

# Taller 1, análisis numérico

Rafael Salvador Frieri, Daniel Hamilton Smith, Laura Juliana Mora Páez

*En el presente documento se encuentran la solución a diversos problemas planteados en la case de análisis numérico.*

## I. PUNTO 1

Evaluar el valor de un polinomio es una tarea que involucra para la maquina realizar un número de operaciones la cual debe ser mínimas. Para cada una de las siguientes polinomios, hallar  $P(x)$  en el valor indicado y el número de operaciones mínimo para hacerlo (sugerencia utilizar el algoritmo Horner).

# Se declara la función para resolver ecuaciones por método de Horner

```
Horner<-function(p,n,x0){
  y=p[1]
  sumas=0
  multis=0
  for(i in c(2:n)){
    y=x0*y+p[i]
    sumas=sumas+1
    multis=multis+1
  }
  tot=c(y,sumas,multis)
  return(tot)
}
```

# Se evalúan los polinomios y se muestra su resultado, número de operaciones óptimas y número de operaciones convencionales

```
Px=c(2,0,-3,3,-4)
n=length(Px) x0=-2
res1=Horner(Px,n,x0)
res2=Evaluation(Px,n,x0)
```

```
cat("El resultado del polinomio 2x^4-3x^2+3x-4 evaluado
en x0=-2 es:",res1[1],"nEl número de operaciones
realizadas evaluando convencionalmente fue
de:",res2[2]+res2[3]," con ",res2[2]," sumas y ",res2[3],"
multiplicaciones.\nEl número de operaciones realizadas con
el método de Horner fue de:",res1[2]+res1[3]," que es el
número de operaciones mínimo para evaluar el polinomio,
con ",res1[2]," sumas y ",res1[3]," multiplicaciones.\n\n")
```

```
Px=c(7,6,-6,0,3,-4)
n=length(Px)
x0=3
res1=Horner(Px,n,x0)
res2=Evaluation(Px,n,x0)
```

```
cat("El resultado del polinomio 7x^5+6x^4-6x^3+3x-4
evaluado en x0=3 es:",res1[1],"nEl número de operaciones
realizadas evaluando convencionalmente fue
de:",res2[2]+res2[3]," con ",res2[2]," sumas y ",res2[3],"
multiplicaciones.\nEl número de operaciones realizadas con el
método de Horner fue de:",res1[2]+res1[3]," que es el
número de operaciones mínimo para evaluar el polinomio, con
",res1[2]," sumas y ",res1[3]," multiplicaciones.\n\n")
```

```
Px=c(-5,0,3,0,2,-4,0)
n=length(Px)
x0=-1
res1=Horner(Px,n,x0)
res2=Evaluation(Px,n,x0)
```

```
cat("El resultado del polinomio -5x^6+3x^4+2x^2-4x
evaluado en x0=-1 es:",res1[1],"nEl número de operaciones
realizadas evaluando convencionalmente fue
de:",res2[2]+res2[3]," con ",res2[2]," sumas y ",res2[3],"
multiplicaciones.\nEl número de operaciones realizadas con el
método de Horner fue de:",res1[2]+res1[3]," que es el número
de operaciones mínimo para evaluar el polinomio, con
",res1[2]," sumas y ",res1[3]," multiplicaciones.\n\n")
```

El código anterior realiza el número de operaciones de un polinomio de forma convencional y con el método de Horner, compara cuales son las cantidades de operaciones utilizadas usualmente y de forma eficiente.

Ejecutándolo se obtiene:

El resultado del polinomio  $7x^5+6x^4-6x^3+3x-4$  evaluado en  $x_0=3$  es: 2030

El número de operaciones realizadas evaluando convencionalmente fue de: 20 , con 5 sumas y 15 multiplicaciones.

El número de operaciones realizadas con el método de Horner fue de: 10 , que es el número de operaciones mínimo para evaluar el polinomio, con 5 sumas y 5 multiplicaciones.

