Taller Analisis Numérico

Maria Paula Covelli Reyes
David Mateo Henao Prieto
Juan Sebastian Pulecio Romero

Septiembre 26, 2020

1. Numero de Operaciones

Las operaciones fundamentales de la arimética de valores reales son la suma + y el producto , que son las operaciones necesarias para evaluar un polinomio $P(x)$ en un valor particular x.Es por esto, que es importante saber cómo se evalua un polinomio y más aun hacerlo de manera eficiente para que el número de sumas y productos requeridas sea mínima.

1.1. Evaluar el polinoio en $x = 1{,}00000000001$ con $P(x) = 1 + x + x^2 + ... + x^{50}$ y la primera derivada. Encuentre el error de cálculo al comparalo con los resultados de la expresión equivalente $Q(x) = (x^{51}1)/(x1)$

Entrada:

$$P(x) = 1 + x + x^2 + ... + x^{50}.$$
$x = 1{,}00000000001$ (Valor que se debe evaluar)
$Q(x) = (x^{51}1)/(x1)$(Expresión equivalente)

Salida:

Resultado con expeción inicial: 51,00000001225
Resultado con expresión final: 51
Con error absoluto: $1{,}22500196653164 * 10^{-8}$
Con error relativo¿$2{,}40196464025811 * 10^{-8}$ %
Error relativo:

Algorithm 1 Evaluación del polinomio

```
        rm(list=ls())
library(Rmpfr)

horner <- function(polinomio, x){

            #implementación del método de Horner, antes
            debemos conocer los coeficientes del polinomio
            a evluar

  eval <- polinomio[1]
  i <-0

  for(j in polinomio[2:length(polinomio)]){
    eval <- x*eval + j
    i <- i + 2
  }
  return (eval)
}

porderivar <- function(polinomio){

             #Esta función se encarga de encontrar los coeficientes
             de cada una de las variables a  derivar, acompañadas
             por su respectivo exponente

  grado <- length(polinomio)-1
  deriv <- c()
  for(i in polinomio[1:length(polinomio)-1]){
    aux <- i*grado
    deriv <- c(deriv, aux)
    grado <- grado - 1
  }
  return (deriv)#retorna un vector con los nuevos coeficientes (derivada)
}


polinomio = c()

for (i in 1:50)#permite crear el polinomio x+x^2+...+x^50
{
  polinomio = c(polinomio,1)
}
```

```
options(digits=15)
Q = ((1.00000000001^51)-1)/(1.00000000001-1)
P = 1+horner(polinomio,1.00000000001)

            #horner solo nos ayuda a evaluar el polinomio
            sin la suma de 1 inicial

derivada<-porderivar(polinomio)
DP=horner(derivada,1.00000000001)

cat("\nEl valor de P(x) es de: ",P)
cat("\nEl valor de la derivada de P es: ",DP)
cat("\nEl valor de Q(x) es de: ",Q)
cat("\nEl error absoluto generado entre P(X) y Q(x) Es: ",abs(P-Q),"\n")
cat("El error relativo generado entre P(x) y Q(x) Es: ",abs(P-Q)/Q*100,"%\n")
```

1.2. Numeros binarios

1.2.1. Encuentre los primeros 15 bits en la representación binaria de π

Entrada:
π

Salida:
11.10110000111 (15 primeros dijitos en binario de π)

Algorithm 2 15 primeros bits de pi en número binario

```
cat("15 primeros bits de pi en número binario")
decimalesPI= pi - 3

cat("11")#para unicamente tener que obtener los valores después de la coma
cat(".")
for (i in 1:13)#ya que tenemos dos digitos de pi por ende el ciclo solo tiene que ir hasta 1
{
  decimalesPI = decimalesPI*2;#binario es base 2

  if(decimalesPI >= 1)
  {
    cat("1")
    decimalesPI = decimalesPI - 1;
  }
  else
  {
    cat("0")
  }
}
```

1.2.2. Convertir los siguientes números binarios a base 10: 1010101; 1011.101; 10111.010101...; 111.1111...

Entrada:
1010101
1011.101
10111.010101...
111.1111...

