Analisis Numerico - Algoritmos Desarrollados

Trabajo realizado por:

Juan Francisco Hamon (hamon_juan@javeriana.edu.co)

Diego Mauricio Bulla (diegobulla@javeriana.edu.co)

Juan Diego Campos Neira (juand.campos@javeriana.edu.co)

Biseccion

El algoritmo de la bisección se utiliza para encontrar las raices de una función, trabajando con un interavalo que se va reduciendo a la mitad para así seleccionar un sub-intervalo donde se encuentra la raíz buscada. Para que el algoritmo sea efectivo se debe tener claro si la funcion $f(x)$ es continua en el intervalo dado.

Entradas del algoritmo

1)Valor del intervalo a la izquierda

2)Valor del intervalo a la derecha

3)Error

Paso a Paso del algoritmo

1)Empezar el ciclo

2)Se verifica que $f(a) * f(b) < 0$, siendo a el valor del intervalo a la izquierda y b el valor del intervalo a la derecha.

3)Se calcula el punto medio c del intervalo

4)Se evalua $f(c)$, si es esto da 0 significa que ya se encontro la raiz y el proceso termina, sino el proceso sigue.

5)De no complirse el paso 4 se verifica que $f(c) * f(a) > 0$ o si $f(c) * f(b) > 0$,en otras palabras, se busca cual de esas dos operaciones tiene signo opuesto.

6)Si se cumple $f(c) * f(a) > 0$ entonces el intervalo se vuelve $[c, b]$, sino, se define como $[a, c]$

7)Se verifica la condición del ciclo, si $b - a < E$ (siendo E el error) entonces el ciclo termina y se da el resultado obtenido. Sino, se repite el proceso de nuevo.

Para probar el algoritmo se utilizó la función $f(x) = e^x - \pi x$ y el intervalo [0,1] con una toleracia de $10e - 8$. A continuación, se mostrará el código del algoritmo y sus salidas.

```r
#Remueve los elementos que estan guardados en memoria
rm(list=ls())
#Limpia la consola para una mejor visualización
cat("\014")
```

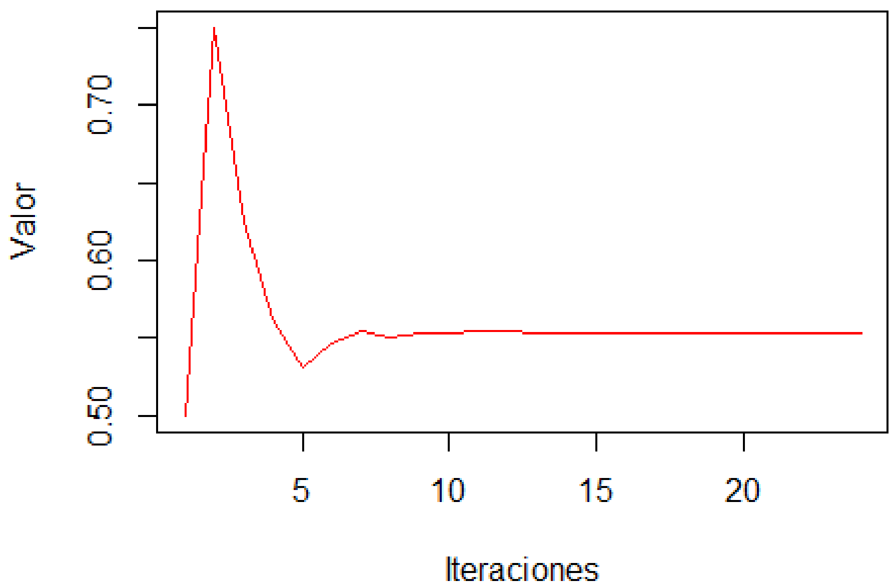
```r
Fx = function(x) ((exp(1)^x)-(pi*x))

biseccion = function(izquierdo, derecho, error){

  xIzquierda = izquierdo
  xDerecha = derecho
  iteraciones = 0
  valorAprox = 0
  valorActual = 0
  errorp = 0
  err = FALSE
  valores = c(0)
  errores = c(0)
  errores1 = c(0)

  while((xDerecha - xIzquierda) > error && !err){

    iteraciones = iteraciones + 1

    if(Fx(xIzquierda)*Fx(xDerecha) < 0){

      valorActual = (xDerecha + xIzquierda)/2
      valores[iteraciones] = valorActual

      if(Fx(valorActual) == 0){

        break

      }else{

        if(Fx(valorActual)*Fx(xIzquierda) > 0){
          errorp = abs((valorActual-xIzquierda)/valorActual)
          errores[iteraciones] = errorp
          if(iteraciones == 1){
            errores1[iteraciones] = 0
          }else{
            errores1[iteraciones] = errores[iteraciones-1]
          }
          xIzquierda = valorActual

        }else{
          errorp = abs((valorActual-xDerecha)/valorActual)
```

```r
          errores[iteraciones] = errorp
          if(iteraciones == 1){
            errores1[iteraciones] = 0
          }else{
            errores1[iteraciones] = errores[iteraciones-1]
          }
          xDerecha = valorActual

        }
      }

    }else{

      err = TRUE

    }
  }
  if(!err){

    valorAprox = valorActual
    iteracion = seq(1, iteraciones)
    tabla = data.frame(iteracion, valores)
    print(tabla)
    plot(iteracion, valores, type = "l",col="red"
         ,main = "Valores v.s Iteraciones"
         ,xlab = "Iteraciones"
         ,ylab = "Valor")
    plot(iteracion, errores, type = "l",col="red"
         ,main = "Errores v.s Iteraciones"
         ,xlab = "Iteraciones"
         ,ylab = "Error")
    plot(errores, errores1, type = "l", col="red"
         ,main = "Error en i v.s Error en i+1"
         ,xlab = "Error en i+1"
         ,ylab = "Error en i")

  }else{

    cat("No se pudo encontrar la raiz")

  }

}

biseccion(0,1,10e-8)
```

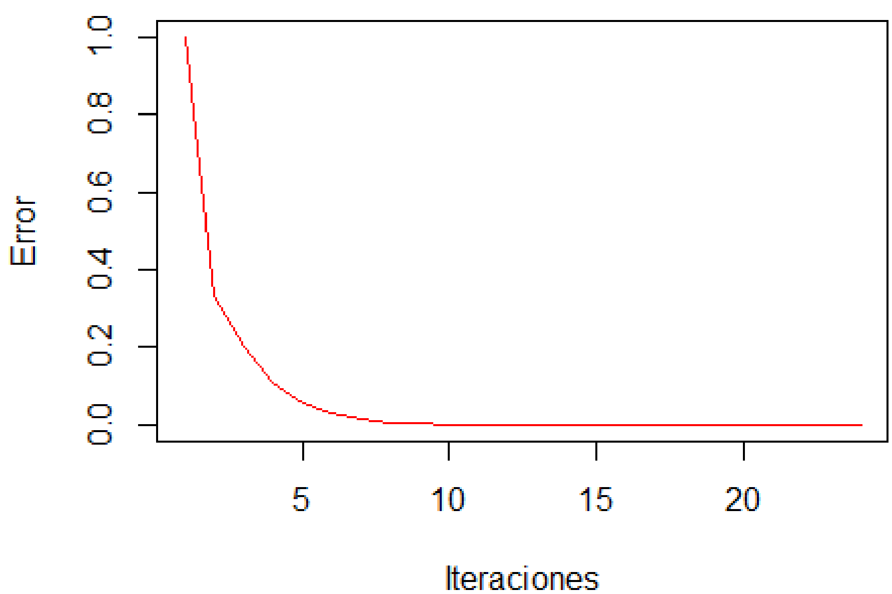
```
##    iteracion   valores
## 1          1 0.5000000
## 2          2 0.7500000
```

