PONTIFICIA UNIVERSIDAD JAVERIANA

ANALISIS NUMÉRICO
TALLER 1 – MÉTODOS DE BISECCIÓN, PUNTO FIJO Y APLICACIONES EN ING. SISTEMAS DE HALLAR CEROS

JOHAN DANIEL ORTEGÓN PARRA

BOGOTÁ D.C
2019

Método de Bisección

Código fuente Bisección

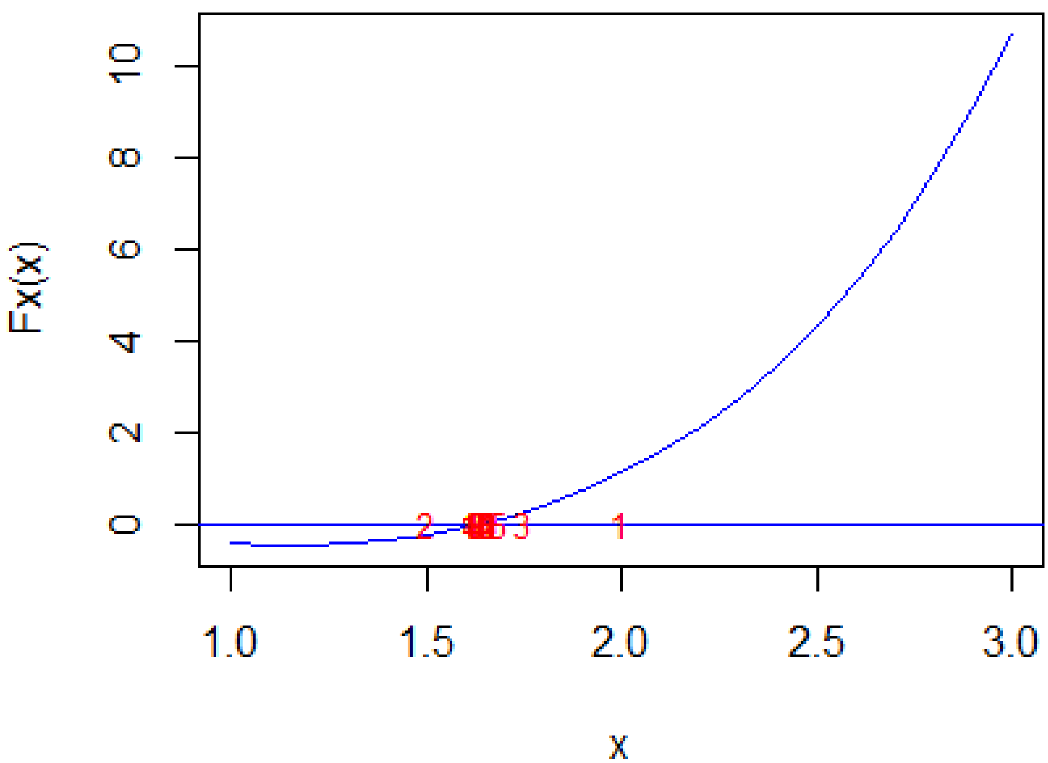
```
#Remueve todos los objetos creados
rm(list=ls())
Fx <- function(x) exp(x)-pi*x

#Halla la raiz de Fx
 biseccion <- function(a,b) {
 x<-seq(a,b,0.1)
 plot(x,Fx(x),type="l",col="blue")
 abline(h=0,col="blue")
 x<-b
 d<-(a+b)/2
 i<-0
 error<-abs(a-b)/2
   while (error > 1.e-4)
   {
     i<-i+1
     if (Fx(x) == 0) break
     if (Fx(x)*Fx(a) < 0) b <- x else {a <- x}
     d<-x
     x<-(a+b)/2
   #points(rbind(c(x,0)),pch=17,cex=0.7,col="red")
     text(x,0,i,cex=0.8,col="red")
     error<-abs(a-b)/2
     cat("X=",x,"\tE=",error,"\t","Iteracion=",i,"\n")
   }
 }
 biseccion(1,3)
```

Impresión en consola

```
X= 2            E= 1                    Iteracion= 1
X= 1.5          E= 0.5                  Iteracion= 2
X= 1.75         E= 0.25                 Iteracion= 3
X= 1.625        E= 0.125                Iteracion= 4
X= 1.6875       E= 0.0625               Iteracion= 5
X= 1.65625      E= 0.03125              Iteracion= 6
X= 1.640625     E= 0.015625             Iteracion= 7
X= 1.632812     E= 0.0078125            Iteracion= 8
X= 1.636719     E= 0.00390625           Iteracion= 9
X= 1.638672     E= 0.001953125          Iteracion= 10
X= 1.637695     E= 0.0009765625         Iteracion= 11
X= 1.638184     E= 0.0004882812         Iteracion= 12
X= 1.638428     E= 0.0002441406         Iteracion= 13
X= 1.63855      E= 0.0001220703         Iteracion= 14
X= 1.638489     E= 6.103516e-05         Iteracion= 15
```

Gráfica



Método de punto fijo

Código fuente

```
f<-function(x)
{
  x=exp(x)/pi
}
x=0
x=f(x)
temp=0
cont=0
err=1
for(i in 1:100)
  if(err>0.00000001)
  {
    temp=x
    x=f(x)
    err=(x-temp)/x
    if(err<0) err=0-err
    cat("X=",x,"\tE=",err,"\t","Iteracion=",cont,"\n")
    cont=cont+1
  } else break
```