Salida:
85
11.5
23.21
7.15

Algorithm 3 convertir numeros binarios a base 10

```
cat("Convertir números binarios a base 10\n")


parteEntera = function(b)
#retorna la parte entera del numero binario en base 10
{
  contadorPotencias = 0
  digito = 0
  entero=0

  while(b > 0)
  {
    digito = b %% 10
    entero = entero + (digito*(2^contadorPotencias))
    contadorPotencias = contadorPotencias + 1
    b = b %/% 10
  }
  return(entero)
}

parteDecimal = function(b)
#retorna la parte decimal del numero binario en base 10
{
  copiab = b
  contadorPotencias = 0
  digitos = 0
  digito = 0
  decimal=0

  while(b > 0)
  {
    digitos = digitos+1
    b = b %/% 10
  }
  contadorPotencias = -digitos;
  while(copiab > 0)
  {
    digito = copiab %% 10
    decimal = decimal + (digito*(2^contadorPotencias))
    contadorPotencias = contadorPotencias + 1
    copiab = copiab %/% 10
  }
  return(decimal)

}
```

```
cat("\nLos resultados obtenidos en base 10 son los siguientes: " )
cat("\nPrimer Numero: ",parteEntera(101010101),"\n")
cat("Segundo Numero: ",parteEntera(1011) + parteDecimal(101),"\n")
cat("Tercer Numero: ",parteEntera(10111) + parteDecimal(0101010101010101),"\n")
cat("Cuarto Numero: ",parteEntera(111) + parteDecimal(11111111111111))
```

1.2.3. Convierta los siguientes números de base 10 a binaria: $11{,}25; \frac{2}{3}, ; 30{,}6; 99{,}9$

Entrada:
11.25
2/3
30.6
99.9

Salida:
1011.11001
0.1000010
11110.110
1100011.1001

Algorithm 4 Pasar base 10 a binarios

```
cat("\nPasar base 10 a binarios\n")

parteDecimal = function(x,cantidadDeBits)
{
  numero=""
  for (i in 1:cantidadDeBits)
  {
    x = x*2;
    if(x >= 1)
    {
      numero = paste(numero,"1",sep="")
      x = x - 1;
    }
    else
    {
      numero = paste(numero,"0",sep="")
    }
  }
  return(numero)
}

parteEntera = function(x)
{
  acomulado=0
  nexp=1
  while(x>0)
  {
    digito = x%%2
    acomulado =  acomulado+nexp*digito
    nexp=nexp*10
    x = x%/%2
  }
  return (acomulado)
}
num1=11.25
num2=0.6666667
num3=30.6
num4=99.9

num1entero=trunc(11.25)

num2entero=trunc(0.6666667)

num3entero=trunc(30.6)
```

```
num4entero=trunc(99.9)

cat("Primer Numero a Base 10: ", parteEntera(num1entero),".",
parteDecimal(num1-num1entero,10), "\n")
cat("Segundo Numero a Base 10: ", parteEntera(num2entero),".",
parteDecimal (num2-num2entero,10), "\n")
cat("Tercer Numero a Base 10: ", parteEntera(num3entero),".",
parteDecimal(num3-num3entero,10), "\n")
cat("Cuarto Numero a Base 10: ", parteEntera(num4entero),".",
parteDecimal (num4-num4entero,10), "\n")
```

1.3. Representación Punto Flotante de los números relaes:

1.3.1. ¿cómo se ajusta un número binario infinito en un número finito de bits?

Para ajustar un numero binario infinito es necesario utilizar el método de truncamiento o el método de redondeo ya que estos métodos nos permiten representar el valor en un número finito de bits.

1.3.2. ¿cuál es la diferencia entre redondeo y corte?

La diferencia más representativa entre el método de corte y el de redondedo, es que el primero elimina los bits que se encuentran después de una posición determinada sin aproximar hacia arriba o hacia abajo dependiendo del numero siguiente. Mientras que, el método de redondeo, aproxima hacia arriba o hacia abajo el bit en la posición determinada. Es decir que en este caso si el bit es el número 1 este se aproxima hacia arriba, de lo contrario simplemente se aproxima hacia abajo.