El resultado del polinomio  $-5x^6+3x^4+2x^2-4x$  evaluado en  $x_0=-1$  es: 4

El número de operaciones realizadas evaluando convencionalmente fue de: 27 , con 6 sumas y 21 multiplicaciones.

El número de operaciones realizadas con el método de Horner fue de: 12 , que es el número de operaciones mínimo para evaluar el polinomio, con 6 sumas y 6 multiplicaciones.

## II. PUNTO 2

La eficiencia de un algoritmo esta denotada por  $T(n)$

```

leer n
  Mientras n>0 repita
    d<- mod(n,2)
    n<-fix(n/2)
  Mostrar d
Fin

```

a) Recorra el algoritmo con  $n=73$ . El algoritmo utilizado fue el siguiente:

```

# Punto 2
alg<-function(n){
  cont=0
  while(n>0){
    d=n%%2
    n=floor(n/2)
    print(d)
    cont=cont+1
  }
  cat("Cantidad de iteraciones:",cont,"\n") }
# Parte a
alg(73)

```

En este punto se escribe el algoritmo propuesto por el problema y se evalúa en 73, se muestra la salida y la cantidad de iteraciones que realiza el algoritmo para dicho número (que hace la conversión a binario). b) Suponga que  $T(n)$  representa la cantidad de operaciones aritméticas de división que se realizan para resolver el problema de tamaño  $n$ . Encuentre  $T(n)$  y exprese la con la notación  $O(\cdot)$ . Para obtener  $T(n)$  observe el hecho de que en cada ciclo el valor de  $n$  se reduce aproximadamente a la mitad. Debido a que la condición para que el algoritmo termine su proceso es la división de  $n$  en 2 y sacar su parte entera hasta que este llegue a ser 0, significa que  $n$  se dividirá por 2 un número  $m$  de veces representado de la siguiente forma  $n/(2^m)$ . El proceso terminará en la iteración siguiente luego de que  $n$  sea 1 ya que es el único entero positivo cuya parte entera de la división por 2 será 0, luego  $n/(2^m)=1$  marca la cantidad  $m$  de iteraciones del algoritmo para llegar a 1, luego  $m=\log_2(n)$ . Este será el número total de iteraciones menos 1 porque aún falta la iteración en donde la parte entera de la división resulte en 0. Luego el número de iteraciones  $i$  será  $i=m+1=\log_2(n)+1$ .  $i$  no siempre será un número entero, entonces al final la cantidad de iteraciones será la parte entera de  $i$ . Si por cada iteración se realizan 2 divisiones que son la del módulo (valor residual de la división) y la división de  $n$  entre 2, la cantidad de divisiones del algoritmo será  $T(n)=2*\text{fix}(\log_2(n)+1)$ . Y su complejidad será de  $O(\log n)$ . Esto aplica si  $n>0$ , si  $n\leq 0$  simplemente la cantidad de iteraciones será 0 y por tanto la cantidad de divisiones será 0. Luego:

$$I(\#) = 2 * (0^{**} + (0^{*}, (\#) + 1)$$

Y se realiza un código para comprobar la cantidad de divisiones (siendo estás 2 veces la cantidad de iteraciones):

```

# Parte b
# Se declara la función T, número de divisiones para un n
T<-function(n){

```

```

  if(n>0){
    return(2*floor(log2(n)+1))
  }else{
    return(0)
  }
}
# Se evalúa en el valor de prueba(73) para comprobar que funciona
cat("Cantidad de divisiones:",T(73),"\n")

```

El resultado obtenido es el siguiente:

Cantidad de divisiones: 14

## III. PUNTO 3

Utilice el método de Newton para resolver el problema, muestre gráficamente cómo se comporta la convergencia a la solución.

Una partícula se mueve en el espacio con el vector de posición  **$R(t)=(2\cos(t), \sin(t), 0)$** . Se requiere conocer el tiempo en el que el objeto se encuentra más cerca de punto  **$P(2,1,0)$** . Utilice el método Newton con cuatro decimales de precisión.

