Taller 1

Por: Rafael Salvador Frieri, Daniel Hamilton-Smith y Laura Juliana Mora

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).

```
# Punto 1

# 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 declara la función para resolver ecuaciones de forma convencional
Evaluation<-function(p,n,x0){
  s=p[n]
  sumas=0
  multis=0
  for(i in c(1:(n-1))){
    s=s+p[i]*x0^(n-i)
    sumas=sumas+1
    multis=multis+n-i
  }
  tot=c(s,sumas,multis)
  return(tot)
}
# Se evaluan 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 convencinalmente 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 convencinalmente 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 convencinalmente 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 2x^4-3x^2+3x-4 evaluado en x0=-2 es: 10
El número de operaciones realizadas evaluando convencinalmente fue de: 14 , con 4 sumas y 10 multiplicaciones.
El número de operaciones realizadas con el método de Horner fue de: 8 , que es el número de operaciones mínimo para evaluar el polinomio, con 4 sumas y 4 multiplicaciones.

El resultado del polinomio 7x^5+6x^4-6x^3+3x-4 evaluado en x0=3 es: 2030
El número de operaciones realizadas evaluando convencinalmente 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 x0=-1 es: 4
El número de operaciones realizadas evaluando convencinalmente 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.

Punto 2

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

```
Leer n
Mientras n>0 repita
    d ← mod(n,2)        Produce el residuo entero de la división n/2
    n ← fix(n/2)        Asigna el cociente entero de la división 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 evalua 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 exprésela con la notación O( ). 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=log2(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=log2(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*fix(log2(n)+1). Y su complejidad será de O(log n). Esto aplica si n>0, si n<=0 simplemente la cantidad de iteraciones será 0 y por tanto la cantidad de divisiones será 0.

Luego:

$$T(n) = 2 * floor(log_2(n) + 1)$$

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 evalua 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
```

Punto 3

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

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

Se utiliza el siguiente código para llegar a la solución:

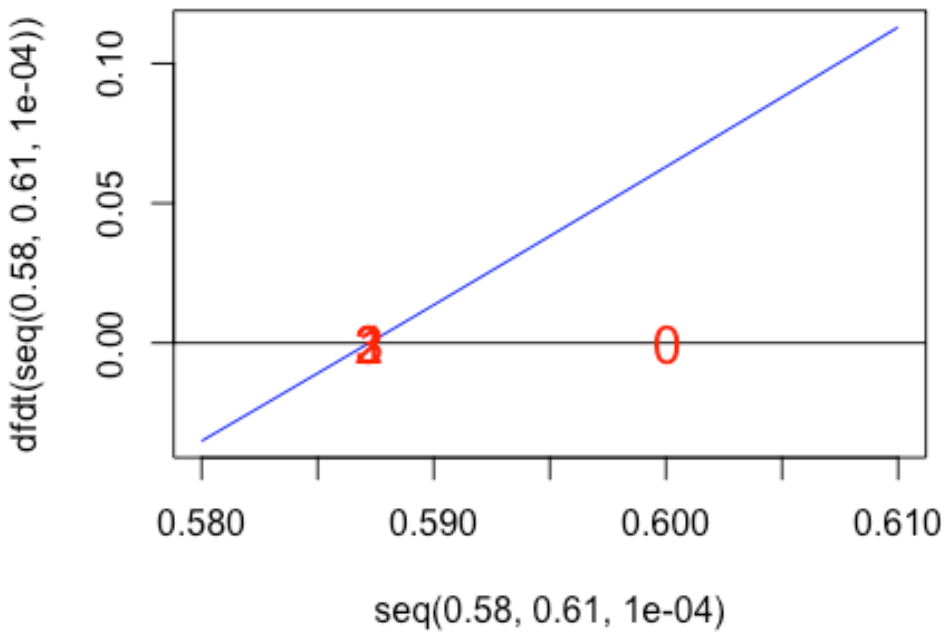
```
# Se utiliza distancia euclidiana para saber cuando la párticula estará más cerca del punto

