Algoritmo de Halley

Thomás Rivera
Nicolás Puerto

February 2021

1 ¿Cuáles son condiciones para aplicar el método?

Podemos encontrar dos condiciones para que el método se pueda cumplir. La primera de ellas es que f(x) debe ser de variable real(dominio los reales) y la segunda es que f(x) debe tener una tener una segunda derivada continua.

2 Proporcione una explicación geométrica del algoritmo

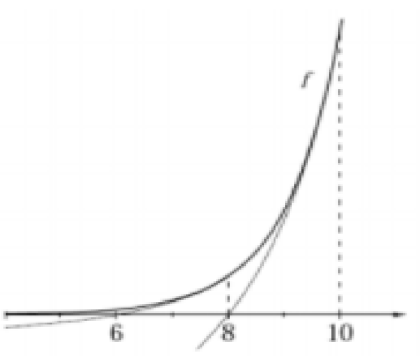

Figure 1: Explicación geométrica

También llamado el método de las hipérbolas tangentes. Consiste en la construcción de una hipérbola tangente a la gráfica de f(x) en el punto Xo próximo a la raíz de la función, que utiliza la función f(x) en cada una de sus iteraciones, para acercarse cada vez más al valor buscado. Cuanto mayor sea el número de estimaciones, mayor precisión en el resultado.

3 Realice un diagrama de flujo que muestre como se debe operar el algoritmo

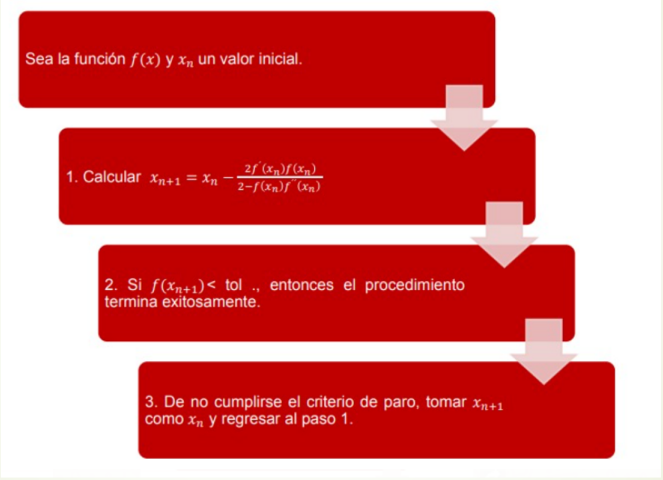

Figure 2: Diagrama de flujo

4 ¿Cuáles son las raíces?

4.1 cos2(x)-x2=0

En el caso de la primera función (figura 3 y 4) encontramos gran cercania al valor esperado dado por wolphram, con tan solo 4 iteraciones en el primer caso y 5 para el resto, es importante aclarar que utilizamos 56 digitos en todas funciones para todos sus diferentes valores de tolerancia

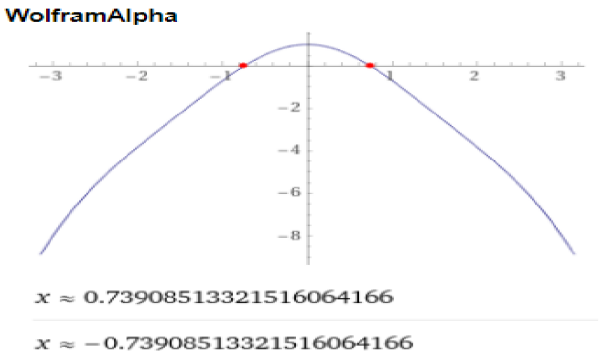

Figure 3: Gráfica 1

A continuación el código de la función y sus resultados en R

Código en R

```
#Metodo de Halley
i=1
VI <- mpfr(0.71875,56)
n=100
tolerancia=10^-8

f <- function(x){
  cos(x)^2-x^2
}
dx <- function(x){
  -(2*(sin(x)*cos(x))+2*x)
}
dxx <- function(x){
  -(2*(cos(x)*cos(x)-sin(x)*sin(x))+2)
}

while(i<=n){
  VS = VI - (f(VI)/dx(VI))*(1 - (f(VI)*dxx(VI)/dx(VI)^2))^(-1)
  if(abs(VS-VI)/abs(VS) < tolerancia){
    cat("Iteracion =",i,"\n Raiz =  ")
    print(VS,56)
    cat("Tolerancia =",tolerancia,"\n")
    break
  }
  i=i+1
  VI = VS
}
```