# Se utiliza distancia euclidiana para saber cuando la partícula estará más cerca del punto

```

f<-function(t){sqrt((2*cos(t)-2)^2+(sin(t)-1)^2)}
# Luego se calcula su derivada para poder calcular sus
raices que serán los puntos de máxima y mínima distancia
dfdt<-function(t){(4*sin(t)-
(3*sin(t)+1)*cos(t))/(sqrt((sin(t))^2-2*sin(t)+4*(cos(t))^2-
8*cos(t)+5))}
# Se realiza método de Newton para llegar al
resultado Newton<-function(a,b,t0){
  if(dfdt(a)*dfdt(b)<0){
    error=100
    ant=t0
    cont=0
    d2fdt2=deriv(~(4*sin(t)-
(3*sin(t)+1)*cos(t))/(sqrt((sin(t))^2-2*sin(t)+4*(cos(t))^2-
8*cos(t)+5)), "t", TRUE)
    if(t0==0.5){
      plot(seq(0.58,0.61,0.0001),dfdt(seq(0.58,0.61,0.0001)),type="l",col="blue")
    }else if(t0==30){
      plot(seq(23,70,0.0001),dfdt(seq(23,70,0.0001)),type="l",col="blue")
    }else{
      plot(seq(a,b,0.0001),dfdt(seq(a,b,0.0001)),type="l",col="blue")
    }
  }
  abline(h=0,col="black")
  while(error>0.0001){
    if(attr(d2fdt2(t0),"gradient")[1]!=0){ t0=t0-
      dfdt(t0)/(attr(d2fdt2(t0),"gradient")[1])
    }else{
      print("Método no converge con dicho valor inicial")
    }
  }
}

```

```

        break
    }
    error=abs(t0-ant)/abs(t0)
    ant=t0
    text(t0,0,cont,cex=1.5,col="red")
    cont=cont+1
}
} else {
    print("Ingrese otro intervalo ya que el ingresado no
    tiene raiz única para ser calculada.")
}
return(t0)
}

```

# Debido a que la ecuación de la posición de la partícula es periódica y tiene infinitas soluciones para cuando está más cerca de la partícula se procedió a calcular y graficar el comportamiento de convergencia del primer tiempo positivo en el que sucede

```

options(digits=4)
min=Newton(0,1,0.5)
min=Newton(0,1,0.5)
cat("El primer tiempo (positivo) en el cual la distancia es
mínima es de:",min,"\nEn este tiempo la distancia de la
partícula al punto es de:",f(min),"\n")

```

Al realizar la entrada de Newton(0,1,0.5) obtenemos como salida

El primer tiempo (positivo) en el cual la distancia es mínima es de: 0.5872198 .

En este tiempo la distancia de la partícula al punto es de:  
0.5577801 .

Con la siguiente gráfica que representa la convergencia:

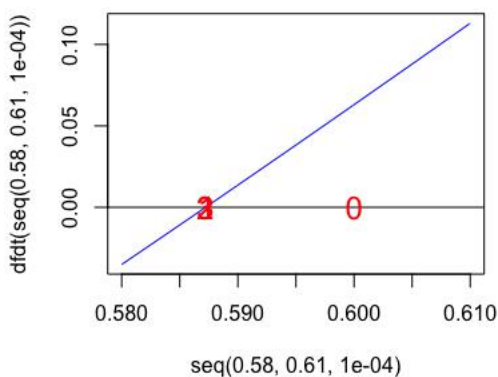


Figura 1. Gráfica de convergencia Newton

Para observar mejor la convergencia también se realizó con otro ejemplo el proceso, el otro ejemplo fue:

```

options(digits=6)
min=Newton(60,70,30)
#Se escogió x0 para que tuviera suficientes iteraciones de forma
que se pudiera observar el comportamiento de la convergencia

```

cat("Otro tiempo en el cual la distancia es mínima es de (como en el primer caso):",min,"\nEn este tiempo la distancia de la partícula al punto es de:",f(min),"\n") Y se obtuvo el siguiente resultado:

Otro tiempo en el cual la distancia es mínima es de (como en el primer caso): 63.4191 .

En este tiempo la distancia de la partícula al punto es de: 0.55778 .

