Parcial 2

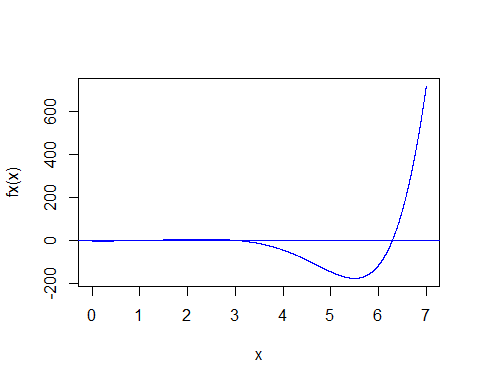
Andres David Gomez B

18 de abril de 2018

## Area entre las curvas

El codigo que permite hallar el area entre las curvas con el metodo de Simpson es: El siguiente ejercicio se ejecutará por pasos para hacer mas comprensible pero se puede integrar en una sola función. Primero se hallan los puntos de corte entre ambas funciones:

rm(list=ls())  
y1=function(x) (exp(x))\*(sin(x))  
y2=function(x) 4+cos(x+1)  
  
  
# programa busca intervalos donde es posible encontrar raices  
  
fx<- function(x) y1(x)-y2(x)  
  
x<-seq(0,7,0.001)  
plot(x,fx(x),type="l",col="blue")  
abline(h=0,col="blue")



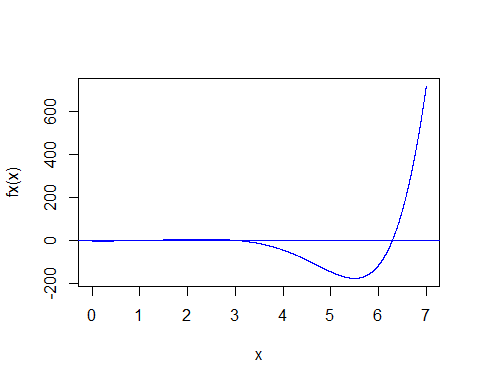
buscador <- function(a,b) {  
 i<-a  
 j<-a  
 x<-fx(a)  
 y<-fx(i)  
 while (b > i) {  
 y<-fx(i)  
 if (x\*y < 0)   
 {  
 cat("intervalo [",j," ",i,"]","\n")   
 x<-fx(i)  
 j<-i  
 }  
 i<-i+1  
 }  
}  
buscador(0,100)

## intervalo [ 0 2 ]   
## intervalo [ 2 3 ]   
## intervalo [ 3 7 ]   
## intervalo [ 7 10 ]   
## intervalo [ 10 13 ]   
## intervalo [ 13 16 ]   
## intervalo [ 16 19 ]   
## intervalo [ 19 22 ]   
## intervalo [ 22 26 ]   
## intervalo [ 26 29 ]   
## intervalo [ 29 32 ]   
## intervalo [ 32 35 ]   
## intervalo [ 35 38 ]   
## intervalo [ 38 41 ]   
## intervalo [ 41 44 ]   
## intervalo [ 44 48 ]   
## intervalo [ 48 51 ]   
## intervalo [ 51 54 ]   
## intervalo [ 54 57 ]   
## intervalo [ 57 60 ]   
## intervalo [ 60 63 ]   
## intervalo [ 63 66 ]   
## intervalo [ 66 70 ]   
## intervalo [ 70 73 ]   
## intervalo [ 73 76 ]   
## intervalo [ 76 79 ]   
## intervalo [ 79 82 ]   
## intervalo [ 82 85 ]   
## intervalo [ 85 88 ]   
## intervalo [ 88 92 ]   
## intervalo [ 92 95 ]   
## intervalo [ 95 98 ]

Como lo indica la función buscador existen multiples raíces por lo que la solución de penderá de los limites que el usuario desee. En este caso, por comodidad, se evalua el intervalo de 0 a 10.

