# Optimización inteligente

### Tarea 3. Búsqueda de Fibonacci.

**Ángel García Báez**

**2024-10-03**

## Breve introducción

Se planteo un ejercicio en clase donde se pedía encontrar el intervalo (espacio de búsqueda) donde sería factible encontrar el mínimo global de la función $f(x) = x^2 + \frac{54}{x}$ haciendo uso del método conocido como ***busqueda de Fibonacci***.

A continuación se muestra el pseudocódigo resumido de lo que busca hacer el algoritmo:

### Método de busqueda de Fibonacci

Algoritmo

Paso 1: Elegir un límite inferior $a$ y un limite superior $b$

$L = b - a$
Elegir un número deseado de iteraciones $N$
$k = 2$

Paso 2: $L_k^* = (F_{n-k+1}/F_{n+1}) * L$

$x_1 = a + L_k^*; x_2 = b - L_k^*$

Paso 3: Calcular $f(x_1)$ ó $f(x_2)$

(el que no se haya evaluado antes)
Usar la propiedad de eliminación de regiones.
Establecer nuevos valores de a y b.

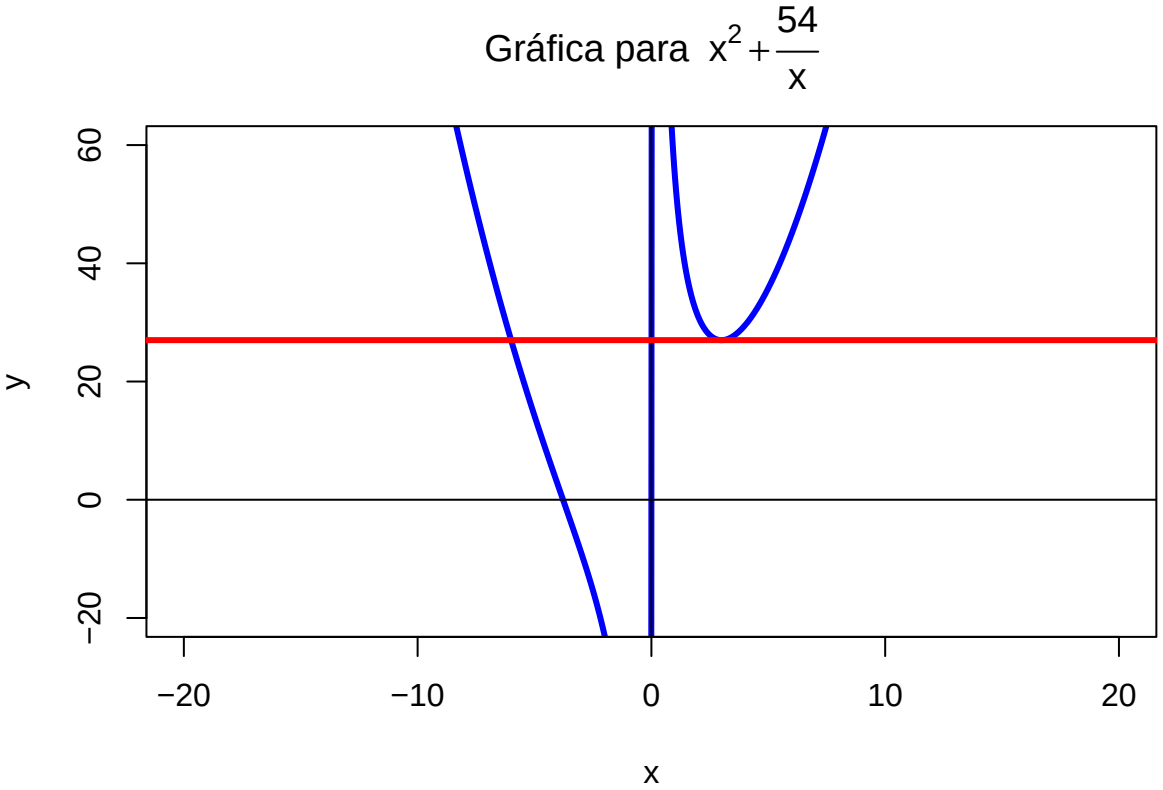
Paso 4: ¿ES $k > N$?

Si no, $k = k + 1$, GOTO Paso 2 ELSE TERMINAR

Una vez explicado lo que se debe de hacer y visto la estructura del método, se realizo la implementación de dicho algoritmo (Anexo 1) en lenguaje $R$ para ponerlo a prueba contra la función dada en un inicio.