Con la gráfica de convergencia:

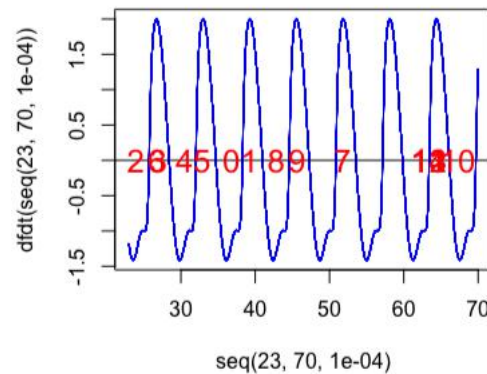


Figura 2. Gráfica de convergencia Newton segunda prueba

En la Figura 2 se puede observar mejor la gráfica en un rango más amplio y su convergencia con mayor claridad

#### IV. PUNTO 4

Resolver por dos métodos diferentes, grafique las soluciones y compare sus soluciones

Encuentre una intersección de las siguientes ecuaciones en coordenadas polares  $+ = 2 + \cos(34)$ ,  $+ = 3 - 7^8$

# Se tienen dos ecuaciones, igualándolas y dejando los términos de un solo lado resulta la siguiente ecuación

```

f<-function(t){cos(3*t)+exp(t)}

```

# Para el método de Newton la derivada de f(t) es

```

dfdt<-function(t){exp(t)-3*sin(3*t)}

```

# Se declara función para cálculo de r luego de obtener t

```

r<-function(t){2-exp(t)}

```

# Se declara otra función para el cálculo de r para poder graficar soluciones

```

r1<-function(t){2+cos(3*t)}

```

#Se resolverá por el método de la secante, método de posición falsa y método de Newton con aceleración de delta de Aitken

```

secante<-function(a,b,x0){
  if(f(a)*f(b)<0){
    error=100
    ant=x0+1
    cont=0
    while(error>0.00000001){
      if(x0!=ant){
        x1=x0-((x0-ant)/(f(x0)-f(ant)))*f(x0)
      } else {

```

```

    print("Se ha llegado a una indeterminación en
el proceso, no hay solución disponible")
    break
  }
  ant=x0
  x0=x1
  error=abs(x0-ant)/abs(x0)
  cont=cont+1
}
cat("Iteraciones método secante:",cont,"\n")
return(x0)
} else {
  print("No existe una solución en el intervalo
[a,b]")
}
}

```

```

posicionFalsa<-function(a,b){
  if(f(a)*f(b)<0){
    c=(f(b)*a-f(a)*b)/(f(b)-f(a))
    error=100
    ant=c
    cont=0
    while(error>0.00000001){
      if(f(a)*f(c)<0){
        b=c
      } else {
        a=c
      }
      if(a==b){
        break
      }
      c=(f(b)*a-f(a)*b)/(f(b)-f(a))
      error=abs(c-ant)/abs(c)
      ant=c
      cont=cont+1
    }
    cat("Iteraciones método posición
falsa:",cont,"\n")
    return(c)
  } else {
    print("No se puede encontrar una raíz a partir de
dicho intervalo.")
  }
}

```

```

deltaAitkenNewton<-function(a,b,x0){
  if(f(a)*f(b)<0){
    error=100
    cont=0
    while(error>0.00000001){
      if(dfdt(x0)!=0){
        Xn1=x0-f(x0)/dfdt(x0)
      } else {
        print("El método no converge correctamente")
        break
      }
      if(dfdt(Xn1)!=0){
        Xn2=Xn1-f(Xn1)/dfdt(Xn1)
      } else {
        print("El método no converge correctamente")
        break
      }
      if(Xn2-2*Xn1+x0!=0){
        xAt=Xn2-(Xn2-Xn1)^2/(Xn2-2*Xn1+x0)
      } else {