Se modifica “buscador”" para que entregue todos los intervalos de existencia presentes entre los limtes

rm(list=ls())  
y1=function(x) (exp(x))\*(sin(x))  
y2=function(x) 4+cos(x+1)  
  
  
# programa busca intervalos donde es posible encontrar raices  
  
fx<- function(x) y1(x)-y2(x)  
  
x<-seq(0,7,0.001)  
plot(x,fx(x),type="l",col="blue")  
abline(h=0,col="blue")

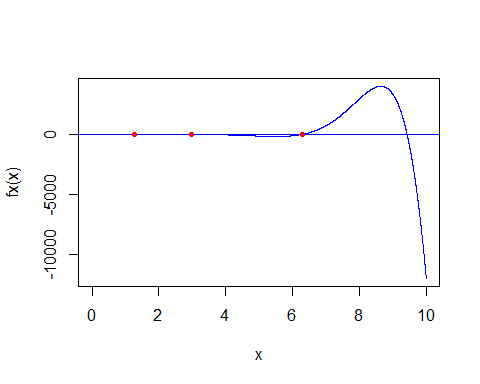


buscador <- function(a,b) {  
 i<-a  
 j<-a  
 x<-fx(a)  
 y<-fx(i)  
 resx=0  
 resy=0  
 k=1  
 while (b > i) {  
 y<-fx(i)  
 if (x\*y < 0)   
 {  
 cat("intervalo [",j," ",i,"]","\n")   
 resx[c(k)]=j  
 resy[c(k)]=i  
 k=k+1  
 x<-fx(i)  
 j<-i  
 }  
 i<-i+1  
 }  
 M=matrix(0,nrow=NROW(resx),ncol=2)  
 M[,1]=resx  
 M[,2]=resy  
 return(M)  
}  
  
M=buscador(0,10)

## intervalo [ 0 2 ]   
## intervalo [ 2 3 ]   
## intervalo [ 3 7 ]

Con los intervalos hallados se usa el metodo de newton, se elige por tener convergencia cuadratica, para hallar las raices de la función. El punto inicial es el 80 por ciento de la distancia entre los dos puntos para garantizar que las raices no se repitan y se maneja 1.e-5 como cota de error

rm(list=ls())  
y1=function(x) (exp(x))\*(sin(x))  
y2=function(x) 4+cos(x+1)  
  
  
# programa busca intervalos donde es posible encontrar raices  
  
fx<- function(x) y1(x)-y2(x)  
F1x = function(x) ((exp(x))\*(sin(x)))+((exp(x))\*(cos(x)))+(sin(x+1))  
  
x<-seq(0,10,0.001)  
plot(x,fx(x),type="l",col="blue")  
abline(h=0,col="blue")  
  
 # Metodo de Newton  
 # Halla la raiz de   
 Newton <- function(x0) {  
 x<-x0-(fx(x0)/F1x(x0))  
 error <-1  
 while (error > 1.e-5) {  
 x<-x-(fx(x)/F1x(x))  
 if (fx(x) == 0) break  
 error<-abs(fx(x)/F1x(x))  
 points(rbind(c(x,0)),pch=19,cex=0.7,col="red")  
 #cat("X=",x,"\t","E=",error,"\n")  
 }  
 cat("X=",x,"\t","E=",error,"\n")  
 return(x)  
 }  
  
  
M=matrix(c(0,2,3,2,3,7),nrow=3,ncol=2)  
  
i=1  
tam=NROW(M)  
#calcula la raiz en los intervalo de existencia encontrados y se guardan en res  
res=0  
while(i<=tam)  
{  
res[c(i)]=Newton(M[i,1]+0.8\*(M[i,2]-M[i,1]))  
 i=i+1  
}