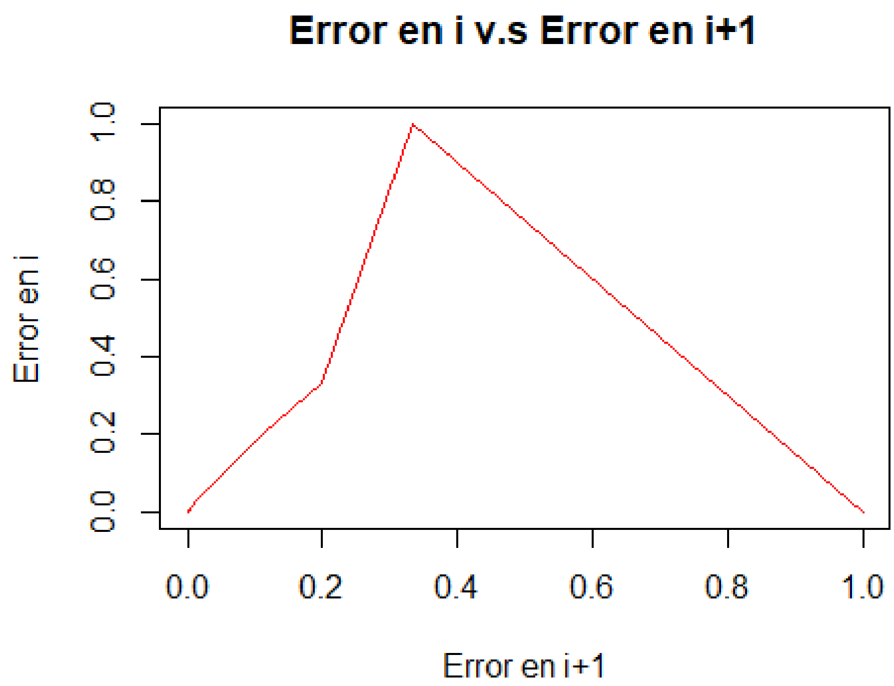
```
## 3           3 0.6250000
## 4           4 0.5625000
## 5           5 0.5312500
## 6           6 0.5468750
## 7           7 0.5546875
## 8           8 0.5507812
## 9           9 0.5527344
## 10         10 0.5537109
## 11         11 0.5541992
## 12         12 0.5539551
## 13         13 0.5538330
## 14         14 0.5537720
## 15         15 0.5538025
## 16         16 0.5538177
## 17         17 0.5538254
## 18         18 0.5538292
## 19         19 0.5538273
## 20         20 0.5538263
## 21         21 0.5538268
## 22         22 0.5538270
## 23         23 0.5538269
## 24         24 0.5538270
```

Valores v.s Iteraciones



Errores v.s Iteraciones





Posicion Falsa

Al igual que en el algoritmo anterior, este metodo utiliza un intervalo inicial para encontrar una raíz de una función, y a medida que se va desarrollando, dicho intervalo se va volviendo cada vez más pequeño en donde se encuentra la raíz buscada.

Entradas del algoritmo

1)Valor del intervalo a la izquierda

2)Valor del intervalo a la derecha

3)Error

Paso a Paso del algoritmo

1)Se entra al ciclo

2)Se evalúa $c = (f(b)a - f(a)b)/(f(b) - f(a))$ donde c es el punto intersección que pasa por $(a, f(a))$ y $(b, f(b))$; a es el valor del intervalo a la izquierda y b es el valor del intervalo a la izquierda.

3)Si $f(c)$ es lo suficientemente pequeño entonces esa es la raíz. Sino, se evalúa $f(c) * f(a) < 0$ y si esto se cumple, el intervalo cambia a $[a, c]$; de lo contrario el intervalo será $[c, b]$.

4)Si el error calculado es mayor que el error aceptado, el ciclo continua. De lo contrario, se repite el proceso.

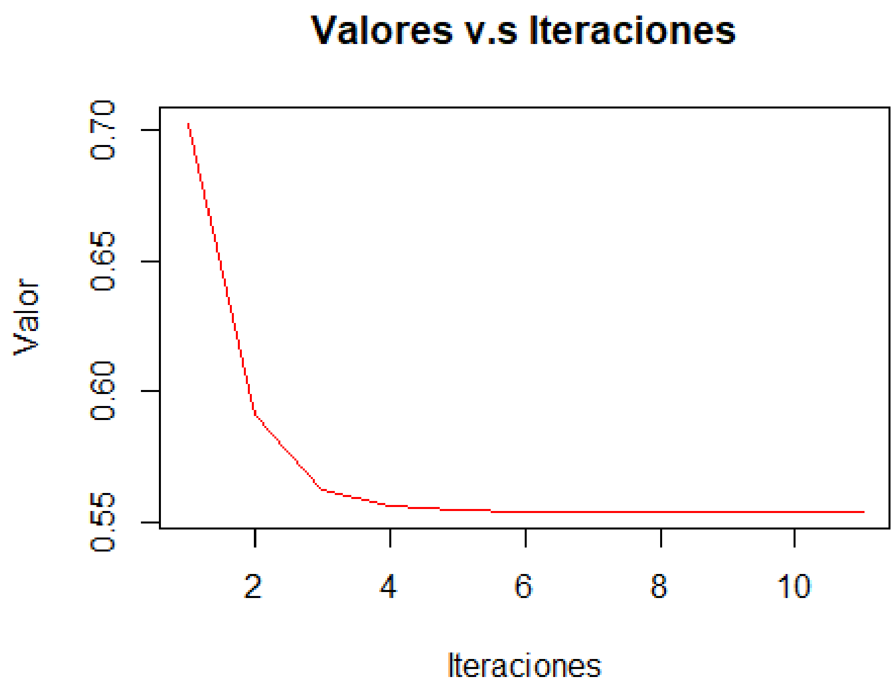
Para probar el algoritmo se utilizó la función $f(x) = e^x - \pi x$ y el intervalo $[0,1]$ con una toleracia de $10e - 8$. A continuación, se mostrará el código del algoritmo y sus salidas.

