\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Monte Carlo Integration\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Entering the functions

|  |
| --- |
| #I=Integrate\_{0}^{Infinity} (abs|cos(x)|/x\*exp^(-(log(x)-3)^2)) dx  #f(x)=exp^(-x) g(x)/f(x)= Integrate\_{0}^{Infinity} (abs|cos(x)|/x\*exp^(-(log(x)-3)^2+x)) dx  f3<- function(X){abs(cos(X))/X\*exp(-1\*((log(X)-3)^2))} #Original Formula  f3a <- function(X){abs(cos(X))/X\*exp(-1\*((log(X)-3)^2-X))} #Pick Random Variable X following Exp~(1)  f3b <- function(X){abs(cos(X))\*sqrt(2\*pi)\*exp(-1/2\*((log(X)-3)^2))} #Pick Random Variable X following Lognormal~(3,1)  curve(f3, 0, 200,xlab= "Q3 function", ylab = "y=f(x)") #plot the curve  #g(x)/f(x)=abs|cos(x)|/x\*exp^(-(log(x)-3)^2+x) |

#Monte Carlo Integration

Integral and the error term choosing X following exponential distribution with mean 1.

|  |
| --- |
| #Exponential  mcie <- function(n,f){ #Set up a function of simulation  U <- runif(n) # Simulate n random numbers  X <- -log((1-U)) #Simulate n values from Exp(1)  Int <- c(mean(f(X)), var(f(X))/n) #Compute g(x\_i)/f(x\_i)=x\_1 for i = 1 to n and the error estimation  Int} #Calculate the sample mean to estimate the population mean  set.seed(77960)  mcie(5000000,f3a) |

Integral and the error term choosing X following lognormal distribution with mean 3 and variance 1.

|  |
| --- |
| #Lognormal  mciln <- function(n,f){ #Set up a function of simulation  X <- rlnorm(n, meanlog=3, sdlog=1) #Simulate n values from Lognormal(0,1)  Int <- c(mean(f(X)), var(f(X))/n) #Compute g(x\_i)/f(x\_i)=x\_1 for i = 1 to n and the error estimation  Int} #Calculate the sample mean to estimate the population mean  set.seed(77960)  mciln(5000000,f3b) |

#Calculate the running time

|  |
| --- |
| runtime=function(fn){  start\_time=Sys.time()  fn  end\_time=Sys.time()  return(-1\*(start\_time-end\_time))  }  runtime(mcie(100000,f3))  runtime(mciln(100000,f3)) |

Results

|  |  |  |
| --- | --- | --- |
| > set.seed(77960)  > mcie(5000000,f3a)  [1] 0.9318097 0.3014599 | > set.seed(77960)  > mciln(5000000,f3b)  [1] 1.128507e+00 1.079433e-07 | > runtime(mcie(100000,f3))  Time difference of -0.118232 secs  > runtime(mciln(100000,f3))  Time difference of -0.03117204 secs |

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Quadrature\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#By substitution Define the function and draw the curve

|  |
| --- |
| fy=function(y){  result=(1/(y\*(1-y)))\*abs(cos((1-y)/y))\*exp(-(log((1-y)/y)-3)^2)  return(result)  }  curve(fy,0,1, ylab = "y=f(x)") #sketch the curve in 0 to 1  curve(fy,0,0.2, ylab = "y=f(x)") #sketch the curve in 0 to 0.2 |

#1.Composite trapezoidal rule

|  |
| --- |
| CTrape<-function(funct,a,b,n){ #Define the parameters #n is the no. of subinterval  h <- (b - a) / n #Calculate the increment  c <- 1:(n - 1) #Create a vector for the order of the increment  xj <- a + c \* h #Transform the order vector to a vector of f(x\_j)  Integral <- (1/2\*h) \* (2 \* sum(funct(xj)))  #Composite Trapezoidal Rule= h/2+[f(a)+Summation\_c=1\_n-1(f(x\_j)+f(b))]    value<-function (funct, x, adj=h/1e6) { #Set up an adjustment valueCompromise on a small error to avoid the infinity value on 0  if (is.nan(funct(x))) { #if condition when f(x)=NaN  if (x==0) return (funct(adj+x)) #add adjustment value to x\_1 when x=0  else return (funct(adj\*(-1)+x)) #prevent any NaN cases of x\_j!=0  }  else {return(funct(x))}  }    Integral=Integral+(h/2\*(value(funct,a)+value(funct,b))) #Calculate the final Integral  (Result= Integral)  }  set.seed(77960)  CTrape(fy,0.0005,1,2000) |

#2. Composite Simpson's rule

|  |  |
| --- | --- |
| CSimp<-function(funct,a,b,n) { #Define the parameters #n is the no. of subinterval  h <- (b - a) / n #Calculate the increment  c<-0:(n-1)  d <- 1:(n - 1) #Create a vector for the order of the increment  xj1 <- a+(c+1/2)\*h #Transform the order vector to a vector of f(x\_j)  xj2<- a+(d\*h)  Integral <- (1/6\*h) \* (4 \* sum(funct(xj1))+2\*sum(funct(xj2)))  #Calculate parts of the formula of Simpson's rule   |  | | --- | | value <- function (funct, x, adj=h/1000000) {  #Set up an adjustment value #Compromise on a small error to avoid the infinity value on 0  if (is.nan(funct(x))) {  if (x==0) return (funct(adj+x))  else return (funct(adj\*(-1)+x))  }  else {return(funct(x))}  }    Integrand=Integral+(h/6\*(value(funct,a)+value(funct,b)))  #Composite Simpson's Rule= h/6\*[f(a)+4\*Summation\_i=0\_n-1(f(a+(i+1/2)\*h))+2\*Summation\_i=1\_n-1(f(a+i\*h)+f(b)]  (Result= Integral) #return the result  }  set.seed(77960)  CSimp(fy,0.0005,1,2000) | |

