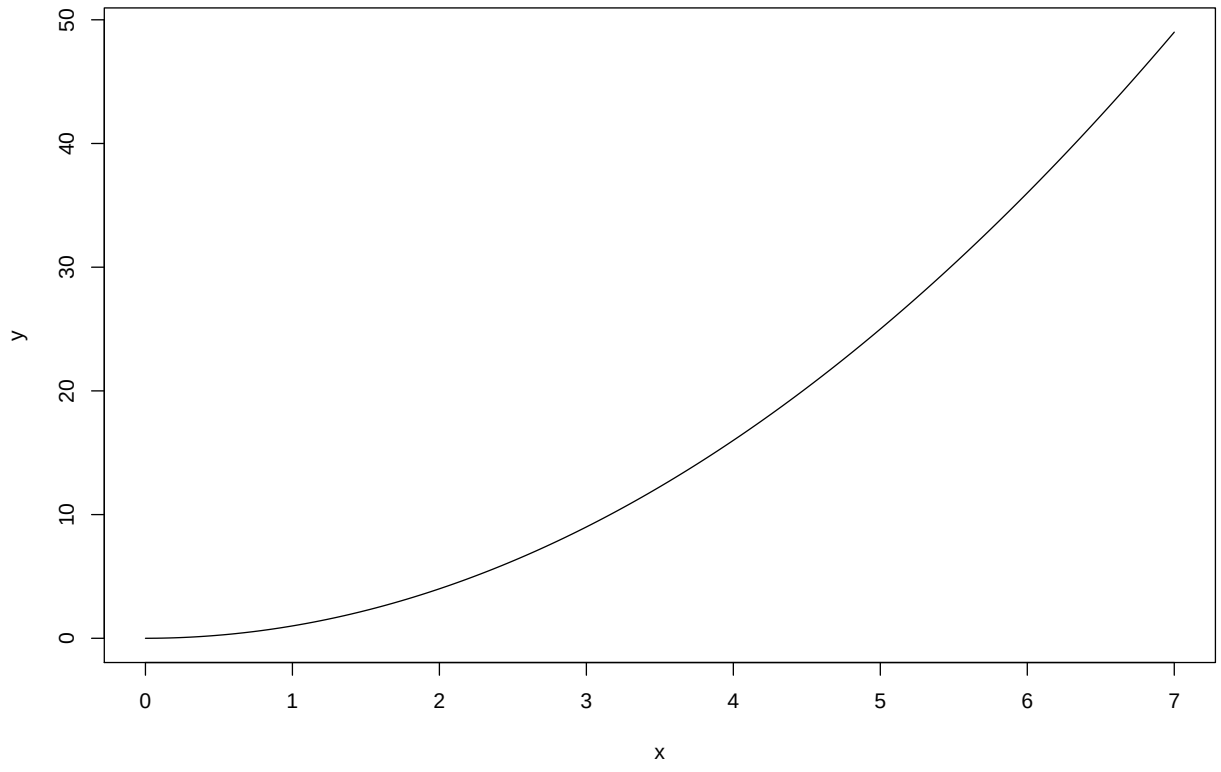
Funciones. Estructuras condicionales e iterativas

Mariela Sued, Jemina García y Ana M. Bianco

17 abril 2020

Hicimos este gráfico continuo

```
x<-seq(0,7,by=0.01) # armo la grilla
y<- x*x             # elevo cada elemeto al cuadrado
plot(x,y,type="l")  # ploteo
```



Una función más complicada

Grafiquemos $f(x) = 15x^4 - 32x^3 + 23x^2 - 8x + 17$ para x en $[-1, 1]$

¿Vamos a calcular punto por punto?

- $f(-1) =$
- $f(-0.8) =$
- $f(-0.5) =$

Function

Una opción es programar la función f y luego usar plot.

Podemos programar la función f usando el comando function.

```
f.funcion<-function(mongo)
{
  salida<-15*mongo^4-32*mongo^3+23*mongo^2-
    8*mongo+17
  return(salida)
}

#llamo a la funcion
f.funcion(-0.8)
```

```
## [1] 60.648
```

Function

Es un tipo de construcción que permite encerrar un pedacito de codigo, para realizar una misma tarea en diferentes situaciones

NombreFuncion<-function(argumentos)

{

cuerpo funcion

return( )