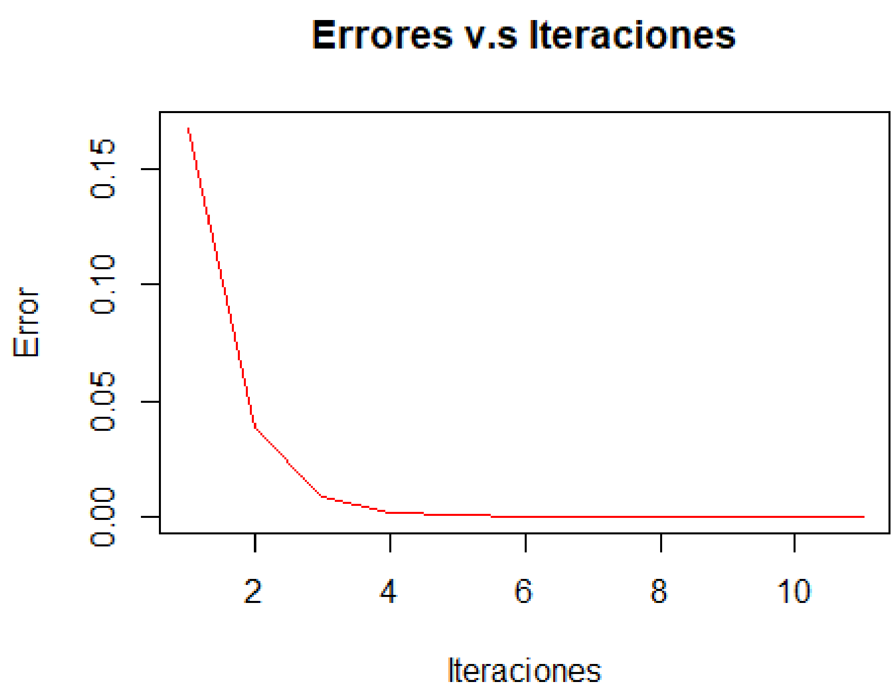
```
#Se limpian los elementos creados con anterioridad
rm(list=ls())
#Se limpia la consola para una mejor visualización
cat("\014")
```

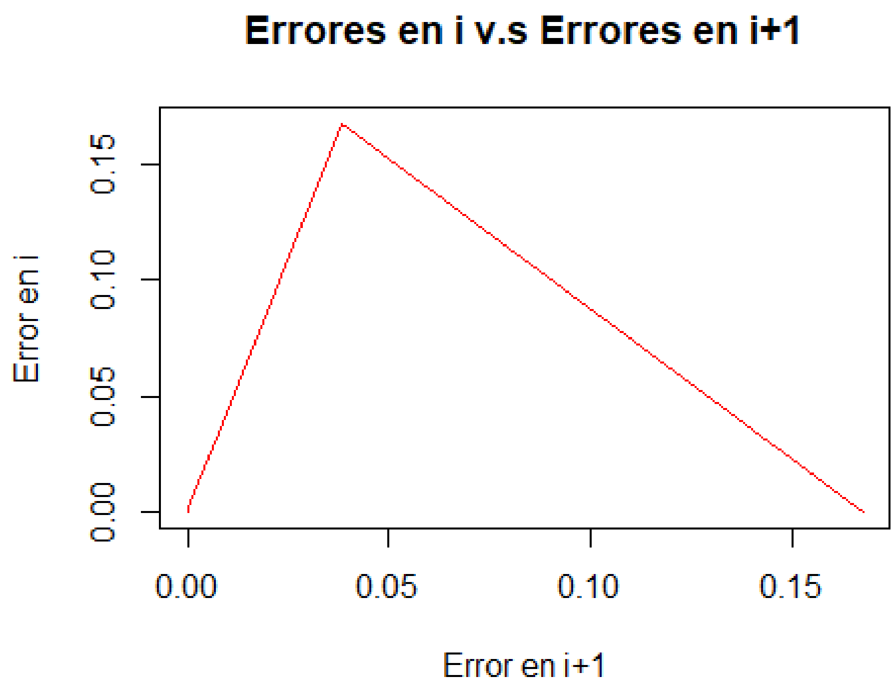
```
Fx = function(x) ((exp(1)^x)-(pi*x))
dFx = function(x) ((exp(1)^x)-pi)

posicionFalsa = function(xIzquierdo, xDerecho, error){

  iteraciones = 0
  errors = 1
  valores = c(0)
  errores = c(0)
  errores1 = c(0)

  while(errors > error){

    iteraciones = iteraciones + 1
    x = (Fx(xDerecho)*xIzquierdo -
Fx(xIzquierdo)*xDerecho)/(Fx(xDerecho)-Fx(xIzquierdo))
    valores[iteraciones] = x

    if(Fx(x) == 0){
      break
    }

    if(Fx(x)*Fx(xIzquierdo) < 0){
      xDerecho = x
    }

    else{
      xIzquierdo = x
    }

    errors = abs(Fx(x)/dFx(x))
    errores[iteraciones] = errors
    if(iteraciones == 1){
      errores1[iteraciones] = 0
    }
    else{
      errores1[iteraciones] = errores[iteraciones-1]
    }
```

```r
  }
  iteracion = seq(1,iteraciones)
  tabla = data.frame(iteracion,valores)
  print(tabla)
  plot(iteracion, valores, type = "l",col="red"
       ,main = "Valores v.s Iteraciones"
       ,xlab = "Iteraciones"
       ,ylab = "Valor")
  plot(iteracion, errores, type = "l",col="red"
       ,main = "Errores v.s Iteraciones"
       ,xlab = "Iteraciones"
       ,ylab = "Error")
  plot(errores,errores1, type = "l", col= "red"
       ,main = "Errores en i v.s Errores en i+1"
       ,xlab = "Error en i+1"
       ,ylab = "Error en i")
}

posicionFalsa(0, 1, 10e-8)
```

```
##    iteracion   valores
## 1          1 0.7025872
## 2          2 0.5912673
## 3          3 0.5624449
## 4          4 0.5557674
## 5          5 0.5542617
## 6          6 0.5539243
## 7          7 0.5538488
## 8          8 0.5538319
## 9          9 0.5538281
## 10        10 0.5538273
## 11        11 0.5538271
```


Valores v.s Iteraciones
Valor
0.70
0.65
0.60
0.55
2
4
6
8
10
Iteraciones


Errores v.s Iteraciones
Error
0.15
0.10
0.05
0.00
2
4
6
8
10
Iteraciones



Newton

Este algoritmo no solamente sirve para encontrar la aproximación de las raices de una función, sino que también funciona para econtrar el máximo o el mínimo de una función (encontrando los ceros de su primera derivada). Este algoritmo es un método abierto, es decir, que no garantiza su convergencia global; la única forma de alcanzar la convergencia es eligiendo un valor inicial lo suficientemente cercano a la raíz buscada. Así mismo, la cercanía del punto inicial a la raíz depende de la naturaleza de la función a evaluar, ya que si hay muchos puntos de inflexión o pendientes muy grandes, las probabilidades de que el algoritmo diverja aumentarán, por lo que el punto inicial debe ser muy cercano a la raíz.

Entradas del algoritmo

1)Error

2)Valor inicial

Paso a Paso del algoritmo

1)Se verifica que $f'(x_0) \neq 0$ donde x_0 es el valor inicial dado (la razón se explicará más adelante). En dado caso que esto ocurra, el proceso termina, pues no se puede calcular la raíz.

2)Se evalúa $x_1 = x_0 - \frac{f(x_0)}{f\prime(x_0)}$. Se verificó el paso 1, pues si $f'(x_0) = 0$ se habría presentado una indeterminación en la operación.

3)Se entra al ciclo.

4)Se verifca que $f'(x_i) \neq 0$ siendo i la iteración actual.En dado caso que esto ocurra, el proceso termina, pues no se puede calcular la raíz.

5)Se evalúa $x_{i+1} = x_i - \frac{f(x_i)}{f\prime(x_i)}$.

6)Si $|x_{i+1} - x_i| > E$ se termina el proceso (siendo x_{i+1} el valor de x en la iteracion $i + 1$; x_i el valor de x en la iteracion i y E el error permitido). De lo contrario, se repite el proceso desde el paso 3.