#Calculate the running time

|  |
| --- |
| runtime=function(fn){  start\_time=Sys.time()  fn  end\_time=Sys.time()  return(end\_time-start\_time)  }  CTrape(fy,0,1,1e6)  runtime(CTrape(fy,0.0005,1,2000))  CSimp(fy,0,1,1e6)  runtime(CSimp(fy,0.0005,1,2000)) |

Results

|  |  |  |
| --- | --- | --- |
| > CTrape(fy,0.0005,1,2000)  [1] 1.123706 | > CSimp(fy,0.0005,1,2000)  [1] 1.134448 | > runtime(CTrape(fy,5e-4,1,2000))  Time difference of 0.0001599789 secs  > runtime(CSimp(fy,5e-4,1,2000))  Time difference of 0.0002520084 secs |

#By maximization Define the function and draw the curve

|  |
| --- |
| f3 <- function(x) {return((abs(cos(x))/x)\*exp(-(log(x)-3)^2)) }#function of question 3  f3d2<- function(x) {return(x^5\*exp(-1\*(log(x)^2)-9)\*(2\*sin(x)\*dist.delta(cos(x))-abs(cos(x)))  +2\*x^3\*exp(-1\*(log(x)^2)-9)\*abs(cos(x))\*(2\*(log(x)^2)-9\*log(x)+x\*(2\*log(x)-5)\*tan(x)+9))}  curve(f3d2, 0,1000 , ylab = "y=f(x)") #sketch the curve  curve(f3, 0, 50 , ylab = "y=f(x)") #sketch the curve |

#1.Composite trapezoidal rule

|  |
| --- |
| CTrape<-function(funct,a,b,n){ #Define the parameters #n is the no. of subinterval  h <- (b - a) / n #Calculate the increment  c <- 1:(n - 1) #Create a vector for the order of the increment  xj <- a + c \* h #Transform the order vector to a vector of f(x\_j)  Intergrand <- (1/2\*h) \* (2 \* sum(funct(xj)))  #Composite Trapezoidal Rule= h/2+[f(a)+Summation\_c=1\_n-1(f(x\_j)+f(b))]    value <- function (funct, x, adj=h/1000000) { #Set up an adjustment valueCompromise on a small error to avoid the infinity value on 0  if (is.nan(funct(x))) { #if condition when f(x)=NaN  if (x==0) return (funct(adj+x)) #add adjustment value to x\_1 when x=0  else return (func(adj\*(-1)+x)) #prevent any NaN cases of x\_j!=0  }  else {return(funct(x))}  }    Intergrand=Intergrand+(h/2\*(value(funct,a)+value(funct,b))) #Calculate the final intergrand  (Result= Intergrand)  }  set.seed(77960)  CTrape(f3, 1/1e7, 5000, 10000000) |

#2. Composite Simpson's rule

|  |  |
| --- | --- |
| CSimp<-function(funct,a,b,n) { #Define the parameters #n is the no. of subinterval  CSimp<-function(funct,a,b,n) { #Define the parameters #n is the no. of subinterval  h <- (b - a) / n #Calculate the increment  c<-0:(n-1)  d <- 1:(n - 1) #Create a vector for the order of the increment  xj1 <- a+(c+1/2)\*h #Transform the order vector to a vector of f(x\_j)  xj2<- a+(d\*h)  Intergrand <- (1/6\*h) \* (4 \* sum(funct(xj1))+2\*sum(funct(xj2)))  #Calculate parts of the formula of Simpson's rule   |  | | --- | | value <- function (funct, x, adj=h/1000000) { #Set up an adjustment valueCompromise on a small error to avoid the infinity value on 0  if (is.nan(funct(x))) {  if (x==0) return (funct(adj+x))  else return (func(adj\*(-1)+x))  }  else {return(funct(x))}  }    Intergrand=Intergrand+(h/6\*(value(funct,a)+value(funct,b)))  #Composite Simpson's Rule= h/6\*[f(a)+4\*Summation\_i=0\_n-1(f(a+(i+1/2)\*h))+2\*Summation\_i=1\_n-1(f(a+i\*h)+f(b)]  return(Intergrand)  }  set.seed(77960)  CSimp(f3, 1/1e7, 5000, 10000000) | |

#Calculate the running time

|  |
| --- |
| runtime=function(fn){  start\_time=Sys.time()  fn  end\_time=Sys.time()  return(-1\*(start\_time-end\_time))  }  runtime(CTrape(f3, 1/1e7, 5000, 10000000))  runtime(CSimp(f3, 1/1e7, 5000, 10000000)) |

Results

|  |  |  |
| --- | --- | --- |
| > CTrape(f3, 1/1e7, 5000, 10000000)  [1] 1.128537 | > CSimp(f3, 1/1e7, 5000, 10000000)  [1] 1.128537 | > runtime(CTrape(f3, 1/1e7, 5000, 10000000))  Time difference of 0.8330648 secs  > runtime(CSimp(f3, 1/1e7, 5000, 10000000))  Time difference of 1.493879 secs |