## X= 1.261451 E= 1.927353e-07   
## X= 2.970267 E= 6.696164e-06   
## X= 6.29158 E= 7.536476e-09

Se calcula el área entre las curvas usando la regla de simpson . Se divide en 10 particiones. No olvidar que se tiene que revisar el sentido de las curvas. Si

rm(list=ls())  
y1=function(x) (exp(x))\*(sin(x))  
y2=function(x) 4+cos(x+1)  
  
fx<- function(x) y1(x)-y2(x)  
gx=function(x) y2(x)-y1(x)  
F1x <- function(x) (2\*exp(x)\*cos(x))+cos(x+1)  
  
  
simpson=function(f,a,b,n)  
{  
 if((n%%2)!=0)  
 {  
 cat("El numero de particiones no es par","\n")  
 }  
 else{  
 particion=(b-a)/n  
 x0=a  
 xn=a+particion  
 sumador=f(a)  
 i=1  
 A=0  
 while(i<n)  
 {  
 if((i%%2)!=0)  
 {  
 A=4\*f(xn)  
 sumador=sumador+A  
 }  
 else{  
 A=2\*f(xn)  
 sumador=A+sumador  
 }  
 xn=xn+particion  
 i=i+1  
 }  
 sumador=sumador+f(xn)  
 A=(b-a)/(3\*n)  
 return(A\*sumador)  
 }  
   
}  
  
  
M=matrix(c(0,2,3,2,3,7),nrow=3,ncol=2)  
tam=NROW(M)  
res=(c(1.261451,2.970267,6.29158))  
i=1  
sum=0  
while(i<=tam-1)  
{  
 a=res[i]+((res[i+1]-res[i])/2)  
 if(y2(a)<y1(a))  
 {  
 aux=simpson(fx,res[i],res[i+1],10)  
 cat("integral ",i," ",aux,"\n")  
 sum=sum+aux  
 }  
 else  
 {  
 aux=simpson(gx,res[i],res[i+1],10)  
 cat("integral ",i," ",aux,"\n")  
 sum=sum+aux  
   
 }  
   
 i=i+1  
}

## integral 1 4.796207   
## integral 2 293.7857

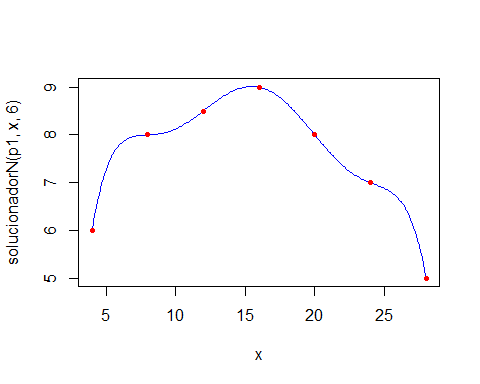
cat("El resultado de la integración es ",sum,"\n")

## El resultado de la integración es 298.5819

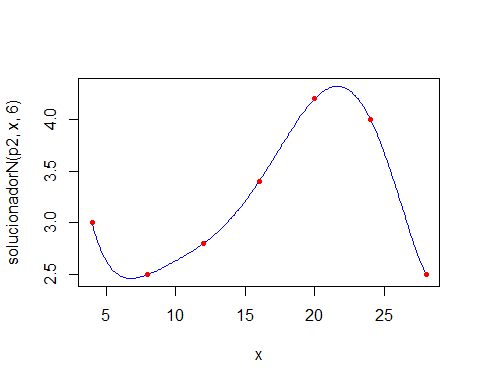
## Lago

Primero llenamos los datos del lago en dos matrices, donde tratamos la parte positiva (encima del eje de referencia) y la parte negativa(debajo del eje de referencia) segun lo explica el ejercicio. Se interpolan los puntos.