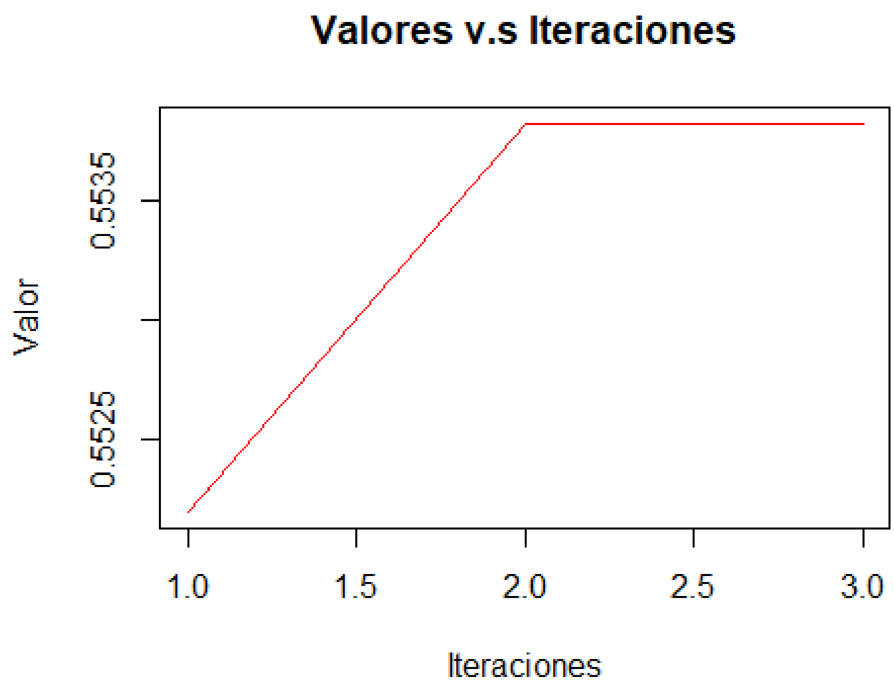
Para probar el algoritmo se utilizó la función $f(x) = e^x - \pi x$ y el valor inicial 0.5 con una toleracia de $10e - 8$. A continuación, se mostrará el código del algoritmo y sus salidas.

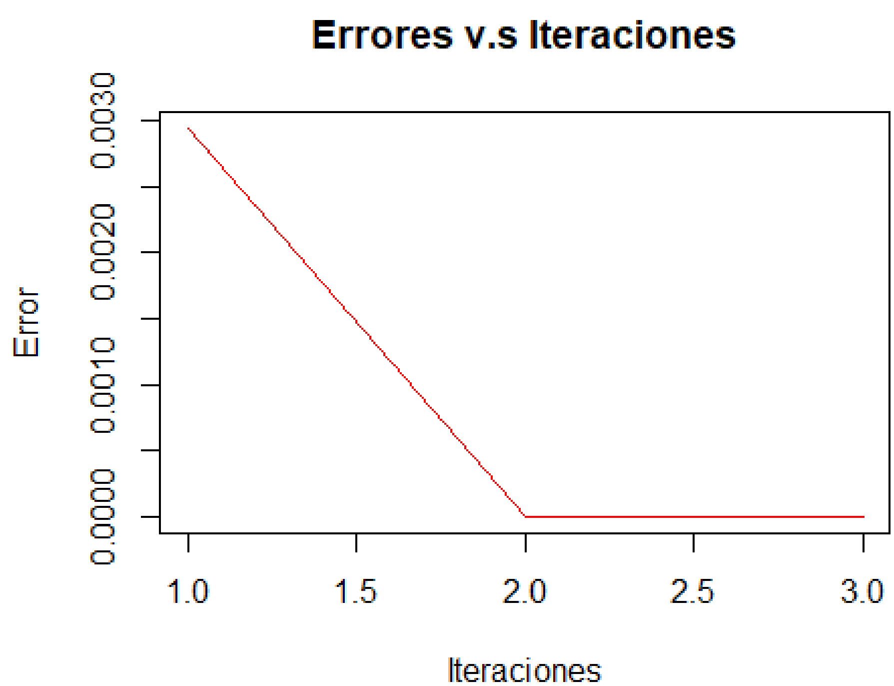
```
#Remueve los elementos que estan guardados en memoria
rm(list=ls())
#Limpia la consola para una mejor visualización
cat("\014")
```

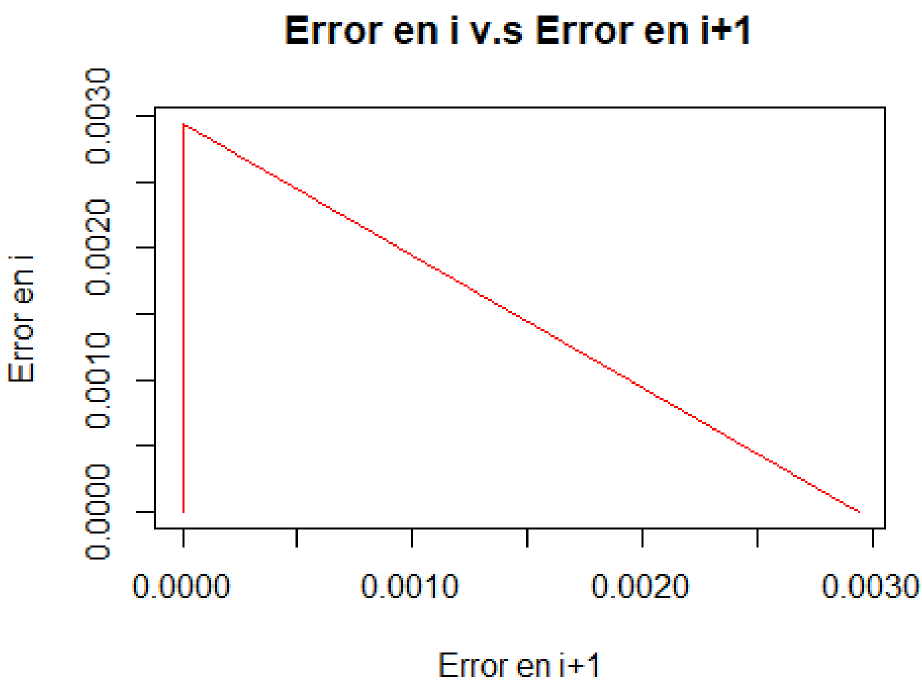
```
Funcion = function(x) ((exp(1)^x)-(pi*x))
dFX = function(x) ((exp(1)^x)-pi)

newton = function(error, valInicial){

  x = valInicial
  iteraciones = 0
  err = FALSE
  rela = 0
  valores = c(0)
  errores = c(0)
  errores1 = c(0)

  #Si la derivada evaluada en x es cero, no se puede continuar
  if(dFX(x) == 0){

    err = TRUE
    cat("No se pudo encontrar un valor aproximado")

  }
  if(err == FALSE){

    r = x - (Funcion(x)/dFX(x))

    while(abs(r - x) > error && err == FALSE){

      #Si la derivada evaluada en x es cero, no se puede continuar
      if(dFX(x) == 0){
```

```
        err = TRUE
        break
      }


      iteraciones = iteraciones + 1
      x = r
      r = x - (Funcion(x)/dFX(x))
      rela = (abs(r - x)/r)
      errores[iteraciones] = rela
      valores[iteraciones] = x
      if(iteraciones == 1){
        errores1[iteraciones] = 0
      }else{
        errores1[iteraciones] = errores[iteraciones-1]
      }

    }

  }
  iteracion = seq(1,iteraciones)
  tabla = data.frame(iteracion,valores)
  print(tabla)
  plot(iteracion, valores, type = "l",col="red"
       ,main = "Valores v.s Iteraciones"
       ,xlab = "Iteraciones"
       ,ylab = "Valor")
  plot(iteracion, errores, type = "l",col="red"
       ,main = "Errores v.s Iteraciones"
       ,xlab = "Iteraciones"
       ,ylab = "Error")
  plot(errores, errores1, type = "l",col="red"
       ,main = "Error en i v.s Error en i+1"
       ,xlab = "Error en i+1"
       ,ylab = "Error en i")
}

newton(10e-8, 0.5)
```

```
##   iteracion   valores
## 1         1 0.5521980
## 2         2 0.5538254
## 3         3 0.5538270
```