## Experimentación con los resultados.

Primero, debido a que la función se puede graficar en un espacio de 2 dimensiones, se recurrió a esto para tener un referente visual del comportamiento de la misma:




La función presenta ciertos comportamientos interesantes, por ejemplo, a medida de que se acerca al 0 por la izquierda, dicha función tiende al infinito negativo (si buscara ahí el mínimo, nunca lo encontraría) y si se acerca al 0 por la derecha, el valor de la función tiende a ir hacia el infinito positivo (ahí tampoco es factible buscar).

Dado que la mayoría de los problemas que tienen que ver con cosas del mundo real implican que los valores sean positivos, se optara por buscar valores mayores o iguales a 0 de X tal que permitan llegar al mínimo que se ubica cuando $x = 3$ que devuelve un $f(x) = 27$.

## Primer acercamiento

Durante la clase, se dieron los parámetros de inicio para la búsqueda: $a = 0$, $b = 5$ y un $N = 3$, los cuales al ser evaluados en el algoritmo implementado, devuelven los siguientes resultados:

| **iter** | **a** | **b** | **fx1** | **fx2** |
|---|---|---|---|---|
| 1 | 2 | 5 | 31 | 27.0 |
| 2 | 2 | 4 | 27 | 29.5 |

El algoritmo con los parámetros dados en clase logra encontrar un intervalo que contiene al valor mínimo de la función, dicho intervalo es relativamente grande PERO logra encontrarlo en apenas 2 iteraciones del método, dicho intervalo es:

$$[2, 4]$$

Este intervalo encontrado bajo los criterios dados en clase logra contener en su interior al valor de $x$ que minimiza la función.

## Exploración y uso del método

Con la finalidad de explorar el comportamiento del método, se propone volver a correr el algoritmo para el mismo problema pero esta vez en un intervalo determinado por $a = 0$ y $b = 10$ y el uso de $N = 3$ para comprobar a que resultado llega si se dobla el intervalo de búsqueda.

A continuación se muestra el resultado de dicha evaluación:

| **iter** | **a** | **b** | **fx1** | **fx2** |
|---|---|---|---|---|
| 1 | 0 | 6 | 29.5 | 45.0 |
| 2 | 2 | 6 | 31.0 | 29.5 |

Los resultados arrojados resultan interesantes, puesto que en su búsqueda logra hacer apenas 2 iteraciones, moviéndose del intervalo de búsqueda $[0, 10]$ al intervalo $[2, 6]$ aunque si lo consideramos, el intervalo sigue siendo relativamente grande, por lo que valdria la pena experimentar con una mayor cantidad de interaciones.

## Propuesta de un intervalo de 0 a 5 pero con $N = 10$

A continuación se explora que pasaría al explorar el intervalo de $[0, 5]$ con una $N = 10$ que permite realizar una mayor cantidad de iteraciones en la búsqueda para encontrar un intervalo más finito en el cual realizar la búsqueda. Los resultados de evaluar el algoritmo así son:

| **iter** | **a** | **b** | **fx1** | **fx2** |
|---|---|---|---|---|
| 1 | 1.909722 | 5.000000 | 31.92340 | 27.02397 |
| 2 | 1.909722 | 3.819444 | 27.02397 | 28.72634 |
| 3 | 2.638889 | 3.819444 | 27.42689 | 27.02397 |
| 4 | 2.638889 | 3.368056 | 27.02397 | 27.37679 |
| 5 | 2.638889 | 3.090278 | 27.02123 | 27.02397 |
| 6 | 2.812500 | 3.090278 | 27.11016 | 27.02123 |
| 7 | 2.916667 | 3.090278 | 27.02123 | 27.00058 |
| 8 | 2.916667 | 3.020833 | 27.00058 | 27.00130 |
| 9 | 2.951389 | 3.020833 | 27.00717 | 27.00058 |

Los resultados mostrados al mantener un intervalo de $[0, 5]$ y aumentar $N = 10$ muestra el como los valores del intervalo se vuelven más finitos, puesto que el ultimo intervalo que logra encontrar ya se encuentra más cerrado alrededor del valor que minimiza la función, dicho intervalo se cierra a $[2.951389, 3.020833]$.

