# Práctica 6: Funciones II

## 1. Implementación del método de Newton-Raphson

### 1.1. Formulación teórica

El método de Newton-Raphson permite encontrar las raíces de una función $f$ arbitraria, esto es, el valor (o valores) de $x$ para los cuales $f(x) = 0$. Se trata de un método numérico muy utilizado en gran parte debido a su simplicidad.

Partiendo de un valor inicial $x_0$, la verdadera raíz de la función $r$ se puede expresar como $r = x_0 + h$, donde $h$ representa lo lejos que el valor inicial se encuentra de la raíz de la función. Si el valor inicial está cerca de la raíz ($h$ es pequeño) entonces podemos considerar la siguiente aproximación:

$$0 = f(r) = f(x_0 + h) \approx f(x_0) + hf'(x_0),$$

de donde, si $f'(x_0) \neq 0$, tenemos que $h \approx -f(x_0)/f'(x_0)$ y por tanto

$$r = x_0 + h \approx x_0 - \frac{f(x_0)}{f'(x_0)}.$$

La aproximación anterior proporciona un algoritmo de aproximación de la raíz $r$, a partir de la solución inicial $x_0$, se van obteniendo iterativamente actualizaciones $x_1, x_2, \ldots$, que responden a la siguiente expresión recursiva:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}. \tag{1}$$

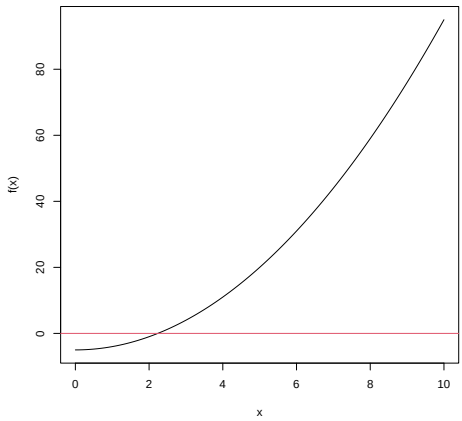
Existen condiciones sobre la función $f$ que garantizan la convergencia del algoritmo anterior, por ejemplo si la segunda derivada de $f$ existe y es continua cerca de la raíz $r$. Sin embargo el método puede fallar si por ejemplo $f'(x_n)$ está muy próximo a cero en alguna iteración.

### 1.2. Implementación en R conocida la derivada $f'(x)$

A partir de la descripción anterior nos proponemos en esta sección construir una función en R que implemente el método de Newton-Raphson. No obstante, para tener una mejor comprensión del método, vamos a comenzar considerando un ejemplo sencillo y vamos a implementar paso a paso algunas iteraciones del algoritmo para ver cómo se comporta.

Consideremos la función $f(x) = x^2 - 5$, y nos planteamos encontrar la solución de $f(x) = 0$, para $x \geq 0$. Aunque en este caso es trivial que la función tiene una raíz en $r = \sqrt{5}$, un buen comienzo para explorar el problema sería representar gráficamente la función:

```
> f<-function(x) x^2-5
> curve(f,0,10)    # curve() permite representar una función en un intervalo
>                  # en este caso lo hacemos en (0,10)
> abline(h=0,col=2)  # abline permite superponer una línea al gráfico
>                    # en este caso horizontal en y=0
```




El gráfico nos confirma que la función cruza el eje $y = 0$ en el intervalo $(2, 3)$. Teniendo en cuenta esto vamos a elegir como valor inicial $x_0 = 2$ y a implementar las primeras iteraciones del algoritmo de Newton-Raphson, usando que $f'(x) = 2x$, como sigue:

```
> f<-function(x) x^2-5
> f.prima<-function(x) 2*x
> # valor inicial
> x0<-2
> # primera iteración
> x1<-x0-f(x0)/f.prima(x0) ; x1

[1] 2.25

> # segunda iteración
> x2<-x1-f(x1)/f.prima(x1) ; x2

[1] 2.236111

> # tercera iteración
> x3<-x2-f(x2)/f.prima(x2) ; x3

[1] 2.236068

> # cuarta iteración
> x4<-x3-f(x3)/f.prima(x3) ; x4

[1] 2.236068
```

Observa que el método converge muy rápido y ya en la tercera iteración hemos encontrado la verdadera raíz que en este caso era $\sqrt{5} = 2.236068$.