1.3.3. Identifique el numero de punto flotante (IEEE) de precisión doble asociado a x, el cual se denota como fl(x); para x(0.4)

Lo primero que es necesario hacer para resolver este punto es pasar el 0.4 a binario, esto lo hacemos utilizando el metodo ya presentado anteriormente donde pasamos un número base 10 a binario, y esto nos da como resultado que: $0,4 = 0{,}100110011001100\ldots$ Pero tambien lo pocemos ver como $(1{,}00110011001100\ldots) * 10^{-1}$lo cual teniendo en cuenta que el sigo es "-1", podemos determinar que la caracteristica es 1023-1=1022. Pasando este números binarios tenemos: 1022=1111111110 en binario Y dado que se ttrabaja con doble precisión la mestisa corresponde a los 52 primeros digitos despues de la coma "," dado que hay

infinitos digitos se puede emplear el metodo de redonde para la cifra número 52 y obtenemos el valor de la matisa: Mantisa:
1001100110011001100110011001100110011001100110011001100110010
Gracias a este valor, es posible observar que se hizo un redondeo hacia arriba y que el bit es par ya que termina en 0. Y de esta manera podemos definir que el número 0.4 en el estandar 754-IEEE:
fl(0.4)=01111111110100110011001100110011001100110011001100110011001100110011010
En binario, mienttras que en base 10 tenemos un valor:
fl(0.4)=0.1000000000000000022204460492503130808472633361816406 25

1.3.4. Error de redondeo: en el modelo de la aritmética de computadora IEEE, el error de redondeo relativo no es de la mitad del épsilon de la máquina: $\frac{fl(x)-x}{x} \leq \frac{1}{2}$ Teniendo en cuenta lo anterior, encuentre el error de redondeo para x-0.4

El error relativo es equivalente a:
$e = \frac{fl(0{,}4) - 0{,}4}{0{,}4} = 5,551115^{-17}$
Podemos decir que el error relativo es menor que la mitad del error de la máquina:
$\frac{1}{2} * 2^{-52} = 1,110223^{-16}$

1.3.5. verificar si el tipo de datos básico de R y Python es de precisión doble IEEE y Revisar en R y Phython el format long

El tipo de dato basico de R y Phyton si es de doble presicion, El tipo long o float de Python y R permite almacenar números de cualquier precisión, limitado por la memoria disponible en la máquina.

1.3.6. Encuentre la representación en número de máquina hexadecimal del núemro real 9.4

El resultado es 9.66666666666, esto se debio a :
$0{,}4_10 = 0{,}66666666666_16$
$9_16 + 0{,}66666666666_16 = 9{,}66666666666_16$
$9{,}4_10 = 9{,}66666666666_16$

1.3.7. Encuentre las dos raíces de la ecuacion cuadrática x2 + 912x = 3 Intente resolver el problema usando la arimética de precisión doble, tenga en cuenta la pérdida de significancia y debe contrarestarla.

Segun lo calculado en wolfram, se encontro que las dos raices son:
x=-912.0032894618195624316239
x=0.0032894618195624316239 23921637

1.3.8. Explique cómo calcular con mayor exactitud las raíces de la ecuación: $x2+bx-10^{-12}=0$. Donde b es un numero mayor que 100

por medio del metodo de newton raphson se puede lograr calcular las raices con mayor precision,el siguiente codigo aplica el metodo, solo se debe mandar el polinomio como parametro y mostrara el resultado:

Algorithm 5 Pasar base 10 a binarios

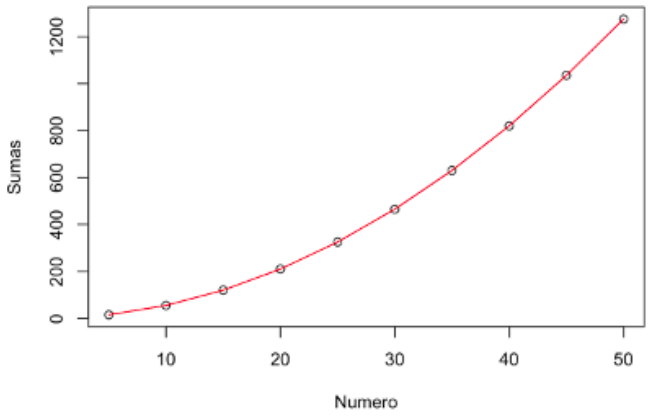
```
newton1 = function(f, fp, x0, tol, maxiter){
k = 0
# Imprimir estado

cat(formatC( c("x_k"," f(x_k)","Error est."),
width = -20, format = "f", flag = " "), "\n")
repeat{
correccion = f(x0)/fp(x0)
x1 = x0 - correccion
dx = abs(x1-x0)
# Imprimir iteraciones
cat(formatC( c(x1 ,f(x1), dx), digits=15,
width = -15, format = "f", flag = " "), "\n")
x0 = x1
k = k+1
# until
if(dx <= tol || k > maxiter ) break;
}

if(k > maxiter){
cat("Se alcanzó el máximo número de iteraciones.\n")
cat("k = ", k, "Estado: x = ", x1, "Error estimado <= ", correccion)
} else {
cat("k = ", k, " x = ", x1, " f(x) = ", f(x1),
" Error estimado <= ", correccion) }
}
## --- Pruebas
f = function(x) x-cos(x)
fp = function(x) 1+sin(x)
options(digits = 15)
newton1(f,fp, 0, 0.0000005, 10)
```