rm(list=ls())  
  
  
interNewton<-function(M){  
 res<-0  
 k<-1  
 res<-M  
 tam<-NROW(res)  
 temp<-0  
 temp[c(1)]<-res[1,2]  
 while(k<tam)  
 {  
 k<-k+1  
 j<-k  
 i<-1  
 while((j-1)<=(tam-1))  
 {  
 #up<-(res[j,2]-res[j-1,2])  
 #down<-(res[j,1]-res[i,1])  
 temp[c(j)]<-(res[j,2]-res[j-1,2])/(res[j,1]-res[i,1])  
 #temp[c(j)]<-up/down  
 i<-i+1  
 j<-j+1  
   
 }  
 #cat("" solución: ",temp,"\n")  
 res[,2]<-temp  
 }  
 return(res)  
}  
  
solucionadorN=function(p,x,n){  
 i=2;  
 j=1;  
 if((0<=n)&&(n<=(NROW(p))))  
 {  
 acum=p[1,2]  
 while(i<=n+1){  
 multi=1  
 while(j<i){  
 multi=multi\*(x-p[j,1])  
 j=j+1;  
 }  
 multi=multi\*p[i,2]  
 i=i+1;  
 j=1  
 acum=acum+multi  
 }  
 return(acum)  
 }  
 else cat("Grado no posible","\n")  
}  
  
#matriz positiva  
Mup=matrix(c(4,8,12,16,20,24,28,6,8,8.5,9,8,7,5),ncol=2,nrow=7)  
#matriz negativa  
MDown=matrix(c(4,8,12,16,20,24,28,3,2.5,2.8,3.4,4.2,4,2.5),ncol=2,nrow=7)  
p1<-interNewton(Mup)  
x<-seq(4,28,0.1)  
p2<-interNewton(MDown)  
x<-seq(4,28,0.1)  
#plot(x,p5(x,p),type="l",col="blue")  
#abline(h=0,col="blue")  
#points(rbind(M),pch=19,cex=0.7,col="red")  
#cat(solucionadorN(p,0,7))  
  
plot(x,solucionadorN(p1,x,6),type="l",col="blue")  
abline(h=0,col="blue")  
points(rbind(Mup),pch=19,cex=0.7,col="red")

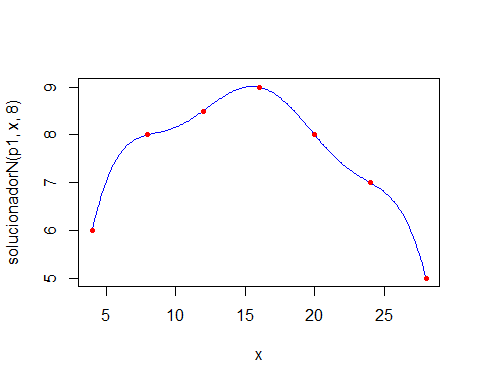


plot(x,solucionadorN(p2,x,6),type="l",col="blue")  
abline(h=0,col="blue")  
points(rbind(MDown),pch=19,cex=0.7,col="red")

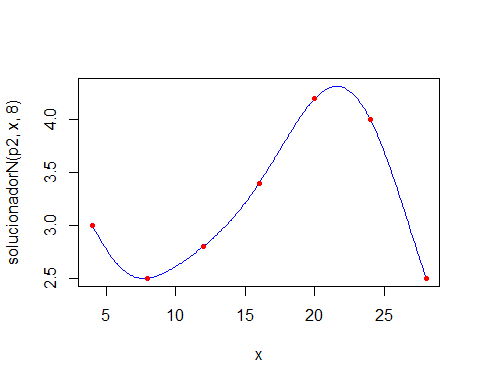


Ahora se halla el área de ambas y se suma para hallar el area total.

rm(list=ls())  
  
  
interNewton<-function(M){  
 res<-0  
 k<-1  
 res<-M  
 tam<-NROW(res)  
 temp<-0  
 temp[c(1)]<-res[1,2]  
 while(k<tam)  
 {  
 k<-k+1  
 j<-k  
 i<-1  
 while((j-1)<=(tam-1))  
 {  
 #up<-(res[j,2]-res[j-1,2])  
 #down<-(res[j,1]-res[i,1])  
 temp[c(j)]<-(res[j,2]-res[j-1,2])/(res[j,1]-res[i,1])  
 #temp[c(j)]<-up/down  
 i<-i+1  
 j<-j+1  
   
