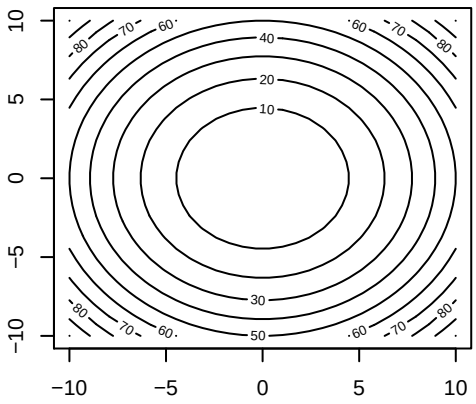
```
> x <- seq(-10, 10, length = 30)
> y <- x
> f <- function(x, y) {0.5 * (x^2 + y^2)}
> z <- outer(x, y, f)
> persp(x, y, z, col = 4)
```



Son muchas las opciones de la función. Por ejemplo, los argumentos `phi` y `theta` se usan para rotar el ángulo de visión de la superfície. Puedes consultarlas todas los argumentos de la función con `help(persp)`.

La función `contour` crea un gráfico de contornos. Su uso es similar al de la función `persp`.

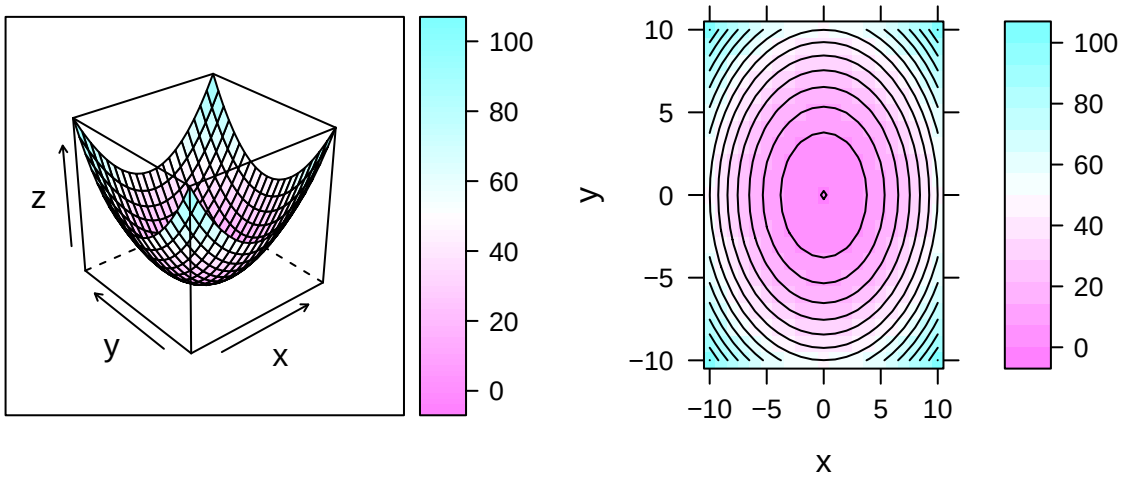
```
> contour(x, y, z)
```



De nuevo, existen muchas opciones para esta función, como por ejemplo, el número de niveles para el trazado de los contornos.

Otra opción para realizar este tipo de representaciones es utilizar la librería `lattice`. Esta librería incluye las funciones `wireframe` y `levelplot`, que proporcionan gráficos similares a los obtenidos con `persp` y `contour`, pero que además nos permiten representar la superfície con un gradiente de color. La llamada a la función es distinta, ya que como argumentos de entrada se requiere una fórmula que relacione `z` con `x` e `y`.

```
> library(lattice)
> g <- expand.grid(x = -10:10, y = -10:10)
> g$z <- 0.5 * (g$x^2 + g$y^2)
> wireframe(z ~ x * y, data = g, drape = TRUE)
> levelplot(z ~ x * y, g, contour = TRUE)
```



2 Algoritmo de descenso de gradiente

Programa el algoritmo de descenso de gradiente para minimizar una funcion objetivo $f : \mathbb{R}^2 \to \mathbb{R}$ del tipo

$$f(x, y) = \frac{1}{2}(x^2 + \gamma y^2)$$

con $\gamma > 0$. Recuerda que el algoritmo es:

Algoritmo: Método de descenso de gradiente
Dado un punto inicial $x \in$ dom(f)
repite
1. $\Delta x = -\nabla f(x)$
2. Line search. Elige un paso $t > 0$
3. Actualiza. $x := x + t\Delta x$
hasta que se cumpla el criterio de parada

Inicia el algoritmo en el punto $x^{(0)} = (\gamma, 1)^t$. Fija un número máximo de iteraciones N y el valor de η para el criterio de parada del algoritmo.