2. Raíces de una ecuación

2.1. Implemente en R o Python un algoritmo que le permita sumar u únicamente los elementos de la sub matriz triangular superior o triangular inferior, dada la matriz cuadrada An. Imprima varias pruebas, para diferentes valores de n y exprese f(n) en notaci on O() con una gra fica que muestre su orden de convergencia.

Entrada:
x= (5,10,15,20,25,30,35,40,45,50)

Salida:



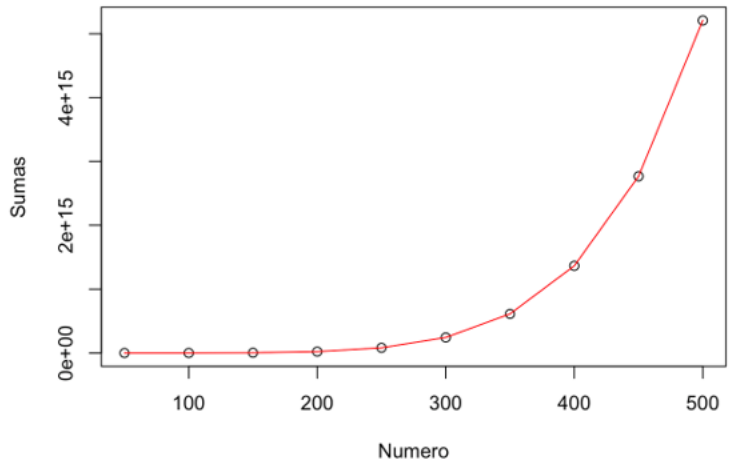
Algorithm 6 suma naturales de submatriz

```
sumaDeNaturales <- function(x)
{
  v <- c()
  for( j in x)
  {
    sum = 0

    for(y in 0:j)
    {
      sum = sum + y
    }
    v = c(v,sum)
  }
  plot(x,v,xlab="Numero",ylab="Sumas")
  lines(x, v, col = "red")



}
x<-c(5,10,15,20,25,30,35,40,45,50)

sumaDeNaturales(x)
```

2.2. Implemente en R o Python un algoritmo que le permita sumar los n2 primeros números naturales al cuadrado. Imprima varias pruebas, para diferentes valores de n y exprese f(n) en notacio n O() con una gra fica que muestre su orden de convergencia.

Entrada:
x= (50,100,150,200,250,300,350,400,450,500)

Salida:



Algorithm 7 suma naturales de n^2

```
sumaDeNaturales <- function(x)
{
  v <- c()
  for( j in x)
  {
    sum = 0

    for(y in 0:j^2)
    {
      sum = sum + y^2
    }
    v = c(v,sum)
  }
  plot(x,v,xlab="Numero",ylab="Sumas")
  lines(x, v, col = "red")



}
x<-c(50,100,150,200,250,300,350,400,450,500)

sumaDeNaturales(x)
```

2.3. Para describir la trayectoria de un cohete se tiene el modelo:

$$y(t) = 6 + 2{,}13t^2 - 0{,}0013t^4$$

Donde, y es la altura en [m] y t tiempo en [s]. El cohete esta colocado verticalmente sobre la tierra. Utilizando dos métodos de solución de ecuación no lineal, encuentre la altura máxima que alcanza el cohete.