 }  
 #cat("" solución: ",temp,"\n")  
 res[,2]<-temp  
 }  
 return(res)  
}  
  
solucionadorN=function(p,x,n){  
 i=2;  
 j=1;  
 if((0<=n)&&(n<=(NROW(p))))  
 {  
 acum=p[1,2]  
 while(i<=n+1){  
 multi=1  
 while(j<i){  
 multi=multi\*(x-p[j,1])  
 j=j+1;  
 }  
 multi=multi\*p[i,2]  
 i=i+1;  
 j=1  
 acum=acum+multi  
 }  
 return(acum)  
 }  
 else cat("Grado no posible","\n")  
}  
  
#matriz positiva  
Mup=matrix(c(0,4,8,12,16,20,24,28,32,0,6,8,8.5,9,8,7,5,0),ncol=2,nrow=9)  
#matriz negativa  
MDown=matrix(c(0,4,8,12,16,20,24,28,32,0,3,2.5,2.8,3.4,4.2,4,2.5,0),ncol=2,nrow=9)  
p1<-interNewton(Mup)  
x<-seq(4,28,0.1)  
p2<-interNewton(MDown)  
x<-seq(4,28,0.1)  
#plot(x,p5(x,p),type="l",col="blue")  
#abline(h=0,col="blue")  
#points(rbind(M),pch=19,cex=0.7,col="red")  
#cat(solucionadorN(p,0,7))  
  
plot(x,solucionadorN(p1,x,8),type="l",col="blue")  
abline(h=0,col="blue")  
points(rbind(Mup),pch=19,cex=0.7,col="red")



plot(x,solucionadorN(p2,x,8),type="l",col="blue")  
abline(h=0,col="blue")  
points(rbind(MDown),pch=19,cex=0.7,col="red")



#simpson  
  
simpsonp1=function(a,b,n)  
{  
 if((n%%2)!=0)  
 {  
 cat("El numero de particiones no es par","\n")  
 }  
 else{  
 particion=(b-a)/n  
 x0=a  
 xn=a+particion  
 sumador=solucionadorN(p1,a,8)  
 i=1  
 A=0  
 while(i<n)  
 {  
 if((i%%2)!=0)  
 {  
 A=4\*solucionadorN(p1,xn,8)  
 sumador=sumador+A  
 }  
 else{  
 A=2\*solucionadorN(p1,xn,8)  
 sumador=A+sumador  
 }  
 xn=xn+particion  
 i=i+1  
 }  
 sumador=sumador+solucionadorN(p1,xn,8)  
 A=(b-a)/(3\*n)  
 return(A\*sumador)  
 }  
   
}  
  
simpsonp2=function(a,b,n)  
{  
 if((n%%2)!=0)  
 {  
 cat("El numero de particiones no es par","\n")  
 }  
 else{  
 particion=(b-a)/n  
 x0=a  
 xn=a+particion  
 sumador=solucionadorN(p2,a,8)  
 i=1  
 A=0  
 while(i<n)  
 {  
 if((i%%2)!=0)  
 {  
 A=4\*solucionadorN(p2,xn,8)  
 sumador=sumador+A  
 }  
 else{  
 A=2\*solucionadorN(p2,xn,8)  
 sumador=A+sumador  
 }  
 xn=xn+particion  
 i=i+1  
 }  
 sumador=sumador+solucionadorN(p1,xn,8)  
 A=(b-a)/(3\*n)  
 return(A\*sumador)  
 }  
}  
positiva=simpsonp1(0,32,100)  
negativa=simpsonp2(0,32,100)  
cat("El área curvas es ", negativa+positiva)

## El área curvas es 302.0079

1. S modifica m para llegar al error de truncamiento