Impresión en consola

```
X= 0.4376131   E= 0.2726227    Iteracion= 0
X= 0.4930638   E= 0.1124614    Iteracion= 1
X= 0.5211767   E= 0.05394128   Iteracion= 2
X= 0.5360364   E= 0.02772145   Iteracion= 3
X= 0.5440612   E= 0.01474984   Iteracion= 4
X= 0.5484448   E= 0.007992706  Iteracion= 5
X= 0.5508542   E= 0.004373964  Iteracion= 6
X= 0.5521831   E= 0.002406516  Iteracion= 7
X= 0.5529173   E= 0.001327955  Iteracion= 8
X= 0.5533234   E= 0.0007339798        Iteracion= 9
X= 0.5535482   E= 0.0004060458        Iteracion= 10
X= 0.5536726   E= 0.0002247406        Iteracion= 11
X= 0.5537415   E= 0.000124425        Iteracion= 12
X= 0.5537797   E= 6.889692e-05        Iteracion= 13
X= 0.5538008   E= 3.815298e-05        Iteracion= 14
X= 0.5538125   E= 2.112893e-05        Iteracion= 15
X= 0.553819    E= 1.17014e-05         Iteracion= 16
X= 0.5538226   E= 6.480435e-06        Iteracion= 17
X= 0.5538246   E= 3.589005e-06        Iteracion= 18
X= 0.5538257   E= 1.987677e-06        Iteracion= 19
X= 0.5538263   E= 1.100826e-06        Iteracion= 20
X= 0.5538266   E= 6.096662e-07        Iteracion= 21
X= 0.5538268   E= 3.376493e-07        Iteracion= 22
X= 0.5538269   E= 1.869992e-07        Iteracion= 23
X= 0.553827    E= 1.035652e-07        Iteracion= 24
```

Aplicación de hallar ceros en la ingeniería de sistemas

En la ingeniería de sistemas el encontrar los ceros en las funciones resulta de utilidad al querer analizar la complejidad de diversos algoritmos, dada la definición

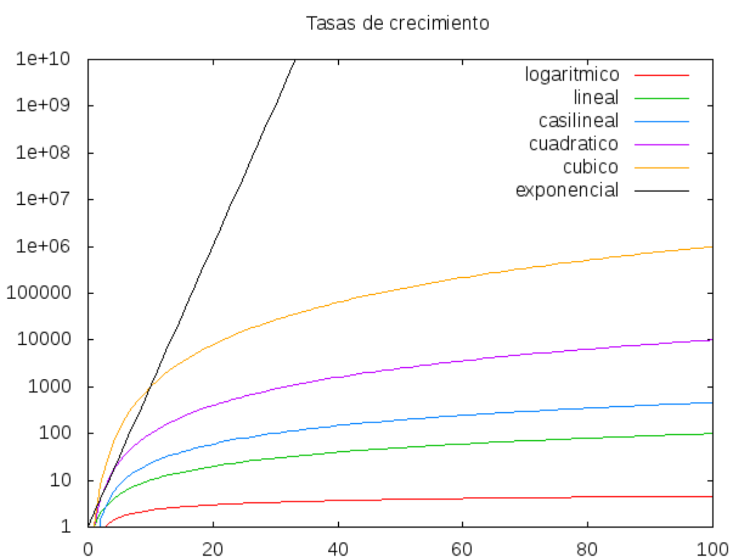
> A menudo se piensa que un algoritmo sencillo no es muy eficiente. Sin embargo, la sencillez es una característica muy interesante a la hora de diseñar un algoritmo, pues facilita su verificación, el estudio de su eficiencia y su mantenimiento. De ahí que muchas veces prime la simplicidad y legibilidad del código frente a alternativas más crípticas y eficientes del algoritmo. Este hecho se pondrá de manifiesto en varios de los ejemplos mostrados a lo largo de este libro, en donde profundizaremos más en este compromiso.
> Respecto al uso eficiente de los recursos, éste suele medirse en función de dos parámetros: el espacio, es decir, memoria que utiliza, y el tiempo, lo que tarda en ejecutarse. Ambos representan los costes que supone encontrar la solución al problema planteado mediante un algoritmo. Dichos parámetros van a servir además para comparar algoritmos entre sí
> (Rosa Guerequeta y Antonio Vallecillo, pg. 1 ,1998)

Podemos ver que el gasto de recursos por parte de un algoritmo puede ser expresado en una función, gracias a esto tenemos la posibilidad de comparar la eficiencia de diversos algoritmos que hacen la misma función, pero con consumos distintos, factores que resultan cruciales en el desarrollo óptimo de software.

conjuntos u órdenes de complejidad	
O (1)	orden constante
O (log n)	orden logarítmico
O (n)	orden lineal
O (n log n)	
O (n^2)	orden cuadrático
O (n^a)	orden polinomial (a > 2)
O (a^n)	orden exponencial (a > 1)
O (n!)	orden factorial

De Análisis de Algoritmos – Complejidad, José A. M añas, 2017

Como podemos observar la tabla anterior enuncia los comportamientos más comunes en la complejidad de algoritmos, si los comparamos entre si nos daremos cuenta de que describen graficas que se interceptan en algún punto, esta intersección significaría que los algoritmos consumen la misma cantidad de recursos dados ciertos criterios de entrada y de estructura del algoritmo, esto puede ayudar a tomar la decisión de que algoritmos se deberían implementar.



De http://www.cs.us.es/~jalonso/cursos/i1m/temas/tema-28.html

Bibliografía

A.Vallecillo. R.Guerequeta,1998, Técnicas de Diseño de Algoritmos , Publicaciones de la Universidad de Málaga

José A. M añas, imagen 1. Tabla de conjuntos u ordenes de complejidad, Lugar de publicación http://web.dit.upm.es/~pepe/doc/adsw/tema1/Complejidad.pdf

Universidad de Sevilla. Imagen 2. Tazas de crecimiento en la complejidad de algoritmos, Lugar de publicación http://www.cs.us.es/~jalonso/cursos/i1m/temas/tema-28.html