Repite el proceso anterior ahora para encontrar una raíz de $f(x) = x^3 - 2x - 5$ en el intervalo $(-5, 5)$. Comienza representando la función en dicho intervalo y observa la dificultad que puede tener el problema. Después implementa una a una las iteraciones eligiendo primero como valor inicial un valor cercano a la verdadera raíz y luego otro más alejado. ¿Qué observas?

Utilizando lo que hemos aprendido con los ejemplos anteriores vamos ya a crear una función que implemente el algoritmo para cualquier función $f$. Dado que se trata de un algoritmo recursivo, necesitamos definir un criterio de convergencia o parada para el mismo (esto en qué iteración $n$ paramos). En los ejemplos anteriores hemos visto cómo el algoritmo convergía rápidamente cuando empezábamos en un valor próximo a la solución (no se producían cambios significativos en las iteraciones a partir de la tercera o cuarta), y que sin embargo no lo hacía tan rápido si el valor inicial no era tan cercano. Teniendo esto en cuenta vamos a definir nuestro criterio de convergencia o de parada como la unión de las dos condiciones siguientes:

(C1) Si la diferencia entre dos iteraciones consecutivas es despreciable, esto es, $|x_{n+1} - x_n| < tol$ donde $tol$ es un valor pequeño (tolerancia prefijada para el algoritmo).

(C2) Si hemos alcanzado un número máximo de iteraciones $nmax$.

Con este criterio ya tenemos todos lo necesario para implementar el algoritmo usando alguna estructura de repetición adecuada (valora si usar `for`, `while` o `repeat`). Esta es la tarea que te proponemos a continuación.

1. Escribe una función con nombre `algoritmo.NR` que implemente el método de Newton-Raphson descrito antes. La función tendrá cinco argumentos: la función cuya de la que se busca la raíz (`f`), su derivada (`f.prima`), el valor inicial (`x0`, la tolerancia (`tol`) y el número máximo de iteraciones (`nmax`). Los dos primeros argumentos serán objetos de tipo función mientras que los últimos corresponderán a valores numéricos. La función devolverá una lista con varias componentes: la raíz encontrada, el nivel de tolerancia conseguido y el número de iteraciones realizadas. Si el algoritmo se paró porque se alcanzó el número máximo de iteraciones (sin que el algoritmo convergiera) entonces deberá mostrar un mensaje de advertencia.

2. Comprueba el resultado de la función con las dos funciones que has explorado antes $f(x) = x^2 - 5$ y $f(x) = x^3 - 2x - 5$.

3. Añade filtros en la definición de la función que permitan controlar posibles errores (e.g. no se introduce alguno de los argumentos o no son apropiados).

4. Incluye un sexto argumento (`dibuja`) de tipo lógico y con valor por defecto `TRUE`. Cuando tome dicho valor entonces la evaluación de la función comenzará ofreciendo un gráfico de la función del tipo que construíamos antes (puedes fijar el intervalo de $x$ para el gráfico a partir del valor inicial o permitir al usuario pasar sus límites como argumentos adicionales a la función).

```
> algoritmo.NR<-function(f,f.prima,x0,tol,nmax)
+ {
+   x<-x0
+   for (i in 2:nmax)
+   {
+     x[i]<-x[i-1]-f(x[i-1])/f.prima(x[i-1])
+     dif<-abs(x[i]-x[i-1])
+     if (dif<tol) break
+   }
+   if (dif>tol) warning("Se ha alcanzado el número máximo´
                        de iteraciones sin llegar a la convergencia")
+   return(list(raiz=x[i],num.iteracions=i,iteraciones=x,tolerancia=dif))
+ }
> f<-function(x) x^2-5
> f.prima<-function(x) 2*x
> algoritmo.NR(f,f.prima,2,1e-4,10)

$raiz
[1] 2.236068
```

```
$num.iteracions
[1] 4

$iteraciones
[1] 2.000000 2.250000 2.236111 2.236068

$tolerancia
[1] 4.31332e-05

> f<-function(x) x^3-2*x-5
> f.prima<-function(x) 3*x^2 -2
> algoritmo.NR(f,f.prima,2,1e-4,10)

$raiz
[1] 2.094551

$num.iteracions
[1] 4

$iteraciones
[1] 2.000000 2.100000 2.094568 2.094551

$tolerancia
[1] 1.663941e-05
```

```
> algoritmo.NR.2<-function(f,f.prima,x0,tol=1e-10,nmax=50,
+                          dibuja=TRUE,lim.x=c(x0-2,x0+2))
+ {
+   if (missing(f) || missing(f.prima) || missing(x0))
+     stop("Debe proporcionar los argumentos requeridos")
+   if (!is.function(f) || !is.function(f.prima))
+     stop("Los argumentos 'f' y 'f.prima' deben ser funciones")
+   if (!is.numeric(x0) || length(x0)>1)
+     stop ("Proporcione un valor inicial adecuado")
+   if (dibuja) {
+     curve(f,lim.x[1],lim.x[2])
+     abline(h=0,col=2)
+   }

+   x<-x0
+   for (i in 2:nmax)
+   {
```

```
+     x[i]<-x[i-1]-f(x[i-1])/f.prima(x[i-1])
+     dif<-abs(x[i]-x[i-1])
+     if (dif<tol) break
+   }
+   if (dif>tol) warning("Se ha alcanzado el número
                        máximo de iteraciones sin llegar a la convergencia")
+   return(list(raiz=x[i],num.iteracions=i,iteraciones=x,tolerancia=dif))
+ }
```