}

*Llamo a la función

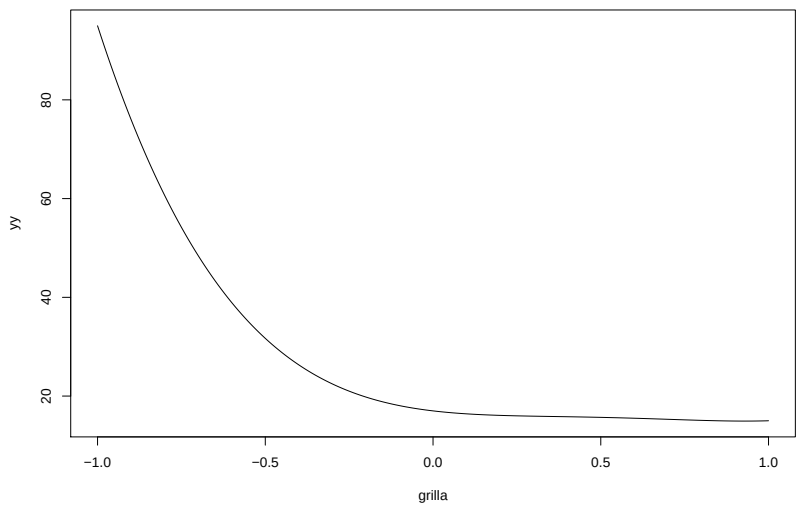
NombreFuncion(valores para los argumentos)

Graficamos

```
grilla<-seq(-1,1,by=0.01)

#haciendo uso de las bondades de R! comandos apply
yy<-lapply(grilla,f.funcion) # aplico la f.funcion a una lista

plot(grilla,yy, type="l")
```



¿Cómo haríamos si quisiéramos graficar la siguiente función?

$$g(x) = \begin{cases} x^2 & \text{si } x \leq 1 \\ 2x - 1 & \text{si } x > 1 \end{cases}$$

Grafiquemos a g en en $[-13, 14]$

Primero la programamos: observemos que hay una condición.

¿Cómo hacemos?

Estructura de control: if

Permite ejecutar una serie de instrucciones si se cumple cierta condición.

```
if ( test_expression) {

statement

}
```

Estructura de control: if

```
if ( test_expression1) {
statement1
} else if ( test_expression2) {
statement2
} else {
statement3
}
```

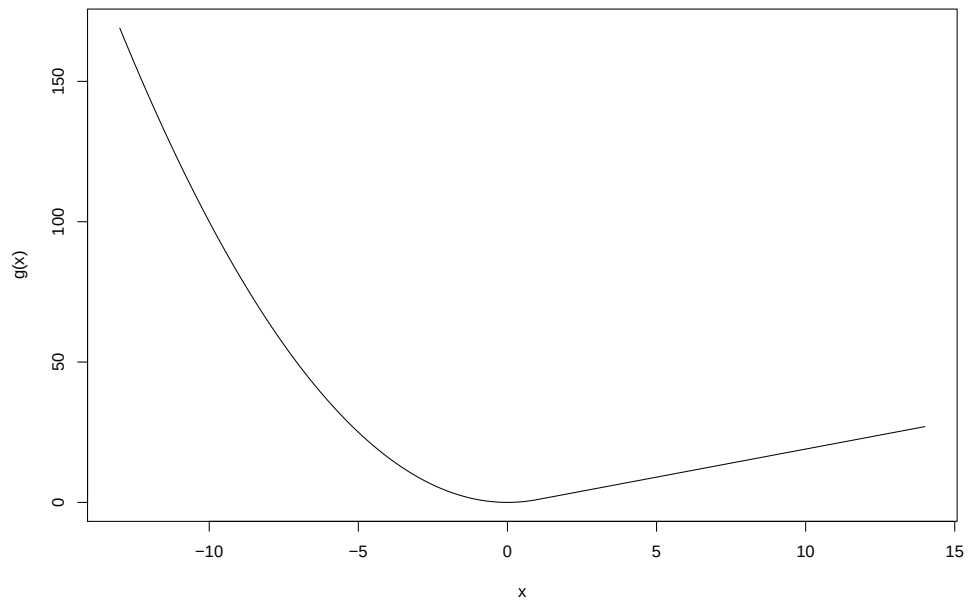
Función g

```
ge<-function(x)
{
  if(x<=1)
  {
    salida<-x^2
  }
else {salida<-2*x-1}
return(salida)
}
```

Graficamos g

```
grilla_1<-seq(-13,14,by=0.01)
yy_1<-lapply(grilla_1,ge)

plot(grilla_1,yy_1, type="l",xlab="x",ylab="g(x)")
```