Mayor # iteraciones, mayor precisión

Resultados con diferentes tolerancias

```
Iteracion = 4
 Raiz =  1 'mpfr' number of precision  56   bits
[1] 0.73908513321516064453753358520771143957972526550292968 75
Tolerancia = 1e-08

Iteracion = 5
 Raiz =  1 'mpfr' number of precision  56   bits
[1] 0.73908513321516064453753358520771143957972526550292968 75
Tolerancia = 1e-16

Iteracion = 5
 Raiz =  1 'mpfr' number of precision  56   bits
[1] 0.73908513321516064453753358520771143957972526550292968 75
Tolerancia = 1e-32

Iteracion = 5
 Raiz =  1 'mpfr' number of precision  56   bits
[1] 0.73908513321516064453753358520771143957972526550292968 75
Tolerancia = 1e-56
```

Difiere con WolframAlpha a partir de la cifra 18

Figure 4: Código 1

4.2 xsin(x)-1=0 de [-1,2]

En el caso de la segunda función (figura 5 y 6) encontramos gran cercania al valor esperado y dado por wolphram, con tan solo 3 iteraciones en el primer valor de tolerancia y 4 para el resto de sus tolerancias, siempre manejando todo conn 56 digitos

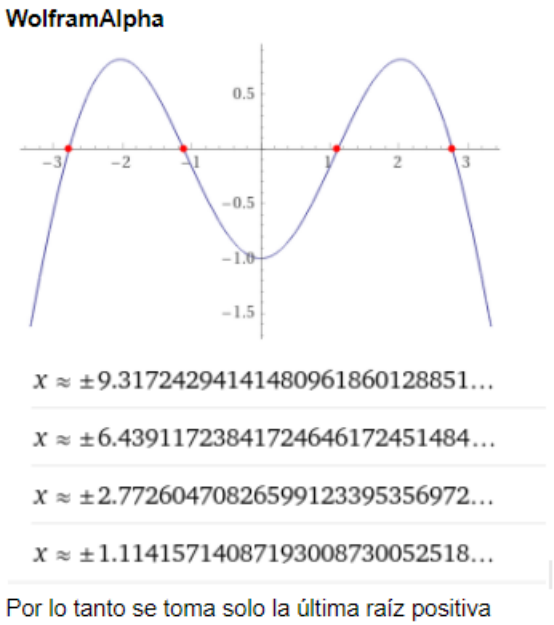

Figure 5: Gráfica 2

A continuación el código de la función y sus resultados en R

Código en R

```r
#Metodo de Halley
i=1
VI <- mpfr(1.09375,56)
n=100
tolerancia=10^-8

f <- function(x){
  x*sin(x)-1
}
dx <- function(x){
  sin(x)+x*cos(x)
}
dxx <- function(x){
  cos(x)+(cos(x)-x*sin(x))
}

while(i<=n){
  VS = VI - (f(VI)/dx(VI))*(1 - (f(VI)*dxx(VI)/dx(VI)^2))^(-1)
  if(abs(VS-VI)/abs(VS) < tolerancia){
    cat("Iteracion =",i,"\n Raiz =  ")
    print(VS,56)
    cat("Tolerancia =",tolerancia,"\n")
    break
  }
  i=i+1
  VI = VS
}
```

Resultados con diferentes tolerancias

```
Iteracion = 3
 Raiz =  1 'mpfr' number of precision  56   bits
[1] 1.1141571408719301017331559933154494501650333404541015625
Tolerancia = 1e-08

Iteracion = 4
 Raiz =  1 'mpfr' number of precision  56   bits
[1] 1.1141571408719300739775803776865359395742416381835937 5
Tolerancia = 1e-16

Iteracion = 4
 Raiz =  1 'mpfr' number of precision  56   bits
[1] 1.1141571408719300739775803776865359395742416381835937 5
Tolerancia = 1e-32

Iteracion = 4
 Raiz =  1 'mpfr' number of precision  56   bits
[1] 1.1141571408719300739775803776865359395742416381835937 5
Tolerancia = 1e-56
```

Con una tolerancia de -16 y mayores, la raíz es exacta

Figure 6: Código 2

4.3 x3-2x2+43x-827=0

En el caso de la tercera función (figura 7 y 8) encontramos gran cercania al valor esperado y dado por wolphram, con tan solo 1 iteración para todos los casos.