## 1.3. Implementación en R usando una aproximación numérica de la derivada

Cuando la derivada de la función no resulta un cálculo trivial podemos utilizar métodos numéricos para su aproximación. El paquete *numDeriv* permite este tipo de aproximaciones. A continuación te mostramos su uso y te proponemos que definas una nueva versión de la función `algorimo.NR` que utilice esta aproximación cuando no se proporcione la derivada.

El primer paso es que instales y cargues el paquete en tu ordenador. Después observa cómo funciona para calcular la derivada de una función en un valor dado. Consideramos $f(x) = x^2 - 5$ como ejemplo aunque en tal caso la derivada sea trivial:

```
> library(numDeriv)
> f<-function(x) x^2-5
> # derivada en x=2
> res<-genD(func=f,x=2)
> res

$D
     [,1]     [,2]
[1,]    4 1.999999

$p
[1] 1

$f0
[1] -1

$func
function(x) x^2-5
<bytecode: 0x00000000110d8c10>
```

```
$x
[1] 2

$d
[1] 1e-04

$method
[1] "Richardson"

$method.args
$method.args$eps
[1] 1e-04

$method.args$d
[1] 1e-04

$method.args$zero.tol
[1] 1.781029e-05

$method.args$r
[1] 4

$method.args$v
[1] 2


attr(,"class")
[1] "Darray"

> # la derivada está en el primer elemento de la componente $D
> res$D[1]

[1] 4
```

Con esto podemos ahora por ejemplo implementar la primera iteración del algoritmo que escribíamos antes como sigue:

```
> # valor inicial
> x0<-2
> # primera iteración
> x1<-x0-f(x0)/genD(func=f,x=x0)$D[1] ; x1

[1] 2.25
```