Entrada:
$y(t) = 6 + 2{,}13t^2 - 0{,}0013t^4$

Salida:
y(t)= 877.8647 metros

Algorithm 8 trayectoria del cohete

```
trayectoria = function()
{
  t <- 0
  valor1 <- 6+2.13*(t^2)-0.0013*(t^4)
  valor2 <- 6+2.13*((t+1)^2)-0.0013*((t+1)^4)
  while(valor1<valor2)
  {

    t <- t+1
    valor1 <- 6+2.13*(t^2)-0.0013*(t^4)
    valor2 <- 6+2.13*((t+1)^2)-0.0013*((t+1)^4)

  }
  cat("altura maxima: ",valor1, "metros")

}
trayectoria()
```

3. Convergencia de metodos iterativos

3.1. Para cada uno de los siguientes ejercicios implemente en R o Python, debe determinar el numero de iteraciones realizadas,una grafica que evidencie el tipo de convergencia del método y debe expresarla en notacion O()

3.1.1. Sean f(x) = ln(x + 2) y g(x) = sin(x) dos funciones de valor real.

Utilice la siguiente formula recursiva con E = 1016 para determinar aproximadamente el punto de intersección:

$$x_n = x_{n-1} - \frac{f(x_{n-1})(x_{n-1} - x_{n-2})}{f(x_{n-1}) - f(x_{n-2})}$$

Aplicar el método iterativo siguiente con E = 108 para encontrar el punto de intersección:

$$x_{n+1} = x_n - f(x_n)\frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})}$$

Codigo solucion para ambos numerales :

Algorithm 9 punto de interseccion

```
Fx<-function(x) log(x+2)
Gx<-function(x) sin(x)

plot(Fx, xlim = c(-3,0),ylim=c(-3,0), col="red", ylab = "y")
par(new = T)
plot(Gx, xlim = c(-3,0),ylim=c(-3,0), col="blue", main = "Gráficas", ylab = "y")

#metodo posicion falsa o regula false

f <- function(x) log(x+2)-sin(x)


PosicionFalsa= function(f, xi, xf, max, t)
{
  fi = f(xi)
  ff = f(xf)
  Ei = c()
  Ej = c()
  issues = c()
  ite = c()
  i = 1
  last = xf
  cat(formatC(c("i","x_i","f(x)","Error est."),
  width = -15, format = "f", flag = " "),"\n")
  while (f(xi) != 0 )
  {
    x2 <-(xi*ff-xf*fi)/(ff-fi)
    f2 <- f(x2)
    if(abs(f2)<= t){
      break
    }
    ite <- c(ite, i)
    Error <-abs(xf - xi)
    issues <- c(issues, Error)
    if(sign(f2) == sign(fi)){
      xi <- x2
      last <- x2
      fi <- f2
    }
    else{
      xf <- x2
      last <- x2
      ff <- f2
    }
```

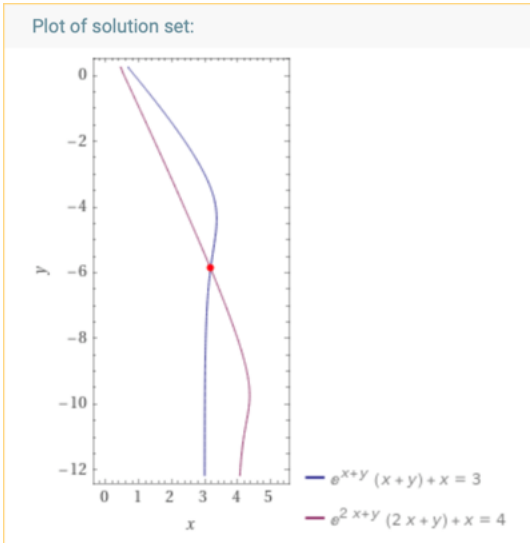
```
    cat(formatC( c(i,x2,f(x2),Error), digits = 12,
    width=-15, format = "f", flag = " "), "\n")
    i <- i+1
  }
  cat("Raíz de funcion: ",x2, " con error <=", round(abs(x2 - last),16)
  , "Iteraciones: ", i)

  cat("\nFinalmente los puntos son X:",x2," y Y igual a:",Fx(x2))
  #Errores Ei vs Ei+1
  for(b in 1:i){
    if(b!=i){
      Ei[b]<-issues[b]
      Ej[b]<-issues[b+1]
    }
  }
  plot(Ei,Ej, type = "b", xlab = "Error i",ylab="Error i+1")
}

PosicionFalsa(f,-1.5,-1,1000,1e-8)
```

3.2. Newton: Determine el valor de los coeficientes a y b tal que $f(1) = 3$ y $f(2) = 4$ con $f(x) = a + (ax + b)e^{ax+b}$. Obtenga la respuesta con $E = 10^{-8}$