```

```

        break
      }
      error=abs(xAt-x0)/abs(xAt)
      x0=xAt
      cont=cont+1
    }
    cat("Iteraciones método Newton con delta de
Aitken:",cont,"\n")
    return(xAt)
  } else {
    print("No existe una solución única en el intervalo
dado, intentar uno diferente.")
  }
}
# Se usan ambos métodos para hallar una solución de
la ecuación (Preferiblemente la misma, ya que tiene
infinitas soluciones por las oscilaciones del coseno)
# Se calcula el punto de intersección con ambos
métodos y se muestran por la consola

```

```

tRaizSe=secante(-1,-0.5,-0.6)
rSe=r(tRaizSe)
cat("La solución dada por el método de la secante fue:
t=",tRaizSe,", r= ",rSe,"\n")

```

```

tRaizPF=posicionFalsa(-1,-0.5)
rPF=r(tRaizPF)
cat("La solución dada por el método de posición falsa
fue: t=",tRaizPF,", r= ",rPF,"\n")

```

```

tRaizNA=deltaAitkenNewton(-1,-0.5,-0.6)
rNA=r(tRaizNA)
cat("La solución dada por el método de Newton con
delta de Aitken fue: t=",tRaizNA,", r= ",rNA,"\n")

```

```

# Realización de la gráfica
plot(seq(-1,-0.5,0.000001),r(seq(-1,-
0.5,0.000001)),type="l",col="blue")
abline(h=0,col="black")
lines(seq(-1,-0.5,0.000001),r1(seq(-1,-
0.5,0.000001)),type="l",col="red")
points(rbind(c(tRaizSe,rSe)),pch=17,cex=1.5,col="red")
points(rbind(c(tRaizPF,rPF)),pch=17,cex=1.5,col="blue")

```

```
}
}
```

Tras ejecutar obtenemos la siguiente gráfica y salida

Iteraciones método secante: 7 .

La solución dada por el método de la secante fue:  $t = -0.6973291$ ,  $r = 1.502087$

Iteraciones método posición falsa: 8 .

La solución dada por el método de posición falsa fue:  $t = -0.6973291$ ,  $r = 1.502087$

Iteraciones método Newton con delta de Aitken: 3 .

La solución dada por el método de Newton con delta de Aitken fue:  $t = -0.6973291$ ,  $r = 1.502087$

```
angulo <- seq(-max(theta),max(theta),by=theta[2]-
theta[1])
y <- r*sin(theta)
x <- r*cos(theta)
plot.window(xlim = c(-max(r), max(r)), ylim = c(-
max(r), max(r)), asp = 1)

aux <- max(r)
while (aux > 0){
  fi <- aux*sin(angulo)
```

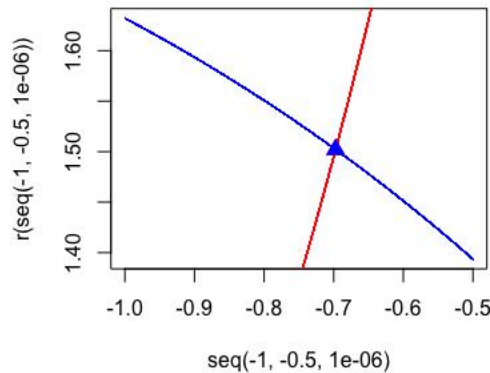


Figura 3. gráfica 1 punto 4

Se puede observar tanto en la consola como en la gráfica que las soluciones dadas por el método de posición falsa y por el método

de la Secante son muy cercanas (comienzan a diferir en luego de

puntos no se puede diferenciar el rojo del azul ya que está uno

encima del otro. Sin embargo, la solución del método de la secante fue más acertada en cantidad de cifras significativas

a la solución real luego de realizar una comparación con una calculadora que tuviera una mayor precisión. Es posible que esto se de debido a que como la convergencia del método de secante es de 1,6 cuando el error se acerca al límite dado este se acerque en su última iteración más a la solución real que el método de posición falsa que tiene convergencia lineal.

Por último se puede observar utilizando el método de delta de Aitken se obtiene una convergencia mucho más acelerada que en los otros dos casos, el cual es el objetivo de dicho método.