Tarea 1: Para realizar ahora

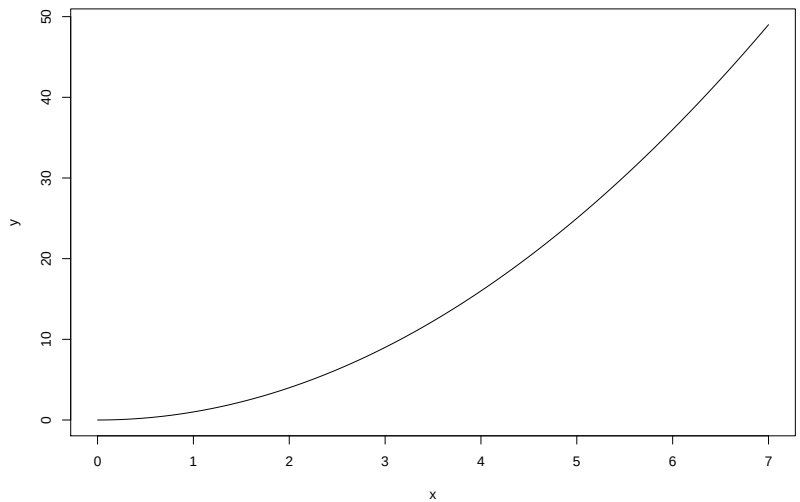
Considerar la función

$$h(x) = \begin{cases} 0 & \text{si } x < 10 \\ \dfrac{x-10}{15-10} & \text{si } 10 \leq x \leq 15 \\ 1 & \text{si } x > 15 \end{cases}$$

- Implementar una función *funcion.h* que dado un número devuelva el valor de la función h en ese número.
- Graficar la función $h(x)$ para $x \in [5, 20]$.
- Si quisiera cambiar las constantes 10 y 15 por parámetros $a < b$, respectivamente, de tal manera que puedan ser fijados por el usuario, ¿como debería reprogramarse la *funcion.h*?

Rebobinemos. . .

```r
x<-seq(0,7,by=0.01) # armo la grilla
y<- x*x             # elevo cada elemeto al cuadrado
plot(x,y,type="l")  # ploteo
```



Cuando hicimos **x*x** repetimos la operación multiplicación tantas veces como **length(x)**: es decir lo hicimos con el primer elemento de **x**, con el segundo elemento de **x**, etc.

Estas iteraciones las podríamos realizar usando el comando **for**.

Bucles: for

Bucles: se utilizan para repetir cierta acción.

Es útil si conocemos de antemano el número de veces que hay que repetir la acción (iteraciones).

for (val in sequence)

{

statement

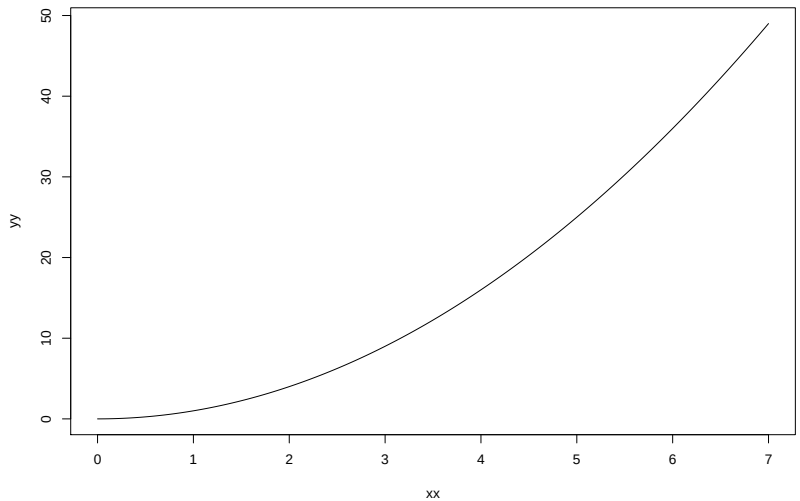
}

Implementación

```
xx <-seq(0,7,by=0.01) #armo grilla

nxx<-length(xx)       #calculo la longitud de la grilla
yy <-rep(NA, nxx)     #aca voy a guardar los cuadrados
for (i in 1:nxx)
{
yy [i]<- xx[i]^2
}

plot(xx,yy,type="l")
```



Reprogramamos sum() de R

```
sumar<- function(v)  # v: vector de números.
{
  suma = 0           # aca voy a ir acumulando
  for (elem in v)    # Ojo: iteramos sobre toda la colección
  {
    suma = suma + elem
  }
  suma
}

vv<- c(10,1,-1,8)
sumar(vv)
```

```
## [1] 18
```

```
sum(vv)
```

```
## [1] 18
```

Tarea 2: Para hacer ahora

Combinemos bucles y condiciones.