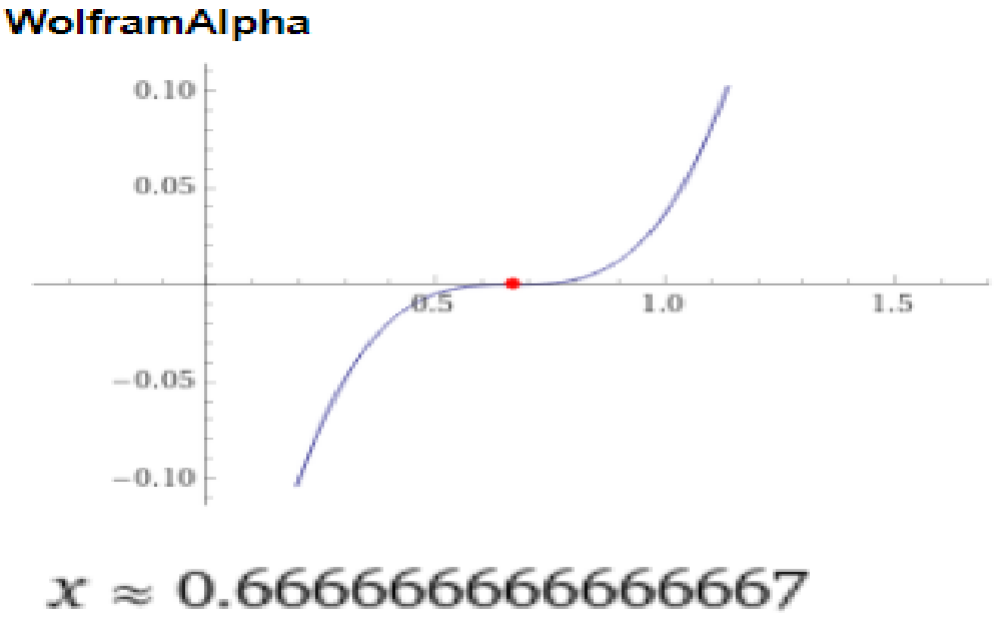

Figure 7: Gráfica 3

A continuación el código de la función y sus resultados en R.Hasta ahora es la función que más diferencia mantiene con el valor dado por WolframAlpha, a pesar del aumento de significancia en el resultado se mantuvo en todos los casos con 1 iteraciones

Código en R

```r
#Metodo de Halley
i=1
VI<-mpfr(0.65625,56)
n=100
tolerancia=10^-8

f <- function(x){
  x^3-2*x^2+(4/3)*x-(8/27)
}
dx <- function(x){
  3*x^2-2*(2*x)+(4/3)
}
dxx <- function(x){
  3*(2*x)-2*2
}

while(i<=n){
  VS = VI - (f(VI)/dx(VI))*(1 - (f(VI)*dxx(VI)/dx(VI)^2))^(-1)
  if(abs(VS-VI)/abs(VS)< tolerancia){
    cat("Iteracion =",i,"\n Raiz =  ")
    print(VS,56)
    cat("Tolerancia =",tolerancia,"\n")
    break
  }
  i=i+1
  VI = VS
}
```

Resultados con diferentes tolerancias

```
Iteracion = 7
 Raiz =  1 'mpfr' number of precision  56   bits
[1] 0.66667001816683790771378426143201068043708801269531 25
Tolerancia = 1e-08

Iteracion = 7
 Raiz =  1 'mpfr' number of precision  56   bits
[1] 0.66667001816683790771378426143201068043708801269531 25
Tolerancia = 1e-16

Iteracion = 7
 Raiz =  1 'mpfr' number of precision  56   bits
[1] 0.66667001816683790771378426143201068043708801269531 25
Tolerancia = 1e-32

Iteracion = 7
 Raiz =  1 'mpfr' number of precision  56   bits
[1] 0.66667001816683790771378426143201068043708801269531 25
Tolerancia = 1e-56
```

Hasta ahora es la función que más diferencia mantiene con el valor dado por WolframAlpha, a pesar del aumento de significancia en el resultado se mantuvo en todos los casos con #7 iteración, falla desde el 5 decimal

Figure 8: Código 3

4.4 667.38/x(1-e**(-0.1468x)))-40=0

En el caso de la cuarta función (figura 9 y 10) encontramos gran cercania al valor esperado y dado por wolphram, con tan solo 11 iteraciones en la primera tolerancia y 29 para casos más grandes e inclusive llega a 30.

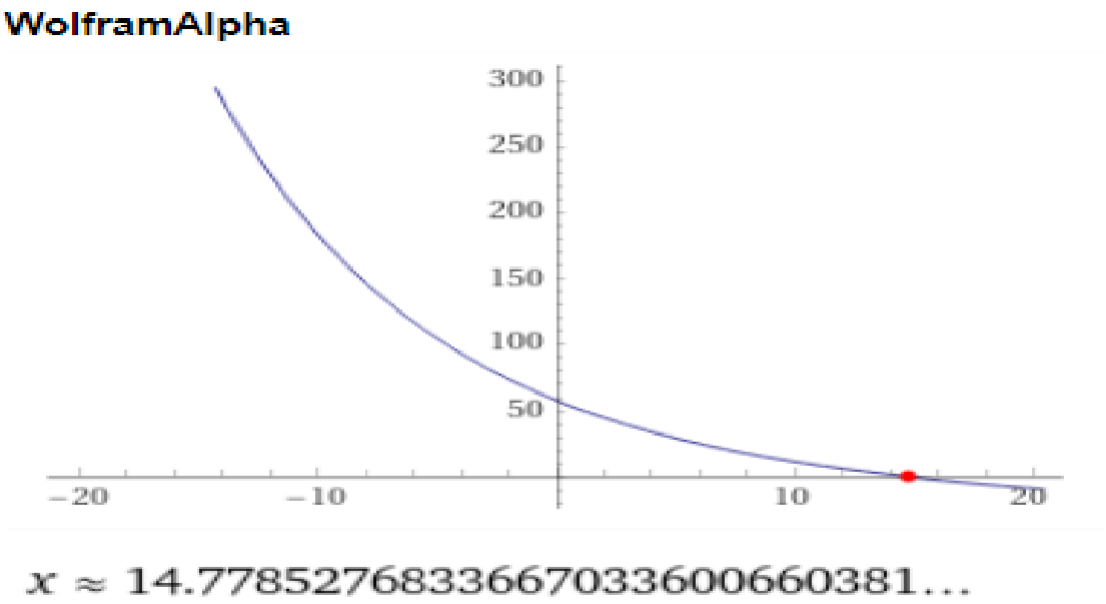

Figure 9: Gráfica 4

A continuación el código de la función y sus resultados en R

Código en R

```r
#Metodo de Halley
i=1
VI<-mpfr(14.78125,56)
n=100
tolerancia=10^-8

f <- function(x){
  ((667.38/x)*(1-exp(-0.1468*x)))-40
}
dx <- function(x){
  (0.1468*exp(-0.1468*x))-(667.38/x^2)
}
dxx <- function(x){
  (-0.02155024*exp(-0.1468*x))+(1334.76/x^3)
}

while(i<=n){
  VS = VI - (f(VI)/dx(VI))*(1 - (f(VI)*dxx(VI)/dx(VI)^2))^(-1)
  if(abs(VS-VI)/abs(VS) < tolerancia){
    cat("Iteracion =",i,"\n Raiz =  ")
    print(VS,56)
    cat("Tolerancia =",tolerancia,"\n")
    break
  }
  i=i+1
  VI = VS
}

```

Resultados con diferentes tolerancias

```
Iteracion = 11
 Raiz =  1 'mpfr' number of precision  56   bits
[1] 14.778527717639533989313349593430757522583007812 5
Tolerancia = 1e-08

Iteracion = 29
 Raiz =  1 'mpfr' number of precision  56   bits
[1] 14.778527683366703415757115180895198136568069458007812 5
Tolerancia = 1e-16

Iteracion = 30
 Raiz =  1 'mpfr' number of precision  56   bits
[1] 14.778527683366703415757115180895198136568069458007812 5
Tolerancia = 1e-32

Iteracion = 30
 Raiz =  1 'mpfr' number of precision  56   bits
[1] 14.778527683366703415757115180895198136568069458007812 5
Tolerancia = 1e-56
```

Difiere con WolframAlpha en la cifra 17

Figure 10: Código 4

4.5 x3-2x-5=0

En el caso de la quinta función (figura 11 y 12) encontramos gran cercania al valor esperado y dado por wolphram, con tan solo 3 iteraciones y 4 para casos más grandes finalizando con 5.