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") #Intervalo escogido para
mejor visualización de la convergencia
    }else if(t0==30){
      plot(seq(23,70,0.0001),dfdt(seq(23,70,0.0001)),type="l",col="blue") #Intervalo escogido para mejor
visualización de la convergencia
    }else{
      plot(seq(a,b,0.0001),dfdt(seq(a,b,0.0001)),type="l",col="blue") #Caso general
    }
    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) #Se escogió el x0 para que convergiera en el primer tiempo de distancia mínima

cat("El primer tiempo (positivo) en el cual la distancia es mínima es de:",min,".\nEn este tiempo la distancia de la párticula al punto es de:",f(min),".\n")
```

Al correr el código se llega al siguiente resultado:

El primer tiempo (positivo) en el cual la distancia es mínima es de: 0.5872 .
En este tiempo la distancia de la párticula al punto es de: 0.5578 .

Con la siguiente gráfica que representa la convergencia:



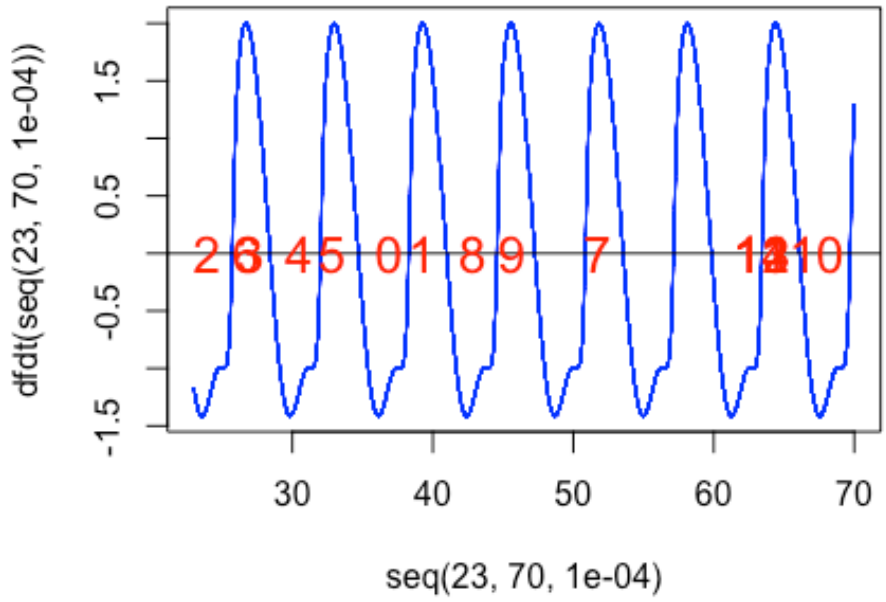
Para observar mejor la convergencia también se realizo con otro ejemplo el proceso, el otro ejemplo fue:

```
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 párticula 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 párticula al punto es de: 0.55778 .

Con la gráfica de convergencia:



Aquí se puede observar mejor la gráfica en un rango más amplio y su convergencia con mayor claridad.

Punto 4

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

Encuentre una intersección de las siguientes ecuaciones en coordenadas polares

$r = 2+\cos(3*t),\ r = 2 - e^{t}$

Se utilizó el siguiente código para encontrar la solución a la ecuación de forma analítica y se graficó la solución (usando dos métodos distintos) en coordenadas rectangulares para poder observar claramente la intersección de las curvas:

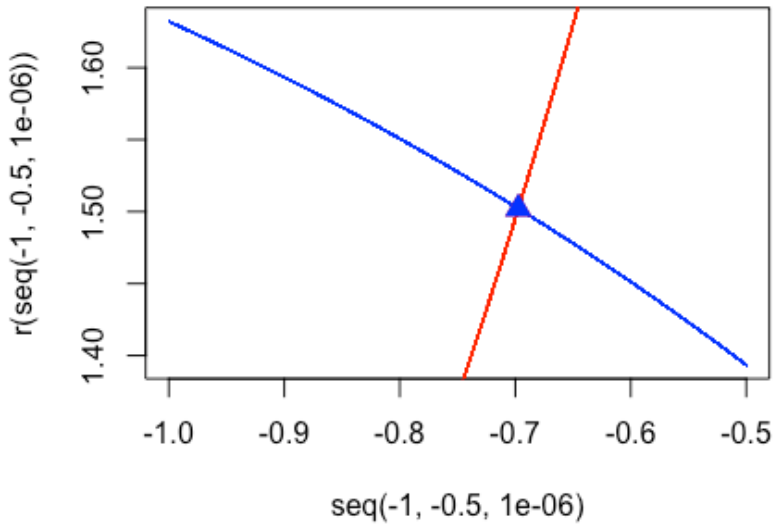
```
# Se tienen dos ecuaciones, igualándolas y dejando los términos de un solo lado resulta la siguiente ecuación
f<-function(t){cos(3*t)+exp(t)}

# 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 bisección y el método de Newton

biseccion<-function(a,b){
  if(f(a)*f(b)<0){
    c=(a+b)/2
    error=100
    ant=c
    while(error>0.00000001){
      if(f(c)==0){
        break
      }else{
        if(f(a)*f(c)<0){
          b=c
        }else{
          a=c
        }
      }
      c=(a+b)/2
      error=abs(c-ant)/abs(c)
      ant=c
    }
  }else{
    print("No hay una solución única en el intervalo escogido")
  }
  return(c)
}

Newton<-function(a,b,t0){
  if(f(a)*f(b)<0){
    error=100
    ant=t0
    while(error>0.00000001){
      if(dfdt(t0)!=0){
        t0=t0-f(t0)/dfdt(t0)
      }else{
        print("El método no converge para el t0 inicial")
      }
      error=abs(t0-ant)/abs(t0)
      ant=t0
```

```
  }
 }else{
  print("El intervalo escogido no tiene una raiz única.")
 }
 return(t0)
}

# 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

tRaizBi=biseccion(-1,-0.5)
rBi=r(tRaizBi)
cat("La solución dada por el método de bisección fue: t=",tRaizBi,", r= ",rBi,"\n")

tRaizNe=Newton(-1,-0.5,-0.6)
rNe=r(tRaizNe)
cat("La solución dada por el método de Newton fue: t=",tRaizNe,", r= ",rNe,"\n")

# Realización de la gráfica
plot(seq(-1,-0.5,0.000001),r(seq(-1,-0.5,0.000001)),type="l",col="blue")
lines(seq(-1,-0.5,0.000001),r1(seq(-1,-0.5,0.000001)),type="l",col="red")
abline(h=0,col="black")
points(rbind(c(tRaizBi,rBi)),pch=17,cex=1.5,col="red")
points(rbind(c(tRaizNe,rNe)),pch=17,cex=1.5,col="blue")
```

Y se obtuvo el siguiente resultado:

```
La solución dada por el método de bisección fue: t= -0.6973 , r=  1.502
La solución dada por el método de Newton fue: t= -0.6973 , r=  1.502
```

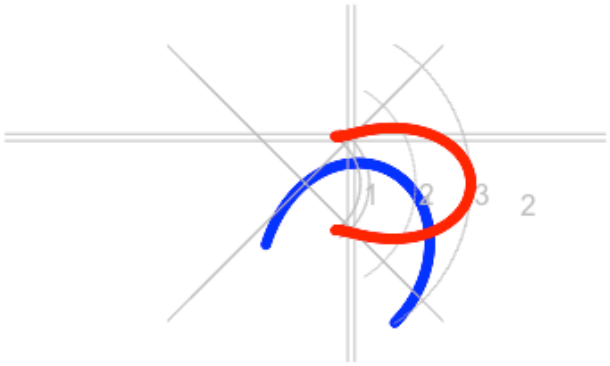
Y la gráfica de su intersección es:



Se puede observar tanto en la consola como en la gráfica que las soluciones dadas por el método de bisección y por el método de Newton son muy cercanas (comienzan a diferir en luego de de 1*10^-8), por lo cual en la gráfica cuando se grafican ambos puntos no se puede diferenciar el rojo del azul ya que está uno encima del otro. Sin embargo, la solución del método de Newton 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 Newton es cuadrática cuando el error se acerca al límite dado este se acerque en su última iteración más a la solución real que el método de bisección que tiene convergencia lineal.

Luego también se observó la intersección en coordenadas polares, la gráfica obtenida a partir de este proceso fue la siguiente (con el siguiente código para graficar):

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

  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)
    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")
```



Punto 13

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 raiz n-ésima de A es B, luego n√(A)=B, luego B^n=A y B^n-A=0, 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 Xn+1=Xn-f(Xn)/f'(Xn), luego Xn+1=Xn-(Xn^n-A)/(n*Xn^(n-1)). Luego de manipular la ecuación, su resultado es Xn+1=(1/n)*((n-1)*Xn+A/Xn^(n-1)).

Luego la ecuación escrita de forma matemáticamente correcta será:

$$X_{n+1} = \left(\frac{1}{n}\right)\left((n-1) * X_n + \frac{A}{(X_n)^{n-1}}\right)$$

Donde n es orden de la raíz que se desea sacar, Xn es la iteración donde se encuentra trabajando el sistema, A es el número del cual se desea calcular la raíz y Xn+1 es 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 x0 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 x0 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 raiz, n el orden de la raiz y x0 el punto donde comienzan las iteraciones
  cont=0
```

```
  ant=x0
  while(cont<100){
    x0=(1/n)*((n-1)*x0+a/(x0)^(n-1))
    ant=x0
    cont=cont+1
  }
  print(x0)
}

#Cálculos de algunas raices de distinto orden y distinto x0
raizN(4,2,0.4)
raizN(-27,3,4)
raizN(3,5,-3)
```

Con los siguientes resultados:

```
[1] 2
[1] -3
[1] 1.246
```

Punto 14

14. El siguiente es un procedimiento 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 raiz 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 intervarlo. 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 Sturm en dicho intervalor 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 raiz 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 x 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 raiz (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){
    ant=f(x)
    d=(b-a)/10
    while(d>E){
      x=x+d
      nuevo=f(x)
      if(ant*nuevo<0){
        x=x-d
        d=d/10
      }else{
        ant=nuevo
      }
    }
    print(x)
  }else{
    print("Ingrese otro intervalo, ya que este no aplica para el método numérico.")
  }
}

# Se calculan ambas raices de la ecuación con precision de 10^-8
calcRaiz(0,1,0.00000001)
calcRaiz(1,2,0.00000001)
```

Se obtienen los siguientes resultados:

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

Punto 15

15. Se propone resolver la ecuación $\int_0^x (5 - e^u)du = 2$ con el **método del punto fijo**

a) Obtenga la ecuación **f(x) = 0** resolviendo el integral

b) Mediante un gráfico aproximado, o evaluando 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.

La solución del punto se encuentra en el siguiente código:

```
# Punto 15

# 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 raices 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){ # Verificacion 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])
```

A partir de los cálculos de error se puede estimar que el error relativo estuvo en al 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.

Se obtuvieron los siguientes resultados a partir del código:

|g'(x0)|= 0.3297442541
|g'(x0)|= 0.3396995739
|g'(x0)|= 0.3430982813
|g'(x0)|= 0.3442663558
|g'(x0)|= 0.3446687195
Resultado: 0.5446687195
Error relativo final: 0.0007387310712
Error absoluto final: 0.0004023637066

Y la siguiente gráfica del intervalo donde se encuentran las raíces: