PRACTICALS-R

RAHUL GOSWAMI
M.sc Statistics and Computing(DST-CIMS)
Semester 1st
Class Roll no. 18
Examination Roll no. 18419STC019

SUBMITTED TO

DR. RAKESH RANJAN SIR

PRACTICAL 1

```
#-------------------------------------------PRACTICAL NO. 1---------------------------------------#
# Obtain the following result for the given dataset.
# 0,0,1,1,1,1,2,2,2,2,3,3,3,3,3,4,4,4,4,5,5,5,5,5,5,5,5,6,6,6,6,6,6,6,6,7,7,7,
# 8,8,8,8,8,8,8,9,9,9,9,9,9,10,10,10,10,10,10,11,11,11,12,12,12,12,12,12,12,12,
# 13,13,13,13,13,14,14,14,14,14,14,14,14,15,15,15,16,16,16,16,16,16,16,16,17,17,
# 17,19,19,19,19,19
#
#(a) Mean,Median,Mode
#(b)Variance
#(c)Absolute deviation about mean and Median
#(d)Skewness and Kurtosis

#------------------------------------------------SOLUTION--------------------------------------------#
#Putting the data in a vector
x<-c(0,0,1,1,1,1,2,2,2,2,3,3,3,3,3,4,4,4,4,5,5,5,5,5,5,5,5,6,6,6,6,6,6,6,6,7,7,7,
     8,8,8,8,8,8,8,9,9,9,9,9,9,10,10,10,10,10,10,11,11,11,12,12,12,12,12,12,12,12,
     13,13,13,13,13,14,14,14,14,14,14,14,14,15,15,15,16,16,16,16,16,16,16,16,17,17,
     17,19,19,19,19,19)

#---------------------------Basic function to run the program smoothly ----------------------#

#Basic function for length
r.length<-function(x)
{
  length<-0;
  for(i in x)
  {
    length<-length+1
  }
  as.numeric(length)
}
#Basic function for Sum
r.sum<-function(x)
{
  kum<-0;
  for(i in x)
  {
    kum=kum+i
  }
  as.numeric(kum)
}
#Basic function for Unique
r.uni<-function(x)
{
  uni<-NULL;
  while(r.length(x)!=0)
```

```
  {
    key<-x[1]
    y<-key==x
    uni<-c(uni,key)
    z<-x[!y]
    x<-z
  }
  uni
}
#Basic function for sorting (Using Quick sort)
qs <- function(vec)
{

  if(r.length(vec) > 1)
  {

    pivot <- vec[1]
    low <- qs(vec[vec < pivot])
    mid <- vec[vec == pivot]
    high <- qs(vec[vec > pivot])

    c(low, mid, high)


  }

  else vec

}
#Basic function for taking absolute
r.abs<-function(x)
  {
    for(i in 1:r.length(x))
    {
      if(x[i]<0)
        x[i]<-(-x[i])
    }
  x

  }
#--------------------Part(a)-------------------------------------------
#Creating Mean function
r.mean<-function(x)
{
  r.sum(x)/r.length(x)
}
#Creating Median Function
r.median<-function(x)
{
  qs(x)
```

```
  if(r.length(x) %% 2==0)
    median<-(x[r.length(x)/2]+x[(r.length(x)/2)+1])/2
  else
    median<-x[(r.length(x)+1)/2]
  median
}
#Creating Mode Function
rahul.ka.mode.function<-function(x)
  {
    d<-NULL;
    counter<-0;
    mode<-NULL;
    a<-NULL
    b<-NULL
    c<-NULL
    d<-NULL
    e<-NULL
    f<-NULL
    g<-NULL
    h<-NULL
    for(i in x)
    {
      a<-i==x
      b<-sum(a)
      d<-c(d,b)
    }
    for(i in d)
    {
      a<-i>=d
      if(r.sum(a)==r.length(d))
      {
        e<-i

      }
      else
      {

      }
    }
    for(i in d)
    {
      if(e==i)
      {
        counter<-counter+1
        f<-c(f,counter)
      }
      else
      {
        counter<-counter+1
      }
```

```
    }
    for(i in 1:length(f))
    {
      g<-x[f[i]]
      h<-c(h,g)
    }

  mode<-r.uni(h)
  mode
  }
#----------------------Part(b)------------------------------------#
#Creating Function for Variance
r.variance<-function(x)
  {
    meantimes<-NULL
    for(i in r.length(x))
    {
      meantimes<-c(meantimes,r.mean(x))
    }
    (r.sum((x-meantimes)*(x-meantimes)))/r.length(x)

  }
#------------------------Part(c)------------------------------------------#
#Creating function for absolute deviation about mean

r.absdevmean<-function(x)
  {
  meantimes<-NULL
  for(i in r.length(x))
  {
    meantimes<-c(meantimes,r.mean(x))
  }
  (r.sum(r.abs(x-meantimes)))/r.length(x)
  }

#Creatin function for absolute deviation about median

r.absdevmedian<-function(x)
  {
    mediantimes<-NULL
    for(i in r.length(x))
    {
      mediantimes<-c(mediantimes,r.median(x))
    }
    r.median(r.abs(x-mediantimes))
  }
#--------------------Part(d)--------------------------------------------------------

#Creating function for Skewness
r.skewness<-function(x)
```

```
  {
    meantimes<-NULL
    for(i in r.length(x))
    {
      meantimes<-c(meantimes,r.mean(x))
    }
    (r.sum((x-meantimes)*(x-meantimes)*(x-
meantimes))/r.length(x))/r.variance(x)^(3/2)
  }

#Creating functions for kurtosis
r.kurtosis<-function(x)
{
  meantimes<-NULL
  for(i in r.length(x))
  {
    meantimes<-c(meantimes,r.mean(x))
  }
  (r.sum((x-meantimes)^4)/r.length(x))/r.variance(x)^(2)
}
```

```
r.mean(x)
[1] 9.47

> rahul.ka.mode.function(x)
[1]  5  6 12 14 16

> r.median(x)
[1] 9

> r.variance(x)
[1] 26.5291

> r.absdevmean(x)
[1] 4.4194

> r.absdevmedian(x)
[1] 0

> r.skewness(x)
[1] 0.03849586

> r.kurtosis(x)
[1] 1.970708
```

PRACTICAL 2

```
#-------------------------PRACTICAL NO. 2-------------------------------------
# Evaluate the integral from 0 to 1 of the function 1/(1+x^2) w.r.t x by following
# methods
#
#(a) Trapezoidal rule
#(b) Simpon's 1/3 rule
#(c) Simpon's 3/8 rule
#(d) Weddle's rule
#----------------------------SOLUTION------------------------------------------


#---------------------Basic function to run the program smoothly ----------------

#Basic function for length
r.length<-function(x)
{
  length<-0;
  for(i in x)
  {
    length<-length+1
  }
  as.numeric(length)
}
#Basic function for Sum
r.sum<-function(x)
{
  kum<-0;
  for(i in x)
```

```
  {
    kum=kum+i
  }
  as.numeric(kum)
}
#Basic function for Unique
r.uni<-function(x)
{
  uni<-NULL;
  while(r.length(x)!=0)
  {
    key<-x[1]
    y<-key==x
    uni<-c(uni,key)
    z<-x[!y]
    x<-z
  }
  uni
}
#Basic function for sorting (Using Quick sort)
qs <- function(vec)
{

  if(r.length(vec) > 1)
  {

    pivot <- vec[1]
    low <- qs(vec[vec < pivot])
    mid <- vec[vec == pivot]
    high <- qs(vec[vec > pivot])
```

```
    c(low, mid, high)



  }

  else vec

}
#Basic function for taking absolute
r.abs<-function(x)
{
  for(i in 1:r.length(x))
  {
    if(x[i]<0)
      x[i]<-(-x[i])
  }
  x

}




#-----------------------------------FUNCTION---------------------------------------------#
#Creating function
f<-function(x)
  {
    1/(1+x^3)
  }
#------------------------------------Part(a)------------------------------------------
#Creating function to approximate (trapezoidal)
```

```
r.trapezoidal<-function(f,start,end,n)
 {
   distance<-(end-start)/n
   z<-f(seq(start,end,distance))
   (distance/2)*(z[1]+z[r.length(z)]+2*(r.sum(z[c(-1,-r.length(z))])))
 }

#------------------------------------Part(b)-------------------------------------------
#Creating function to approximate (Simpsons 1/3)

r.simpsons_one_by_three <-function(f,start,end,n)
 {
   distance<-(end-start)/n
   z<-f(seq(start,end,distance))
   res<-0
   for(i in 1:r.length(z))
   {
    if (i == 1 || i==r.length(z))
    {
      res <- res + z[i]
    }
    if(i%%2 ==0)
    {
      res<- res+4*z[i]
    }
    else
    {
      res<-res+2*z[i]
    }
```

```
    }
    res*(distance/3)
  }
#------------------------------------Part(c)------------------------------------------
#Creating function to approximate (Simpsons 3/8)

r.simpsons_three_by_eight <-function(f,start,end,n)
{
  distance<-(end-start)/n
  z<-f(seq(start,end,distance))
  res<-z[1]+z[r.length(z)]
  p<-z[c(-1,-r.length(z))]
  for(i in 1:r.length(p))
  {
    if(i%%3 ==0)
    {
      res<- res+2*p[i]
    }
    else
    {
      res<-res+3*z[i]
    }

  }
  res*((3*distance)/8)
}
#------------------------------------Part(d)------------------------------------------
#Creating function to approximate (Weddle)

r.weddle <-function(f,start,end,n)
{
```

```
  distance<-(end-start)/n
  z<-f(seq(start,end,distance))
  res<-0
  for(i in 1:r.length(z))
  {
    if(i ==1 || i==r.length(z))
    {
      res<- res+z[i]
    }
    if(i%%4==0)
    {
      res<-res+6*z[i]
    }
    if(i%%2==0 && i%%4 !=0)
    {
      res<-res+5*z[i]
    }
    else
    {
      res<-res+z[i]
    }

  }
  res*((3*distance)/10)
}
```

```
> r.trapezoidal(f,0,1,500)
[1] 0.8356486

> r.simpsons_one_by_three(f,0,1,1000)
[1] 0.8366488
```

```
> r.simpsons_three_by_eight(f,0,1,2000)
[1] 0.8358049


> r.weddle(f,0,1,25000)
[1] 0.8774553
```