Para cada iteración k, deberás:

- Seleccionar el paso $t^{(k)} > 0$
- Evaluar el gradiente de la función objetivo en $x^{(k-1)}$
- Calcular el nuevo punto $x^{(k)}$

2.1 Elección del paso

1. En primer lugar, programa el algoritmo utilizando un paso t fijo. ¿Cómo afecta la elección de t a la convergencia?

2. Programa el algoritmo utilizando como paso de cada iteración el obtenido por el método de búsqueda exacta (exact line search). En ese caso, recuerda que:

$$t = \arg\min_{s \geq 0} f(x + s\Delta x)$$

   Utilizando este método responde a las siguientes preguntas:

   - ¿Cuántas iteraciones necesita el método para converger cuando $\gamma = 1$?
   - Analiza como es la velocidad de convergencia cuando $1/3 < \gamma < 3$, $\gamma \gg 1$ y $\gamma \ll 1$.

3. Programa el algoritmo utilizando como paso de cada iteración el obtenido por el método backtracking line search. En ese caso, recuerda que:

Algoritmo: Backtracking line search
dada una dirección descendente Δx para f en el valor $x \in \text{dom}(f)$, $\alpha \in (0, 0.5]$, $\beta \in (0, 1)$ $t := 1$ **mientras que** $f(x + t\Delta x) > f(x) + \alpha t \nabla f(x)^t \Delta x$, $t := \beta t$

   - ¿Cómo afecta la elección de $\alpha \in (0, 0.5]$ y $\beta \in (0, 1)$?

2.2 Demostración del algoritmo de descenso de gradiente

La librería `animation` incluye una función `grad.desc` que nos permite visualizar gráficamente el método de descenso de gradiente con paso t fijo.

A continuación se muestra un ejemplo de ejecución para la función $f(x, y) = \frac{1}{2}(x^2 + 2y^2)$, punto incial $(2, 0)^t$ y $t = 0.2$.

```
> library(animation)
> oopt = ani.options(interval = 0.3, nmax = ifelse(interactive(), 50, 2))
> xx = grad.desc(FUN = function(x, y) 0.5 * (x^2 + 2 * y^2), init = c(2, 1), gamma = 0.2)
> xx$par  # solución
> xx$persp(col = "lightblue", phi = 30)  # Gráfico de superfície
```

3 Programación en R

3.1 Funciones

El lenguaje R permite al usuario definir sus propias funciones. El esquema de definición de una función es muy sencillo.

```
nombre_func <- function(arg_1, arg_2, ...) {
    expr
}
```

Si queremos que por ejemplo `arg_2` tome por defecto el valor `val` bastaría escribir

```
nombre_func <- function(arg_1, arg_2 = val, ...) {
    expr
}
```

Todo argumento que no tenga un valor por defecto será obligatorio. Una función tomará el valor de la expresión, es decir, el valor del último comando ejecutado en `expr` (que no tiene por qué ser el último comando de `expr`, la sentencia `return` puede ser utilizada para devolver un valor por el medio de una expresión y terminar la ejecución de la función).

Las funciones en R pueden ser recursivas; una función puede llamarse a si misma.

En una función en R se distinguen los siguientes tipos de variables:

- Parámetros formales: Son los argumentos de la función, cualquier modificación que se haga sobre los mismos se pierde al salir de la función.
- Variables locales: Aparecen en la función al serles asignado algún valor, desaparecen al salir de la función.
- Variables libres: Son variables que aparecen en una función sin que sean parámetros ni se les haya asignado previamente ningún valor. R busca de dentro de la función hacia fuera (podemos tener funciones anidadas) hasta que encuentra alguna variable que con su nombre. Esto se conoce como "alcance lexicográfico"

3.2 Sentencias condicionales

Como en todos los lenguajes de programación, esto se hace con las sentencias `if`. Estas sentencias en R tienen la siguiente sintaxis:

```
if (expr_1) {expr_2}
else if (expr_3) {expr_4}
else {expr_5}
```

donde `expr_1` y `expr_3` deben devolver un valor lógico. En R también se pueden construir sentencias `if` anidadas

Por ejemplo

```
> if (10>5) {cat("Hola")} else cat ("Adiós")
```

```
## Hola
```

3.3 Bucles

R admite los bucles `for`, `repeat` y `while`, su sintaxis es la siguiente:

- `for`: La sintaxis es : `for (name in expr1) expr2`. Por ejemplo, un bucle que mueva un índice `i` desde 1 hasta `n` se escribiría como: `for (i in 1:n) {expr}`.
- `repeat`: La sintaxis es `repeat expr`. En este caso hay que asegurarse de que en algún momento de `expr` se llega a un `break/return` que nos saca del bucle.
- `while`: La sintaxis es `while(cond) expr`.