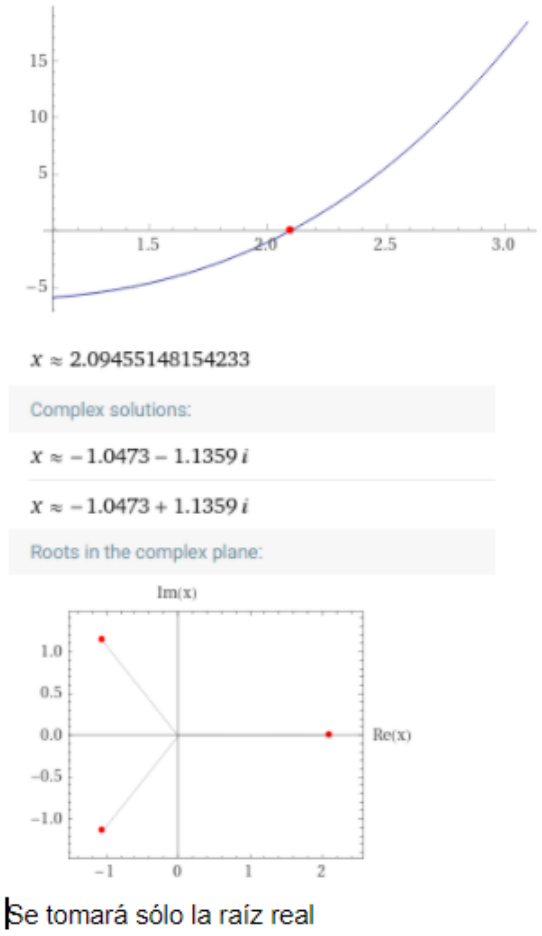

Figure 11: Gráfica 5

A continuación el código de la función y sus resultados en R

Código en R

```r
#Metodo de Halley
i=1
VI<-mpfr(2.09375,56)
n=100
tolerancia=10^-8

f <- function(x){
  x^3-2*x-5
}
dx <- function(x){
  3*x^2 -2
}
dxx <- function(x){
  3*(2*x)
}

while(i<=n){
  VS = VI - (f(VI)/dx(VI))*(1 - (f(VI)*dxx(VI)/dx(VI)^2))^(-1)
  if(abs(VS-VI)/abs(VS) < tolerancia){
    cat("Iteracion =",i,"\n Raiz =  ")
    print(VS,56)
    cat("Tolerancia =",tolerancia,"\n")
    break
  }
  i=i+1
  VI = VS
}

```

Resultados con diferentes tolerancias

```
Iteracion = 3
 Raiz =  1 'mpfr' number of precision  56   bits
[1] 2.0945514815423265653215878501214319840073585510253906 25
Tolerancia = 1e-08

Iteracion = 4
 Raiz =  1 'mpfr' number of precision  56   bits
[1] 2.0945514815423266208327390813792590051889419555664062 5
Tolerancia = 1e-16

Iteracion = 5
 Raiz =  1 'mpfr' number of precision  56   bits
[1] 2.0945514815423266208327390813792590051889419555664062 5
Tolerancia = 1e-32

Iteracion = 5
 Raiz =  1 'mpfr' number of precision  56   bits
[1] 2.0945514815423266208327390813792590051889419555664062 5
Tolerancia = 1e-56
```

Difiere con WolframAlpha a partir del dígito 14

Figure 12: Código 5

5 ¿Cómo se comporta el método en cuanto: pérdida de significancia, el número de iteraciones, la convergencia, en cada caso?

Problema 1: El algoritmo cumple con su deber y calcula de forma eficiente la raíz de la función. Le toma en principio 3 iteraciones y al aumentar la tolerancia llega a un máximo de 5 iteraciones sin diferir mucho del resultado dado por WolframAlpha. En cuanto a la convergencia halley tiene convergencia cúbica y eso facilita el proceso.
Problema 2: Al método le toma pocas iteraciones llegar a la raíz de la función, teniendo como máximo 3 y una pérdida de significancia mínima, lo que permitió encontrar la raíz de manera exacta comparada con el resultado de WolframAlpha. En la tolerancia 10-16 y en adelante la convergencia se mantuvo constante, con un mismo número de iteraciones.
Problema 3: Hasta ahora es la función que más diferencia mantiene con el valor dado por WolframAlpha, a pesar del aumento de significancia, el resultado se mantuvo en todos los casos con 1 iteración y con convergencia constante.
Problema 4: Es el caso donde mayor número de iteraciones le toma al método encontrar la raíz, debido a la naturaleza intrínseca de la función. El resultado varía respecto WolframAlpha en la última cifra, con un máximo de 29 iteraciones.
Problema 5: Al igual que el primer caso el método logra hallar la raíz de la función con pocas iteraciones, con menos de 5, el resultado difiere en los últimos 3 decimales y deja la convergencia constante con una tolerancia de 10-16 en adelante.

6 ¿Cómo se puede solucionar el problema de significancia, es remediable o está destinado al fracaso, en los casos que se presente el problema?

Si es remediable y en ese mismo orden de ideas se podría utilizar una precisión extendida para obtener números con 15 o 16 dígitos decimales más de precisión, ya que el método no es muy complejo y así mejorar los resultados. Para ello se emplea la librería Rmpfr para aumentar la cantidad de dígitos decimales obtenidos (utilizamos 56), sin embargo a pesar de ello, las raíces no llegan a ser completamente exactas.
Por el contrario, no consideramos que el algoritmo esté destinado al fracaso ya que, es de los mejores algoritmos que se presentan en la clase; es el que menos iteraciones presenta y en ese orden nos da una solución mucho más rápida, más precisa y "inalcanzable" para los demás algoritmos que en los mismo valores llegaban a tener 50 iteraciones, 30 o 20.

7 ¿Qué pasa con el método cuando hay más de dos raíces, explique su respuesta, encontrar la multiplicidad (sugerencia utilice Wólfram para factorizar)?

En la siguiente tabla encontramos los números de iteraciones presentados para cada función teniendo en cuenta la tolerancia
Podemos observar que los números de las iteraciones son demasiado estables, lo cual no representa un problema a pesar de que estamos cambiando bastante los valores de la tolerancia. Además también podemos inferir que la cuarta función demanda muchas más iteraciones debido a que es más compleja de lo normal puesto que maneja Euler elevado a ciertos valores.

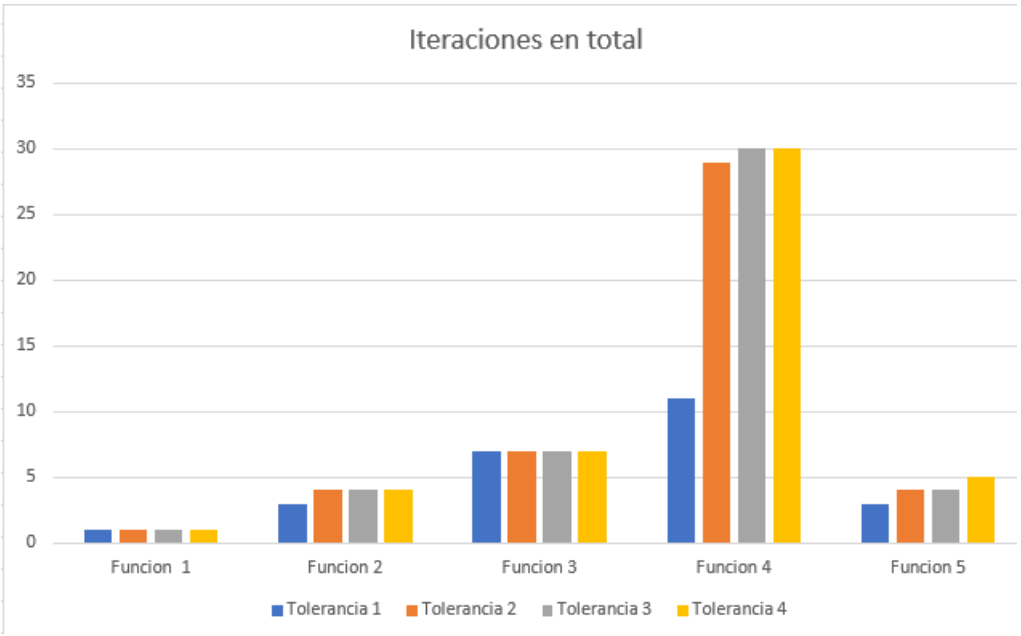

Figure 13: Iteraciones vs ToleranciaPorFuncion