PRACTICAL 3

#--PRACTICAL NO. 3--

Consider any non-singular square matrix A of p*p dimension where p>0 and find the following

results

#

#(a) A+A^T   : here T is denoting transpose

#(b) A-A^T   : here T is denoting transpose

#(c) Determinant of A

#(d) Inverse and Adjoint of A

#--SOLUTION--

#------------------------------Basic function to run the program smoothly ------------------------

```
#Basic function for length
r.length<-function(x)
{
  length<-0;
  for(i in x)
  {
    length<-length+1
  }
  as.numeric(length)
}
#Basic function for Sum
r.sum<-function(x)
{
  kum<-0;
  for(i in x)
```

```
  {
    kum=kum+i
  }
  as.numeric(kum)
}
#Basic function for Unique
r.uni<-function(x)
{
  uni<-NULL;
  while(r.length(x)!=0)
  {
    key<-x[1]
    y<-key==x
    uni<-c(uni,key)
    z<-x[!y]
    x<-z
  }
  uni
}
#Basic function for sorting (Using Quick sort)
qs <- function(vec)
{

  if(r.length(vec) > 1)
  {

    pivot <- vec[1]
    low <- qs(vec[vec < pivot])
    mid <- vec[vec == pivot]
    high <- qs(vec[vec > pivot])
```

```
    c(low, mid, high)



  }

  else vec

}
#Basic function for taking absolute
r.abs<-function(x)
{
  for(i in 1:r.length(x))
  {
    if(x[i]<0)
      x[i]<-(-x[i])
  }
  x

}




#-------------------------------------Part (a & b)----------------------------------------------
#Creating function to transpose

r.transpose<-function(matrix)
  {
    for(i in 1:dim(matrix)[1])
    {
      for(j in i:dim(matrix)[2])
      {
```

```
      s<-matrix[i,j]
      matrix[i,j]<-matrix[j,i]
      matrix[j,i]<-s
     }
    }
    matrix
  }
#-------------------------------------Part (c)----------------------------------------------
#Creating function to determinant

r.determinant<-function(x)
{
  a<-dim(x);
  if(a[1] == 1 && a[2] == 1)
    return(x[1,1])
  if(a[1]==2 && a[2]==2)
    return(x[1,1]*x[2,2]-x[1,2]*x[2,1])
  else
  {
    det<-0
    for(i in 1:a[1])
    {
      det<-det+(-1)^(1+i)*x[1,i]*r.determinant(x[-1,-i])
    }
  }
  return(det)
}
#-------------------------------------Part (d)----------------------------------------------
#Creating function to Adjoint

r.adjoint<-function(x)
```

```
{
  a<-dim(x);
  if(a[1] == 1 && a[2] == 1)
    return(x[1,1])
  if(a[1]==2 && a[2]==2)
  {
    t<-x[1,1]
    x[1,1]<-x[2,2]
    x[1,2]<--x[1,2]
    x[2,1]<--x[2,1]
    r.transpose(x)
  }
  else
  {
    b<-x
    for(i in 1:dim(b)[1])
    {
      for(j in 1:dim(b)[2])
      {
        x[i,j]<-((-1)^(i+j))*r.determinant(b[-i,-j])

      }
    }
    r.transpose(x)
  }

}

#Creating function for Inverse
r.inverse<-function(x)
  {
```

```
  if(r.determinant(x)==0)
  {
    print("Non invertible function")
  }
  else
  {
    r.adjoint(x)/r.determinant(x)
  }
}
```

```
> A<-matrix(1:16,4,4)
> A
     [,1] [,2] [,3] [,4]
[1,]    1    5    9   13
[2,]    2    6   10   14
[3,]    3    7   11   15
[4,]    4    8   12   16
```

```
> A+r.transpose(A)
     [,1] [,2] [,3] [,4]
[1,]    2    7   12   17
[2,]    7   12   17   22
[3,]   12   17   22   27
[4,]   17   22   27   32
```

```
> A-r.transpose(A)
     [,1] [,2] [,3] [,4]
[1,]    0    3    6    9
[2,]   -3    0    3    6
[3,]   -6   -3    0    3
[4,]   -9   -6   -3    0
```

```
> r.determinant(A)
[1] 0
```

```
> r.adjoint(A)
     [,1] [,2] [,3] [,4]
[1,]    0    0    0    0
[2,]    0    0    0    0
[3,]    0    0    0    0
[4,]    0    0    0    0
```

```
> r.inverse(A)
[1] "Non invertible function"
```

PRACTICAL 4

```
#------------------------------Practical no. 4-------------------------------------#
#Find the root of x-exp(-x)=0 using following methods
#
#(a)Bisection
#(b)Newton Ralphson
#(c)Regula Falsi
#
#----------------------------------SOLUTION------------------------------------#
#---------------------Basic function to run the program smoothly ----------------

#Basic function for length
r.length<-function(x)
{
  length<-0;
  for(i in x)
  {
    length<-length+1
  }
  as.numeric(length)
}
#Basic function for Sum
r.sum<-function(x)
{
  kum<-0;
  for(i in x)
  {
    kum=kum+i
```

```
  }
  as.numeric(kum)
}
#Basic function for Unique
r.uni<-function(x)
{
  uni<-NULL;
  while(r.length(x)!=0)
  {
    key<-x[1]
    y<-key==x
    uni<-c(uni,key)
    z<-x[!y]
    x<-z
  }
  uni
}
#Basic function for sorting (Using Quick sort)
qs <- function(vec)
{

  if(r.length(vec) > 1)
  {

    pivot <- vec[1]
    low <- qs(vec[vec < pivot])
    mid <- vec[vec == pivot]
    high <- qs(vec[vec > pivot])

    c(low, mid, high)
```

```
  }

  else vec

}
#Basic function for taking absolute
r.abs<-function(x)
{
  for(i in 1:r.length(x))
  {
    if(x[i]<0)
      x[i]<-(-x[i])
  }
  x

}
#-----------------------------Part(a)---------------------------------------------
#Creating function
  y<-function(x)
  {
    x-exp(-x);
  }
#Creating function for bisection method

bisection<-function(y,m,n)
{
  if(y(m)*y(n)<0)
  {
    g<-(m+n)/2
    if(abs(y(g))>0.001 && y(g)>0)
```

```
    {
      n<-g
      bisection(y,m,n)
    }
    else if(abs(y(g))>0.001 && y(g)<0)
    {
      m<-g
      bisection(y,m,n)
    }
    else
    {
      print(g)
    }
  }
  else
  {
    print("invalid input")
  }

}

#---------------------------------Part(c)---------------------------------------
#Creating function for regula falsi
  f<-function(x)
  {
    x-exp(-x);
  }

regula<-function(f,m,n)
{
  if(f(m)*f(n)<0 )
```

```
 {
   slope<-(f(m)-f(n))/(m-n)
   g<-m-(f(m)/slope)
   if(abs(f(g))>0.001 && f(g)>0)
   {
     n<-g
     regula(f,m,n)
   }
   else if(abs(f(g))>0.001 && f(g)<0)
   {
     m<-g
     bisection(f,m,n)
   }
   else
   {
     print(g)
   }
 }
 else
 {
   print("invalid input")
 }

}
```

```
> bisection(y,0,2)
[1] 0.5673828
> regula(y,0,2)
[1] 0.5673202
```

Practical 5

#------------------------------Practical no. 5---#

#Using the given bivariate data obtain the following result

#

#--

X   12.4 14.3 14.5 14.9 16.1 16.9 16.5 15.4 22.4 19.4 15.5 16.7 17.3 18.4 19.2 17.4 17.0 17.9 18.8 20.3 19.5 19.7 21.2

#--

Y   11.2 12.5 12.7 13.1 14.1 14.8 14.4 13.4 19.6 16.9 14.0 14.6 15.1 16.1 16.8 15.2 14.9 15.6 16.4 17.7 17.0 17.2 18.6

#--

#

#(a) Karl Pearson Correlation Coefficient

#(b) Spearman's Rank Correlation

#(c) Regression Line of X on Y

#(d) Regression Line of Y on X

#(e) Scatterplot of X and Y and also draw the regression lines on same plot

#----------------------------SOLUTION--

#Putting the value of X any Y in vector x and y respectively

```
x<-c(12.4,14.3,14.5,14.9,16.1,16.9,16.5,15.4,22.4,19.4,15.5,16.7,17.3,18.4,19.2,17.4,17.0,17.9,18.8,20.3,19.5,19.7,21.2)

y<-c(11.2,12.5,12.7,13.1,14.1,14.8,14.4,13.4,19.6,16.9,14.0,14.6,15.1,16.1,16.8,15.2,14.9,15.6,16.4,17.7,17.0,17.2,18.6)
```

#--------------------Basic function to run the program smoothly ----------------

```
#Basic function for length
r.length<-function(x)
{
 length<-0;
 for(i in x)
 {
  length<-length+1
 }
 as.numeric(length)
}
#Basic function for Sum
r.sum<-function(x)
{
 kum<-0;
 for(i in x)
 {
  kum=kum+i
 }
 as.numeric(kum)
}
#Basic function for Unique
r.uni<-function(x)
{
 uni<-NULL;
 while(r.length(x)!=0)
 {
  key<-x[1]
  y<-key==x
  uni<-c(uni,key)
  z<-x[!y]
  x<-z
```

```
  }
  uni
}
#Basic function for sorting (Using Quick sort)
qs <- function(vec)
{

  if(r.length(vec) > 1)
  {

    pivot <- vec[1]
    low <- qs(vec[vec < pivot])
    mid <- vec[vec == pivot]
    high <- qs(vec[vec > pivot])

    c(low, mid, high)



  }

  else vec

}
#Basic function for taking absolute
r.abs<-function(x)
{
  for(i in 1:r.length(x))
  {
    if(x[i]<0)
      x[i]<-(-x[i])
  }
```

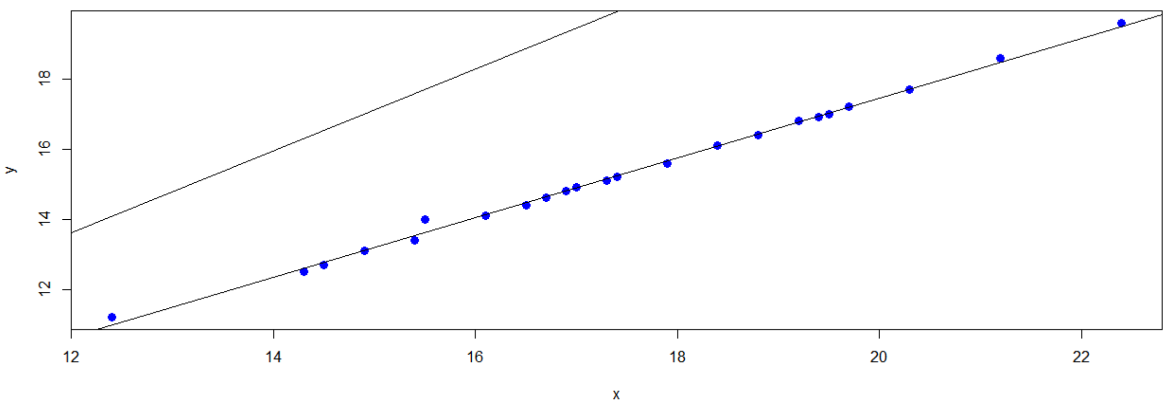
```
  x

}
#Creating Mean function
r.mean<-function(x)
{
  r.sum(x)/r.length(x)
}
#-----------------------------------------------Part(a)---------------------------------------------------
#Calculating function to calculate Covariance
r.covariance<-function(x,y)
  {
    (r.sum((x-r.mean(x))*(y-r.mean(y))))/r.length(x)
  }
#Calculating function to calculate Standard deviation
r.sd<-function(x)
{
  ((r.sum((x-r.mean(x))^2))/r.length(x))^(0.5)
}
#Calculating function to calculate Karl pearson correlation coefficient
r.kpcc<-function(x,y)
{
  r.covariance(x,y)/(r.sd(x)*r.sd(y))
}
#-----------------------------------------------Part(b)---------------------------------------------------
#Creating function to Rank
r.rank<-function(x)
{
  i<-1
  while(i<=length(x))
  {
```

```
    t<-qs(x)

    for(j in 1:r.length(x))

    {

      x[j]<-r.mean((r.sum(t<x[j]))+(1:r.sum(t==x[j])))

    }

    i<-i+1

  }

  x

}

#Creating function to calculate Spearman's Rank correlation coefficient

r.rc<-function(x,y)

  {



      l<-r.length(x)

      d<-r.sum((r.rank(x)-r.rank(y))^2)



      1-((6*d)/(l*(l^2-1)))



  }
```

```
> x
 [1] 12.4 14.3 14.5 14.9 16.1 16.9 16.5 15.4 22.4 19.4 15.5 16.7 17.3 18.4
19.2 17.4 17.0 17.9
[19] 18.8 20.3 19.5 19.7 21.2
> y
 [1] 11.2 12.5 12.7 13.1 14.1 14.8 14.4 13.4 19.6 16.9 14.0 14.6 15.1 16.1
16.8 15.2 14.9 15.6
[19] 16.4 17.7 17.0 17.2 18.6
> r.covariance(x,y)
[1] 4.743913
> r.sd(x)
[1] 2.360842
> r.sd(y)
[1] 2.012353
> r.kpcc(x,y)
[1] 0.9985405
```

```
> r.rank(x)
 [1]  1  2  3  4  7 10  8  5 23 18  6  9 12 15 17 13 11 14 16 21 19 20 22
> r.rank(y)
 [1]  1  2  3  4  7 10  8  5 23 18  6  9 12 15 17 13 11 14 16 21 19 20 22
```

```
> r.rc(x,y)
[1] 1


> lm(x ~ y)

Call:
lm(formula = x ~ y)

Coefficients:
(Intercept)            y
    -0.4582       1.1715

> lm(y ~ x)

Call:
lm(formula = y ~ x)

Coefficients:
(Intercept)            x
     0.4346       0.8511


> plot(x, y, pch = 16, cex = 1.3, col = "blue")
> abline(lm(y ~ x))
> abline(lm(x ~ y))
```



PRACTICAL 6

```
#------------------------------------Practical 6 --------------------------------------#
#Sort the following numbers usig the given algorithm
#(a) Bubble Sort algorithm
#(b) Insertion Sort
#(c) Recursive Sort
#
# 5.637,4.942,4.861,3.469,5.009,7.702,5.473,3.613,3.444,4.509,5.171,3.680,2.365
# -4.959,5.030,4.815,4.564,4.224,4.426,4.471

#---------------------------SOLUTION------------------------------------------
#Putting the value in x
x<-c(5.637,4.942,4.861,3.469,5.009,7.702,5.473,3.613,
   3.444,4.509,5.171,3.680,2.365,-4.959,5.030,4.815,
   4.564,4.224,4.426,4.471)
#-----------------------------Bubble Sort ------------------------------#
 r.bs <- function(x)
{
  n <- length(x) # better insert this line inside the sorting function
  for (k in n:2) # every iteration of the outer loop bubbles the maximum element
    # of the array to the end
  {
    i <- 1
    while (i < k)       # i is the index for nested loop, no need to do i < n
      # because passing j iterations of the for loop already
      # places j maximum elements to the last j positions
    {
```

```
    if (x[i] > x[i+1]) # if the element is greater than the next one we change them
    {
      temp <- x[i+1]
      x[i+1] <- x[i]
      x[i] <- temp
    }
    i <- i+1          # moving to the next element
   }
  }
  x            # returning sorted x (the last evaluated value inside the body
  # of the function is returned), we can also write return(x)
}


#------------------------------Insertion Sort----------------------------------------#
insertionsort_function <- function(A){
  for (j in 2:length(A)) {
    key = A[j]
    # insert A[j] into sorted sequence A[1,...,j-1]
    i = j - 1
    while (i > 0 && A[i] > key) {
      A[(i + 1)] = A[i]
      i = i - 1
    }
    A[(i + 1)] = key
  }
  A
}
#--------------------------Quick sort using Recursion -------------------------------#
qs <- function(vec)
{
```

```
 if(r.length(vec) > 1)
 {

   pivot <- vec[1]
   low <- qs(vec[vec < pivot])
   mid <- vec[vec == pivot]
   high <- qs(vec[vec > pivot])

   c(low, mid, high)

 }

 else vec

}
```

```
> x
 [1]  5.637  4.942  4.861  3.469  5.009  7.702  5.473  3.613  3.444  4.509
[11]  5.171  3.680  2.365 -4.959  5.030  4.815  4.564  4.224  4.426  4.471
> r.bs(x)
 [1] -4.959  2.365  3.444  3.469  3.613  3.680  4.224  4.426  4.471  4.509
[11]  4.564  4.815  4.861  4.942  5.009  5.030  5.171  5.473  5.637  7.702
> insertionsort_function(x)
 [1] -4.959  2.365  3.444  3.469  3.613  3.680  4.224  4.426  4.471  4.509
[11]  4.564  4.815  4.861  4.942  5.009  5.030  5.171  5.473  5.637  7.702
> qs(x)
 [1] -4.959  2.365  3.444  3.469  3.613  3.680  4.224  4.426  4.471  4.509
[11]  4.564  4.815  4.861  4.942  5.009  5.030  5.171  5.473  5.637  7.702
```

PRACTICAL 7

```
#-----------------------------------Practical 7 -------------------------------------#
#Obtain the MLE for the location parameter of cauchy distribution. Use the following
#to get your result
#(a) Newton Ralphson Method
#(b) Method of scoring
#
#
# 5.637941,4.942002,4.861254,3.469588,5.009333,
#7.702125,5.473228,3.613141,3.444167,4.509174,
#5.171716,3.680117,2.365371
# -4.959420,5.030187,4.815630,4.564628,4.224900,4.426912,4.471680

#---------------------------SOLUTION------------------------------------------
#Putting the value in x
x<-c(5.637941,4.942002,4.861254,3.469588,5.009333,
    7.702125,5.473228,3.613141,3.444167,4.509174,
    5.171716,3.680117,2.365371,-4.959420,5.030187,
    4.815630,4.564628,4.224900,4.426912,4.471680)

int_theta = median ( x )# consistent estimator of theta
calculate = function (a, samp )
  {
    new_a=0
    sum =0
    for (i in 1: length ( samp ))
```

```
      sum = sum +((( samp [i]-a)/ (1+( samp [i]-a) ^2) )*(2/ 15) )
      new_a=a +(2 * sum )
      return ( new_a)
   }
i=0
b=0
new_theta = int_theta
while ( round (b ,2) != round ( new_theta ,2) )
   {
     i=i+1
     b= calculate ( int_theta , x )
     cat (" Value of theta in iteration ",i," is : ",round (b ,4) ,"\n")
     if(b== int_theta )
       break
     else
       {
         new_theta = int_theta
         int_theta =b
         }
     }
```

```
 Value of theta in iteration 1  is :  4.0749
 Value of theta in iteration 2  is :  5.019
 Value of theta in iteration 3  is :  4.0749
 Value of theta in iteration 4  is :  5.019
 Value of theta in iteration 5  is :  4.0749
 Value of theta in iteration 6  is :  5.019
 Value of theta in iteration 7  is :  4.0749
```