Valores v.s Iteraciones
Valor
0.5525
0.5535
1.0
1.5
2.0
2.5
3.0
Iteraciones


Errores v.s Iteraciones
Error
0.0000
0.0010
0.0020
0.0030
1.0
1.5
2.0
2.5
3.0
Iteraciones



Punto Fijo

El algoritmo del punto fijo, se utiliza para encontrar los ceros de una función, reescribiendo $f(x) = 0$ de la forma $x = g(x)$.

Entradas del algoritmo

1)Valor inicial

2)Error

Paso a Paso del algortimo

1)Se entra al ciclo

2)Se evalua $x = g(x)$.

3)Si $|g(x) - x| > E$ (siendo E el error permitido), el proceso termina. En caso contrario, el proceso vuelve a iniciar.

Para probar el algoritmo se utilizo la función $g(x) = \frac{e^{\pi}}{\pi}$ y el valor incial 0.5 con una tolerancia de $10e - 8$. A continuación, se mostrará el código del algoritmo y sus salidas.

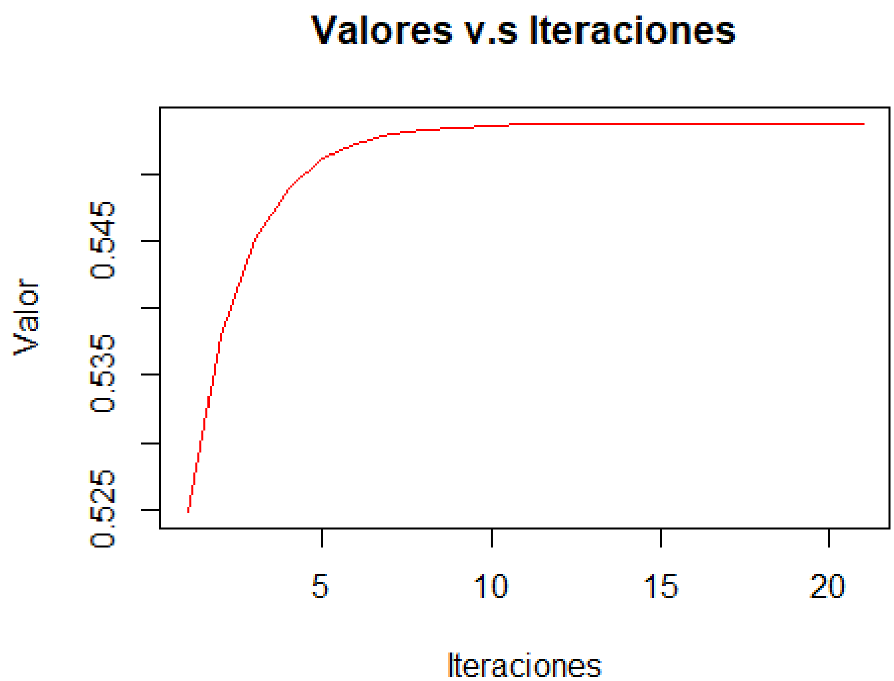
```
#Se limpian los elementos creados con anterioridad
rm(list=ls())
```

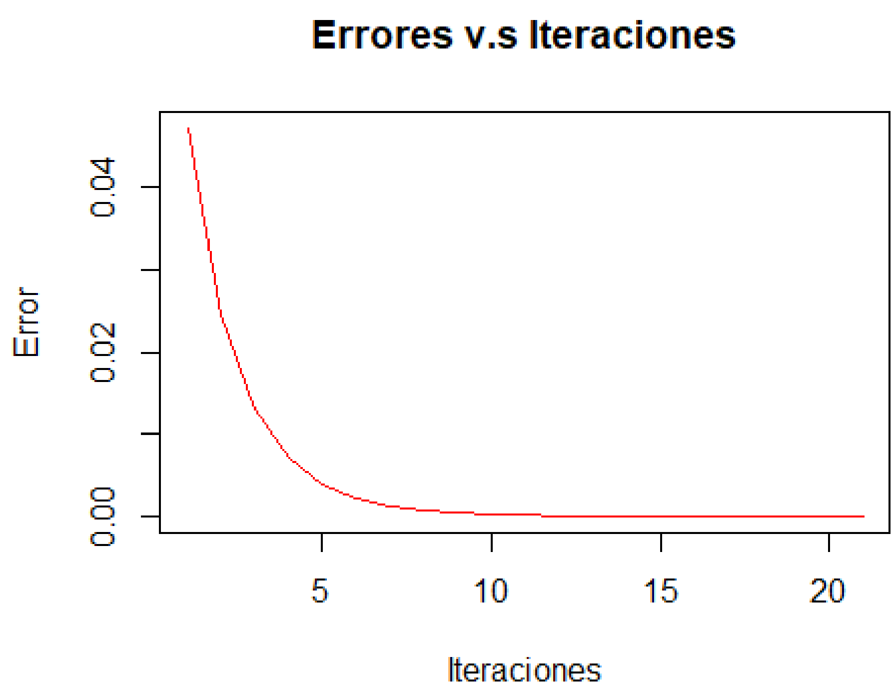
```r
#Se limpia la consola para una mejor visualización
cat("\014")
```

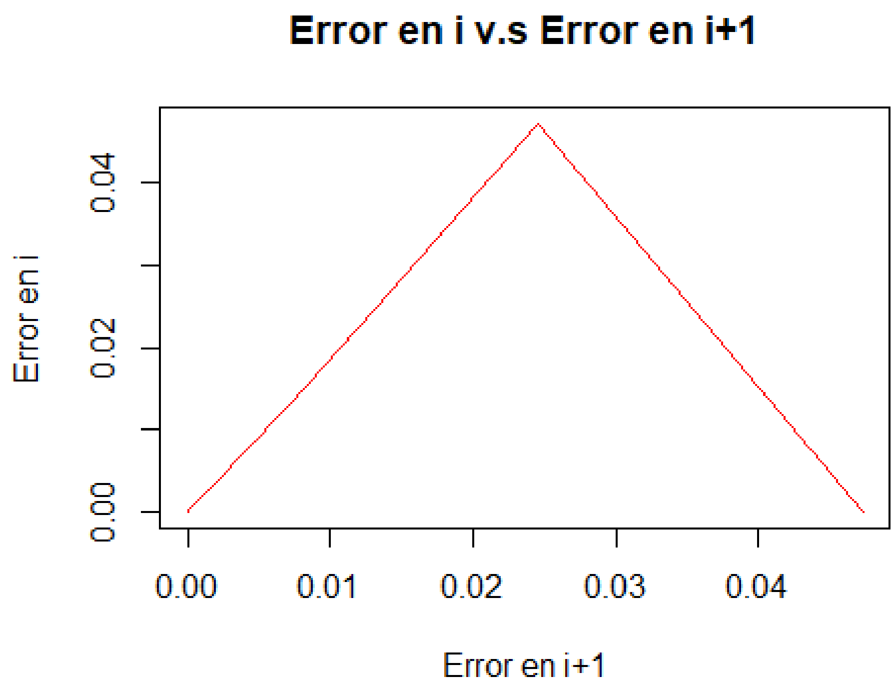
```r
g = function(x) ((exp(1)^x)/pi)
f = function(x) (log(pi*x))


punto_fijo = function(x, error_permitido)
{

  iteraciones = 0
  valores = c(0)
  errores = c(0)
  errores1 = c(0)

  while(abs(g(x) - x) > error_permitido)
  {

    iteraciones = iteraciones + 1
    r = (abs(g(x) -x )/g(x))
    errores[iteraciones] = r
    x = g(x)
    valores[iteraciones] = x
    if(iteraciones == 1){
      errores1[iteraciones] = 0
    }else{
      errores1[iteraciones] = errores[iteraciones-1]
    }

  }

  iteracion = seq(1,iteraciones)
  tabla = data.frame(iteracion,valores)
  print(tabla)
  plot(iteracion, valores, type = "l",col="red"
       ,main = "Valores v.s Iteraciones"
       ,xlab = "Iteraciones"
       ,ylab = "Valor")
  plot(iteracion, errores, type = "l",col="red"
       ,main = "Errores v.s Iteraciones"
       ,xlab = "Iteraciones"
       ,ylab = "Error")
  plot(errores, errores1 , type = "l",col="red"
       ,main = "Error en i v.s Error en i+1"
       ,xlab = "Error en i+1"
       ,ylab = "Error en i")
}
```

```r
punto_fijo(0.5, 10e-8)
```

```
##    iteracion   valores
## 1          1 0.5248043
## 2          2 0.5379845
## 3          3 0.5451221
## 4          4 0.5490269
## 5          5 0.5511750
## 6          6 0.5523602
## 7          7 0.5530153
## 8          8 0.5533776
## 9          9 0.5535782
## 10        10 0.5536892
## 11        11 0.5537507
## 12        12 0.5537848
## 13        13 0.5538036
## 14        14 0.5538141
## 15        15 0.5538199
## 16        16 0.5538231
## 17        17 0.5538248
## 18        18 0.5538258
## 19        19 0.5538264
## 20        20 0.5538267
## 21        21 0.5538268
```


Valores v.s Iteraciones
Valor
0.525
0.535
0.545
5
10
15
20
Iteraciones


Errores v.s Iteraciones
Error
0.00
0.02
0.04
5
10
15
20
Iteraciones



Secante

Este algoritmo sirve para encontrar las raíces de una función en una forma iterativa. El mismo es una variación del método de Newton, en donde en vez de calcular la derivada de la función a evaluar, se aproxima la pendiente a la recta que une a la función evaluada en un primer punto y en el punto de una iteración anterior. Este algoritmo es de mucha utilidad cuando calcular y evaluar la derivada de la función resulta muy complejo (en terminos computacionales), lo que lleva a que no se pueda utilizar el método de Newton.

En pocas palabras, este algoritmo utiliza una serie de líneas secantes para así poder aproximar el valor de una raíz de una función f.

Entradas del algortimo

1)Punto inicial de la izquierda

2)Punto inicial de la derecha

3)Error

Paso a Paso del algoritmo

1)Se entra al ciclo

2)Se evalúa $x_{i+1} = x_i - \frac{x_i - x_{i-1}}{f(x_i) - f(x_{i-1})} f(x_i)$, donde i es la iteración actual, x_{i+1} el valor de x en la iteración $i+1$, x_i el valor de x en la iteración i y x_{i-1} el valor de x en la iteración $i-1$.

3)Si el error calculado es mayor que el error permitido, entonces el proceso termina. En caso contrario, se vuelve a iniciar el proceso.

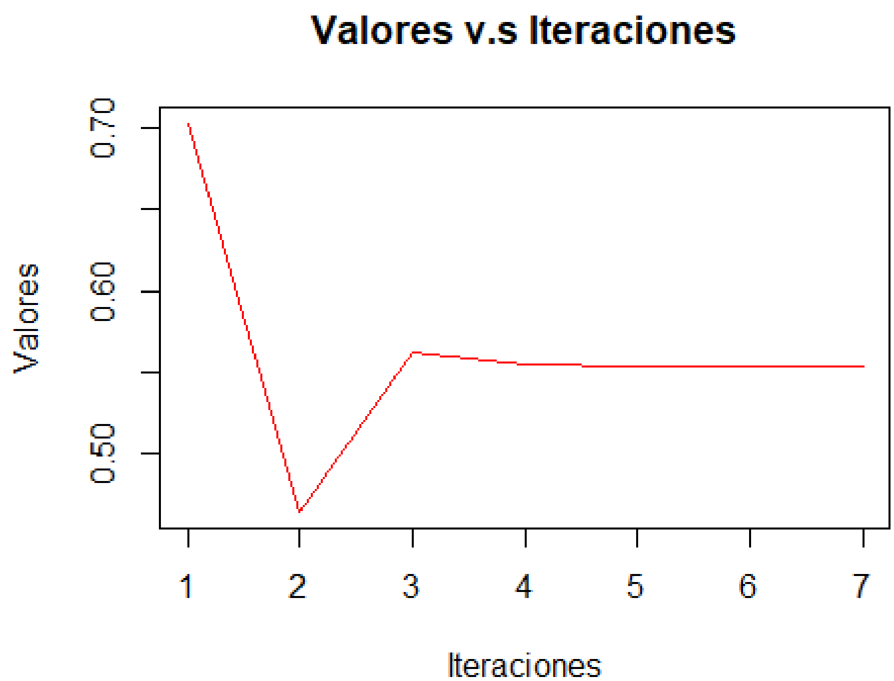
Para probar el algoritmo se utilizó la función $f(x) = e^x - \pi x$, el punto 0 como x_0 (punto inicial a la izquierda) y 1 como x_1 (punto inicial a la derecha) con una toleracia de $10e-8$. A continuación, se mostrará el código del algoritmo y sus salidas.

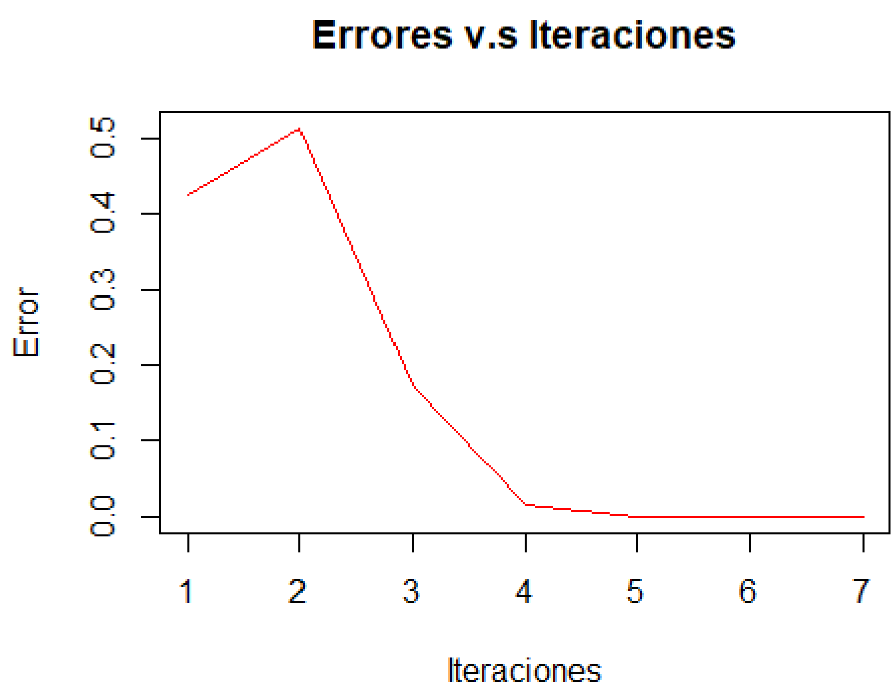
```
#Remueve los elementos que estan guardados en memoria
rm(list=ls())
#Limpia la consola para una mejor visualización
cat("\014")
```

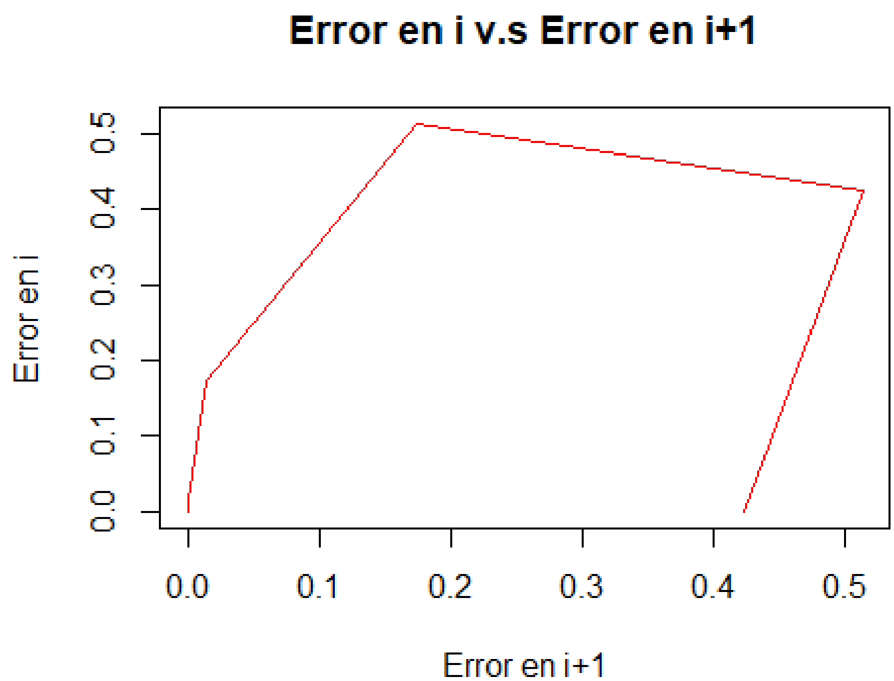
```
funcion_1 = function(x) ((exp(1)^x)-(pi*x))


secante = function(x0, x1, error_permitido)
{
  iteraciones = 0
  valores = c(0)
  errores = c(0)
  errores1 = c(0)


  repeat
  {
    iteraciones = iteraciones + 1
    x = x1 - (funcion_1(x1)*(x0 - x1))/(funcion_1(x0)-funcion_1(x1))
    error = error = abs((x - x1)/x)
    x0 = x1
    x1 = x
    valores[iteraciones] = x
    errores[iteraciones] = error

    if(iteraciones == 1){
      errores1[iteraciones] = 0
    }
    else{
      errores1[iteraciones] = errores[iteraciones-1]
    }

    if(error < error_permitido){
      break
    }
```

```r
  }
  iteracion = seq(1,iteraciones)
  tabla = data.frame(iteracion,valores)
  print(tabla)
  plot(iteracion, valores, type = "l",col="red"
       ,main = "Valores v.s Iteraciones"
       ,xlab = "Iteraciones"
       ,ylab = "Valores")
  plot(iteracion, errores, type = "l" ,col="red"
       ,main = "Errores v.s Iteraciones"
       ,xlab = "Iteraciones"
       ,ylab = "Error")
  plot(errores, errores1, type = "l",col="red"
       ,main = "Error en i v.s Error en i+1"
       ,xlab = "Error en i+1"
       ,ylab = "Error en i")
}

secante(0,1,10e-8)
```

```
##   iteracion   valores
## 1         1 0.7025872
## 2         2 0.4643490
## 3         3 0.5626182
## 4         4 0.5542804
## 5         5 0.5538245
## 6         6 0.5538270
## 7         7 0.5538270
```


Valores v.s Iteraciones
Valores
0.50
0.60
0.70
1
2
3
4
5
6
7
Iteraciones


Errores v.s Iteraciones
Error
0.0
0.1
0.2
0.3
0.4
0.5
1
2
3
4
5
6
7
Iteraciones



Steffensen

Este algoritmo se utiliza para encontrar los ceros de una función. El método de Steffensen puede considerarse como una combinación entre el algoritmo del punto fijo y el método de Aikten. Este método presenta una convergencia rápida y no requiere la evaluación de ninguna derivada (como si toca hacer en el algoritmo de Newton); sin embargo, si requiere un punto inicial para inicar el proceso, por lo que se presenta el mismo problema que se dió en el método de Newton, pues si no se elige un punto lo suficientemente cercano a la raíz, el algoritmo no convergerá (algo que también dependerá de la naturaleza de la función).

Entradas del algoritmo

1)Punto inicial

2)Error

Paso a Paso del algoritmo

1)Se calcula $f(x_0)$.

2)Se entra al ciclo.

3)Se debe calcular $x_{i+1} = x_i - \frac{[f(x_i)]^2}{f(x_i+f(x_i))-f(x_i)}$. Sin embargo, primero se calcula el denominador para ver si este se vuelve 0.

4)Si el denominador es 0 el proceso termina pues se daría una indeterminación. En caso contrario, el proceso continúa.

5)Una vez revisado el denominador, ahora si se evalúa $x_{i+1} = x_i - \frac{[f(x_i)]^2}{f(x_i+f(x_i))-f(x_i)}$

6)Se evalúa $f(x_{i+1})$

7)Si $|f(x_{i+1})| > E$ (donde E es el error permitido), el proceso termina. En caso contrario el proceso se repite desde el paso 2.