## Propuesta de un intervalo de 0 a 10 pero con $N = 20$

A continuación se explora que pasaría al explorar el intervalo de $[0, 10]$ con una $N = 20$ que le permita tener aun más iteraciones para lograr encontrar un intervalo aun más pequeño en el cual buscar el valor que minimiza la función. Los resultados de dicha busqueda son:

| iter | a | b | fx1 | fx2 |
|---|---|---|---|---|
| 1 | 0.000000 | 6.180340 | 28.72719 | 46.93398 |
| 2 | 0.000000 | 3.819660 | 28.44758 | 28.72719 |
| 3 | 1.458980 | 3.819660 | 39.14077 | 28.44758 |
| 4 | 2.360680 | 3.819660 | 28.44758 | 27.02057 |
| 5 | 2.360680 | 3.262379 | 27.02057 | 27.19545 |
| 6 | 2.705099 | 3.262379 | 27.27986 | 27.02057 |
| 7 | 2.917961 | 3.262379 | 27.02057 | 27.00728 |
| 8 | 2.917961 | 3.130823 | 27.00728 | 27.04991 |
| 9 | 2.917961 | 3.049517 | 27.00000 | 27.00728 |
| 10 | 2.968212 | 3.049517 | 27.00305 | 27.00000 |
| 11 | 2.968212 | 3.018463 | 27.00000 | 27.00102 |
| 12 | 2.987409 | 3.018463 | 27.00048 | 27.00000 |
| 13 | 2.987409 | 3.006606 | 27.00000 | 27.00013 |
| 14 | 2.994749 | 3.006606 | 27.00008 | 27.00000 |
| 15 | 2.994749 | 3.002089 | 27.00000 | 27.00001 |
| 16 | 2.997572 | 3.002089 | 27.00002 | 27.00000 |
| 17 | 2.999266 | 3.002089 | 27.00000 | 27.00000 |
| 18 | 2.999266 | 3.000960 | 27.00000 | 27.00000 |
| 19 | 2.999266 | 3.000395 | 27.00000 | 27.00000 |

Los resultados encontrados aumentando el $N = 20$ para el intervalo son sorprendentes, puesto que se observa como los valore del intervalo convergen lentamente a partir de la iteración numero 14, en donde el intervalo es tan finito que apenas cambian unas diez milesimas. La mejor solución que logra encontrar esta exploración es el intervalo: $[2.999266, 3.000395]$

**Evaluando en valores negativos**

Como experimento adicional, se plantea la idea de ver que ocurre si se explora un intervalo en los números negativos, como lo puede ser para $a = -5$, $b = 0$ y un $N = 15$:

| iter | a | b | fx1 | fx2 |
|---|---|---|---|---|
| 1 | -3.090169 | 0 | -7.925627 | -24.62730 |
| 2 | -1.909831 | 0 | -24.627300 | -44.35640 |
| 3 | -1.180338 | 0 | -44.356404 | -73.49187 |
| 4 | -0.729493 | 0 | -73.491875 | -119.57174 |
| 5 | -0.450845 | 0 | -119.571738 | -193.71561 |
| 6 | -0.278647 | 0 | -193.715614 | -313.56308 |
| 7 | -0.172198 | 0 | -313.563075 | -507.27102 |
| 8 | -0.106450 | 0 | -507.271021 | -821.30996 |
| 9 | -0.065748 | 0 | -821.309963 | -1,326.73681 |
| 10 | -0.040701 | 0 | -1,326.736805 | -2,155.94937 |
| 11 | -0.025047 | 0 | -2,155.949373 | -3,449.51975 |
| 12 | -0.015654 | 0 | -3,449.519755 | -5,749.19991 |
| 13 | -0.009393 | 0 | -5,749.199912 | -8,623.79996 |
| 14 | -0.006262 | 0 | -8,623.799961 | -17,247.59999 |

Se sabia de antemano que la función presenta una asintota en 0, cuando se aproxima por la izquierda tiende a valer $-\infty$, las decisiones que toma para seguir cortando se centran más del lado derecho, pues encuentra que a medida que se acerca al 0, la función disminuye aun más, esto seguirá ocurriendo indefinidamente, pero resulta interesante el intervalo relativamente pequeño que logra encontrar en 14 iteraciones:

$$[-0.006262, 0]$$

## Conclusiones y comentarios finales

El implementar el algoritmo de búsqueda Fibonacci consistió en un reto, porque de base fue necesario implementar primero una función que me diera los números de Fibonacci necesarios para poder usarlos en el método, después de ello, el método como tal fue sencillo de implementar al tener de antecedente lo aprendido al implementar el método de eliminación de regiones.