Practical 9

```
#------------------------------Practical no. 9-------------------------------------#
#Fit Binomial distribution for following data also check  goodness of fit using
#chi sqauare goodness of fit test
#
# X | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8
# F | 5 | 9 | 22| 29| 36| 25| 10| 3 | 1
#

#----------------------------------SOLUTION------------------------------------#
#--------------------Basic function to run the program smoothly ----------------

#Basic function for length
r.length<-function(x)
{
  length<-0;
  for(i in x)
  {
    length<-length+1
  }
  as.numeric(length)
}
#Basic function for Sum
r.sum<-function(x)
{
  kum<-0;
  for(i in x)
  {
```

```
    kum=kum+i
  }
  as.numeric(kum)
}
#Basic function for Unique
r.uni<-function(x)
{
  uni<-NULL;
  while(r.length(x)!=0)
  {
    key<-x[1]
    y<-key==x
    uni<-c(uni,key)
    z<-x[!y]
    x<-z
  }
  uni
}
#Basic function for sorting (Using Quick sort)
qs <- function(vec)
{

  if(r.length(vec) > 1)
  {

    pivot <- vec[1]
    low <- qs(vec[vec < pivot])
    mid <- vec[vec == pivot]
    high <- qs(vec[vec > pivot])

    c(low, mid, high)
```

```
  }

  else vec

}
#Basic function for taking absolute
r.abs<-function(x)
{
  for(i in 1:r.length(x))
  {
    if(x[i]<0)
      x[i]<-(-x[i])
  }
  x

}




#----------------------------------fitting Poisson distribution----------------------------------------#
binomial <- function(x, p) {
  n<-x[r.length(x)]
  r<-r.length(x)-1
  probfn <- factorial(n)/(factorial(0:r)*factorial(n-(0:r)))*p^(0:r)*(1-p)^(n-0:r)
  return(probfn)
}

X <-c ( 0 , 1 , 2 , 3 , 4 , 5 , 6 , 7 , 8)
F <-c ( 5 , 9 , 22, 29, 36, 25, 10, 3 , 1)
```

mean<- r.sum(X*F)/r.sum(F)

mean<-mean/x[r.length(x)]

expected_freequency<-round(r.sum(F)*binomial(X,mean))

#--------------------------------- Chi square goodness of fit--#

chi_square<-r.sum(((F-expected_freequency)^2)/expected_freequency)

```
> expected_freequency
[1]  0  0  0  1  7 21 42 46 22
> chi_square
[1] Inf
```

PRACTICAL 10

#-----------------------------Practical no. 10------------------------------------#

```
#Fit Poisson distribution for following data also check  goodness of fit using
#chi sqauare goodness of fit test
#
# X | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8
# F |162|193|115| 83| 44| 24| 19| 8 | 2
#

#----------------------------------SOLUTION-----------------------------------#
#--------------------Basic function to run the program smoothly ----------------

#Basic function for length
r.length<-function(x)
{
  length<-0;
  for(i in x)
  {
    length<-length+1
  }
  as.numeric(length)
}
#Basic function for Sum
r.sum<-function(x)
{
  kum<-0;
  for(i in x)
  {
    kum=kum+i
  }
  as.numeric(kum)
}
```

```
#Basic function for taking factorial

r.factorial<-function(x)

{

 if(x==0)

   return(1)

  return(x*r.factorial(x-1))

}
```

```
#--------------------------------fitting Poisson distribution----------------------------------------#

poisson <- function(x, lambda) {

  probfn <- (exp(-lambda) * (lambda ^ x)) / factorial(x)

  return(probfn)

}


X <-c ( 0 , 1 , 2 , 3 , 4 , 5 , 6 , 7 , 8)

F <-c (162,193,115, 83, 44, 24, 19, 8 , 2)

mean<- r.sum(X*F)/r.sum(F)


expected_freequency<-round(r.sum(F)*poisson(X,mean))


#--------------------------------  Chi square goodness of fit----------------------------------------#

chi_square<-r.sum(((F-expected_freequency)^2)/expected_freequency)
```

```
> expected_freequency
[1] 110 196 174 103  46  16   5   1   0
> chi_square
[1] Inf
```