Este problema se maneja por medio de la solucion de un sistema de ecuaciones dado que nos presentan la ecuacion a solucionar la cual tiene dos variables como incognitas (a y b). Dado que nos dan resultados de la evaluación de valores para x=1 y x=2 el sistema de ecuaciones queda de la siguiente manera:
$1)a + (a + b)e^(a + b) = 3$
$2)a + (2a + b)e^(2a + b) = 4$

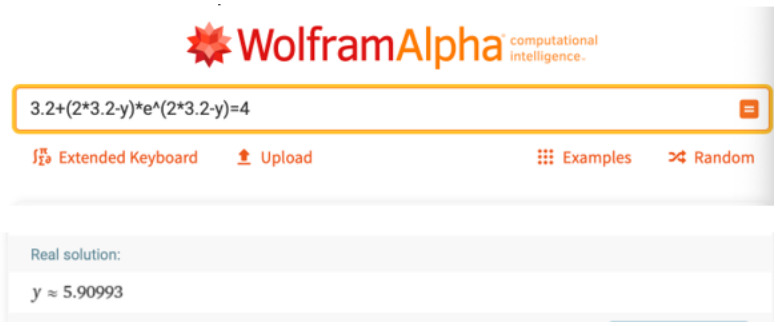
Lo cual nos da a conocer que ese sistema si cuenta con soluciones dado que tenemos la misma cantidad de incognitas y de ecuaciones; Utilizando herramientas como WolframAlpha obtenemos lo siguiente:

El sistema nos arroja la siguiente gráfica:



La gráfica mostrada anteriormente nos da posibles valores a la solución, sin embargo estos valores no cuentan con la exactitud que se desea o busca.

Sin embargo podemos ver que los valores a tratar son muy cercanos a y=-6 y a x=3, a simple vista podemos notar que el valor en x está un poco más adelantado y que el valor en y esta un poco atrasado al valor menos 6 por lo cual vamos a reemplazar el valor de x por 3.2 y así obtendremos el valor de y.



3.3. Sea $f(x) = e^x x1$

3.3.1. Demuestre que Tiene un cero de multiplicidad 2 en x = 0

Codigo solucion:

Algorithm 10 metodo newton

```
demostrarmultiplicidad<-function(Fx,x){
 valor<-Fx(x)
  cat("\nEl resultado de reemplazar ",x," en la ecuacion es: ",valor,"\n")
}

Newton = function(Fx,Gx,x0,tol,maxiter){
  listaErrorAnt = c()
  listaErrorAct = c()
  k = 0
  dx = 0
  cat(formatC( c("x"," f(x)","Error Ngen ", "Error New"),
  width = -14, format = "f", flag = " "), "\n")
  repeat{
    correccion = Fx(x0)/Gx(x0)
    x1 = x0 - correccion
    eAnterior = dx
    dx = abs(x1-x0)
    cat(formatC( c(x1 ,Fx(x1), dx,eAnterior), digits=10,
    width = -10, format = "f", flag = " "), "\n")
    x0 = x1
    k = k+1

    if(k>1){
      listaErrorAnt = c(listaErrorAnt , eAnterior)
      listaErrorAct = c(listaErrorAct, dx)
    }

    if(dx <= tol || k > maxiter ) break;}
  points(listaErrorAnt, listaErrorAct, col = "blue")
  lines(listaErrorAnt, listaErrorAct, col = "blue")
```

```
  if(k > maxiter){
    cat("Se alcanzó el máximo número de iteraciones.\n")
  }
}

Fx <- function(x){
  return (exp(1)^x - x - 1 )
}
Gx <- function(x){
  return (exp(1)^x - 1)
}
#Demostrar multiplicidad 2 cuando x=0
x<-0
demostrarmultiplicidad(Fx,x)
#-------------------------------------
plot.function(Fx, xlim=c(0,0.5), ylim=c(0,0.5),
main = "Error i+1 vs Error i",xlab = " Error i ",ylab = "
Error i+1 ",col="white")
options(digits = 10)
Newton(Fx,Gx, 2, 1e-8, 30)
```

3.3.2. Utilizando el método de Newton generalizado, mejora la tasa de rendimiento? explique su respuesta

Si mejora la tasa de rendimiento, pues es eficiente en la solución de sistemas de ecuaciones no lineales, converege de modo muy rapido y proporciona una muy buena precisión en los resultado, por tanto si se calcula usando este metodo, siempre buscara la manera mas rapida de hacerlo.