Para observarlo en coordenadas polares se agrego el siguiente código

```
plot.new()
polar <- function (theta, r,
  color=4){ y <- 0
  x <- 0
  ejex <- 1
  for (i in 1:length(r)){
    if(is.nan(rfil)== T){
```

```

cir <- aux*cos(angulo)
points(cir,fi,pch="-"
",col="gray",cex=0.3) text(ejex+0.2,-
0.2,ejex,col="gray") ejex <- ejex + 1
aux <- aux - 1
}

abline(v=((max(cir)+min(cir))/2),col="gray")
abline(h=((max(cir)+min(cir))/2),col="gray") segments(-
max(r)+0.5,-max(r)+0.5,max(r)-0.5,max(r)-
0.5,col="gray")
segments(-max(r)+0.5,max(r)-0.5,max(r)-0.5,-
max(r)+0.5,col="gray")

points(x,y,pch=20,col=color,cex=1)
}

dim <- seq(-1,1,by=0.01)
r2=r(dim)
polar(dim,r2,"blue")
r3=r1(dim)
polar(dim,r3,"red")

```

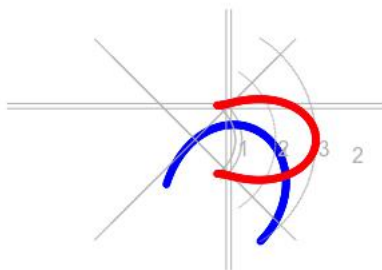


Figura 4. gráfica punto 4 en coordenadas polares

## V. PUNTO 13

Encuentre una fórmula iterativa de convergencia cuadrática y defina un intervalo de convergencia apropiado para calcular la raíz real  $n$ -ésima de un número real. El algoritmo solamente debe incluir operaciones aritméticas elementales.

Para encontrar la fórmula iterativa de convergencia cuadrática se parte de que la raíz  $n$ -ésima de  $A$  es  $B$ , luego  $n\sqrt[n]{A}=B$ , luego  $B^n=A$  y  $B^{n+1}=A$ , y se desea encontrar valor de  $B$ , por lo cual sería la variable de la función. Pasándolo a términos de  $x$ , se tendría que  $f(x)=x^n-A$  y su derivada  $f'(x)=n*x^{n-1}$ , luego se utiliza el método de Newton para cumplir con el orden de la convergencia donde  $X_{n+1}=X_n-f(X_n)/f'(X_n)$ , luego  $X_{n+1}=X_n-(X_n^n-A)/(n*X_n^{n-1})$ . Luego de manipular la ecuación, su resultado es  $X_{n+1}=(1/n)*((n-1)*X_n+A/X_n^{n-1})$ . Luego la ecuación escrita de forma matemáticamente correcta será:

Donde  $n$  es

$$n = \frac{1}{\frac{1}{n-1} + \frac{1}{n-2} + \dots + \frac{1}{1}}$$

orden de la raíz que se desea sacar,  $X_n$  es la iteración el sistema,

cual se desea calcular la raíz y  $X_{n+1}$  es la  $X$  de la siguiente iteración. Convergencia: Para la convergencia de la fórmula iterativa es necesario que el número  $n$  sea un número diferente de 0, ya que de ser 0 habrá una indeterminación al inicio de la

ecuación. Luego si  $n$  es un número entero impar positivo, a puede tomar cualquier valor, pero si este es un entero par, un número racional, un número irracional o un impar negativo es necesario que  $a$  sea un número positivo o 0 para la convergencia de la ecuación. Por último es importante para la convergencia que el número con que se comienza a iterar  $x_0$  sea distinto de 0 (o cercano) para evitar una indeterminación al inicio de las iteraciones o que el resultado aumente descontroladamente, y que si se van a calcular raíces con  $n$  negativo es importante que  $x_0$  esté cerca de la raíz, porque de otra forma por la naturaleza de la ecuación esta divergirá. En estos casos la convergencia se asegura y es además cuadrática. Luego se plantea el siguiente código para encontrar raíces y se utilizan algunos ejemplos:

```

raizN<-function(a,n,x0){ #Donde a es el número del que se
sacará raíz, n el orden de la raíz y x0 el punto donde
comienzan las iteraciones
if((n%%2==0 && x0>0)|| (n%%2!=0)){
  cont=0
  ant=x0
  while(cont<100){
    x0=(1/n)*((n-1)*x0+a/(x0)^(n-1))
    ant=x0
    cont=cont+1
  }
  print(x0)
}else{
  print("El método no presenta una solución correcta, escoger
un valor inicial distinto (si es raíz par, un número positivo)")
}
}