Los resultados obtenidos por el método de búsqueda de Fibonacci resultaron ser competentes al lograr dar un intervalo relativamente pequeño para realizar la búsqueda, sin embargo se demora una cantidad de iteraciones similar que el método de búsqueda por regiones, ademas de tener el extra de calcular los números de Fibonacci en su ejecución, lo cual le añade un coste computacional extra.

## Anexo 1: Código fuente del algotimo de busqueda de Fibonacci.

```
## # Mecanismo generador de fibonacci
## FIBO = function(n){
##   # Crear una lista inicial con los valores F_0=1 y F_1 = 1
##   FV = c(1,1)
##   if(n>=2){
##     # Hacer el ciclo que la construya poco a poco
##     for(i in 3:(n+1) ){
##       # Desde el elemnto 3 hasta el elemento n, suma los 2 anteriores
##       FV[i] = FV[i-1] + FV[i-2]
##     }
##     # Nombrar los datos en orden
##     names(FV) = 0:(n)
##     FV[paste0(n)] |> as.numeric()
##   }else{
##     names(FV) = 0:1
##     FV[paste0(n-1)] |> as.numeric()
##
##   }
##
##   #Devolver el ultimo numero
##
## }
##
## #### Definir los parámetros de entrada para el método ####
##
## # fx = función a evaluar
## # a = limite inferior
## # b = limite superior
## # N = Cantidad de iteraciones -1
##
## ### Definir la función a evaluar
##
## fibonacci = function(fx,a,b,N){
##   #### Paso 1 ####
##   L = b-a;L # ESTO VA
##   k = 2 # ESTO VA FIJO
##   # Llevar el control interno
##   contador = 0
##   # Tabla bonita para los resultados
##   resu = data.frame(iter = 0, a = 0, b = 0, fx1 = 0, fx2 = 0)
##   #### Paso 2 ####
##   while(k<=N){
##     # Calcular la razon con numeros de fibonacci para la distancia
##     Lke = (FIBO(N-k+1)/FIBO(N+1))*L;Lke
##     x1 = a+Lke
##     x2 = b-Lke
##     #### Paso 3, eliminación de regiones ###
##
##     # Evalua ambaos lados
##     fx1 = fx(x1)
##     fx2 = fx(x2)
##     # Decidir que hacer si
```

```
##     if(fx1>fx2){ #Si fx1 es mayor a fx2, corta la izquierda
##       a = x1
##     } else{
##       if(fx1<fx2){ #Si fx1 es menor a fx2, corta la derecha
##         b = x2
##       }else{ # Si no se cumple ninguna, corta ambos lados
##         a = x1
##         b = x2
##       }
##     }
##     ## Evaluar si K == N
##     k = k+1
##     resu[k-2,] = data.frame(iter = k-2, a, b , fx1 , fx2)
##
##   }
##
##   # Devolver la función
##   return(resu)
## }
```

## Anexo 2: Código para hacer las evaluaciones del método.

```
## rm(list=ls())
##
## ##Cargar el script con el método de busqueda Fibonacci ####
##
## source("Tarea 3. Busqueda de Fibonacci MAIN.R")
##
## ##Función que se desea acotar ####
##
## fx = function(x){(x^2) +  (54/x)}
##
##
## #### Declaro la función que se desea minimizar ####
## fx = function(x){(x^2) +  (54/x)}
##
##
## #### Evaluación para el ejercicio de clase ####
## # a = 0
## # b = 5
## # N = 3
## res1 = round(fibonacci(fx,0,5,3),6)
## res1
##
## #### Evaluación propuesta de aumentar el rango de búsqueda ####
## # a = 0
## # b = 10
## # N = 3
## res2 = round(fibonacci(fx,0,10,3),6)
## res2
##
## #### Evaluación propuesta de clase y aumentar N = 10 ####
## # a = 0
## # b = 5
## # N = 10
## res3 = round(fibonacci(fx,0,5,10),6)
## res3
##
## #### Evaluación propuesta con a = 0, b = 10  y aumentar N = 20 ####
## # a = 0
## # b = 10
## # N = 20
## res4 = round(fibonacci(fx,0,10,20),6)
## res4
##
## #### Evaluación propuesta en negativos con a = -5, b = 0  y aumentar N = 15 ####
## # a = -5
## # b = 0
## # N = 15
## res5 = round(fibonacci(fx,-5,0,15),6)
## res5
```