4. convergencia acelerada

4.1. Dada la sucesion $\{x_n\}_{n=0}^{\infty}$ con $x_n = cos(1/n)$

4.1.1. Verifique el tipo de convergencia en x = 1 independiente del origen

$x_n = cos(1/n)$ si converge linealmente a x=1, verificando su convergencia tenemos la definicion de esta tecnica:
Dada la sucesion $\{x_n\}_{n=0}^{\infty}$ se define la diferencia Δ_{xn} mediante:

$$\Delta_{xn} = x_n + 1 - x_n \text{ para } k \geq 2$$

Debido a la definición,

$$\begin{aligned}\Delta^2 x_n &= \Delta(x_{n+1} - p_n) = \\ &= \Delta x_{n+1} - \Delta p_n = \\ &= (x_{n+2} - x_{n+1}) - (x_{n+1} - x_n) = \\ &= x_{n+2} - 2x_{n+1} + x_n \ .\end{aligned}$$

.

Por lo tanto, la fórmula para A_n dada se puede escribir como

$$A_n = p_n - \frac{(\Delta x_n)^2}{\Delta^2 x_n} \quad \text{para toda } n \geq 0 \ .$$

de este modo verificando que x=1 si converge con $x_n = cos(1/n)$.

4.1.2. Compare los primeros terminos con la sucesion $\{A_n\}_{n=0}^{\infty}$

usando los calculos anteriormente dichos nos dan las siguientes cifras:

n	x_n	A_n
1	0.54030	0.96178
2	0.87758	0.98213
3	0.94496	0.98979
4	0.96891	0.99342
5	0.98007	0.99541
6	0.98614	
7	0.98981	

por tanto es evidente que $\{A_n\}_{n=0}^{\infty}$ converge mas rapidamente que $\{x_n\}_{n=0}^{\infty}$

4.2. Utilice el algoritmo de Steffensen para resolver $x^2 cosx$ y compararlo con el metodo de Aitken con Tolerancia de $10^8, 10^{16}$, realice una grafica que muestre la comparación entre los metodos

Codigo solucion:

Algorithm 11 algoritmo de steffensen

```
#Método Steffensen
f = function(x){
  return (x^2-cos(x))
}

Gx = function(x) return (sqrt(cos(x)))
aux =0
puntoFijo = function(a,b,i){
  if((Gx(a)-a)*(Gx(b)-b) < 0){
    x=(a+b)/2
    iteraciones = 0
    aux = 0
    dx = 0
    tol = 1e-8
    while (Gx(x) != x & iteraciones < i) {
      dx=abs(a-b)/2
      if(dx > tol){
        if (Gx(x) < x){
          b = x
        }
        else {a = x}
      }
      else {
        break
        }
      x=(a+b)/2
      aux = c(aux,x)
      iteraciones = iteraciones+1
    }
    vectorAux <<- aux
    return(x)
  }
  else{
    cat("No tiene raiz la funcion en ese intervalo\n")
  }
}
aitken = function(x,x1,x2){
  resultado = x2 - (((x2 - x1)^2)/(x2 -2*x1+x))
  return(resultado)
}

i =18
iteracion = 3
puntoFijo(0,1,i)
```

```
resultadosAitken<-c()
iteraciones<-c()
while(iteracion < i ){
  cat(iteracion,"", aitken(vectorAux[iteracion-2],
  vectorAux[iteracion-1], vectorAux[iteracion]),"\n")
  iteracion = iteracion +1
  }
cat("Resultado sin aceleracion: ", puntoFijo(0,1,i), "\n")
cat("Resultado con aceleracion: ", aitken(vectorAux[i-3],
vectorAux[i-2], vectorAux[i-1]), "\n")
cat("Valor de la solución de Wolfram: 0.82413","\n")
```

Bibliografia

Faires, B. y. (2011). Análisis Numérico. Ciudad de México: Cengage Learning.

Schnabel, D. y. (1996). Numerical Methods for uncontrained optimization and non lineal equations. No especifica: SIA.

Saleri, S. y. (2000). Numerical Mathematics . New York: Springer.

OConnor, J. L. (Septiembre de 2017). Ingeniería de los algoritmos y métodos numéricos España.

V.Muto (2005). CAPITULO X. ANALISIS DE ERROR Y TECNICAS DE ACELERACION.

V.Muto (2005). CURSO DE METODOS NUMERICOS. segunda parte. Ecuaciones de una variable: Preliminares. capitulo V