```

#Cálculos de algunas raíces de distinto orden y distinto  $x_0$   
 raizN(4,2,0.4)  
 raizN(-27,3,4)  
 raizN(3,5,-3)

Los resultados fueron los siguientes:

```

[1] 2
[1] -3
[1] 1.245731

```

## VI. PUNTO 14

El siguiente es un proceso intuitivo para calcular una raíz real positiva de la ecuación  $f(x)=0$  en un intervalo  $[a,b]$  con precisión  $E$ :

A partir de  $x=a$  evalúe  $f(x)$  incrementando  $x$  en un valor  $d$ . Inicialmente  $d=(b-a)/10$ . Cuando  $f$  cambie de signo, retroceda  $x$  al punto anterior  $x-d$ , reduzca  $d$  al valor  $d/10$  y evalúe nuevamente  $f$  hasta que cambie de signo. Repita este procedimiento hasta que  $d$  sea menor que  $E$ .

- a) De manera formal escriba las condiciones necesarias para que la raíz exista, sea única y pueda ser calculada.  
 b) Indique el orden de convergencia y estime el factor de convergencia del método.  
 c) Describa el procedimiento anterior en notación algorítmica, o en MATLAB o en Python.

- a) Para asegurar la existencia de una raíz en el intervalo real positivo  $[a,b]$ , se debe cumplir que  $f(a)*f(b)<0$  y que  $f$  sea continua en dicho intervalo, ya que esto denota al menos un cambio de signo de  $f$  en el intervalo. Para que dicha solución sea única, la cantidad de cambios de signo en el intervalo  $[a,b]$  de la función  $f$  debe ser exclusivamente 1, o en otras palabras que la secuencia de Strum en dicho intervalo tenga solo un cambio de signo, lo que significa que hay un único  $c$ , tal que  $f(c)=0$ . Si la función cumple con las condiciones anteriores, su raíz siempre podrá ser calculada, sin embargo su valor exacto solo se podrá hallar si se trata de un número racional, porque de lo contrario el resultado será una aproximación de la solución.
- b) El orden de convergencia del método es lineal, y el factor de convergencia (pendiente de la recta de caída del error) debe ser de  $1/10$ , ya que este es el factor mediante el cual  $z$  se va acercando o alejando de la raíz de la ecuación.
- c) Se utiliza el siguiente código para describir el proceso en R y utilizan ejemplos para demostrar su funcionamiento:

```
# Parte c
# Se declara una función para calcular su raíz (Puede ser cualquiera)
f<-function(x){exp(x)-pi*x}
# Código del proceso descrito
calcRaiz<-function(a,b,E){
  x=a
  if(a>=0 && b>0 &&
    f(a)*f(b)<0){
    x=x+d
    nuevo=f(x)
    if(ant*nuevo<0){
      x=x-d
      d=d/10
    }else{
      ant=nuevo
    }
  }else{
    print("Ingrese otro intervalo, ya que este no aplica para el método numérico")
  }
}
#Se calculan ambas raíces de la ecuación con precisión de 10^-8
calcRaiz(0,1,0.00000001)
calcRaiz(1,2,0.00000001)
```

Se obtienen los siguientes resultados:

```
[1] 0.55382703
[1] 1.63852841
```

## VII. PUNTO 15

Se propone resolver la ecuación  $\int_H^G (5 - 7^D)EF = 2$  con el método del punto fijo

- a) Obtenga la ecuación  $f(x)=0$  resolviendo el integral  
 b) Mediante un gráfico aproximado, o evaluado directamente, localice la raíces reales.  
 c) Proponga una ecuación equivalente  $x=g(x)$  y determine el intervalo de convergencia para calcular una de las dos raíces.  
 d) Del intervalo anterior, elija un valor inicial y realice 5 iteraciones. En cada iteración verifique que se cumple la condición de convergencia del punto fijo y estime el error de truncamiento en el último resultado.