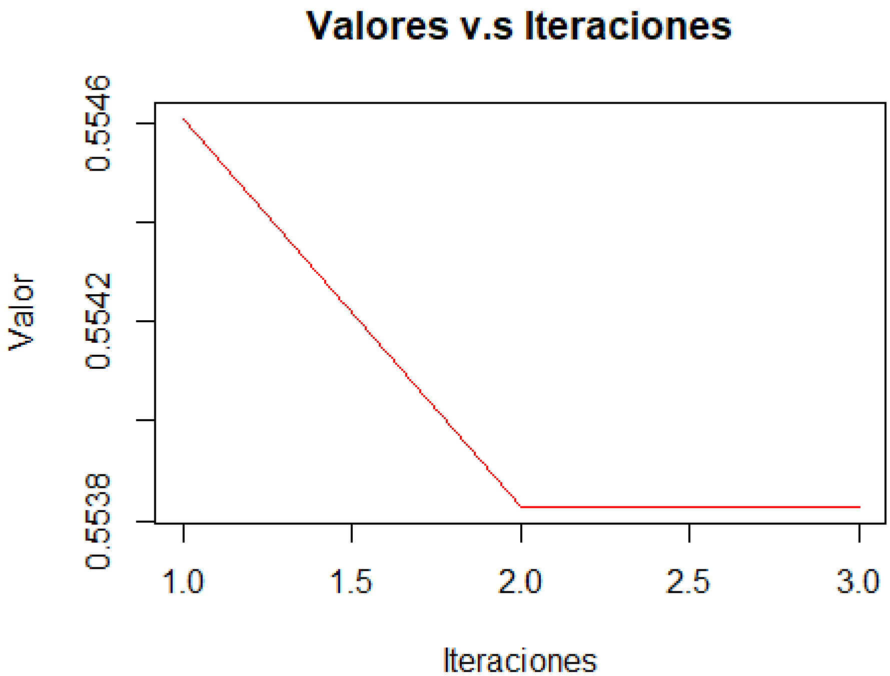
Para probar el algoritmo se utilizo la función $f(x) = e^x - \pi x$ y el valor incial 0.5 con una tolerancia de $10e - 8$. A continuación, se mostrará el código del algoritmo y sus salidas.

```
#Remueve los elementos que estan guardados en memoria
rm(list=ls())
#Limpia la consola para una mejor visualización
cat("\014")
```

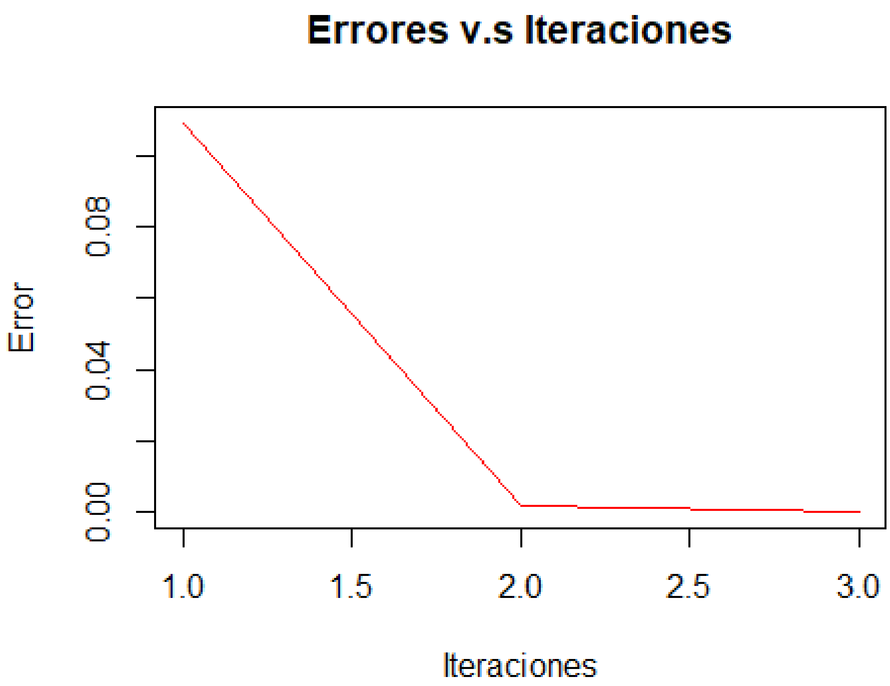
```
Fx = function(x) ((exp(1)^x)-(pi*x))

Steffensen = function(x0, error){

  iteraciones = 0
  x1 = 0
  aux = 0
  err = FALSE
  f1 = Fx(x0)
  valores = c(0)
  errores = c(0)
  errores1 = c(0)

  while(abs(f1) > error && err == FALSE){

    iteraciones = iteraciones + 1
    aux = Fx(x0 + f1) - f1

    if(aux == 0){

      cat("No se puede calcular la raiz")
      err = TRUE

    }else{

      x1 = x0-f1*f1/aux
      valores[iteraciones] = x1
      r = (abs(x0-x1)/x0)
      x0 = x1
```

```r
      f1 = Fx(x0)
      errores[iteraciones] = r
      if(iteraciones == 1){
        errores1[iteraciones] = 0
      }else{
        errores1[iteraciones] = errores[iteraciones-1]
      }
    }

  }

  if(err == FALSE){
    iteracion = seq(1,iteraciones)
    tabla = data.frame(iteracion, valores)
    print(tabla)
    plot(iteracion, valores, type = "l",col="red"
         ,main = "Valores v.s Iteraciones"
         ,xlab = "Iteraciones"
         ,ylab = "Valor")
    plot(iteracion, errores, type = "l",col="red"
         ,main = "Errores v.s Iteraciones"
         ,xlab = "Iteraciones"
         ,ylab = "Error")
    plot(errores, errores1, type = "l",col="red"
         ,main = "Error en i v.s Error en i+1"
         ,xlab = "Error en i+1"
         ,ylab = "Error en i")
  }

}

Steffensen(0.5, 10e-8)
```

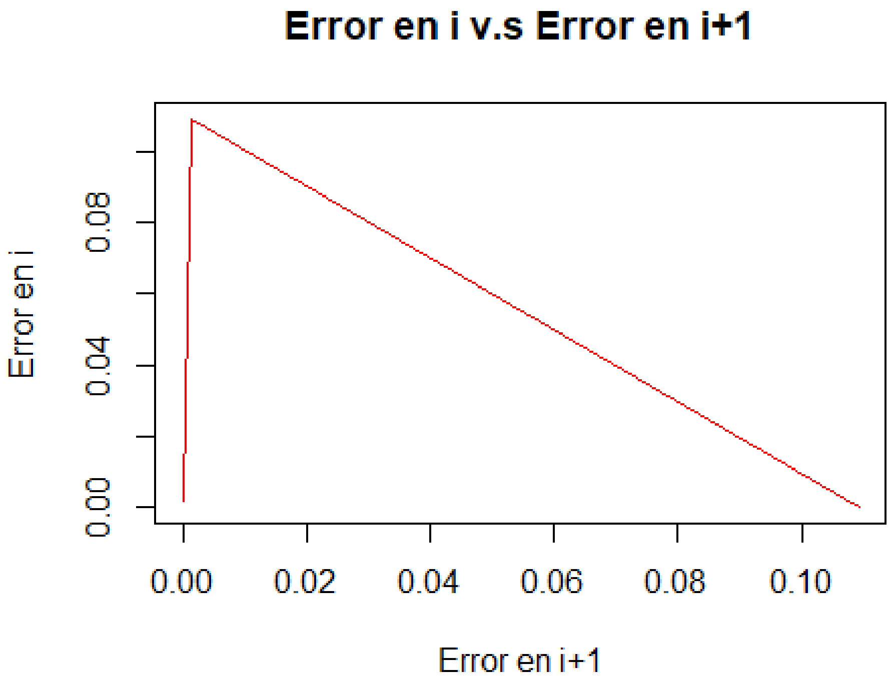
```
##   iteracion   valores
## 1         1 0.5546101
## 2         2 0.5538272
## 3         3 0.5538270
```



Valores v.s Iteraciones
Valor
0.5546
0.5542
0.5538
1.0
1.5
2.0
2.5
3.0
Iteraciones



Errores v.s Iteraciones
Error
0.08
0.04
0.00
1.0
1.5
2.0
2.5
3.0
Iteraciones


Error en i v.s Error en i+1
Error en i
0.00
0.04
0.08
0.00
0.02
0.04
0.06
0.08
0.10
Error en i+1