- Implementar una función **suma.positivos**
  que dado un vector v suma sus componente positivas.

Comando sample

¿Qué hace?

```
sample(1:5, size=5, replace=T)
```

```
## [1] 2 3 1 1 1
```

```
sample(1:5, size=5, replace=T)
```

```
## [1] 5 1 1 5 3
```

```
#control de semilla
set.seed(123)
sample(1:5, size=5, replace=T)
```

```
## [1] 2 4 3 5 5
```

```
set.seed(123)
sample(1:5, size=5, replace=T)
```

```
## [1] 2 4 3 5 5
```

Comando sample

```
sample(c("José", "Pedro", "Zoe"), size=2, replace=T)
```

```
## [1] "José"  "Pedro"
```

```
# Esto fue el muestreo con reposición.

sample(c("José", "Pedro", "Zoe"), size=2, replace=F)
```

```
## [1] "Zoe"   "Pedro"
```

```
# ¿Cómo hago para mezclar("shuffle") un vector?
sample(1:5, size=5, replace=F)   # replace=F
```

```
## [1] 3 4 2 5 1
```

```
x <- c("José", "Pedro", "Zoe", "Maria") #caso genérico

sample(x, size=length(x), replace=F)
```

```
## [1] "José"  "Zoe"   "Maria" "Pedro"
```

Vamos a jugar con los dados

- En un cubilete ponemos 5 dados de distinto color (Rojo, Azul, Verde, Negro y Blanco) con 6 caras numeradas del 1 al 6.
- Batimos el cubilete y arrojamos los dados ¿cuánto suman las caras obtenidas?

Me olvide decirles que trajeron los dados de estos colores!!! ¿Qué hacemos. . . . . ?

Idea: Vamos a simular con el R lo que sale en el cubilete!!!

Vamos a jugar con R

- voy a identificar un vector de 5 componentes con números de 1 al 6 como el resultado de arrojar un cubilete, asociando lo que ocurre en cada componente a un color: la componente 1 al color Rojo, en la 2 al Azul, en la 3 al Verde, en la 4 al Negro y en la 5 al Blanco.
- Asi el vector $c(6, 6, 1, 2, 4)$ representa que salió un 6 en el Rojo, un 6 en el Azul, un 1 en el Verde, un 2 en el Negro y un 4 en el Blanco.
- Para simular que batimos y arrojamos el cubilete usamos la función **(**sample).

Vamos a jugar con R

```r
#un lanzaminento de los 5 dados
cubilete<- sample(1:6, size=5, replace=T)
cubilete
```

```
## [1] 2 6 6 5 4
```

```r
sum(cubilete)
```

```
## [1] 23
```

Tarea 3: Para hacer ahora en clase

- crear una función suma.cubilete que simule arrojar el cubilete y dé por resultado la suma de las 5 caras obtenidas.
- todos aplican la función una vez y escriben en la planilla el resultado. (deben quedar aproximadamente 40 valores...)
- leemos la planilla y hacemos un histograma con los datos leídos.
- cada uno simula todo el proceso ahora con R: genera los 40 resultados del cubilete, calcula la suma y grafica un histograma para su simulación.
- repetir en lugar de 40, 1000 veces el ítem anterior.

Un poquito más de Bucles

Hay dos tipos de bucles dependiendo de si conocemos de antemano el número de veces que hay que repetirlo (iteraciones).

- La construcción **for** es muy útil especialmente cuando se sabe el total de trabajo a realizar (por ejemplo sumar todos los elementos de un vector).
- El **while** también permite *recorrer*, pero es más flexible respecto a cuando dejar de hacerlo.
- El **while** ofrece más posibilidades para elegir condiciones de corte del ciclo interesantes.

Juguemos un poco a la Generala

(. . . . la cuarentena se hace larga. . . .)

Creemos una función que permita responder esta pregunta:

- Arrojamos el cubilete hasta que sale generala ¿cuántos tiros son necesarios?
- Notemos que estamos repitiendo una acción, pero ahora no sabemos cuántas veces vamos a iterar.

while

```
while (condition) {
statement
}
```

Generala

```
 n.generala<- function(ese=5){
   ene=0
   generala=0
 while(generala==0){
    cubilete<- sample(1:6, size=ese, replace=T)
    if(min(cubilete)==max(cubilete)){generala=1}
    ene<- ene+1
 }
   n.generala<- ene
   n.generala
}

n.generala(5)
```

```
## [1] 1902
```

Aplicación

- Arrojar el cubilete hasta que sale generala, repetir esta acción mil veces y calcular el promedio de veces que fue necesario tirar el cubilete.
- ¿Cómo lo haríamos?