# Parte a

# Resolviendo y despejando la ecuación se obtiene la siguiente función

```
f<-function(x){
  5*x-exp(x)-1
}
```

# Parte b

# Se grafica la función

```
inter=seq(0,3,0.0001)
plot(inter,f(inter),type="l",col="blue")
abline(h=0,col="black")
```

# Mediante el gráfico aproximado mostrado, se puede observar que las raíces reales se encuentran aproximadamente en un poco más de 0,5 y un poco menos de 2,5.

# Parte c

# Se propone utilizar la siguiente ecuación despejada  $g(x)=x$

```
g<-function(x){
  (1+exp(x))/5
}
```

# Para que la serie converja la magnitud de la derivada de  $g$  debe ser menor o igual a 1 en cada iteración.

# Si se deriva  $g(x)$ , se obtiene  $g'(x)=(e^x)/5$ , e igualando a 1,  $x<=\ln(5)$  para que el método converja con dicho despeje

# Parte d

# Se declara la derivada de  $g$ , que se usará para verificar que se cumple la condición de convergencia en cada iteración

```
dgdx<-function(x){
  exp(x)/5
}
```

# Se declara función del método de punto fijo

```
solicitado puntoFijo<-function(a,b,x0){
  if((g(a)>a && g(b)<b)|| (g(a)<a && g(b)>b)){
    ant=x0
    for(i in c(1:5)){
```

```

    if(abs(dgdx(x0))>1){ # Verificación de criterio de
convergencia del punto fijo
    print("Se ha dejado de cumplir la condición de
convergencia.")
    break
    }
    cat("|g'(x0)|=",abs(dgdx(x0)),"\n") # Se muestra
magnitud de derivada para verificar criterio
    x0=g(x0)
    error=abs(x0-ant)/abs(x0)
    ERROR=abs(x0-ant)
    ant=x0
    }
    return(c(x0,error,ERROR))
} else {
    print("Dicho intervalo no es apto para el método de punto
fijo, utilizar otro.")
}
}
# Se llama la función con un intervalo apropiado, y un
valor inicial que cumple con los criterios de convergencia,
luego se muestran resultados finales
res=puntoFijo(0,1,0.5)
cat("Resultado:",res[1],"\nError relativo
final:",res[2],"\nError absoluto final:",res[3])

```

Tras ejecutar el código obtenemos la siguiente salida con la siguiente gráfica:

```

|g'(x0)|= 0.3297443
|g'(x0)|= 0.3396996
|g'(x0)|= 0.3430983
|g'(x0)|= 0.3442664
|g'(x0)|= 0.3446687
Resultado: 0.5446687
Error relativo final: 0.0007387311
Error absoluto final: 0.0004023637

```

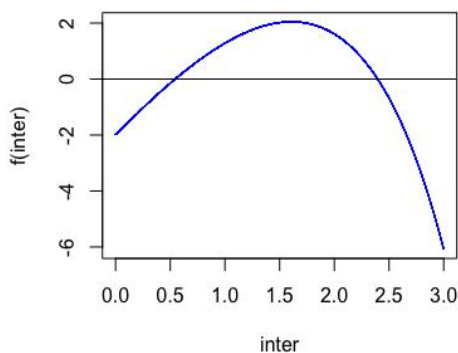


Figura 5. gráfica punto 15

A partir de los cálculos de error se puede estimar que el error relativo estuvo en el rededor de 0.000739 con respecto al último resultado y 0.000402 de error absoluto con respecto al último y penúltimo resultado. Además luego de comparar el resultado de la última iteración con un resultado más exacto dado por una calculadora de alto desempeño se encontró que el error fue aún menor, siendo de al rededor de 0.0002.