1. Crea una nueva versión de la función `algorimo.NR` que utilice la aproximación numérica de la derivada anterior cuando no se proporcione la expresión de la derivada, esto es, el argumento `f.prima` está *missing*.
2. Comprueba tu función con $f(x) = x^2 - 5$, $f(x) = x^3 - 2x - 5$ y $f(x) = e^{2x} - x - 6$.
3. En la práctica anterior utilizamos la función `uniroot` del paquete *stats* para encontrar la raíz de una función. La función implementa otro algoritmo para el problema que evita aproximar numéricamente la derivada. Compara tu solución con la que da esta función en los tres ejemplos anteriores.

```
> algoritmo.NR.3<-function(f,f.prima,x0,tol,nmax)
+ {
+   x<-x0
+   for (i in 2:nmax)
+   {
+     if (missing(f.prima)) x[i]<-x[i-1]-f(x[i-1])/genD(func=f,x=x[i-1])$D[1]
+     else x[i]<-x[i-1]-f(x[i-1])/f.prima(x[i-1])
+     dif<-abs(x[i]-x[i-1])
+     if (dif<tol) break
+   }
+   if (dif>tol) warning("Se ha alcanzado el número máximo
                         de iteraciones sin llegar a la convergencia")
+   return(list(raiz=x[i],num.iteracions=i,iteraciones=x,tolerancia=dif))
+ }
> f<-function(x) exp(2*x)- x-6
> algoritmo.NR.3(f,x0=2,tol=1e-4,nmax=10)

$raiz
[1] 0.97087

$num.iteracions
[1] 7

$iteraciones
[1] 2.0000000 1.5693185 1.2259106 1.0286213 0.9742996 0.9708827 0.9708700

$tolerancia
[1] 1.263511e-05

> uniroot(f,c(-5,5))
```

```
$root
[1] 0.9708526

$f.root
[1] -0.0002249775

$iter
[1] 10

$init.it
[1] NA

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

# 2. Ejercicio propuesto

1. Crear una función con nombre `dif.eq` que devuelva los primeros $n$ elementos, $(x_1, x_2, \ldots, x_n)$, de la sucesión definida por la siguiente expresión:

$$x_{n+1} = r x_n (1 - x_n) \quad (n \geq 1)$$

   La función tendrá tres argumentos: `x1`, que corresponde al primer elemento de la sucesión, el coeficiente `r`, y el número de elementos de la sucesión a calcular (`n`).

   *a*) Ejecuta la función para $r = 2$ y $0 < x_1 < 1$. Deberías obtener que $x_n$ tiende a 0.5 cuando $n$ tiende a infinito.

   *b*) Representa gráficamente los primeros $n = 500$ de la sucesión para $x_1 = 0.95$ y $r = 2.99$. La gráfica la puedes obtener usando la función `plot` con la sintaxis `plot(1:500,v)`, siendo `v` el vector devuelto por la función.

   *c*) Escribe una segunda función `dif.eq2` con argumentos `x1` y `r` que devuelva una lista incluyendo, además del vector anterior (`v`), el número de iteraciones (`n`) necesarias hasta alcanzar el criterio de convergencia siguiente:

$$|x_{n+1} - x_n| < 0.02$$

   Comprueba tu función evaluándola en $x_1 = 0.95$ y $r = 2.99$, en cuyo caso deberías obtener que $n = 84$.

```
> dif.eq<-function(x1,r,n)
+ {
+   xn<-numeric(n)
+   xn[1]<-x1
+   for (i in 2:n) xn[i]<-r*xn[i-1]*(1-xn[i-1])
+   return(xn)
+ }
> dif.eq(x1=0.95,2,10)

 [1] 0.9500000 0.0950000 0.1719500 0.2847664 0.4073490 0.4828316 0.4994105 0.49999
 [9] 0.5000000 0.5000000

> plot(1:500,dif.eq(0.95,2.99,500))
>
>
> dif.eq2<-function(x1,r)
+ {
+   xn<-x1
+   i<-1
+   dif<-1
+   while (dif>=0.02)
+   {
+     i<-i+1
+     xn[i]<-r*xn[i-1]*(1-xn[i-1])
+     dif<-abs(xn[i]-xn[i-1])
+   }
+   return(list(iter=i-1,xn=xn))
+ }
>
> dif.eq2(0.95,2.99)

$iter
[1] 84

$xn
 [1] 0.9500000 0.1420250 0.3643432 0.6924757 0.6367298 0.6916018 0.6377333 0.69077
 [9] 0.6386749 0.6900001 0.6395608 0.6892631 0.6403967 0.6885635 0.6411870 0.68789
[17] 0.6419361 0.6872639 0.6426474 0.6866587 0.6433240 0.6860801 0.6439689 0.68552
[25] 0.6445843 0.6849952 0.6451726 0.6844855 0.6457357 0.6839957 0.6462752 0.68352
[33] 0.6467929 0.6830710 0.6472902 0.6826337 0.6477683 0.6822119 0.6482284 0.68180
[41] 0.6486717 0.6814112 0.6490990 0.6810308 0.6495113 0.6806626 0.6499094 0.68030
[49] 0.6502941 0.6799609 0.6506660 0.6796262 0.6510259 0.6793016 0.6513743 0.67898
```

```
[57] 0.6517117 0.6786808 0.6520388 0.6783838 0.6523559 0.6780952 0.6526635 0.67781
[65] 0.6529620 0.6775418 0.6532519 0.6772764 0.6535335 0.6770181 0.6538071 0.67676
[73] 0.6540731 0.6765218 0.6543317 0.6762833 0.6545833 0.6760510 0.6548281 0.67582
[81] 0.6550664 0.6756037 0.6552984 0.6753884 0.6555243
```